Final Report: Longboard-Riding Robot

Matthew Keeter

6.832, Spring 2012

Introduction

This report presents a simple dynamics model for a longboard-riding robot and a controller that allows it to push itself across a flat surface. In addition, we show the result of stochastic gradient descent on controller parameters, which produces a faster but more erratic controller.

System Model

The primary motivation for this project was to investigate pushing and gliding, so the chosen model ignores some of the complex system dynamics involved in a full humanoid. The model has two legs: The standing leg rests on the board and the pushing leg swings back and forth or pushes against the ground.

There are two main simplifications in the legs: We transform the shin and thigh of the standing leg into a vertical beam which can change its y position (but is always vertical), and the pushing leg as a straight beam with a spring as its bottom half.

This model has three state variables: x, y, and α. These state variables are shown in Figure 1A. The state variable x represents distance travelled (with $\dot{x}$ representing forward velocity); y represents hip height; α represents leg swing angle.

The model has two actuators, shown in Figure 1B. The first actuator is a hip height actuator; as mentioned above, this is an abstraction of foot and knee in the standing leg. This actuator exerts a force F on the hip joint in the y direction. The second actuator exerts a torque τ on the swinging leg.

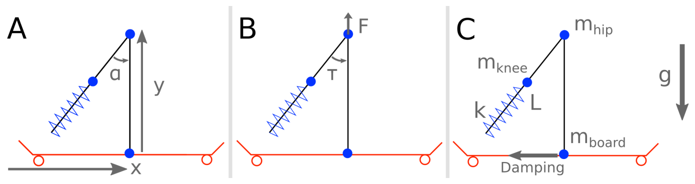


Figure 1: State variables (A), control inputs (B), and parameters (C) of the model system. Note that $\alpha < 0$ in this example state.

The model system is a hybrid system; it switches between gliding and pushing modes.

The mode is determined by the location of the toe: if y_{toe} is below ground level, then the system switches into pushing mode; if y_{toe} is above the ground then it switches into gliding mode. In pushing mode, x_{toe} is locked at $x_{\text{toe}} = 0$ to simplify dynamics.

To summarize, the model system has state variables $\mathbf{q} = [x, y, \alpha]'$ and co-located actuators $\mathbf{u} = [F, \tau]'$ causing accelerations at y and α, respectively.

Equations of Motion

The equations of motion for this system were generated using Lagrangian dynamics[1]. Separate derivations were necessary for gliding and pushing modes. In both modes, the velocities of each mass contribute to kinetic energy, and the y position of each mass contributes to potential energy.

In pushing mode, there is extra potential energy due to compression of the pushing leg's spring. In addition, a virtual angular spring is used to enforce straightness of the pushing leg. This spring penalizes differences between knee and hip angles, with a spring constant of $100k$.

Software tools were developed to automatically generate equations of motion from a Lagrangian L, state vector $\mathbf{q}$, and control input vector $\mathbf{u}$. For each state variable q_i and co-located actuator input u_j, the Lagrangian equation[2]

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = u_j$$

was generated and stored (where $u_j = 0$ for a state if q_i lacks a co-located actuator).

This results in a system of equations which can be solved by a computer algebra system. For this project, Sage[3] was used. The equations were solved twice: once for accelerations $\ddot{x}$, $\ddot{y}$, and $\ddot{\alpha}$; and once for the variables $\ddot{x}$, F, and τ.

The first set of solutions gives the system's dynamics. The second set of solutions allows for partial feedback linearization. The solution set is in terms of $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{y}$, and $\ddot{\alpha}$. Since $\mathbf{q}$ and $\dot{\mathbf{q}}$ are known, the desired values for $\ddot{y}$, and $\ddot{\alpha}$ can be plugged in, resulting in required values of F and τ to achieve those accelerations.

The tools developed for this project automatically export dynamics and PFL functions to MATLAB scripts. This allows for rapid model prototyping and development.

Controller Design

The system controller was designed using prior knowledge, based on self-observation while riding. It switches between four stages of motion, shown in Figure 2.

