It Takes Two Neurons To Ride a Bicycle

Matthew Cook[*]

Abstract

Past attempts to get computers to ride bicycles have required an inordinate amount of learning time (1700 practice rides for a reinforcement learning approach [1], while still failing to be able to ride in a straight line), or have required an algebraic analysis of the exact equations of motion for the specific bicycle to be controlled [2, 3]. Mysteriously, humans do not need to do either of these when learning to ride a bicycle.

Here we present a two-neuron network[1] that can ride a bicycle in a desired direction (for example, towards a desired goal or along a desired path), which may be chosen or changed at run time.

Just as when a person rides a bicycle, the network is very accurate for long range goals, but in the short run stability issues dominate the behavior. This happens not by explicit design, but arises as a natural consequence of how the network controls the bicycle.

1 Introduction

The task of riding a bicycle presents an interesting challenge, whether for human or for computer. We do not have great insight as to how we ride a bicycle, and we do not have much useful advice for someone who is learning.

In fact, in the course of this project, I had the chance to ride a "virtual bicycle" on the computer, and I was surprised to find how counterintuitive it is. I had thought that, knowing perfectly well how to ride a bicycle in real life, it would be no problem in simulation. However, in real life there must be additional inertial cues that I sense or leaning actions that I make which are missing from the simulation, since I had to learn, as if from scratch, what cues to attend to and how to react to them. I even thought at first that there must be a bug in the simulator, since to turn right I found I had to push the handlebars to the left. Of course, if you stop to think about it, that is exactly correct. To turn right, the bicycle has to *lean* to the right, and the only way to make that happen[2] is to shift the point of contact with the ground to the left, which requires an initial push to the left. But then, once the bicycle is leaning to the right, it will itself push the handlebars to the right due to how it is constructed for stability,[3] with a force even greater than your initial push to the left, so maintaining a

[*]California Institute of Technology, Mail Stop 136-93, Pasadena, CA 91125 *cook@paradise.caltech.edu*

[1]Actually, the title of this paper is unproven. We have not ruled out the possibility that a single neuron could ride a bicycle.

[2]This is ignoring the torque effect due to the spinning of the front wheel, but if you take that into account, it too has exactly the same effect as the effect described above (pushing to the left makes you lean to the right).

[3]See footnote 7 on page 6.

constant gentle leftward push does indeed cause the bicycle to turn to the right. Similarly, to come out of the rightward turn (or even to maintain it), you need to push the handlebars gently to the *right*.

In this paper we outline the various portions of our project, which has led us to find a surprisingly competent two-neuron network. Different portions of this work are likely to be of interest to different people. The reader should feel free to skip over sections that are not of interest—the paper has been organized so that skipping ahead should not result in a loss of understanding when reading later sections.

2 Methodology: Overview of the Simulator System

2.1 The Physics

In order to allow us to experiment with different bicycle controllers, we first have to set up a virtual bicycle for them to control. The equations of motion for a bicycle are somewhat complex [2], so it seems no more complicated and much more useful to just write a general robot simulator, which can read a description of an arbitrary robot (rigid bodies linked by hinge-like connections), and simulate how that robot will move given the forces being applied to it. This entails calculating the moments of inertia for each rigid body, simulating the motion of a single rigid body given forces acting on it [5], and solving a system of equations at each step for how the hinge-like connections can apply forces to the parts of the robot so that the alignment and co-location requirements of the hinges are met.[4]

2.2 The Bicycle Robot

Once we have such a general purpose physics simulator, then we can turn to setting up a robot, in this case a bicycle. A bicycle is composed of four rigid bodies: the two wheels, the frame, and the front fork (the steering column). Each adjacent pair of parts is connected with a joint that allows rotation along a defined axis, and the wheels are connected to the ground by requiring that their lowest point must have zero height and no horizontal motion (no sliding).

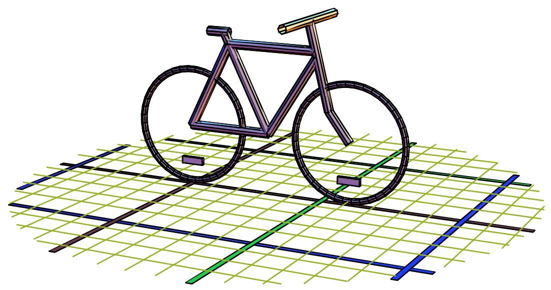

Figure 1: The virtual bicycle.

