Quantification and Design of Robust High-Level Control for Walking Robots via Meshing

Cenk Oguz Saglam and Katie Byl

November 12, 2014

Abstract

In this work, we present tractable tools to estimate steps-to-failure for walking systems on stochastic terrain, via Markov chain modeling. Our particular concern is to provide methods that extend to high-dimensional systems, and so one particular focus is on demonstrating robustness to errors introduced by the discretization of creating a Markov model. Our approach relies on the existence of a set of low-level controllers, which, as we demonstrate, drive the step-to-step dynamics onto quasi-2D manifolds. We present a meshing technique that systematically maps out the entire reachable state space, given control actions are limited to a set of low-level controllers. We examine several approaches for high-level control that employ either or both exteroceptive (terrain estimation) or proprioceptive (state estimation) information, and we explore the use of various numbers of available low-level controllers. Our goal in doing so is to illustrate a means of quantifying such questions as, "how many low-level controllers are really necessary?" and "how valuable is each source of sensor information?" Finally, we examine robustness, identifying two dominant sources of meshing error (i.e., current state and upcoming terrain slope discretizations) and present robust methods to cope with each. We conclude that our approach is applicable to a large class of high-dimensional systems.

1 Introduction

Achieving bipedal robots that can walk like humans requires energy efficiency, agility, and stability. Designs exploiting the dynamic nature of human walking seem an intuitive path toward efficiency and agility, but they also complicate stability. We posit the way forward is to quantify and optimize appropriate metrics to improve performance. For energetics, Cost of Transport (COT), explained by Tucker (1975) as the non-dimensionalized energy expenditure per unit weight and unit distance, is a commonly accepted measure. For walking robots, stability is typically viewed as a 1 or 0 notion. According to the popular definition by Vukobratovic and Borovac (2004), stability is achieved if the center of pressure is inside the support polygon of the feet. Classic Zero Moment Point

(ZMP) techniques rely on preserving stability at all times as applied by Sakagami et al. (2002) and Erbatur et al. (2008). However, this approach results in energy inefficiency as pointed out by Kuo (2007). This fact motivates *dynamic walking*, where the existence of underactuation is a key factor. Although not all human walking is underactuated, studying point-foot robots has proved to be extremely useful towards dynamic human-like walking, as observed by Ames (2012). For dynamic walking robots, Westervelt et al. (2003) uses Poincaré map analysis to show stable limit cycles, and McGeer (1990) demonstrates that even passive robots can walk in a stable manner. Motivated by this, researchers have demonstrated a range of powered walkers based on exploiting natural dynamics as explained in Collins et al. (2005). This approach, inspiring Bhounsule et al. (2012), has led to Cornell Ranger's record-breaking performance in walking with only on-board power (energetically autonomous). Collins et al. (2005) states that while Ranger has a COT of around 0.2, the COT of Asimo (using ZMP-based control) is estimated to be 3.2. However, focusing on energy efficiency while proving the existence of a stable limit cycle only for unrealistically deterministic conditions (e.g., flat terrain) has resulted in designs that are sensitive to perturbations, thus performing poorly on rough terrain. A key reason, we argue, is because stability is still viewed as a 1 or 0 notion. However, even humans fall from time to time; for example, imagine walking on a rocky or icy road. Byl and Tedrake (2009) suggest that stability under stochastic conditions (e.g., rough terrain) should be measured via the expected number of steps before falling, or Mean First Passage Time (MFPT). Here, high MFPT means better stability.

The present work is an extension of Byl and Tedrake (2009) and Chen and Byl (2012) in number of ways. (1) MFPT calculation relies on meshing the state space to model the dynamics of walking as a Markov Decision Process. In this paper, we show how we are able to avoid the curse of dimensionality by illustrating our methods on a 5-Link biped, for which the state is 10 dimensional. (2) If multiple qualitatively different low-level controllers are available, a robot may incorporate high-level behavioral algorithms to increase MFPT by appropriately switching among controllers. While switching using only internal (proprioceptive) state information (blind-walking) is advantageous, a dramatic improvement is obtained by also including upcoming terrain information. (3) We introduce and illustrate the issue of robustness of high-level policies to meshing, and we propose an intuitive solution, which works reasonably well.

