Robotica: (2021) volume 39, pp. 107–122. © Cambridge University Press 2020
doi:10.1017/S0263574720000211

Control Policies for a Large Region of Attraction for Dynamically Balancing Legged Robots: A Sampling-Based Approach

Pranav A. Bhounsule†*, Ali Zamani†, Jeremy Krause†, Steven Farra‡ and Jason Pusey¶

†*Department of Mechanical and Industrial Engineering, University of Illinois at Chicago, 842 W Taylor St, Chicago, IL 60607, USA. E-mails: pranav@uic.edu, alizamani.mecheng@gmail.com, robotics68@gmail.com*
‡*Department of Mechanical Engineering, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA. E-mail: zvo618@my.utsa.edu*
¶*Vehicle Technology Directorate, U.S. Army Research Laboratory, Aberdeen Proving Grounds, Aberdeen, MD 21001, USA. E-mail: jason.l.pusey.civ@mail.mil*

(Accepted March 26, 2020. First published online: May 5, 2020)

SUMMARY
The popular approach of assuming a control policy and then finding the largest region of attraction (ROA) (e.g., sum-of-squares optimization) may lead to conservative estimates of the ROA, especially for highly nonlinear systems. We present a sampling-based approach that starts by assuming an ROA and then finds the necessary control policy by performing trajectory optimization on sampled initial conditions. Our method works with black-box models, produces a relatively large ROA, and ensures exponential convergence of the initial conditions to the periodic motion. We demonstrate the approach on a model of hopping and include extensive verification and robustness checks.

KEYWORDS: Region of attraction; Orbital Lyapunov function; Poincaré map; Periodic motion; Dynamically balancing legged robots; Deep learning neural nets.

1. Introduction

Dynamically balancing legged robots are characterized by a small footprint and thus have to constantly move to stay balanced. The most well-studied approach to controlling these robots is to first find a periodic trajectory,[1] and second, develop a feedback control policy to stabilize the periodic motion. Also, one may estimate the set of system states that converge back to the periodic motion, known as the region of attraction (ROA), to provide formal stability certificates.

The conventional approach for finding the ROA of a periodic motion is shown in Fig. 1(a). Given a periodic motion, a linear control policy (e.g., linear quadratic regulator (LQR)[2]) is used for stabilization. The use of a linear control policy allows one to use convex optimization tools such as sum-of-squares optimization to find a Lyapunov function and the corresponding ROA.[3] One issue with the approach is that a linear control policy may be too restrictive and lead to a small ROA, this is especially true for highly nonlinear systems.

We propose a different approach that involves inverting the conventional approach and is shown in Fig. 1(b). We assume a candidate Lyapunov function and an ROA (typically a big region) and then

* Corresponding author. E-mail: pranav@uic.edu

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

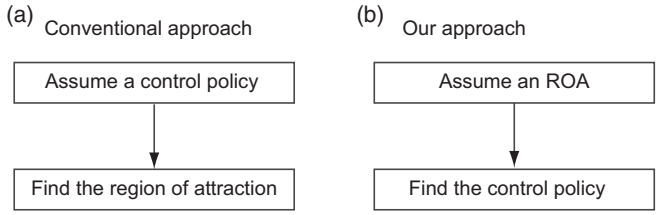


Fig. 1. Types of control approaches: (a) The conventional approach assumes a control policy (usually a linear policy) followed by optimization to find the largest ROA (e.g., using sum-of-squares optimization to find the largest level set for the Lyapunov function). (b) In our approach, we assume an ROA followed by optimization to find the control policy (a higher order polynomial or neural network control policy gives good results for nonlinear systems). Our method uses sample-driven trajectory optimization followed by nonlinear regression, which provides an enlarged ROA even for highly nonlinear systems.

find a control policy (usually a nonlinear one) by performing trajectory optimization for sampled initial conditions. We demonstrate that a higher order polynomial or a neural network-based control policy is able to guarantee the ROA. Another advantage of our method, specifically when applied to periodic systems, is that we recast the trajectory tracking problem into a regulation problem by defining the ROA at the Poincaré section, thus keeping the method computationally inexpensive. The net result is an enlarged ROA with reasonable computational cost. Once the ROAs are computed for multiple cyclic or steady-state gaits, they may be sequenced together by reasoning about overlapping regions to create nonsteady or agile gaits using heuristics[4] or more systematic motion planning algorithms such as rapidly exploring random trees.[5]

The flow of the paper is as follows. First, we give some background and related work on the ROA for legged systems in Section 2. Our overall approach is presented in Section 3 including the model of hopping, which is our test example. The results are in Section 4, followed by the discussion in Section 5, and finally the conclusion in Section 6.

2. Background and Related Work

The most simple dynamically balancing legged robots are those created by McGeer in the early 1990s.[6,7] McGeer showed that by tuning the natural dynamics (mass distribution, geometry, foot shape, etc.), it is possible to get natural-looking walking motion on a shallow incline. This concept is known as passive dynamic walking. Garcia et al.[8] simplified McGeer's model by using a point mass hip, massless feet, and straight legs. The resulting model, called the simplest walker, has a single free parameter, the ramp slope. They demonstrated that as the ramp slope increased, the model displayed period-doubling leading to chaos, a phenomenon seen in many other nonlinear dynamical systems.[9]

Although passive dynamic walkers are energetically efficient (energy cost of movement is zero), they are extremely fragile and knocked down by the slightest disturbance. For a dynamical system, the set of initial conditions that converge back to the periodic motion is called the basin of attraction (BOA)[A]. Schwab and Wisse[10] used the cell mapping method[11] to find the BOA of the simplest walker. The cell mapping method relies on extensive sampling and forward simulations to estimate the BOA of the system. First, the state space is divided into cells. Then initial conditions are chosen in the middle of the cells. For each initial condition, the system is simulated to check if it converges back to the periodic motion. The process is repeated for all initial conditions chosen on the cells. The resolution of the cells determines the accuracy of the method. A finer resolution increases accuracy but also increases the computational cost. It was found that the BOA for the simplest walker is very narrow, which is understandable given that the system is completely passive. A similar brute force approach was used by Heim and Sprowitz[12] to find the BOA of the spring-loaded inverted pendulum (SLIP) model of hopping. They have shown that a nonlinear spring increases the BOA over the linear spring model. However, unlike the passive dynamic walkers, the SLIP model uses active control of the foot placement angle before touchdown.

A broader definition of stability is captured by the viability theory, which defines the viability kernel defined as the set of all states from which it is possible to avoid falling down.[13] Unfortunately,

[A]ROA is an estimate of the BOA.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

it is computationally challenging to find this set even for simplest legged systems. A computationally tractable approach was taken by Pratt et al.[14] by defining a capture region, which is the region where the robot needs to step to come to a complete stop in one or more steps. The use of simple models of locomotion such as linear inverted pendulum model enables easy computation of the capture region and subsequent implementation in hardware.[15] Zaytsev et al.[16] did a broader analysis using a simple model of locomotion that generalizes capture regions to a broader set of targets rather than just standing still and in the spirit of viability theory and computes the states and controllers that avoid falling.

A global method for finding control policy and ROA is to use dynamic programming. Dynamic programming returns a global policy and the value function or the cost of executing the control policy. The issue with this approach is the method suffers from the curse of dimensionality, that is, as the system complexity grows, the storage and computations grow exponentially. The common way to deal with this issue is to reduce the system to a simple abstraction, also known as a template,[17] and use it for controller design. For example, Whitman[18] simplified a humanoid robot into three simple independent models: a sagittal, a lateral, and a frontal model. Controllers were developed for the separate models independently but combined during implementation using time as the phase variable. Another simplification approach taken by Mandersloot et al.[19] is to use dynamic programming to choose states and control actions at the Poincaré section instead of the time trajectory.

Simulation-based tabulation of controllers followed by the online implementation is one way of getting past the computational burden. Raibert[20] used simulations to tabulate the controls as a function of the system state. Then, the polynomial surface was used to fit the table. This allowed efficient storage of the table and for real-time implementation. In a similar vein, Da et al.[21] used simulation to create controllers as functions of system states and terrain height. The data were then inputted into a supervised learning framework to learn a policy, which was then implemented in hardware. Our method is similar to the latter; we generate state, control pairs using optimization followed by learning a function from the states to the controls using regressors such as a deep neural network.

Local control methods involve creating a local control policy for the stabilization of the periodic motion. For example, Tedrake et al.[2] used an LQR to stabilize the periodic motion. Then, using the LQR cost as a value function in combination with sum-of-squares optimization,[3] they found a Lyapunov function and the corresponding ROA. The work by Manchester et al.[22] split the dynamics into transverse and tangential components and then searched a Lyapunov function and ROA on the transverse dynamics using sum-of-squares optimization. These approaches lead to the creation of the ROA as a tube along the trajectory.

In our work, we are primarily interested in stabilizing a periodic motion. Our work is different from past work in several ways.