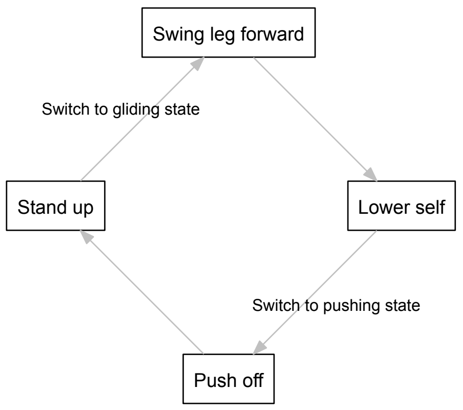


Figure 2: The four stages of motion for the system's controller.

Partial feedback linearization (PFL) is used at each of these stages, reducing control to that of the double integrator. Linearized accelerations are limited to $\ddot{y} \leq \ddot{y}_{\max}$ and $\ddot{\alpha} \leq \ddot{\alpha}_{\max}$; note that this is not equivalent to a force/torque limit on the actuators, as the required controller inputs to match the desired acceleration also depend on $\mathbf{q}$ and $\dot{\mathbf{q}}$.

The system is parameterized by three values: α_{hit} is the angle at which the leg collides with the ground; α_{stand} is the angle at which the robot begins to stand up, and $\ddot{\alpha}_{\text{swing}}$ is the angular acceleration as the robot swings its leg forward.

The following paragraphs give details on system behavior in each stage.

Swing leg foward: In this stage, the controller swings the leg forward to an angle α_{start}. This angle is chosen so that when swung downward, the toe will touch the ground with $\dot{x}_{\text{toe}} = -\texttt{max}(0.1, 1.2\dot{x})$; this encourages the system to match or slightly exceed the ground's velocity. The value of α_{start} is restricted to be less than $\pi/3$, to produce more human movement.

The controller enters the next state when $|\alpha - \alpha_{\text{start}}| < 0.01$.

Lower self: In this state, the leg is accelerated at $\ddot{\alpha} = -\ddot{\alpha}_{\max}$. Hip force F is controlled using PFL and double integrator control so that the swinging foot touches the ground with $\alpha = \alpha_{\text{hit}}$, $\dot{y} = 0$. The controller enters the next state when $y_{\text{toe}} < 0$.

Push off: While pushing, the leg continues to be accelerated with $\ddot{\alpha} = -\ddot{\alpha}_{\max}$ and height is maintained at $y = L\cos\alpha_{\text{hit}}$. The controller enters the next state when $\alpha < \alpha_{\text{stand}}$.

Stand up: While standing up, the leg is controlled to $\alpha = -\alpha_{\text{hit}}$, $\dot{\alpha} = 0$, $y = L$, $\dot{y} = 0$ using the same double integrator control as before. The controller enters the next state when $|y - L| < 0.01$.

Controller Performance

Figure 3 shows time-slices of the controller with parameters $\alpha_{\text{hit}} = \frac{\pi}{5}$, $\alpha_{\text{stand}} = -\frac{\pi}{10}$, $\ddot{\alpha}_{\text{swing}} = 75$ rad/sec^2. These values were chosen empirically based on experimentation.

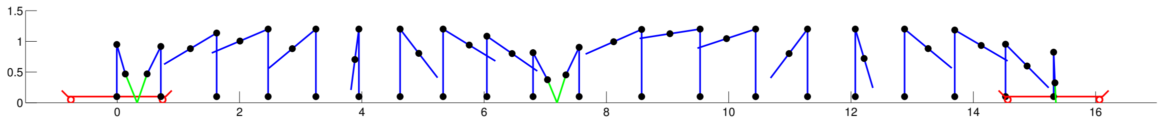
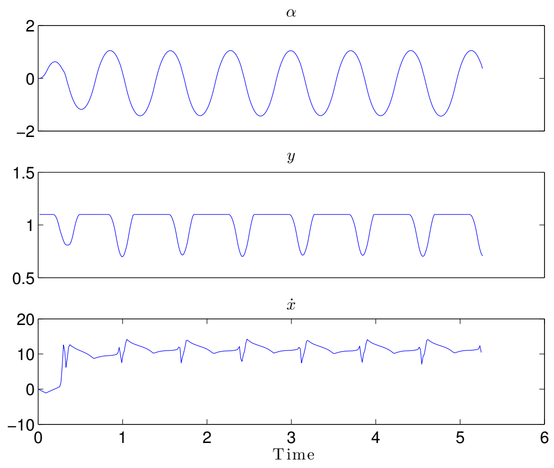


Figure 3: Time slices of trajectory.

