Improved Dynamic Stability Using Reinforcement Learning

R. Tedrake
Brain & Cognitive Sciences Department, MIT, Cambridge MA, USA
Leg Laboratory, MIT, Cambridge MA, USA
H.S. Seung
Brain & Cognitive Sciences Department, MIT, Cambridge MA, USA
Howard Hughes Medical Institute, Cambridge MA, USA

ABSTRACT

Many researchers studying legged locomotion have applied tools from reinforcement learning/optimal control to minimize characteristics of a walking gait, most notably the energy consumption. In this paper, we use similar tools to maximize the region of stability of the controller - defined as the set of initial conditions from which the robot maintains its balance for at least 5 seconds. Experiments were run on a *simulation* of a planar one-legged hopping robot. After a large number of iterations, the 'learned' controller is able to maintain balance from a much larger region of initial conditions than the original controller proposed by Raibert[1].

1 INTRODUCTION

Despite recent advances in walking and running robots, legged systems in biology continue to outperform our robots in terms of energetic efficiency and their ability to cope with rough terrain. Many researchers believe that this success can be attributed to the ability to learn. Learning systems have been applied successfully to dynamically stable legged robots in simulation[2,3,4] and occasionally to physical robots[5,6]. In these experiments, the system is typically trained from a small range of initial conditions, and it is unclear how the learned system would behave in very different configurations. In this paper, we address the issue of robustness. By rewarding the robot for keeping its balance from a large range of initial conditions, we show that a very simple learning system can actually uncover a surprisingly 'stable' controller.

In the biological motor control literature, stability has been proposed as a guiding principle for motor system design[7] and motor learning[8]. To study these issues in the context of locomotion, we focus on a simulation of the planar one-legged hopping robot (Figure 1) introduced by Raibert et al[1,9]. This simple legged system captures the essence of dynamic

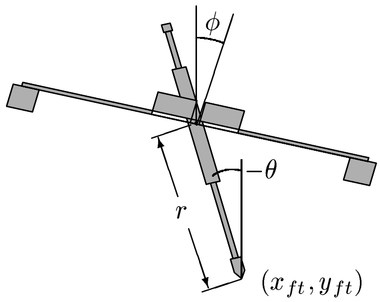

Fig. 1 The planar one-legged hopper. (x_{ft}, y_{ft}) represents the position of the point foot in world coordinates. θ represents the absolute angle of the leg, ϕ represents the absolute angle of the body, and r represents the length of the leg.

stability but avoids the complication of coordinating multiple legs or solving complex leg kinematics. Just as a cat always lands on its feet, the goal here is to make the planar hopper balance (dynamically) from a huge range of initial conditions.

This is not the first work to combine stability and optimization. In [10], energy optimization is performed over a parameter range in which the controller is provably stable. This approach yields powerful theoretical results, but the focus is on guaranteeing stability in a small region around the steady-state trajectory, not maximizing the region of stability. In [11], they describe an optimal feedforward control law for stable hopping in the spring-loaded inverted pendulum model. In this paper, we find a feedback controller for the complete hopper model.

In the following sections, we will outline a procedure for optimizing a neural network feedback controller to maximize the region of stability of the planar one-legged hopping robot. We define this region as the set of initial conditions from which the robot maintains its balance for at least 5 seconds. Using this very simple notion of stability and relatively naive optimization techniques, we are able to obtain a surprisingly 'stable' controller.

2 SIMULATION

The planar one-legged hopping robot (Figure 1) is simulated as a rigid body and a springy leg that pivot around a pin-joint at the hip. Both the body and the leg have considerable mass, although the moment of inertia of the body around the hip is 15 times larger than that of the leg. The leg is actually modelled as a position actuator in series with the main leg spring and also a very stiff spring with damping that acts as a mechanical stop. The controller commands a torque at the hip and the velocity of the position actuator in the leg. All together, the model has 5 state variables $\mathbf{q} = [x_{ft}, y_{ft}, \theta, \phi, r]^T$ and 2 control variables $\mathbf{u} = [u_0, u_1]^T$. More details can be found in [1]. Notice that our u_1 is equivalent to $\dot{u}_1$ in the original paper. The velocity of the leg actuator necessary to produce thrust has a much lower spatial frequency (over the state space), and is consequently a much easier feedback controller to learn.