1. We define the Lyapunov function and the ROA at the Poincaré section, called as the discrete or the orbital Lyapunov function.[23] Thus, we are solving a regulation problem, and the ROA is a single surface of dimension $n - 1$ (n is the number of states of the system) at the Poincaré section compared to the tracking problem and the ROA for t samples along the trajectory, which will be $n \times t$ states which define a tube along the trajectory (see Tedrake[2]).
2. The control policies we find allow exponential convergence to the periodic motion using an orbital Lyapunov function. Previous methods based on LQR generate asymptotic convergence to the periodic motion, which is relatively slower.
3. Our method can work with black-box models of the system to find a nonlinear control policy. This is in contrast to past approaches that need a specific structure in the model and/or controllers (e.g., polynomial models for sum-of-squares optimization).
4. Our method is able to find a relatively larger ROA compared to past approaches but at the expense of finding a more complex control policy (e.g., nonlinear control policy).

3. Methods

The details of our approach are shown in Fig. 2 and are elaborated in this section. First, as shown in Fig. 2(a), we define preliminaries needed for analysis of cyclic gaits. Second, as shown in Fig. 2(b), we assume an ROA and use an exponential control orbital Lyapunov function within a trajectory optimization framework for controller design. Third as seen in Fig. 2(c), the results of trajectory

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

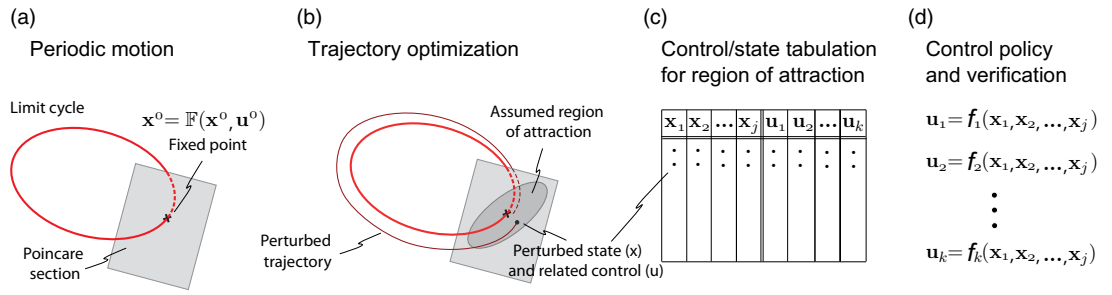


Fig. 2. Overview of our sampling-based optimization approach: (a) A gait that is periodic repeats itself in one or more steps, (b) an ROA is assumed at the Poincaré section and control actions (set points, gains, amplitudes, etc) are found that lead to exponential convergence to the periodic motion at the section, (c) trajectory optimization is repeated for sampled points in the ROA to generate a look-up table for initial states and corresponding control actions, (d) regression is performed to fit a control policy for each control action as a function of state followed by verification.

optimization on samples chosen in the ROA are tabulated. Finally, as shown in Fig. 2(d), we obtain a control policy from the tabulated data and followed by extensive verification of the control policy by simulating the system under external disturbances and modeling errors.

3.1. Periodic motion

In this paper, we are interested in a one-step periodic motion, which is a movement pattern that repeats itself after a single step (see Fig. 2(a)). A Poincaré section is an instant in the gait cycle (e.g., mid-stance and foot-strike). A Poincaré map $\mathbb{F}$ is a function that maps the state at the Poincaré section to itself after one step and is given by

$$\mathbf{x}^{i+1} = \mathbb{F}(\mathbf{x}^i, \mathbf{u}^i), \tag{1}$$

where i is the step number, $\mathbf{x}^i$ are the states at step i, and $\mathbf{u}^i$ are the control actions at step i. In order to find a periodic motion, for a given nominal state $\mathbf{x}^0$, there is a corresponding nominal control $\mathbf{u}^0$ such that

$$\mathbf{x}^0 = \mathbb{F}(\mathbf{x}^0, \mathbf{u}^0). \tag{2}$$

3.2. Trajectory optimization

First, we introduce the stability metric called the orbital Lyapunov function, and then we formulate the trajectory optimization problem.

3.2.1. Exponential convergence using control orbital Lyapunov function. We define a Lyapunov function (V) at the Poincaré section as follows:

$$V(\Delta\mathbf{x}^i) = (\Delta\mathbf{x}^i)^T \mathbf{S}_0 \Delta\mathbf{x}^i = (\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0), \tag{3}$$

where $\mathbf{x}^0$ is the fixed point of the periodic motion, $\mathbf{x}^i \neq \mathbf{x}^0$ is a system state at the Poincaré section that needs to be stabilized, and $\mathbf{S}_0$ is a positive definite matrix that determines the shape of the ROA. The condition for exponential stabilization is

$$V(\Delta\mathbf{x}^{i+1}) - V(\Delta\mathbf{x}^i) \leq -\alpha V(\Delta\mathbf{x}^i), \tag{4}$$

where $0 < \alpha < 1$ is the rate of decay of the Lyapunov function between steps. Thus, the condition for exponential stability can be rewritten in terms of control using Eqs. (3) and (4) as follows:

$$\begin{aligned} & V(\Delta\mathbf{x}^{i+1}) - (1-\alpha)V(\Delta\mathbf{x}^i) \leq 0, \\ \Longrightarrow \quad & (\mathbf{x}^{i+1} - x^0)^T \mathbf{S}_0 (\mathbf{x}^{i+1} - \mathbf{x}^0) - (1-\alpha)(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) \leq 0, \\ \Longrightarrow \quad & \big(\mathbb{F}(\mathbf{x}^i, \mathbf{u}^i) - \mathbf{x}^0\big)^T \mathbf{S}_0 \big(\mathbb{F}(\mathbf{x}^i, \mathbf{u}^i) - \mathbf{x}^0\big) - (1-\alpha)(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) \leq 0. \end{aligned} \tag{5}$$

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

Equation (5) is the condition on the orbital Lyapunov function for exponential orbital stabilization (step-to-step stabilization). Specifically, we select $\mathbf{u}_i$ such that the above condition is met. The variable α is set to 0.9 in all simulations. The rationale is that a value of 0.9 gives a good compromise between rate of convergence and robustness to modeling errors (see ref. [23] for more details).

3.2.2. Trajectory optimization problem formulation. The trajectory optimization is done for a given initial condition at the Poincaré section $\mathbf{x}^i \neq \mathbf{x}^0$ (see Fig. 2(b)) as follows:

$$\underset{\mathbf{u}^i}{\text{minimize}} \quad \text{Mechanical cost of transport (MCOT)} = \frac{\text{Mechanical work per step}}{\text{Weight} \times \text{step length}} \tag{6}$$

$$\text{subject to:} \quad \mathbf{x}^{i+1} = \mathbb{F}(\mathbf{x}^i, \mathbf{u}^i) \tag{7}$$

$$\left(\mathbb{F}(\mathbf{x}^i, \mathbf{u}^i) - \mathbf{x}^0\right)^T \mathbf{S}_0 \left(\mathbb{F}(\mathbf{x}^i, \mathbf{u}^i) - \mathbf{x}^0\right) - (1-\alpha)(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) \leq 0. \tag{8}$$

In the above problem, the control actions $\mathbf{u}^i$ are control parameters that are set once per step. Some examples of control actions are feedback gains, set points, and amplitude of suitable time-based functions. All these control actions, the number and type of actions, are designer's choice (also see Fig. 4).

The optimization problem defined by Eqs. (6)–(8) may be solved using parameter optimization using collocation or shooting methods (for a review, see ref. [24]). We use a direct shooting method in all optimizations.

3.3. Control/state tabulation for the ROA

The ROA of the controller is the set of all initial conditions $\mathbf{x}^i$ that would converge to the fixed point, $\mathbf{x}^0$. As noted earlier, in our method, we start by assuming an ROA. Then, we sample the ROA to generate a set of initial conditions $\mathbf{x}^i$ as follows. We choose a level set of the Lyapunov function, $(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) = c$, where c is a constant and has a small value to start with. We choose n equally spaced points on this level set. This process is repeated for multiple level sets of increasing value for c to generate the set of initial conditions $\mathbf{x}^i$. The control/state tabulation is done as follows. For each initial condition $\mathbf{x}^i$, we solve the trajectory optimization problem given in Section 3.2.2 to obtain the corresponding control action $\mathbf{u}^i$. The control/state pairs are saved in a tabular format (see Fig. 2(c)).

3.4. Control policies and verification

Based on the control/state tabulation, we are interested in fitting a control law for each control action as follows:

$$u_k = f_k(x_1, x_2, x_3, \ldots, x_J) \qquad \text{k} = 1, 2, 3, \ldots, \text{K}, \tag{9}$$

where k denotes the index for each control action (total K actions), and j indicates the index for each state (total J states). The function f is a designer's choice. We use linear, quadratic, and a neural networks in our results and compare the fit with each other by doing additional simulation using additional randomly chosen initial conditions. In addition, we simulate the system to disturbances and modeling errors to check the robustness of the control policy (see Fig. 2(d)).