Selected state trajectories are plotted in Figure 4. It appears to show the robot settling into a periodic trajectory.



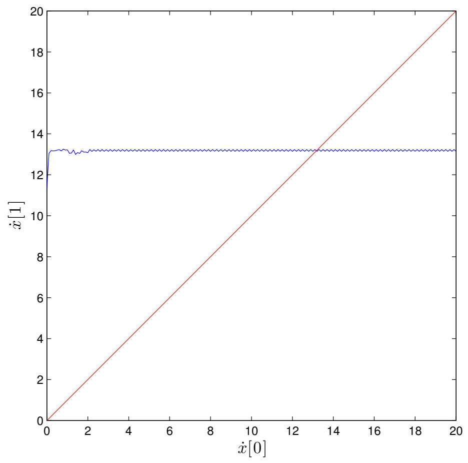


Figure 4: State trajectories

Figure 5: Slice of return map

To examine stability, a slice of the velocity return map was generated. For a range of $\dot{x}[0]$ values, the system was initialized at state $\mathbf{q} = [0, L, -\alpha_{\text{hit}}]'$, $\dot{\mathbf{q}} = [\dot{x}[0], 0, 0]'$ and simulated until completion of the "stand up" control stage, at which point $\dot{x}[1]$ was recorded. The return map is shown in Figure 5; it shows a controller that is very nearly dead-beat with respect to velocity.

Controller Optimization

The controller was optimized using stochastic gradient descent (SGD) over the three parameters α_{hit}, α_{stand}, and $\ddot{\alpha}_{\text{swing}}$. The "score" for an evaluation was distance travelled, where higher scores are better.

For a given set of parameters $\mathbf{p}[i]$, we calculated distance travelled for $\mathbf{p}[i]$ and $\mathbf{p}[i]+\mathbf{D}$, where $\mathbf{D}$ is sampled from the vector of normal distributions $\left[\mathcal{N}(0,\frac{1}{5}),\mathcal{N}(0,\frac{1}{5}),\mathcal{N}(0,2)\right]'$. On every iteration, $\mathbf{p}[i+1]$ was calculated as $\mathbf{p}[i] \pm 0.2\mathbf{D}$ (where 0.2 is the learning rate), depending on whether $\mathbf{D}$ improved or worsened performance.

SGD often pushed the controller into surprising states, producing unusual trajectories where the robot would spin its leg up and over its head. Pathological cases (e.g. where the hip dipped below the ground) were penalized with a distance score of 0. Score and parameter values are shown in Figures 6 and 7.

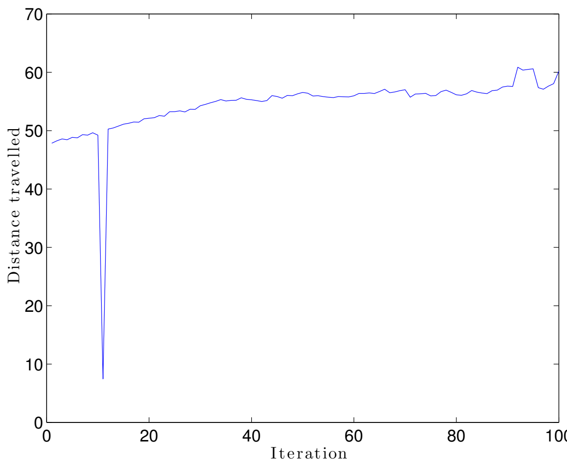


Figure 6: SGD score

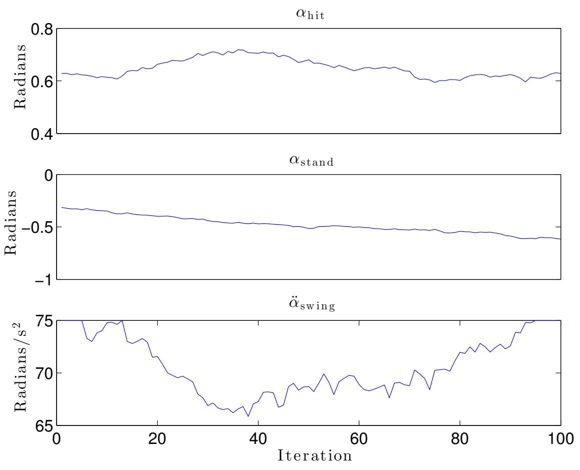


Figure 7: SGC parameter values

The final trajectory is shown in Figure 8; this trajectory travels 25% farther than the original in the same amount of time. The trajectory involves the back leg swinging high in the air; though this is not plausible for a human, it is perfectly feasible for the model. It would not be difficult to enforce a more human trajectory by tweaking the score function.