Using the original control equations provided by [1], when the robot is initialized with $|\phi| >> 0$, it falls down. The large moment of inertia of the body serves to minimize the effect of the swing leg during the aerial phase of the steady-state trajectory, but consequently complicates the problem of dealing with disturbances around the hip. The original control equations were based on an analysis of steady-state hopping conditions augmented with simple linear feedback, and they simply can't recover from these disturbances. [12] reported improved robustness using a more comprehensive model of hip displacements, suggesting that a nonlinear controller may improve the situation.

3 OBJECTIVE FUNCTION: REWARDING STABILITY

The feedback controller to be learned is a function $\pi(\mathbf{w}) : \mathbf{x} \rightarrow \mathbf{u}$, where $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$ and the vector w contains the parameters of the controller. We would like to reward control parameters that maintain the robot's balance from a large range of initial conditions. This can be done by simulating the robot from a set of initial conditions, $\mathbf{X}_0$, and accumulating the total amount of time that it took the robot to fall down. The objective is to select the $\mathbf{w}$ that maximizes the reward function,

$$R(\mathbf{w}, \mathbf{X}_0, T) = \sum_{\mathbf{x}_0 \in \mathbf{X}_0} \int_0^{\min(t_f(\mathbf{w}, \mathbf{x}_0), T)} 1 dt,$$

where T is the finite-horizon time, and $t_f(\mathbf{w}, \mathbf{x}_0)$ is the time that the center of mass of the robot passes through the ground using controller $\mathbf{w}$ and starting from the initial condition $\mathbf{x_0}$. In practice, this is a hard function to optimize, because explicit gradient information is only available for control states where the robot is about to fall down. This is a well-known problem in reinforcement learning and is often treated with *reward shaping*[13]. In reward shaping, we add terms to the objective function that provide more direct gradient information for all configurations of the robot. The actual function that we optimize is

$$R(\mathbf{w}, \mathbf{X}_0, T) = \sum_{\mathbf{x}_0 \in \mathbf{X}_0} \int_0^{\min(t_f(\mathbf{w}, \mathbf{x}_0), T)} \left[1 - \gamma(\phi^2 + (y_{com} - d_{height})^2 + (\dot{x}_{com} - d_{hvel})^2\right]^+ dt$$

where $[z]^+ = \max(z, 0)$, $\phi \in [-\pi, \pi]$ is the body angle in radians, d_{height} is the desired average vertical position (approximately half the hopping height), d_{hvel} is the desired horizontal velocity, and $\gamma > 0$ is a scalar which weights the contribution of the shaping terms. Strictly, the terms added to the objective function during reward shaping should not change the optimal policy[13]. Although we are only applying reward shaping in a loose sense, we do select γ so that the contributions of the shaping terms are relatively small $(\gamma = 0.05)$.

In the examples in this paper, we use a finite-horizon time, T, of 5 seconds. This is enough time for the hopper to land and hop at least 5 times. $\mathbf{X}_0$ represents 50 to 100 points selected from the distribution described in section 5.3. Since our focus is simply on balance, not forward locomotion, we set $d_{hvel} = 0$ m/s, and $d_{height} = 2.0$ m to encourage a hopping solution.

This objective function should be compared to the indicators of dynamic instability typically used in walking systems, namely the zero-moment point (ZMP) and the foot-rotation indicator (FRI)[14]. The value function (also known as the cost-to-go function) associated with Equation 1, taken over the entire state space and with an infinite horizon, would represent the amount of time until the robot falls down given the current controller and the current state. Both ZMP and FRI could be considered elegant analytical approximations of this function, which work for walking robots with feet. Minimizing Equation 1 numerically is more general in the sense that it is well-defined for the current system, which has a point foot and an aerial phase.

4 CONTROLLER REPRESENTATION

The feedback controller $\pi(\mathbf{w})$ is represented using a small 3-layer neural network with 9 inputs, 25 hidden units, and 2 outputs. The inputs are $(y_{ft}, \theta, \phi, r, \dot{x}_{ft}, \dot{y}_{ft}, \dot{\theta}, \dot{\phi}, \dot{r})$, scaled and shifted linearly so that most inputs are within the range $[-1.0, 1.0]$. The outputs are (u_0, u_1), which are computed from the network using the standard sigmoid activation function and then linearly scaled and shifted from the range [0.0,1.0] to the desired actuator range. The parameter vector $\mathbf{w}$ contains the connection strengths between layers and the threshold biases for each unit, a total of 302 values.