3.5. Model of hopping

We demonstrate our method on a model of hopping shown in Fig. 3(a). The model consists of a point mass body with mass $m = 80\,\text{kg}$ and maximum leg length $\ell_0 = 1\,\text{m}$. Gravity points downwards and is denoted by $g = 9.81\ \text{m/s}^2$. There is a prismatic actuator that can generate an axial force F along the leg and a hip actuator that can place the swing leg at an angle θ.

The states of the model are given by $\{x, \dot{x}, y, \dot{y}\}$, where x and y are the x- and y- position of the center of mass and $\dot{x}$ and $\dot{y}$ are the respective velocities. A single step of the hopper shown in Fig. 3(a)–(d) is given by the following equation:

$$\underbrace{\text{Flight} \overset{\text{apex}}{\longrightarrow} \text{Flight} \overset{\text{touchdown}}{\longrightarrow} \text{Stance compression} \longrightarrow \overset{\text{mid-stance}}{\longrightarrow} \text{Stance restitution} \longrightarrow \overset{\text{takeoff}}{\longrightarrow} \text{Flight}}_{\text{one step/ one-step periodic motion}} \overset{\text{apex}}{\longrightarrow} \text{Flight.}$$

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

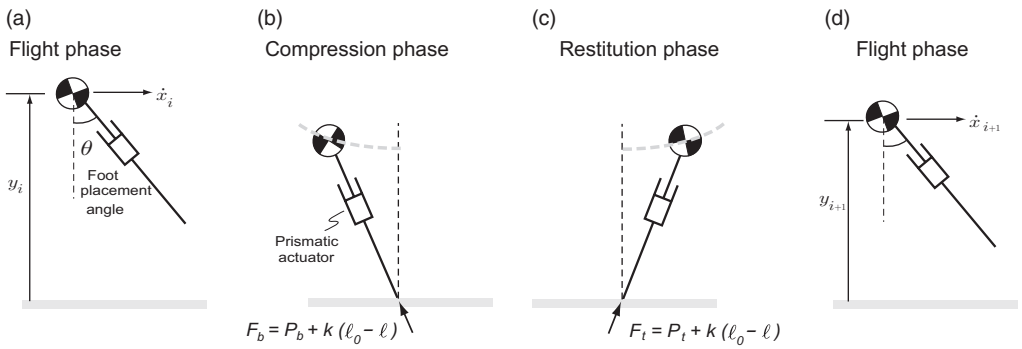


Fig. 3. A complete step for the hopping model: The model starts in the flight phase at the apex position (vertical velocity is zero), followed by the stance phase, and finally ending in the flight phase at the apex position of the next step. The hopping model has a prismatic actuator that is used to provide an axial braking force F_b in the compression phase and axial thrust force F_t in the restitution phase, and a hip actuator (not shown) that can place the swinging leg at an angle θ with respect to the vertical as the leg lands on the ground.

We explain the above equation in detail next. The model starts at the apex (see Fig. 3(a)) where the state vector is $\{x_i, \dot{x}_i, y_i, 0\}$. The model then falls under gravity

$$\ddot{x} = 0, \qquad \ddot{y} = -g, \tag{10}$$

till contact with the ground is detected by the condition $y - \ell_0 \cos(\theta) = 0$, where θ is the foot placement angle and measured relative to the vertical. Thereafter, the ground contact interaction is given by (see Fig. 3(b) and (c))

$$m\ddot{x} = F\frac{(x - x_c)}{\ell}, \qquad m\ddot{y} = F\frac{y}{\ell} - mg, \tag{11}$$

where x_c is the ground-foot contact point that needs to be set at every step depending on the ground-foot contact point at touchdown, and $F > 0$ is the linear actuator force along the leg. For the first half of the stance phase from touchdown to mid-stance (defined by $\dot{y} = 0$[B]), called the compression phase, the actuator force is a braking force $F = F_b = P_b + k(\ell_0 - \ell)$. For the second half of the stance phase from mid-stance to takeoff, called the restitution phase, the actuator force is a thrust force $F = F_t = P_t + k(\ell_0 - \ell)$. P_b and P_t are constant control forces during compression and restitution, respectively[C]. In the above equations, $\ell = \sqrt{(x - x_c)^2 + y^2}$ is the instantaneous leg length measured relative to the contact point and k is constant (fixed) gain analogous to the spring constant. In all simulations, we take $k = 32,000$ N/m. The takeoff phase occurs when the leg is fully extended, that is, $\ell - \ell_0 = 0$. Thereafter, the model has a flight phase and ends up in the next apex state, $\{x^{i+1}, \dot{x}^{i+1}, y^{i+1}, 0\}$ (see Fig. 3(d)).

Figure 4 shows the block diagram of our controller. There are two control loops, an inner loop that does fast continuous control and an outer loop that does slow once-per-step control. The inner loop has a tracking controller that tracks the swing leg angle during flight phase as a function of time and the stance leg force during stance phase as a function of leg length. The slow loop sets the foot placement angle (θ^i) and the stance leg force ($P^i = \{P_t, P_b\}$) once per step. This paper focusses exclusively on the outer, slow controller.

4. Results

4.1. Periodic motion and trajectory optimization for sampled points in the assumed ROA

We define the Poincaré map, $\mathbb{F}$, at the apex. The apex is defined by the condition, $\dot{y} = 0$. Thus, the state at the apex at step i, $\mathbf{x}^i = \{\dot{x}^i, y^i\}$, and the control actions that are tuned once per step as discussed in

[B]We could also have used a slightly different condition $\dot{\ell} = 0$.

[C]The notion of compression and restitution is used with embellishment because during the compression phase there could be a net positive work due to $P_b\dot{\ell} > 0$ and similarly during restitution phase there could be a net negative work due to $P_t\dot{\ell} < 0$, but the effect would be very small for hopping considered here.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

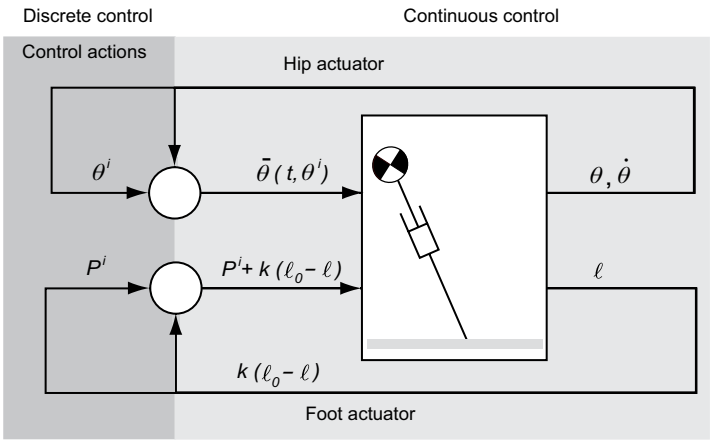


Fig. 4. Block diagram of the controller: There are two control loops. A high-bandwidth continuous-control inner loop that does a time-based position control of the swing leg and force control of the stance leg, and a low-bandwidth (once per step) discrete-control outer loop that sets the foot placement angle (θ^i) and thrust/braking force ($P^i = \{P_t, P_b\}$).

Section 3.5, is $\mathbf{u}^i = \{\theta^i, P_b^i, P_t^i\}$ (where θ^i, P_b^i, and P_t^i are foot placement angle with respect to vertical, the constant forces during compression and restitution, respectively, see Eq. (11)). The step-to-step dynamics are given by $\mathbf{x}^{i+1} = \mathbb{F}(\mathbf{x}^i, \mathbf{u}^i)$. We do not have an analytical expression for $\mathbb{F}$, but we build a simulation of the system using MATLAB's ordinary differential equation solver *ode113* with an integration tolerance of 10^{-9}.

We arbitrarily chose the apex state for the periodic motion to be $\mathbf{x}^{i+1} = \mathbf{x}^i = \mathbf{x}^0 = \{5.0, 1.3\}$ (units of m/s and m, respectively) and numerically evaluated the control $\mathbf{u}^i = \mathbf{u}^0 = \{\theta^0, P_b^0 = 0, P_t^0 = 0\}$ necessary to ensure periodic motion given by the condition $\mathbf{x}^0 = \mathbb{F}(\mathbf{x}^0, \mathbf{u}^0)$. The control necessary to achieve the periodic motion is $\mathbf{u}^0 = \{0.3465, 0, 0\}$

Next, for the trajectory optimization, we chose the positive definite matrix for the Lyapunov function to be $\mathbf{S}_0 = diag\{1, 11.1\}$, exponential convergence rate $\alpha = 0.9$, and the ROA to be $(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) = 1$. These are design variables, and the rationale for these choices is as follows: The S_0 in combination with the ROA $(\mathbf{x}^i - \mathbf{x}^0)^T \mathbf{S}_0 (\mathbf{x}^i - \mathbf{x}^0) = 1$ ensures that speed variations within ± 1 m/s and height variation ± 0.3 m may be stabilized. The choice for α determines how fast the initial condition at the apex decays to the periodic motion. For $\alpha = 0.9$, all initial conditions would reduce by a factor of 90%.