This paper extends work presented at RSS 2014, in Saglam and Byl (2014*c*), which focuses on tractable meshing, increasing robustness to discretization, and quantification of upcoming terrain information. Additional material is presented to provide a more complete presentation that demonstrates that our problem framework, requiring some set of low-level controllers, curtails the curse of dimensionality. Specifically, we demonstrate that the step-to-step dynamics of rough terrain walking converge rapidly to quasi-2D surface, making meshing tractable. Some of this additional material was earlier presented in Saglam and Byl (2013*a*) and Saglam and Byl (2013*b*), but is more clearly and cohesively presented in this paper, importantly using more recent meshing techniques that

are both more robust and efficient.

The rest of this paper is organized as follows. Sections 2 and 3 define our problem framework and walking robot model. Our control framework uses a set of low-level controllers, which are described in Sec. 4. More detailed discussion of low-level control is a topic of future publications. Briefly, low-level control can also be optimized via quantification of MFPT; however, this work focuses instead on design and quantification of high-level switching policies. An algorithm for generating and analyses of the dimensionality of meshes of states are presented in Section 5, and theory for MFPT determination is reviewed briefly in 6. Section 7 presents a detailed study on various design options for high-level control, and finally Sections 8, 9, and 10 discuss practical applicability to real robots, list topics for future work, and summarize our conclusions.

2 Problem Overview

Figure 1 presents the "big picture" for the problem we study. Consider a legged robot that is walking, running, and/or hopping. Say we focus on stability, so value (or cost) must represent how stable walking is. Perhaps the most intuitive choice is the average number of steps before failure, which corresponds to MFPT. Unfortunately, estimating MFPT is cumbersome; however, in this work we will illustrate that it can be done even for high DOF robots, thanks to the hierarchal nature of our problem framework. This process will be independent from the particular low-level controller structure adopted. Also, it will *not* require the existence of a high-level controller as in Figure 1. Having a single controller and/or not using the slope estimation are just special cases.

Secondly, assume there are multiple low-level controllers available, each having advantages and disadvantages under different circumstances. Each of them might be obtained by different methods, and/or optimal for different cost functions (including energetics, speed, etc.). We study how to optimally and robustly switch among these low-level controllers given state information and/or slope estimation to maximize the value. This works assumes we only change controllers at impacts, i.e., the controller is fixed for each step.

We note briefly here that it is the existence of low level controllers that contracts the reachable region of state space to a size and dimensionality that allow for discreteization (meshing) to analyze and optimize performance.

3 Model

The methods of following sections are applicable to various robots. To show applicability to higher degree of freedom (DOF) robots (compared to the 2-Link walker in Byl and Tedrake (2009) and the 3-Link walker of Chen and Byl (2012)), the analysis in this paper will be carried out with a 5-link biped as shown in Figure 2. It has point feet and there are actuators only at internal joints, so it is underactuated by 1 DOF. The angles shown in the figure form

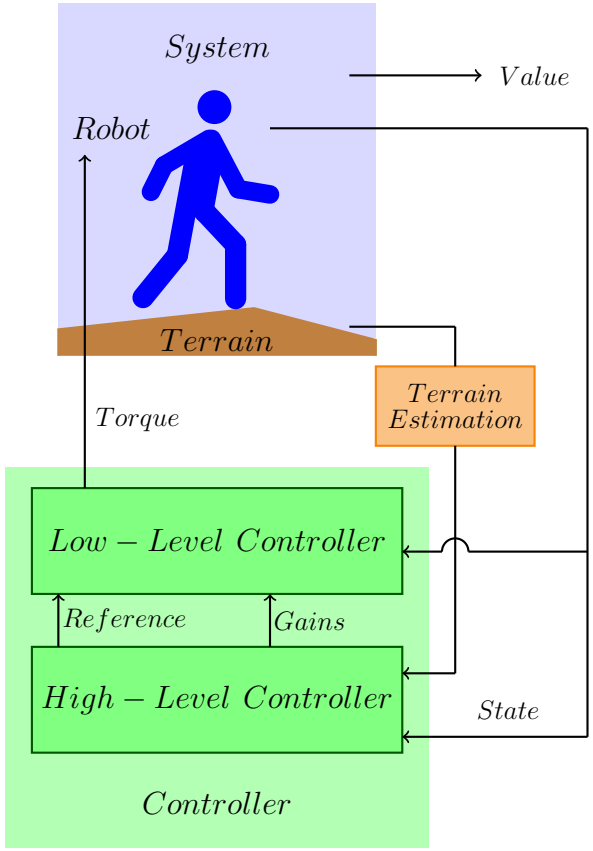

Figure 1: Cartoon representing the "big picture" for our framework.