This representation is very common for standard function approximation. Many robot learning papers, though, use a more local representation of the state space[15,16]. In our experiments, the local representations required a much larger number of parameters to achieve similar training errors. The number of simulations that we perform during our optimization scales proportionally with the size of the parameter space, so we opted to have fewer parameters and a global representation.

5 NONLINEAR OPTIMIZATION

The task is to find the control parameter vector $\mathbf{w}$ that maximizes the reward function $R(\mathbf{w}, \mathbf{X}_0, T)$ for some $\mathbf{X}_0$ selected randomly from a distribution of initial conditions (section 5.3) and for some finite-horizon time T. Our optimization procedure is composed of two parts - a standard algorithm for finding local maxima, and a set of heuristics for initializing the local search near an acceptable solution. To find the local maxima, we use the Downhill Simplex method[17,section 10.4]. This particular algorithm was selected because we do not have analytical gradient information and because it can be implemented efficiently with a distributed algorithm. Whenever possible, simulations of the robot were run simultaneously on all of the machines in our lab.

To initialize the search, we combine a backpropagation algorithm (section 5.1) with a heuristic linear search (section 5.2). We have also observed experimentally that shorter finite-horizon time problems seem to have less local maxima. To capitalize on this, we train the controller over a slowly expanding horizon time, until we reach the desired $T = 5$ seconds. The following is an algorithm that slowly but reliably converges on a controller that is good in practice:

- Pretrain the robot to mimic the modified (autonomous) Raibert equations
- Select 50 random initial conditions
- For $T = 0.5 : 0.25 : 4.75$

- Run heuristic linear search in 75 directions.
- Run the downhill simplex algorithm with very loose convergence criteria

- Run the downhill simplex algorithm with $T = 5$, and force it to truly converge

5.1 Pretraining Algorithm

Although we hope to improve upon the control equations presented in [1], we can certainly use those equations as a starting point. The pretraining algorithm uses the backpropagation algorithm to initialize the network controller to resemble the Raibert controller.

The neural network feedback controller is autonomous, but the control equations that we wish to mimic are based on a state machine, making them dependent on time. It turns out that the most important states of the machine can be easily identified by observing the state. Therefore, it is possible to approximate the state machine controller using an autonomous feedback controller. We simply consider the robot to be in the FLIGHT phase when the foot is above the ground $(y_{ft} > 0)$, in the COMPRESSION phase when the foot is on the ground and the leg is compressing ($y_{ft} < 0$ and $\dot{r} < 0$), or in the THRUST phase any other time the leg is compressed ($r < r_{s0}$, where r_{s0} is the rest length of the mechanical stop in the leg). This allows us to write the following autonomous control equations:

$$u_0 = \begin{cases} k_{fp}\left[\theta + \arcsin\left(\frac{\dot{x}T_s}{2r} + \frac{k_{\dot{x}}(\dot{x} - \dot{x}_d)}{r}\right)\right] + b_{fp}\dot{\theta} & \mathit{FLIGHT} \\ k_{att}\left(\frac{\theta}{2} - \phi\right) - b_{att}\dot{\phi} & \mathit{COMPRESSION}, \mathit{THRUST} \\ 0 & \mathit{otherwise} \end{cases}$$

$$u_1 = \begin{cases} v_{retract} & \mathit{FLIGHT} \\ v_{thrust} & \mathit{COMPRESSION} \\ 0 & \mathit{otherwise} \end{cases}$$

In my simulation, this autonomous approximation to the state-machine controller works very nearly as well as the original equations.

5.2 Heuristic Linear Search

Our line minimization is designed to perform a coarse search in control parameter space for desirable basins of attraction. It is completely independent from the downhill simplex algorithm. The method works by sampling the line in discrete intervals and then evaluating the reward function R at each of those points. We then low-pass filter the results and select the $\mathbf{w}$ which achieves the maximum reward in a small neighborhood around the maximum of the filtered curve. The idea is to encourage maxima with broad basins of attraction. Not only will our hill-climbing algorithms perform best in these regions, we are also rewarding control strategies that are less sensitive to the exact control parameters.

Evaluating all of the sampled points along the line is a luxury that we have because we can evaluate the reward function in parallel on many machines across the lab. This brute force approach is not as susceptible to local minima as bracket- or gradient-based methods.