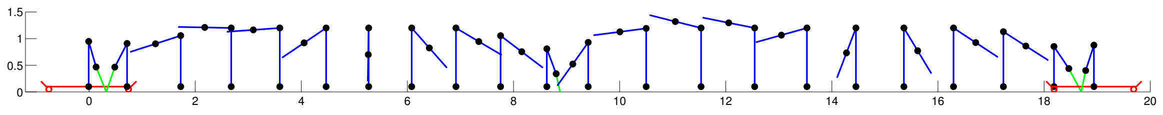


Figure 8: Trajectory after optimization

The state trajectories (Figure 9) and return map (Figure 10) for the optimized controller, are less regular than for the original controller, which was more conservative.

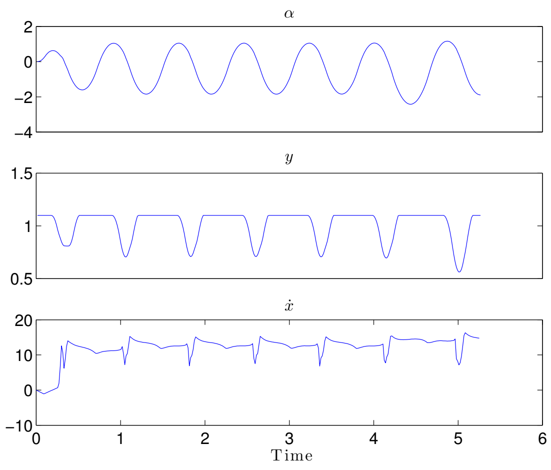


Figure 9: State trajectories for optimized controller

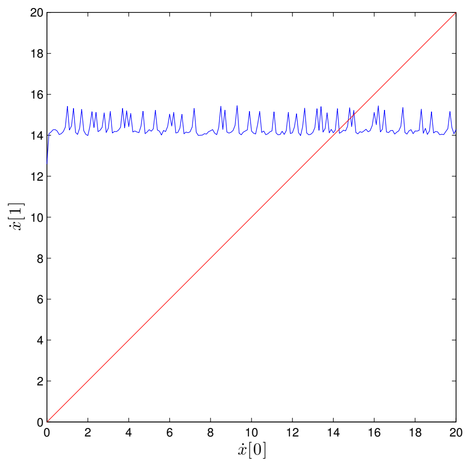


Figure 10: Return map for optimized controller

Conclusions

Throughout the course of this project, I developed a simplified system model for a longboard-riding robot, wrote tools to generate equations of motion, designed high-level control behavior, and used gradient descent to optimize the controller parameters.

Generating the equations of motion was surprisingly difficult. In particular, I had difficulty enforcing the straightness of the pushing leg until I came up with the idea of adding an extra angular spring.

Developing the controller was fairly simple, as I had an high-level idea of what it needed to do. Using feedback linearization made the control straight-forward, although it doesn't necessarily enforce realistic limits on force and torque.

Using SGD to optimize the controller parameters worked without too much effort, although I had to add special cases to catch and discard controllers that produced pathological behavior.

Overall, I am satisfied with how this project turned out. Though I didn't apply any of the more advanced optimization tools, my simple controller performed surprisingly well and appears to be robust.

References

[1] D. Morin, “The Lagrangian Method” in *Introduction to Classical Mechanics*, Cambridge University Press, 2008.

[2] R. M. Murray et al., “Robot Dynamics and Control” in *A Mathematical Introduction to Robotic Manipulation*, CRC Press, 1994.

[3] W. A. Stein et al., *Sage Mathematics Software (Version 4.8)*, The Sage Development Team, 2012, `http://www.sagemath.org`.