We use the energy metric called the MCOT defined as energy used per unit weight per unit distance travelled

$$\begin{aligned} \text{MCOT} &= \text{MCOT}_k + \text{MCOT}_b + \text{MCOT}_t \\ &= \frac{E_k}{mgD} + \frac{E_b}{mgD} + \frac{E_t}{mgD} \\ &= \frac{\int |k(\ell_0 - \ell)\dot{\ell}|dt}{mgD} + \frac{\int |P_b \dot{\ell}|dt}{mgD} + \frac{\int |P_t \dot{\ell}|dt}{mgD}, \end{aligned} \tag{12}$$

where $|x|$ is the absolute value of x, D is the step length, and $\dot{\ell} = \frac{(x - x_c)\dot{x} + y\dot{y}}{\ell}$. The absolute value is a non-smooth function of its argument, so we smooth it using square-root smoothing.[25] That is, $|x| = \sqrt{x^2 + \epsilon^2}$, where ϵ is a small number (we set $\epsilon = 0.01$).

Finally, the optimization problem given by Eqs. (6)–(8) is solved from initial conditions sampled on the ROA and described in detail in Section 4.2.

4.2. Control/state tabulation

We generated 451 data points within the ellipse given by $(\mathbf{x}_i - \mathbf{x}_0)^T \mathbf{S}_0 (\mathbf{x}_i - \mathbf{x}_0) = 1$ (see Section 4.2) and performed the trajectory optimization for each sampled data point (see Section 3.2.2). For the trajectory optimization, we recast the problem as a parameter optimization problem on the control actions and used a single shooting method to evaluate the cost function and system state at the end

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

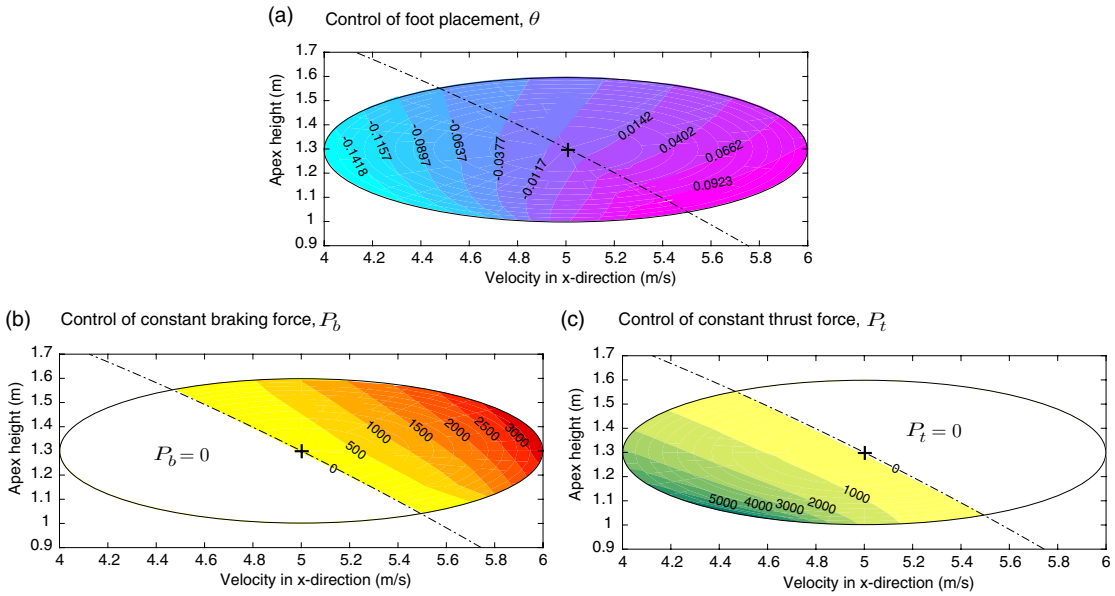


Fig. 5. Contour plots for control actions as a function of horizontal velocity and apex height: (a) foot placement angle; (b) constant braking force in the stance phase; and (c) constant thrust force in the restitution phase.

of a single step. The parameter optimization problem was solved using a constrained optimization software called Sparse Nonlinear OPTimizer,[26] which is based on solving sequential quadratic programs. The total optimization time for 451 initial conditions was 131 min on a laptop (circa 2012). Thus, it took about 17 s for each optimization to complete.

The controls/state combinations are tabulated for further processing but are presented here as plots. Figure 5 shows the three control actions as a function of apex state, the height y, and the horizontal velocity $\dot{x}$. The total energy at the apex is given by $\mathrm{TE}^i = 0.5m(\dot{x}^i)^2 + mgy^i$. The corresponding value for the periodic motion is given by $\mathrm{TE}^0 = 0.5m(\dot{x}^0)^2 + mgy^0$. Next, we find the total energy curve corresponding to the fixed point (shown as the dashed black line) by solving for $\dot{x}^i$ and y^i on the curve $TE^0 = 0.5m(\dot{x}^i)^2 + mgy^i$. This curve divides the ellipsoids into two halves: top-right half has higher total energy than the nominal, $\mathrm{TE}^i > \mathrm{TE}^0$ (where i is an initial condition in the top-right half) and bottom-left half has lower total energy than the nominal, $\mathrm{TE}^i < \mathrm{TE}^0$. Thus for top-right half, the braking force P_b is nonzero and serves to brake or extract energy from the system. Similarly, for the bottom-left half, the control strategy is to apply a constant thrust force to add energy to the system. The foot placement angle maintains the total energy of the system but converts the potential energy to the kinetic energy and *vice versa*. The three control actions perform three distinct roles: foot placement cannot change the total energy but can convert the potential energy to the kinetic energy and *vice versa*, the braking force can only decrease the total energy, and thrust force can only increase the total energy. A detailed analysis of how these separate control actions combine to increase the ROA is provided elsewhere[27].

Figure 6 shows the associated MCOT (see Eq. (12)) for individual control actions and the total. Figure 6(a) shows the total MCOT. The total MCOT for the fixed point (shown by the + sign) is 0.39. The MCOT is mostly flat along the x-axis or the horizontal velocity axis but increases monotonically along the y-axis or the vertical height axis. Figure 6(b)–(d) plots the contribution of individual control actions while (a) is the sum of (b), (c), and (d).

4.3. Finding control policies and verification

Next, we find a control policy ($u_k = f_k(\mathbf{x}^i)$, one for each control action k) from the tabulated data (Section 4.2). The rationale is that such a control policy offers a compact representation of the controls and is easier to store for hardware implementation.

We use the observation from Fig. 5 that we can segregate force control (P_b and P_t) based on the location of the initial condition and the total energy. We have the following three parameterizations in the increasing level of complexity (as given by the number of parameters).

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

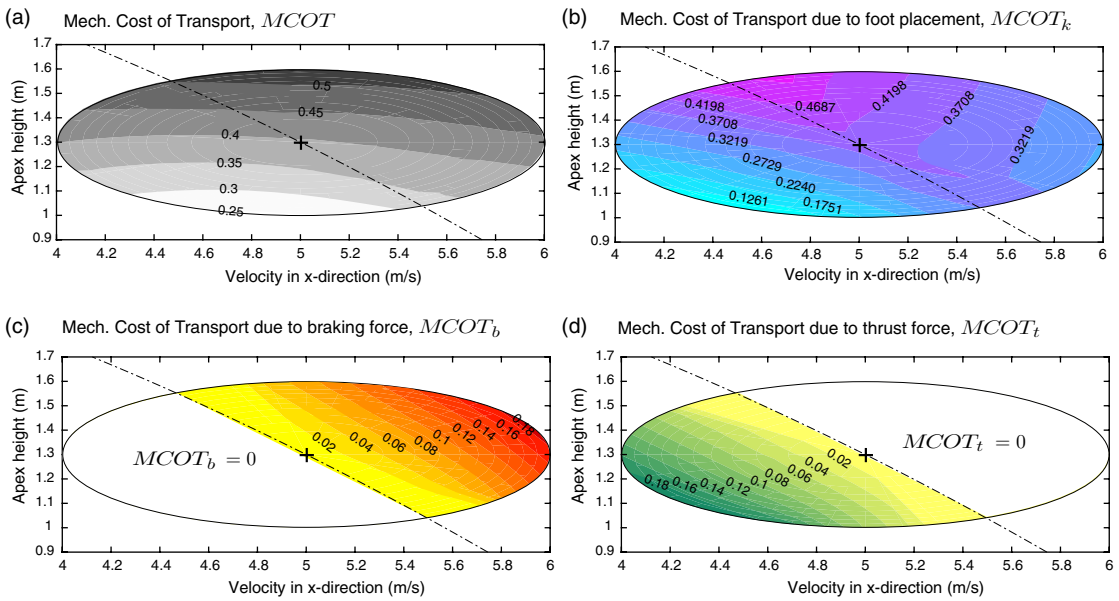


Fig. 6. Contour plots for MCOT function of horizontal velocity and apex height: MCOT (a) total, (b) due to springy leg force k and foot placement angle θ, (c) due to braking force P_b, and (d) due to thrust force P_t.