5.3 Initial Conditions

Due to the large moment of inertia of the body relative to the hip, the robot is most sensitive to initial conditions where $|\phi| >> 0$. For this reason, random initial conditions for ϕ were selected from a uniform distribution over $[-\pi, \pi]$ radians. Similarly, the internal angle between the body and the leg $(\phi - \theta)$ was selected from a uniform distribution over $[-\pi/2 + 0.25, \pi/2 - 0.25]$ radians. The remaining state variables were set to ensure that the center of mass of the robot was stationary at the point $x = 0$ m, $y = 2.0$ m. These random initial conditions correspond to the robot being dropped from 2.5m with arbitrary leg and body angles.

6 RESULTS

After a very large number of simulations ($\approx 10^6$), the learning algorithm had converged on a surprising controller. Because our simulation did not model collisions between the body and the leg, the final controller recovered from large hip angles by spinning the body nearly 360 degrees around the hip to return to vertical.

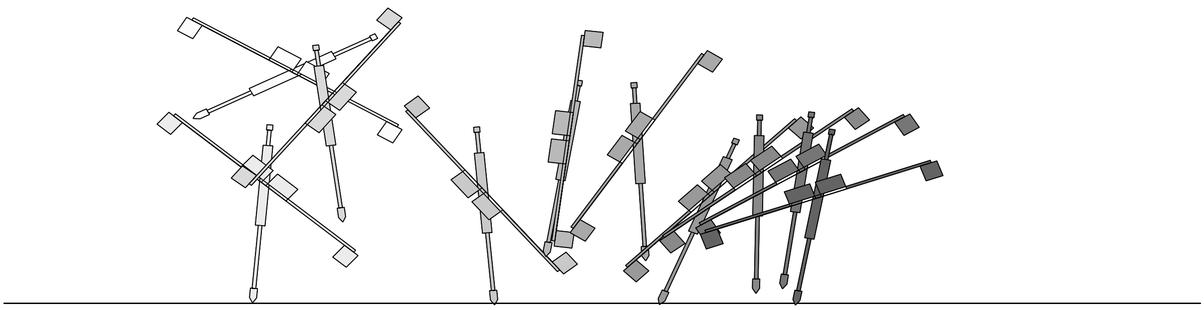

Fig. 2 This figure illustrates the hopper recovering from an initial condition (white fill at the top left) where both the hip and leg angles are very far from vertical. The shading of the robot gets darker as time progresses.

The goal of the optimization was to maximize the region of initial conditions from which the robot can recover. For comparison, Figure 3 plots the region of initial conditions for the two controllers.

7 DISCUSSION

Using brute force optimization techniques, we were able to train a very small neural network controller that notably expanded the region of stability of the hopping robot. This was accomplished by evaluating the robot from a relatively small sample (50-100) of random initial conditions taken from a very broad distribution. Although we did not attempt to optimize any energetic criteria, nor to drive the robot along a specific trajectory, these could easily be taken

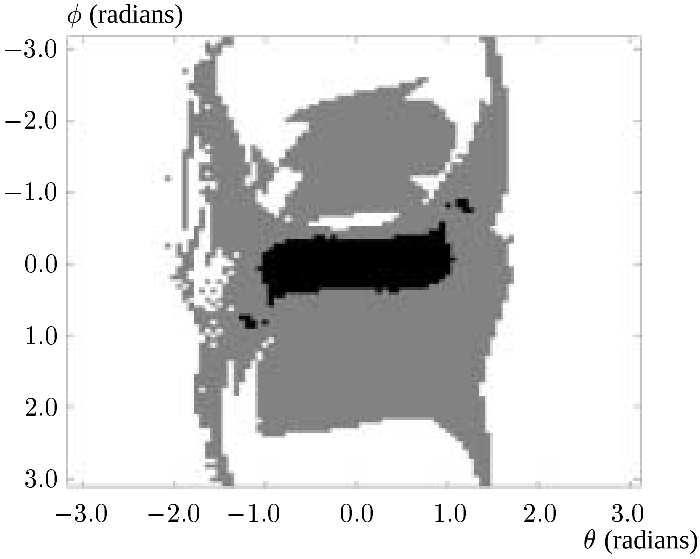

Fig. 3 Regions of stability for the optimized controller $\pi(\mathrm{w})$ in gray, and for the original hopper control equations in black.

into account by adding terms to the reward function. Furthermore, none of the algorithms presented here depend on the robot model and therefore could be applied directly to a simulation of a different robot.