Beyond specifying the construction and connections that form the bicycle, we need to decide what sensory input should be available to the controller, and how the controller's outputs should be converted into forces on the bicycle (in robotics terms, what the sensors

[4]Even the support of the wheel by the ground counts as a hinge-like connection for this purpose.

and actuators should be for this non-holonomic under-actuated system). For our bicycle, we allow all the easily perceivable quantities to be available to an interested controller: Position, heading, speed, angle of the handlebars (and its rate of change), and the amount the bicycle is leaning (and its rate of change). For actuators, we allow a torque on the back wheel and a torque on the handlebars. Humans also make good use of leaning to one side or the other when they ride, but we will not have such a control on the riderless bicycle. Also, we do not allow the controller to know the specifics of the bicycle, such as its exact proportions or the masses of its parts.

2.3 The Controller

Once we have set up the robot bicycle, we can turn to the task of interest: Designing a controller for the bicycle. We want the controller to solve the same problem that a human solves when riding the bicycle. The human knows where they are, which way they are going, how fast they are going, how the bicycle is leaning, and so on, but as we know from experience the human does not need to know the specifics of the construction of the bicycle.

Here we are finally faced with a problem that we do not, *a priori*, know how to solve. So we stare at the ceiling for a while, and whenever we are struck with some inspiration, we quickly write a controller based on it.

There are three main styles of controller (prescient, human, and two-neuron) that have led us to interesting results or observations, and we will discuss them in the next three sections. None of them made significant use of the speed—they all managed to control the bicycle using just the handlebars. We will not discuss here those controllers which did nothing but crash the bicycle at every opportunity.

3 The Prescient Controller: A Look at Reinforcement Learning

One interesting idea for a controller, given that the entire system is being simulated, is to let the controller cheat by giving it access to the simulator. This could not be done with a controller for a bicycle in the real world, so it is not of interest for applications, but we can certainly try it in the simulated world to see what happens.

In particular, we can try the following algorithm for the controller: At each step, first simulate and compare three actions. The actions only differ in how the handlebars are pushed at the first instant: pushed left, pushed right, or not touched. The remainder of each of the three actions is to do nothing until the bicycle crashes. These three actions can then be compared on the basis of which one causes the bicycle to remain upright for the longest time, which one results in the most progress to the right, or whatever other criterion one decides to optimize. After simulating the results of the three actions, the controller decides what to do at this instant based on those results. (Each different criterion is thus the basis for a different controller.)

These simulations were tried with and without random mild forces (“wind”) being applied to the bicycle. The original motivation for this was so that the controller would not be able to rely on an absolutely perfect prediction of the future. It might also help the controller to have a more “continuous” behavior, since over the course of several consecutive instants, it would be getting a rough estimation of the probability distribution for success of each of its actions, leading to the controller taking a similarly distributed action. However, such wind turned out in fact to have no significant effect on the results.

In the language of reinforcement learning, such a controller is exactly what you would get after one step of policy iteration, if you start with the null policy of never touching the handlebars, and allow yourself three actions at each step (push left, push right, or no

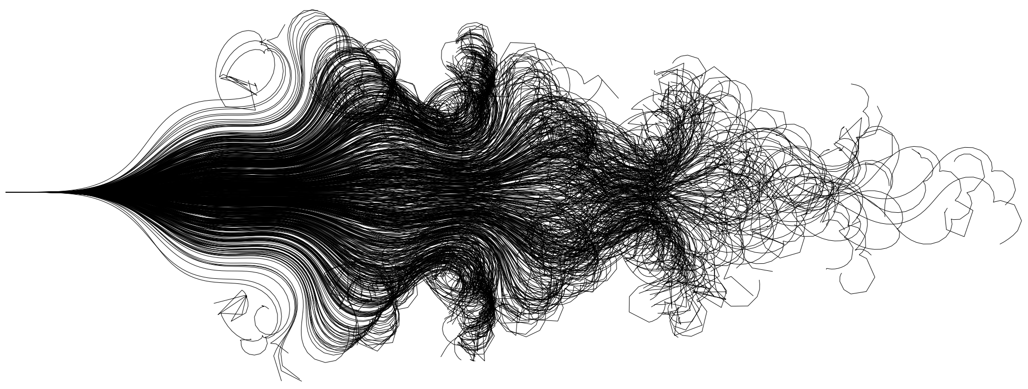

Figure 2: Instability of an unsteered bicycle. This shows 800 runs of a bicycle being pushed to the right. For each run, the path of the front wheel on the ground is shown until the bicycle has fallen over. The unstable oscillatory nature is due to the subcritical speed of the bicycle, which loses further speed with each oscillation.