Linear policy: The linear policy has nine parameters as shown below.

$$\theta^i = a_0 + a_1\Delta y + a_2\Delta\dot{x}$$

$$P_b^i = \begin{cases} 0 & \text{TE}^i \le \text{TE}^0 \\ b_0 + b_1\Delta y + b_2\Delta\dot{x} & \text{otherwise} \end{cases}$$

$$P_t^i = \begin{cases} 0 & \text{TE}^i \ge \text{TE}^0 \\ c_0 + c_1\Delta y + c_2\Delta\dot{x} & \text{otherwise.} \end{cases}$$

Quadratic policy: The quadratic policy has 18 parameters as shown below.

$$\theta^i = a_0 + a_1\Delta y + a_2\Delta\dot{x} + a_3(\Delta y)^2 + a_4(\Delta\dot{x})^2 + a_5\Delta y\Delta\dot{x}$$

$$P_b^i = \begin{cases} 0 & \text{TE}^i \le \text{TE}^0 \\ b_0 + b_1\Delta y + b_2\Delta\dot{x} + b_3(\Delta y)^2 + b_4(\Delta\dot{x})^2 + b_5\Delta y\Delta\dot{x} & \text{otherwise} \end{cases}$$

$$P_t^i = \begin{cases} 0 & \text{TE}^i \ge \text{TE}^0 \\ c_0 + c_1\Delta y + c_2\Delta\dot{x} + c_3(\Delta y)^2 + c_4(\Delta\dot{x})^2 + c_5\Delta y\Delta\dot{x} & \text{otherwise.} \end{cases}$$

Neural network policy: Each neural network in the policy has 12 hidden layers and the total parameters are 484.

$$\theta = \text{Neural Net 1}$$

$$P_b^i = \begin{cases} 0 & \text{TE}^i \le \text{TE}^0 \\ \text{Neural Net 2} & \text{otherwise} \end{cases}$$

$$P_t^i = \begin{cases} 0 & \text{TE}^i \ge \text{TE}^0 \\ \text{Neural Net 3} & \text{otherwise,} \end{cases}$$

where $\Delta y = y^i - y^0$, $\Delta\dot{x} = \dot{x}^i - \dot{x}^0$, a's, b's, and c's are all constants. The constants were fit using the data from Section 4.2. The linear and quadratic policies were fit using nonlinear least squares function

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

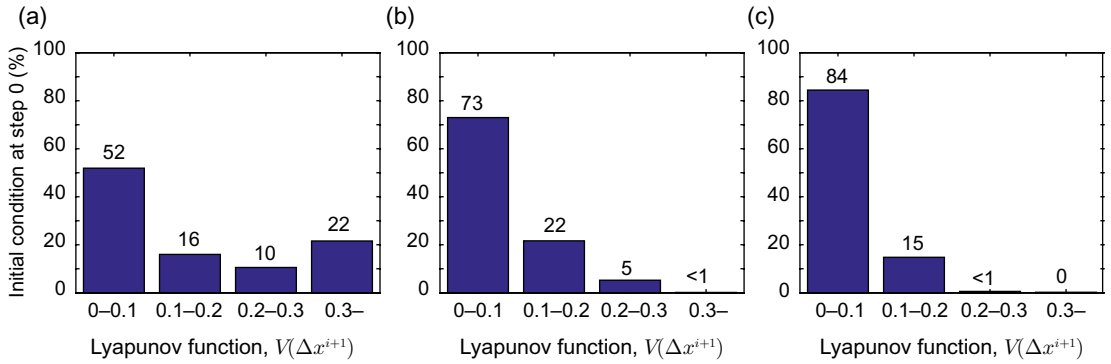


Fig. 7. Verification for different control policies. A histogram showing Lyapunov function after one step ($V(\Delta \mathbf{x}^{i+1})$) for randomly chosen initial conditions ($\mathbf{x}^i$) in the ROA. Each figure represents a different fit between control actions and initial conditions at the apex. The fits are based on (a) linear, (b) quadratic, and (c) neural networks.

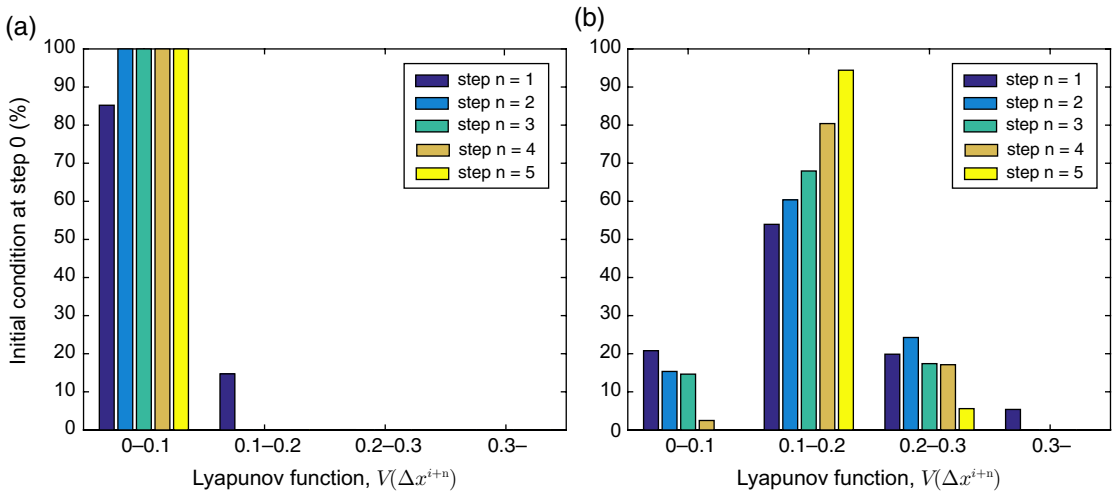


Fig. 8. Effect of simulating the system with damping. Lyapunov function $V(\Delta \mathbf{x}^{i+n})$ for $n = 1, 2, 3, 4, 5$ steps starting from randomly chosen initial conditions within the ROA for damping factor of (a) $c_v = 100$ and (b) $c_v = 200$.

lsqnonlin in MATLAB, and the neural network policy was trained using Levenberg–Marquardt using the Deep Learning Toolbox in MATLAB.

4.3.1. Verification. Using the three control fits discussed above, we do the verification as follows. We choose about 1500 initial conditions inside the ROA $V(\Delta \mathbf{x}^i) \leq 1$ (note that our fit is based on only 451 points). Next, for each of the initial conditions, we did a forward simulation for one step using each control policy. We plotted the histogram of the Lyapunov function after one step for each of the three fits as shown in Fig. 7. We have also indicated the percentages above each bar of the histogram. It can be seen that 52%, 73%, and 84% of the initial conditions are in the range $0 < V(\Delta \mathbf{x}^{i+1}) < 0.1$ for the linear, quadratic, and neural network fit, respectively. Also, almost 99% of initial conditions are within the range $0 < V(\Delta \mathbf{x}^{i+1}) < 0.3$ for the quadratic fit and within the range $0 < V(\Delta \mathbf{x}^{i+1}) < 0.2$ for the neural network fit. These results indicate that the neural network provides the best fit for the data followed by the quadratic fit and finally the linear fit.

4.3.2. Robustness. We choose the quadratic control policy for additional robustness checks. The rationale behind using the quadratic policy over the neural network was that the former uses a few parameters while providing a comparable fit as demonstrated in Fig. 7. To check robustness, we looked into effects such as unmodeled dynamics, noisy sensors/actuators, and external disturbances. These are discussed next.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

To check the robustness to unmodeled dynamics, we added a damping force ($F_v = -c_v\dot{\ell}$) in addition to the spring force on the stance leg. Then, we simulated the system for 1414 randomly chosen points in the ROA, $(\mathbf{x}_i - \mathbf{x}_0)^T\mathbf{S}_0(\mathbf{x}_i - \mathbf{x}_0) \leq 1$. Figure 8 shows the Lyapunov function (averaged over the 1414 simulations) for five consecutive steps for two damping constants: (a) $c_v = 100$ Ns/m and (b) $c_v = 200$ Ns/m. These damping values have the effect of reducing the total energy of the periodic motion at the next step by 6% and 10%, respectively. It can be seen that for $c_v = 100$, all initial conditions decay to and stay within the range $V(\Delta\mathbf{x}^{i+2}) \leq 0.1$, that is, in two steps. However, when damping constant is doubled to $c_v = 200$, all initial conditions are within the range $V(\Delta\mathbf{x}^{i+2}) \leq 0.3$.