The absence of specific stability criterion and of trajectory constraints in the reward function actually allowed the learning algorithm to generate some 'creative' results. Although the robot did not fall down in the first 5 seconds, it was not possible (in either case) to describe the solution as asymptotically stable to a single nominal trajectory. The controller may actually be more stable because it was not limited to a form of stability with strict theoretical properties.

Learning to recover from a large range of initial conditions may assist our robots in locomoting on rough terrain. A first approximation to rough terrain walking might involve being as stable as possible on flat ground and to simply treating the terrain as a disturbance. In addition, by coding the controller as a highly parameterizable function, we provide the opportunity for adjusting the controller online as we experience rough terrain. Finding an initial satisfactory controller was computationally very expensive, but adjusting that controller to account for small changes in the model may be something than can be done easily online.

In the coming months, we plan to experiment with more elegant solutions to this optimization problem using the value function approximation tools that are more characteristic of the reinforcement learning approach. Using these tools, we hope to extend our results to online learning, and in the near future to begin experimenting on a real robot.

8 CONCLUSIONS

Stability is clearly a concern for any controller that is automatically generated using a learning algorithm. By optimizing over a large range of initial conditions, we were able to generate a neural network controller that actually increased the region of stability for the simulated planar one-legged hopping robot. We hope that this work paves the way for more elegant learning systems that emphasize the importance of having a large region of stability.

ACKNOWLEDGEMENTS

This work was supported by the David and Lucille Packard Foundation (contract 99-1471) and by the National Science Foundation (grant CCR-0122419).

REFERENCES

[1] Marc H. Raibert. Hopping in legged systems: Modeling and simulation for the 2d one-legged case. *IEEE Trans. Systems, Man, and Cybernetics*, 14:451–463, 1984.
[2] Jih-Gau Juang and Chun-Shin Lin. Gait synthesis of a biped robot using backpropagation through time algorithm. pages 1710 – 1715. IEEE International Conference on Neural Networks, 1996.
[3] Russell Smith. *Intelligent Motion Control with an Artificial Cerebellum*. PhD thesis, University of Auckland, New Zealand, 1998.
[4] Chee-Meng Chew. *Dynamic Bipedal Walking Assisted by Learning*. PhD thesis, Massachusettes Institute of Technology, 2000.
[5] W. Thomas Miller, III. Real-time neural network control of a biped walking robot. *IEEE Control Systems Magazine*, 14(1):41–48, Feb 1994.
[6] Hamid Benbrahim and Judy A. Franklin. Biped dynamics walking using reinforcement learning. *Robotics and Autonomous Systems*, 22:283–302, 1997.
[7] J.J.E. Slotine andW. Lohmiller. Modularity, evolution, and the binding problem: A view from stability theory. *Neural Networks*, 14(2), February 2001.
[8] Emanuel Todorov. Optimal feedback control as a theory of motor coordination. *Submitted to Nature Neuroscience*, 2002.
[9] Marc H. Raibert. *Legged Robots That Balance*. The MIT Press, 1986.
[10] E.R. Westervelt and J.W. Grizzle. Design of asymptotically stable walking for a 5-link planar biped walker via optimization. International Conference on Robotics and Automation, 2002.
[11] Andre Seyfarth, Hartmut Geyer, Michael G¨unther, and Reinhard Blickhan. A movement criterion for running. *Journal of Biomechanics*, 35(5):649–655, May 2002.
[12] J. Vermeulen, D. Lefeber, and H. De Man. A control strategy for a robot with one articulated leg hopping on irregular terrain. pages 399–406. CLAWAR, Professional Engineering Publishing, 2000.
[13] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. Proceedings of the Sixteenth International Conference on Machine Learning, 1999.
[14] A. Goswami. Postural stability of biped robots and the foot rotation indicator (fri) point. *International Journal of Robotics Research*, 18(6), 1999.
[15] W.T. Miller III, F.H. Glanz, and L.G. Kraft III. Cmac: an associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567, 1990.
[16] Stefan Schaal, Christopher Atkeson, and Sethu Vijayakumar. Scalable techniques from nonparametric statistics for real-time robot learning. *Applied Intelligence*, 2001.

[17] William H. Press, Saul A. Teukolsky,William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.
[18] M.H. Raibert and H.B. Brown Jr. Experiments in balance with a 2d one-legged hopping machine. *Journal of Dynamic Systems, Measurement, and Control*, 106:75–81, March 1984.