push). Then if the controller learns the value function for this policy (which in practice would require lots of experience with not touching the handlebars, but which we simulate by giving the controller access to the simulator), it can then act greedily with respect to that value function. This amounts to one step of policy iteration, and at least for the goal of not falling over, an optimal policy is indeed obtained after a single iteration (i.e., it successfully doesn't fall down). However, it does not do this in a conventional way, say by riding in a straight line, but rather manages to maintain stability at near-zero speed by doing stunts with the front wheel, for example by spinning the handlebars in circles (the handlebars and front wheel do not bump into the frame for our bicycle, and there are no cables to get twisted, so why not?). A movie of this bizarre behavior can be seen at:
`http://www.paradise.caltech.edu/~cook/Warehouse/RecursiveBike.avi`

Despite many attempts at formulating a sensible value function, we found it difficult to get sensible behavior out of the bicycle. By rewarding uprightness, the bicycle would stop riding normally and start doing stunts as described above. If we tried to discourage this by rewarding speed, the bicycle would swoop from side to side, where each swoop results in a temporary increase in speed. If we tried to discourage this by rewarding going in a straight line, the bicycle would do this very nicely, but of course it would fall over right away, as avoiding the fall would have required deviating from the straight line. Of course, one could try weighted combinations of these or other ideas, but then the question starts to be not how long it will take the controller to learn to ride the bicycle, but how long it will take us to learn how to program the controller to get it to ride normally. As has been pointed out by people who have worked with reinforcement learning, it can be a very tricky business trying to pick a good value function.

Even if we had had marvelous success with this method, it would not be immediately applicable to a more realistic situation, due to its reliance on getting answers from the simulator to "what if?" questions, effectively having access to an oracle. However, it would have provided a hint that reinforcement learning could be used effectively. As it was, the hint we got was that reinforcement learning would have a hard time of it. This is also the hint we were getting from Randløv and Alstrøm's nice paper [1], where they tried various reinforcement learning methods to get their controller to learn to ride a bicycle, but even with the best methods it took thousands of practice rides to learn not to fall down, and thousands more to learn to ride towards a goal, and even after having "learned," the controller appeared to have a rather drunken behavior at best.

4 The Human Controller

One venerable method of learning is to be taught by an expert, perhaps by example. The obvious expert in this case would be a skilled human. To enable this, we had the computer present a real-time graphical display of the bicycle, allowing a human to use the keyboard to control the pedaling and the pushing of the handlebars. As for myself, this led to the experience described in the introduction (on page 1), where I discovered that controlling the bicycle is quite counter-intuitive.[5]

The human controllers who tried to learn to drive the virtual bicycle found subjectively that it was important to pay attention to the angle at which the bicycle was leaning, and to concentrate on manipulating this angle. This observation became the basis for the two-neuron controller described below. None of the humans became proficient at getting the bicycle to travel in a precise direction, a skill demonstrated nicely by the two-neuron controller. One cannot rule out the possibility that the humans might have gained this skill with further practice.

There are two ways in which we tried to use the human expertise. One way was by recording what was going on during one of their rides. Collecting this data and then analyzing it for pertinent correlations yielded surprisingly little in the way of useful information.

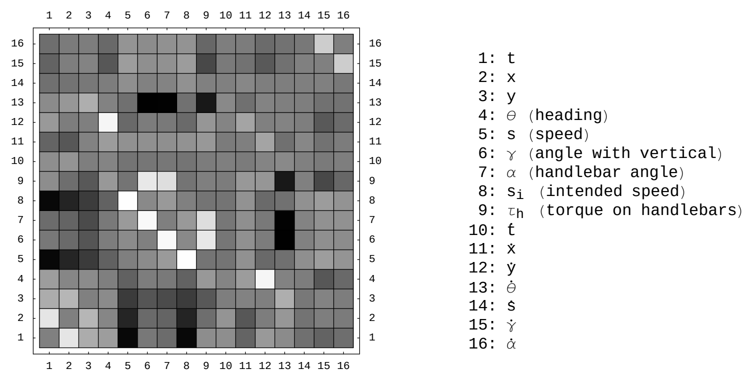

Figure 3: The covariance between many of the quantities available to the controller. The main diagonal has been set to zero so as not to be so visually distracting. There is good correlation between the controller's output τ_h (9) and the amount of turning $\dot{\theta}$ (13), which might seem to solve the problem immediately, but unfortunately the causality in this relationship goes the wrong way: the controller needs to adjust τ_h in response to γ to maintain stability, where γ (6) and $\dot{\theta}$ (13) happen to be strongly correlated. (This correlation between γ (6) and $\dot{\theta}$ (13) can be immediately understood based on the simplest physics of centrifugal force, but we do not want our controller to need knowledge of any physical laws.) The quite useful causal relation between τ_h (9) and $\dot{\gamma}$ (15) is visible, but there is little here to clue one in to its importance for successful control.