We did stochastic simulations to ascertain robustness to several other factors: noisy actuators (foot placement angle and stance forces), noisy sensors (apex height and horizontal velocity), and external disturbance (step change in height at touchdown). We consider each of these five factors separately. That is, only one factor was varied by keeping the other four fixed to the nominal value. For each factor, we choose a standard deviation σ_d (where $d = \theta$, P, y, $\dot{x}$, and h are foot placement angle, stance forces, apex height, apex horizontal speed, and step disturbance, respectively). The standard deviation was varied from zero to a maximum value in certain increments appropriate to the specific factor. For a given σ_d, we created 10 runs, each of 100 steps. In each of these runs, the noise or disturbance was applied *every step.* For sensor noise, foot placement control actuation noise, and step height disturbance, we used uniform random distribution $\{-\sigma_d, \sigma_d\}$, and for stance force noise, we used uniform distribution $\{0, \sigma_d\}$ to create perturbations over the nominal values for each of these parameters. For each of these runs, the hopper started at the same initial state, $\mathbf{x}^0$.

Figure 9 shows the results for the stochastic simulations. Each row corresponds to a single factor. The column corresponds (from left to right) to the number of steps successfully taken, the average foot placement angle, and the average push-off force. The robustness is best explained using the first column, the number of steps successfully taken: the maximum deviation σ_d for which the hopper can take average 100 steps (the maximum number of steps simulated) indicates no failure. This is achieved for foot placement angle $\sigma_\theta \approx 0.04$ rad $= 2.3°$ (about 11% of nominal θ), for stance force $\sigma_P \approx 800$ N, for apex horizontal velocity $\sigma_{\dot{x}} \approx 0.4$ m/s (about 8% of the nominal apex horizontal velocity), for apex height $\sigma_y \approx 0.12$ m (about 9.2% of the nominal apex height), and step disturbance $\sigma_h < 0.01$ m $= 1$ cm. The control actions θ (column 2) and stance forces P_b and P_t (column 3) for values that indicate no failure (i.e., steps traversed is 100) provide an indication of stabilization strategies. In particular, for increasing σ for push-off (row 2), apex horizontal speed (row 3), and apex height (row 4), the foot placement angle decreases, but the push-off force increases to stabilize the system. However, increasing σ for foot placement angle (row 1) does not change either the push-off or foot placement angle (columns 2 and 3), thus indicating that the system relies on the stability properties of the nominal gait for stabilization. Finally, there is no clear trend for an increase in σ for step height (row 5) because of the low tolerance for rejecting step height disturbances. Our main conclusion is that the system is most robust to noisy push-off forces, moderately robust to noise in apex height and velocity and to noise in foot placement angle, and least robust to step height disturbance.

Figure 10 gives a better understanding of the evolution of the Lyapunov function across multiple steps for an increasing standard deviation for each of the five parameters. To generate each plot, we started the hopper from the same initial condition, but a different σ based on the parameter that we considered, and simulated the system for multiple steps. Each dot in the figure represents the Lyapunov function at subsequent steps. The light gray ellipse is the ROA (i.e., $(\mathbf{x}_i - \mathbf{x}_0)^T\mathbf{S}_0(\mathbf{x}_i - \mathbf{x}_0) \leq 1$), and the dark gray ellipse in the middle corresponds to $(\mathbf{x}_i - \mathbf{x}_0)^T\mathbf{S}_0(\mathbf{x}_i - \mathbf{x}_0) \leq 0.1$ or the region inside which all initial conditions will decay for a perfect model, perfect sensors, and no external disturbances. The columns are arranged in the increasing order of σ from left to right; we chose small, medium, and large model uncertainty/disturbance that the controller is able to sustain. It can be seen that as the deviation σ increases the Lyapunov function over multiple steps stays within the ROA, but not inside the ellipse with a boundary defined by 0.1. The effect is most prolonged for the large disturbance (rightmost column), especially for the apex height variation, where the hopper is on the verge of moving out of the assumed ROA, but not necessarily falling.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

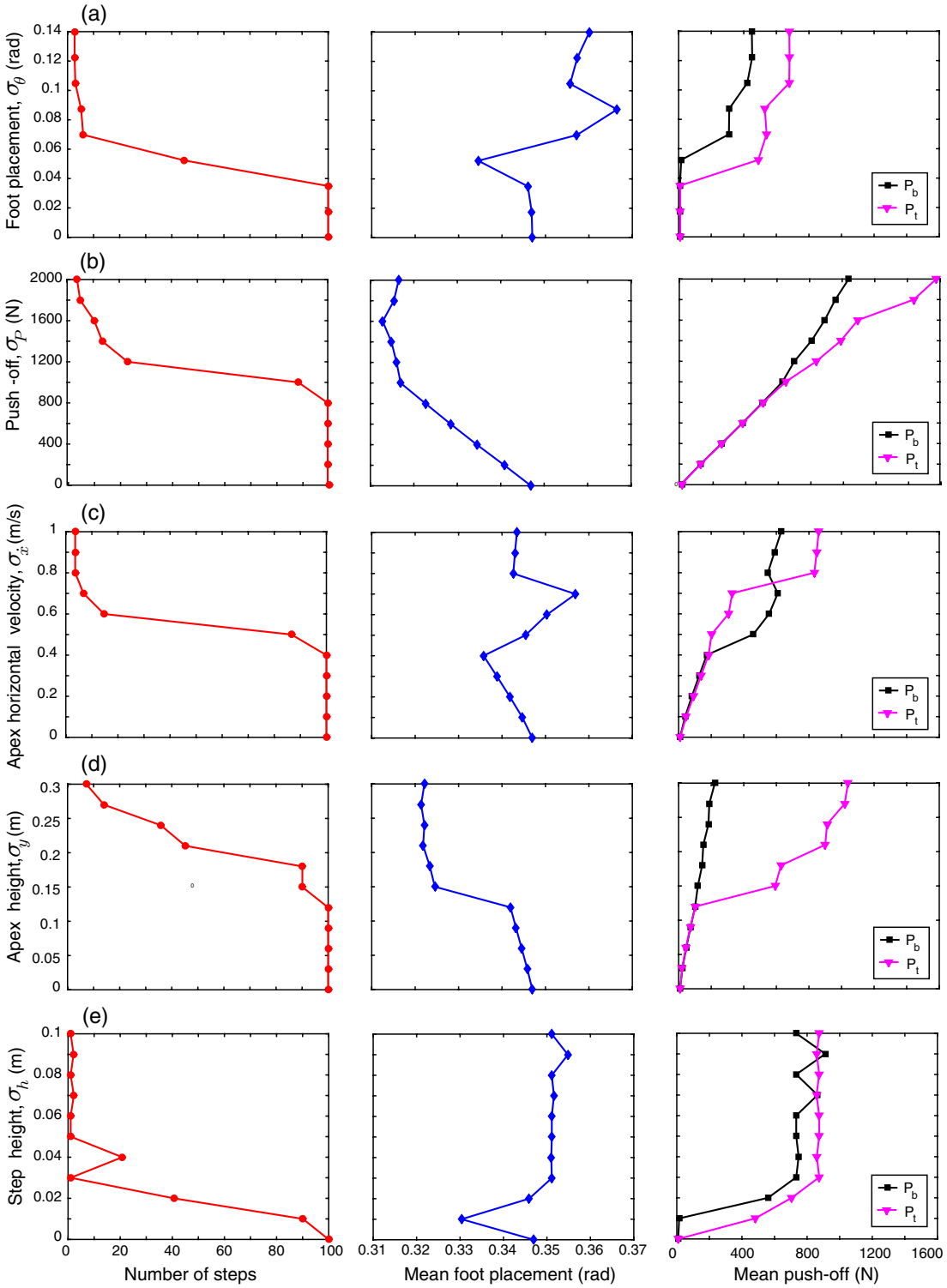


Fig. 9. Robustness to noisy actuators (a–b), noisy sensors (c–d), and external disturbance (e). Each row corresponds to a single parameter and the corresponding effect on the number of steps traversed (column 1), foot placement angle (column 2), and force (column 3). The parameters are (a) noise in foot placement angle, (b) noise in control force, (c) noise in apex horizontal velocity measurement, (d) noise in apex height measurement, and (e) step change in height.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