$q := [q_1 \;\; q_2 \;\; q_3 \;\; q_4 \;\; q_5]^T$. The ten dimensional state of the robot is defined as $x := [q^T \;\; \dot{q}^T]^T$. We restrict our attention to planar motion and assume links are rigid. sh denotes the height of the swing foot. The model parameters are taken from RABBIT (Westervelt et al. (2007)) and listed in Table 1.

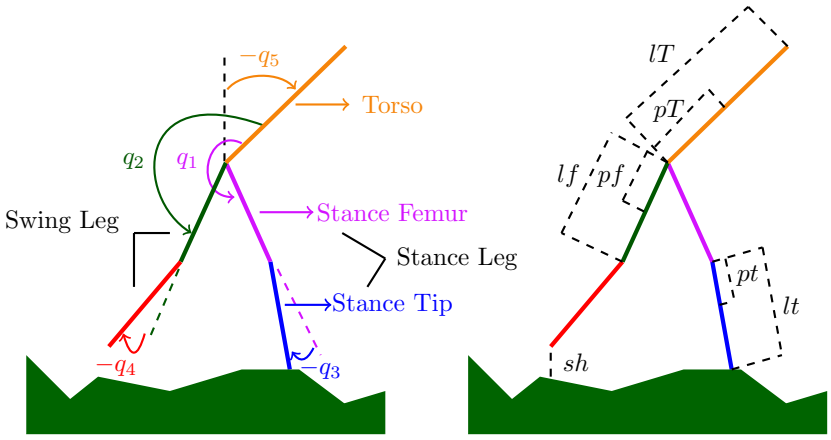

Figure 2: Illustration of the five-link robot with identical legs

Depending on the number of legs in contact with the ground, the robot will be either in the single or double support phase. Walking consists of these two phases in sequence. The single support (swing) phase has continuous dynamics, which can be derived in the following canonical form using a Lagrangian approach.

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu, \tag{1}$$

where u is the input. An important point is that q consists of the five angles depicted in Fig. 2 whereas u has only four elements. The system has this degree of underactuation because of the passive joint at the stance tip. The swing dynamics can be equivalently expressed as:

$$\dot{x} = \begin{bmatrix} \dot{q} \\ D^{-1}(-C\dot{q} - G + Bu) \end{bmatrix} =: f(x) + g(x)u. \tag{2}$$

During the swing phase of a successful step, the swing leg takes off from the ground, passes the stance leg and lands on a further point on the ground. So, each single support phase starts and ends with double support phases. As the robot has point feet, the double support (stance) phase can be well captured as an instantaneous impact event. The robot will experience this impact whenever the swing foot hits the ground ($sh = 0$) from above ($\dot{sh} < 0$). Let us denote this impact surface, aka the jump set, by IS. Then we have

$$x^+ = \Delta(x) \qquad x \in IS, \tag{3}$$

where x and x^+ are the states just before and after the impact respectively. Conservation of energy and the principle of virtual work gives the mapping

Table 1: Model Parameters for the Five-Link Robot

Description	Parameter Label	Value
Torso Mass	m_T	12 kg
Femur Mass	m_f	6.8 kg
Tip Mass	m_t	3.2 kg
Torso Inertia	I_T	1.33 $kg\ m^2$
Femur Inertia	I_f	0.47 $kg\ m^2$
Tip Inertia	I_t	0.20 $kg\ m^2$
Torso Length	l_T	0.63 m
Femur Length	l_f	0.4 m
Tip Length	l_t	0.4 m
Torso Mass Center	p_T	0.24 m
Femur Mass Center	p_f	0.11 m
Tip Mass Center	p_t	0.24 m
Gravitational Acceleration	g_0	9.81 m/s^2
Gear Ratio	n_g	50
Ground Friction Coefficient	μ_s	0.6
Saturation Limit	u_{sat}	50 Nm

Δ (Westervelt et al. (2007), Hurmuzlu and Marghitu (1994)). Essentially, this model assumes instantaneous, inelastic collisions between the swing leg tip and the ground, with instantaneous changes in velocities to reflect the effects of impulsive forces exerted on the robot. Although, the robot's position and orientation do not actually change according to the impact model, we relabel the legs every step, so the previous swing leg becomes the new stance leg and vice versa, i.e.,

$$\begin{bmatrix} q_1^+ & q_2^+ & q_3^+ & q_4^+ & q_5^+ \end{bmatrix} = \begin{bmatrix} q_2^- & q_1^- & q_4^- & q_3^- & q_5^- \end{bmatrix}. \tag{4}$$