The other way in which we tried to use human expertise was by having the humans subjectively describe how they were trying to control the bicycle. Since they had just gone through the experience of having to learn how to do it, they were reasonably well able to describe

[5]This at least explains in part why it takes a certain amount of practice for a human to become a proficient bicycle rider in the real world.

what their algorithm was, and as it turned out they had developed similar techniques, based on carefully adjusting the angle at which the bicycle is leaning.

Based on the reports of the humans, the two-neuron network controller was implemented, and it worked almost immediately, having few parameters, and not being overly sensitive to any of them. This method is the subject of the next section.

5 The Two-Neuron Network

Here we present a two-neuron network which can operate the bicycle competently over a range of speeds. The output of the first neuron is fed into the second neuron, whose output is connected to an actuator which applies the specified amount of torque to the handlebars. As inputs to the network, we provide the desired heading θ_d, as well as the current heading θ and the degree to which the bicycle is currently leaning γ, along with their derivatives $\dot{\theta}$ and $\dot{\gamma}$.[6]

Due to the nature of the problem, we will use a network that is continuous both in time and in values. A unit's output is determined simply by a thresholding function of a weighted sum of the unit's inputs. If necessary, this can be interpreted as a mean-firing-rate model, but we will not explore issues of network realism here. Given that such a small network of this type suffices, there seems little doubt that more realistic networks could solve the problem as well.

The task for the network will be to make the bicycle travel in the desired direction. This can then be used by higher-level planning systems to make the bicycle head towards a goal, or to follow a path by heading towards a sequence of waypoints.

In order to set the bicycle's heading θ as desired, we need to be able to control $\dot{\theta}$. We know from figure 3 that $\dot{\theta}$ is strongly related to γ, the amount the bicycle is leaning, so we can try to control $\dot{\theta}$ indirectly by simply controlling γ.[7]

To control γ, we need control over $\dot{\gamma}$. And indeed, our actuator, which can exert a desired torque on the handlebars, happens to have reasonable control of $\dot{\gamma}$. There is not a direct or fixed correspondence, but as a general rule, during stable riding, a higher clockwise torque on the handlebars will cause the bicycle to start leaning more to the left. Thus, by setting the torque according to how we would like γ to change, we should be able to have γ converge towards its desired value. (Note that it doesn't make a big difference if the actual convergence is just towards some approximation of γ's desired value—the exact desired value is not critical for this method of control to succeed.)

The first neuron in our circuit will output the desired γ, with the nonlinearity being applied so that the bicycle doesn't try to lean too far over.[8] The second neuron in our circuit will output the desired torque to be applied to the handlebars.

The first neuron will take as inputs θ and θ_d (which one can assume to be within $\pm\pi$ of θ,

[6]Actually, as we will see, the network does not even need to use $\dot{\theta}$.

[7]One of the reasons that controlling γ works is due to the realistic bicycle geometry. Real bicycles are designed to be stable, which allows a rider to ride without holding the handlebars, simply by controlling the amount the bicycle is leaning. We note that one typical important factor in stability is that the axis of rotation of the front fork should pass below the hub of the front wheel but above its point of contact with the ground, a feature we have duplicated on the virtual bicycle.

[8]Although most of us do not have direct experience with this, bicycles can become quite unstable if they are in a state of extreme leaning. In real life, usually the wheels skid out from under us before this point is reached. In this simulator, skidding does not occur, but since this controller specifically avoids states of extreme leaning, it thereby avoids the problem entirely, regardless of whether the problem is that of slipping or that of becoming unstable.

although this is not essential), and simply calculate the desired change in heading $\theta_d - \theta$, multiply by a constant, apply a threshold, and then output the result, which we will denote by γ_d.

The second neuron will take as inputs γ_d, γ, $\dot{\gamma}$, and its own output, which we will denote by ϕ. It will calculate the amount by which γ should be changed, $\gamma_d - \gamma$, and compare that to a constant times the current rate $\dot{\gamma}$ at which γ is changing. The difference between these, how much $\dot{\gamma}$ should be adjusted by, is then scaled and sent as the controller's output τ_h of how much torque to apply to the handlebars.

$$\begin{aligned} \gamma_d &= \sigma(c_1\theta_d - c_1\theta) \\ \tau_h &= c_2\gamma_d - c_2\gamma - c_3\dot{\gamma} \end{aligned}$$

Figure 4: The equations for the two-neuron network. σ denotes a thresholding function. The three constants c_i need to be set by the implementor in light of the bicycle's stability characteristics, but the network's behavior is not too sensitive to their precise values, so it is actually quite easy to get a working network.