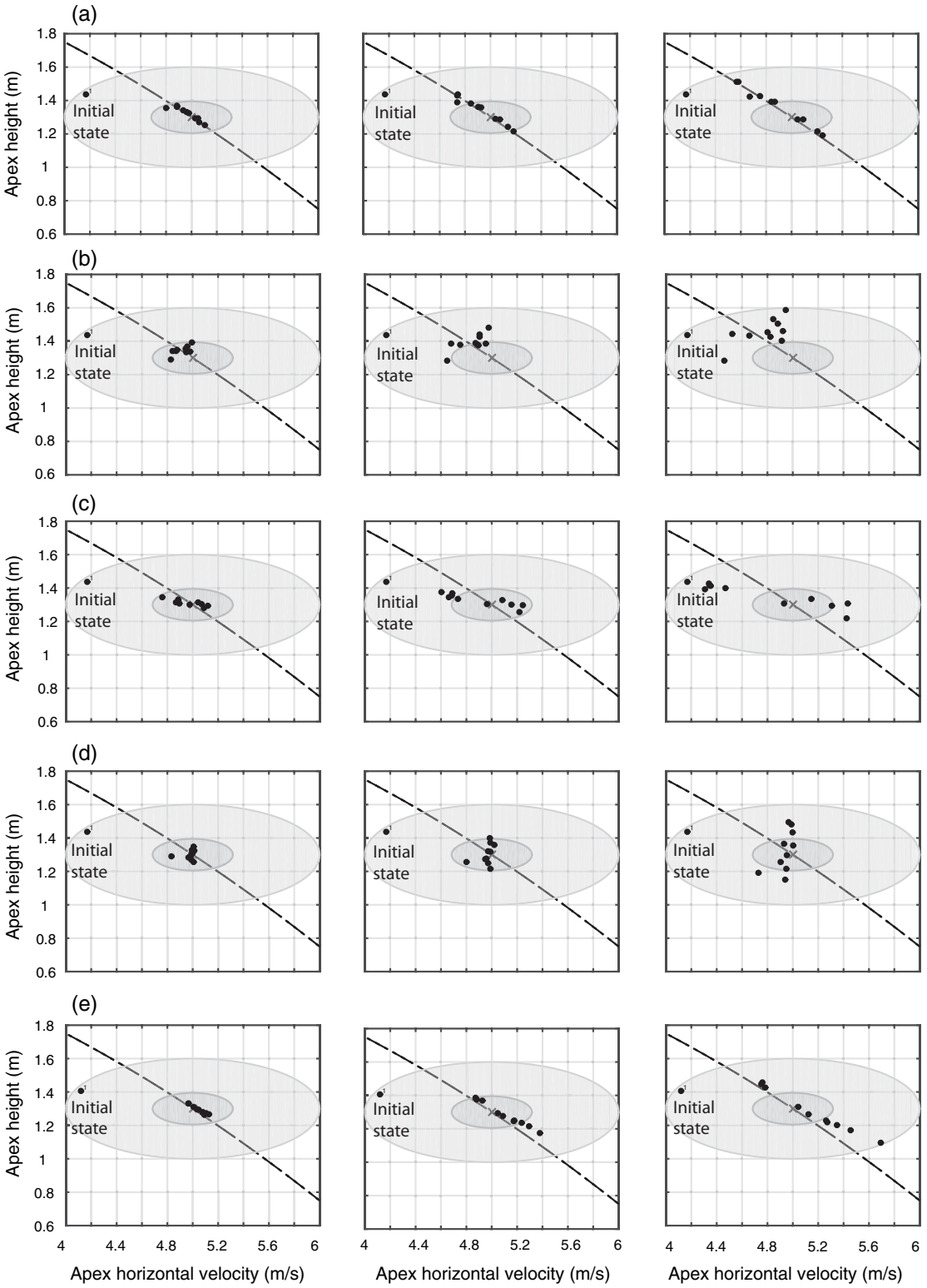


Fig. 10. Effect of size of standard deviation (σ) of various parameters to the evolution of the Lyapunov function. The hopper starts from the same initial conditions for all runs. Each row corresponds to a specific external parameter, and each column corresponds to increasing standard deviation for the particular parameter. The rows and columns in that order are (a) foot placement control for σ_θ of 1°, 2°, and 3°, (b) push-off control for σ_P of 500, 1000, and 1500 all in N, (c) apex velocity sensor for $\sigma_{\dot{x}}$ of 0.1, 0.25, and 0.5 all in m/s, (d) apex height sensor for σ_y of 5, 10, and 20 all in cm, (e) step height disturbance for σ_h of 2, 5, and 8 all in cm.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

5. Discussion

In this paper, we have presented a sampling-based framework to find control policies for an assumed ROA and demonstrated the approach on a model of hopping. The methodology was as follows. We performed a coarse-sampling of the initial conditions at the Poincaré section and used trajectory optimization on each initial condition to find control actions (e.g., gain, amplitudes, and set points) that ensured a reduction of a suitably defined Lyapunov function after a single step. The result was a tabulation of the initial conditions and the corresponding control actions. We fitted each control action as a function of the initial conditions, assuming various functional representations (e.g., linear, quadratic, and neural network) to find the control policy. Finally, we verified the control policy on a finely sampled set of initial conditions and did additional robustness checks.

The traditional formal control approach started by assuming a control policy (e.g., LQR) and then found the largest ROA (e.g., using sum-of-squares optimization). Our sampling-based approach worked in the reverse: we started by assuming an ROA and then found the control policy using extensive simulations and regression. The formal control approach leads to a simple policy (e.g., the linear policy) at the expense of the possibility of generating a small ROA, while our approach guarantees a large ROA at the expense of more complicated control policy. For linear systems, the formal control approach based on a linear control policy would be more effective computation- and performance-wise, but for nonlinear systems where a simple linear controller could potentially produce a small ROA, our method would potentially lead to a larger ROA.

Another major difference between our and prior work is in the way we define the ROA. Tedrake[2] and Manchester[22] considered the ROA along the trajectory, while we considered the ROA at the Poincaré section. In these prior works, the system was linearized about the trajectory, and the time-varying LQR controller was used for tracking purposes. The ROA was estimated by finding multiple Lyapunov functions along the trajectory using the sum-of-squares optimization guaranteeing local stability along the trajectory. In our case, there was a single Lyapunov function at the Poincaré section and gave the ROA for orbital or step-to-step stability for the nominal cyclic motion. In contrast to the trajectory tracking in the prior work, we solved a computationally simpler regulation problem. In other words, using the Poincaré map-based Lyapunov function, we created a discrete controller that operates in a step-to-step fashion, a common practice in the control of machines exhibiting periodic motion.[1]

Our approach blended control theory and machine learning; more specifically, control theory-based stability metrics such as the control Lyapunov function and machine learning-based methods such as sampling and regression (polynomials and deep neural networks) for designing control policies. In this regard, it is important to note that control theory approaches based on the sum-of-squares optimization[2] may only work with polynomial model and control policies. In contrast, our method worked with generic models, including black-box models.

The control actions and Lyapunov function are the designer's choices that affected the final outcomes. For the hopper, our choice of control actions was foot placement angle, braking force, and thrust force, while the Lyapunov function was an ellipse. Our choice of control actions not only led to a large ROA, but it also led to further simplification as only two of the three control actions were needed to stabilize the system based on the system energy. We chose a quadratic Lyapunov function with principal axes along the speed (x-axis) and height (y-axis), and this allowed us to stabilize a relatively wide range of horizontal speeds (range of ± 1 m/s) for the same apex height, the rationale being that regulating the speed is one of the goals of legged locomotion in the presence of disturbances. In comparison, the work by Heim and Sprowitz[12] used a single control action, the foot placement angle, which limited the ROA to a relatively small area.

We have two comments about our chosen ROA. One, we chose our ROA to be $(\mathbf{x}_i - \mathbf{x}_0)^T \mathbf{S}_0 (\mathbf{x}_i - \mathbf{x}_0) \leq c$, with $c = 1$. The choice for c is arbitrary. In principle one could choose a large value of c, perhaps encompassing the entire state space. Then do optimization for sampled points in the entire space followed by regression to fit a control policy ($\mathbf{u}_i = f(\mathbf{x}_i)$). We hypothesize that the resulting control policy will be highly nonlinear and could potentially need a large number of free parameters to describe it adequately. Often a simple control policy (e.g., quadratic or neural network with few parameters) is ideal. Second, although we chose the ROA $c = 1$ and found the control policy for that region, the actual ROA is likely much larger. It would be fairly straightforward to find the actual ROA by doing additional simulations over larger level sets $c > 1$. After finding the actual ROA, one can

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

repeat the control policy evaluation for the new c and repeat the process until no further expansion of the ROA is possible.

We advocated offline trajectory optimization followed by online implementation. The average time for the trajectory optimization for the simple model is about 17 s (circa 2012 laptop), which is too slow for online implementation. Since the trajectory optimization is done for a range of initial conditions, the resulting data (initial conditions and corresponding control actions) may be saved as a lookup table for online implementation. This simple strategy worked for a few ROAs, but as the number of ROAs grows (e.g., for different gaits and fixed points and as the system dimensions scale up), the lookup table might be too large. Thus, we advocate mining the data to find a simple functional representation, a control policy of the form, $\mathbf{u} = \mathbf{f}(\mathbf{x})$, which is easier to store and use for real-time control.