6 Results

The two-neuron network controller does remarkably well at controlling the bicycle, as we can see in figure 5.

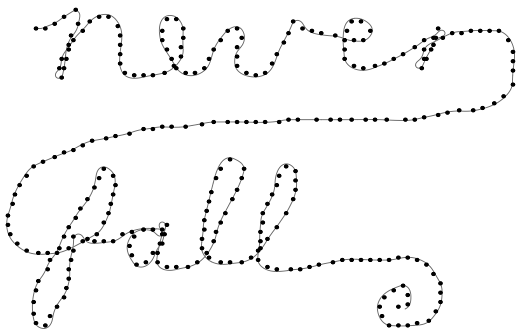

Figure 5: The path taken by the bicycle when told to aim at the successive waypoints shown. Each time the bicycle got within a certain distance of its current target waypoint, the target would change to become the next waypoint in the sequence. This distance can be seen as the distance between the last waypoint and the end of the bicycle's path shown. The irregularity in the writing is due to the author's messy handwriting when trying to write with a mouse, and is not the fault of the bicycle.

Although the two-neuron network controller works well for a range of speeds, one thing the controller does *not* do is to try to dampen the instabilities that can arise when riding too slowly or in too sharp of a turn. (This would probably require a third neuron that is dedicated to this task.)

7 Future Directions: More Automated Learning

The future directions of interest to us have to do with using this system to understand more about learning. There are other interesting future directions, such as building a real bicycle robot, which we probably will not pursue.

One obvious thing to do with this system is to have it learn and tune its parameters with experience. Ideally, if the system is placed on a slightly different bicycle, it should quickly[9] learn how to successfully operate the new bicycle.

This network was designed in an ad-hoc, if traditional, way. First, a human tried to control the bicycle with the simulator. After many attempts, the human finally became a somewhat skilled operator of the bicycle, able to avoid falling down but still not able to head reliably towards a desired goal. Nonetheless, the human at this point was able to describe the key parameters which were being attended to, and based on this, the two-neuron network was designed. The skill of this network immediately exceeded the human's skill.

However, we would like to take the human out of this loop. We would like the computer to be able to figure out on its own what simple network might work, using a minimum of experience (i.e., a minimum number of crashes before mastery of the bicycle), and using no detailed knowledge of the physical system. Our attempts so far have been statistically based, and have not been very successful. We feel that we need to have a causal model for what is observed, where the direction of causality is part of what is observed. Causal networks (belief propagation networks) might be a good representation for such knowledge, but the real issue is how to have such a network be automatically designed for us. It is our opinion that a general solution to this problem would have many applications.

Acknowledgments

I would like to thank Shuki Bruck and Erik Winfree for many helpful discussions. This work was supported in part by the "Alpha Project" that is funded by a grant from the National Human Genome Research Institute (Grant No. P50 HG02370).

References

[1] *Learning to Drive a Bicycle using Reinforcement Learning and Shaping*, Jette Randløv and Preben Alstrøm, PROCEEDINGS OF THE FIFTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, (ISBN:1-55860-556-8), 1998, pp. 463–471.

[2] *Control for an Autonomous Bicycle*, Neil H. Getz and Jerrold E. Marsden, IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1995.

[3] *Descriptor Predictive Control: Tracking Controllers for a Riderless Bicycle*, D. von Wissel, R. Nikoukhah, F. Delebecque, and S. L. Campbell, PROC. COMPUTATIONAL ENGINEERING IN SYSTEMS APPLICATIONS, Lille, France, 1996, pp. 292–297.

[4] *Steering Control System Design and Implementation of a Riderless Bicycle*, Chi-Da Chen, and C. C. Tsai, JOURNAL OF TECHNOLOGY, vol. 16, no. 2, pp. 243-251 (July 2001). NSC89-2213-E-005-052 [Note: I have been unable to locate a copy of this paper, but I am including the information I have on it here anyway.]

[5] *Accurate and Efficient Simulation of Rigid Body Rotations*, Samuel R. Buss, JOURNAL OF COMPUTATIONAL PHYSICS, vol. 164, no. 2, pp. 377–406 (November 2000).

[9] Before falling over even once, one would hope.