Our work has several limitations that we list next. Our sampling-based method is computationally expensive and may not scale well for a high-dimensional system (e.g., quadrupeds and humanoids). The challenge then is to find either simple models (templates of complex robots known as anchors[17]) with a small number of inputs or find clever control approaches to reduce the system dimensionality (e.g., hybrid zero dynamics[28]). Another issue is that the Lyapunov function, and the control actions are design choices and may need trial and error to arrive at good choices. Our thumb rule for control actions is to have ones that have distinct effects on the system dynamics. For example, in the hopping model, the thrust and braking forces allowed us to modify the total system energy, while the foot placement angle allowed speed and/or height regulation without affecting system energy. Thus, there were multiple combinations that regulated the height and speed (redundancy). Our approach of doing step-to-step control at a higher level, a type of low-bandwidth control, is susceptible to failure for large disturbances. Specifically, we measure the system state at the Poincaré section, then choose control actions that remain fixed for the entire step till the next Poincaré section. However, if there are large disturbances between steps, then the system might fail before it reaches the next Poincaré section. One suggestion for circumventing this issue is to have multiple sections along the trajectory and use those sections to modify the control actions more than once per step. Our choice of control policies affected our results. For example, the linear fit is the simplest control policy but does not give adequate convergence rate when we do the verification; while the neural network with about 10 times more parameters is complex but gives good convergence rates. Finally, we have not considered actuator limits. When the actuator limits are reached, it may not be possible to stabilize the system in a single step and two or more steps might be needed. It has been shown that with realistic actuator limits, it takes about two steps to reach any target speed from any other speed.[29]

6. Conclusion

The paper presented a sampling-based approach to find nonlinear control policies for candidate Lyapunov functions defined on the Poincaré section and assumed ROA that ensures exponential convergence of the Lyapunov function between steps. The approach was demonstrated on a model of hopping. We assumed a quadratic Lyapunov function, and then using trajectory optimization and the exponential convergence of the Lyapunov function, we tabulated the control actions as a function of the initial conditions. Then, using regression and deep learning, we approximated the tabulated data to find a control policy. Finally, we verified the control policy using a finer grid and did robustness checks by varying the model parameters, introducing sensor and actuator noise, and disturbances. We conclude that the proposed approach can successfully find control policies (usually nonlinear) for the assumed ROA. This is significant, especially for nonlinear systems where past approaches of assuming a linear control policy lead to a relatively small ROA.

Acknowledgements

This work was supported by National Science Foundation grants 1566463 and 1946282 to Pranav Bhounsule.

References

1. D. Hobbelen and M. Wisse, "Limit Cycle Walking," **In**: *Humanoid Robots Human-Like Machines* (M. Hackel, ed.) (IntechOpen, 2007) pp. 277–294.
2. R. Tedrake, "LQR-Trees: Feedback Motion Planning on Sparse Randomized Trees," *Papers of the Fifth Annual Robotics: Science and Systems Conference*, June 28-July 1, 2009, University of Washington, Seattle, USA.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press

3. S. Prajna, A. Papachristodoulou and P. A. Parrilo, "Introducing Sostools: A General Purpose Sum of Squares Programming Solver," *Proceedings of the the 41st IEEE Conference on Decision and Control* (2002) pp. 741–746.
4. P. A. Bhounsule, A. Zamani and J. Pusey, "Switching Between Limit Cycles in a Model of Running Using Exponentially Stabilizing Discrete Control Lyapunov Function," *Proceedings of the Annual American Control Conference* (2018) pp. 3714–3719.
5. A. Zamani, J. D. Galloway II and P. A. Bhounsule, "Feedback Motion Planning of Legged Robots by Composing Orbital Lyapunov Functions Using Rapidly-Exploring Random Trees," *Proceedings of the IEEE International Conference on Robotics and Automation* (IEEE, 2019).
6. T. McGeer, "Passive dynamic walking," *Int. J. Robot. Res.* **9**(2), 62–82 (1990).
7. T. McGeer, "Passive Dynamic Biped Catalogue," *Proceedings of the 2nd International Symposium on Experimental Robotics* (1991) pp. 465–490.
8. M. Garcia, A. Chatterjee, A. Ruina and M. Coleman, "The simplest walking model: stability, complexity, and scaling," *ASME J. Biomech. Eng.* **120**(2), 281–288 (1998).
9. S. Strogatz, *Nonlinear Dynamics and Chaos* (Addison-Wesley Reading, Boston, MA, 1994).
10. A. Schwab and M. Wisse, "Basin of Attraction of the Simplest Walking Model," *Proceedings of the ASME Design Engineering Technical Conference* (2001) pp. 531–539.
11. C. S. Hsu, *Cell-to-Cell Mapping: A Method of Global Analysis for Nonlinear Systems* (Springer-Verlag, New York, NY, 1987).
12. S. Heim and A. Spröwitz, "Beyond Basins of Attraction: Evaluating Robustness of Natural Dynamics," *arXiv preprint arXiv:1806.08081* (2018).
13. P.-B. Wieber, "Viability and Predictive Control for Safe Locomotion," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008) pp. 1103–1108.
14. J. Pratt, J. Carff, S. Drakunov and A. Goswami, "Capture Point: A Step Toward Humanoid Push Recovery," *Proceedings of the 6th IEEE-RAS International Conference on Humanoid Robots* (2006) pp. 200–207.
15. J. Pratt, T. Koolen, T. D. Boer, J. Rebula, S. Cotton, J. Carff, M. Johnson and P. Neuhaus, "Capturability-based analysis and control of legged locomotion. Part 2: Application to M2V2, a lower-body humanoid," *Int. J. Robot. Res.* **31**(10), 1117–1133 (2012).
16. P. Zaytsev, W. Wolfslag and A. Ruina, "The boundaries of walking stability: Viability and controllability of simple models," *IEEE Trans. Robot.* **34**(2), 336–352 (2018).
17. D. Koditschek and J. Robert, "Templates and anchors: Neuromechanical hypotheses of legged locomotion on land," *J. Exp. Biol.* **202**(23), 3325–3332 (1999).
18. E. C. Whitman, Coordination of Multiple Dynamic Programming Policies for Control of BipedalWalking *Ph.D. Thesis* (Carnegie Mellon University, 2013).
19. T. Mandersloot, M. Wisse and C. G. Atkeson, "Controlling Velocity in Bipedal Walking: A Dynamic Programming Approach," *Proceedings of the 6th IEEE-RAS International Conference on Humanoid Robots* (2006) pp. 124–130.
20. M. H. Raibert and F. C. Wimberly, "Tabular control of balance in a dynamic legged system," *IEEE Trans. Syst., Man, Cybern.* **1**(2), 334–339 (1984).
21. X. Da, R. Hartley and J. W. Grizzle, "Supervised Learning for Stabilizing Underactuated Bipedal Robot Locomotion, with Outdoor Experiments on the Wave Field," *Proceedings of the IEEE International Conference on Robotics and Automation* (2017) pp. 3476–3483.
22. I. R. Manchester, M. M. Tobenkin, M. Levashov and R. Tedrake, "Regions of Attraction for Hybrid Limit Cycles of Walking Robots," *Proceedings of the 18th World Congress The International Federation of Automatic Control* (2011) pp. 5801–5806.
23. P. A. Bhounsule and A. Zamani, "A discrete control lyapunov function for exponential orbital stabilization of the simplest walker," *J. Mech. Robot.* **9**(5), 051011–8 (2017).
24. J. Betts, "Survey of numerical methods for trajectory optimization," *J. Guid. Control Dyn.* **21**(2), 193–207 (1198).
25. M. Srinivasan, Why Walk and Run: Energetic Costs and Energetic Optimality in Simple Mechanics-based Models of a Bipedal Animal *Ph.D. Thesis* (Cornell University, 2006).
26. P. Gill, W. Murray and M. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM J. Optim.* **12**(4), 979–1006 (2002).
27. A. Zamani and P. Bhounsule, "Control synergies for rapid stabilization and enlarged region of attraction for a model of hopping," *Biomimetics* **3**(3), 1–13 (2018).
28. J. Grizzle, G. Abba and F. Plestan, "Asymptotically stable walking for biped robots: Analysis via systems with impulse effects," *IEEE Trans. Autom. Control* **46**(1), 51–64 (2001).
29. P. Zaytsev, S. J. Hasaneini, and A. Ruina, "Two Steps is Enough: No Need to Plan Far Ahead for Walking Balance," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2015) pp. 6295–6300.

https://doi.org/10.1017/S0263574720000211 Published online by Cambridge University Press