Without relabeling, a periodic walking gait would have two steps as its period. Since a step consists of a single support phase and an impact event, it has hybrid dynamics as illustrated in Figure 3. In our modeling, we assume the impact event occurs first, but the order in the definition of a step is an arbitrary choice, so long as one remains consistent after deciding. As seen in Figure 3, for a step to be successful, certain "validity conditions" must be satisfied, which we list next. After impact, the former stance leg must lift from ground with no further interaction with the ground until the next impact. Also, the swing foot must have progressed past the stance foot before the impact of the next step occurs. Only the feet should contact the ground. Furthermore, the force on the stance tip during the swing phase, and the force on the swing tip at the impact

should satisfy the following friction constraint:

$$F_{friction} = F_{normal}\ \mu_s > |F_{transversal}|. \tag{5}$$

If validity conditions are not met, the step is labeled as "unsuccessful" and the system is modeled as transitioning to an absorbing failure state. This is a conservative model, because in reality violating these conditions does not necessarily mean failure.

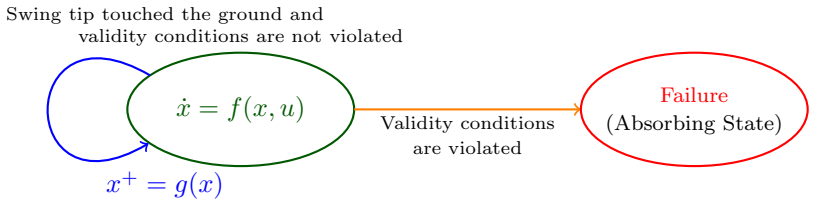

Figure 3: Hybrid model of a step and the failure state

4 Low-Level Control

The majority of the bipedal locomotion research has arguably focused on designing a low-level controller acting as desired. As a result, there are many schemes available. Our methods in the following sections are designed to work with any choice. In this section we will explain the particular structure we will use throughout this paper. We have found using piece-wise constant references to work very well (Saglam and Byl (2013*b*)), and for finite time convergence, we tack these references using a Sliding Mode Control (SMC) scheme, as explained in Sabanovic and Ohnishi (2011) and applied in Tzafestas et al. (1996).

Remember that the robot is underactuated, because there are five angles, but only four actuators. Thus, we define four variables to control. As a result of our experience (Saglam and Byl (2013*b*)), we proceed with

$$q_c := [\theta_2 \quad q_3 \quad q_4 \quad q_5]^T, \tag{6}$$

where $\theta_2 := q_2 + q_5$ is an absolute angle (swing leg femur) and q_c stands for angles to be controlled (See Figure 2). We select the input to the system in the following form.

$$u = (ED^{-1}B)^{-1}(v + ED^{-1}(C\dot{q} + G)),$$
$$where\ E = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

Substituting this u into (1) and noting $q_c = E\ q$, we obtain a simple equation for the second derivatives of the angles to be controlled:

$$\ddot{q}_c = v. \tag{8}$$

We then design v such that q_c acts as desired (i.e., converging toward q_c^{ref}). The error is given by

$$e = q_c - q_c^{ref}, \tag{9}$$

and the generalized error is defined as

$$\sigma_i = \dot{e}_i + e_i/\tau_i, \quad i = \{1, 2, 3, 4\}, \tag{10}$$

where τ_i values are time constants for each dimension of q_c. Note that when the generalized error is driven to zero, i.e. $\sigma_i = 0$, we have

$$0 = \dot{e}_i + e_i/\tau_i. \tag{11}$$

The solution to this equation is given by

$$e_i(t) = e_i(t_0)\ exp(-(t - t_0)/\tau_i), \tag{12}$$

which drives q_c to q_c^{ref} exponentially fast and justifies the name time constant for τ_i values. It can also be viewed as the ratio of proportional and derivative gains of a PD controller. Then, v in (8) is chosen to be

$$v_i = -k_i|\sigma_i|^{2\alpha_i - 1} sign(\sigma_i), \quad i = \{1, 2, 3, 4\}, \tag{13}$$

where $k_i > 0$ and $0.5 < \alpha_i < 1$ are called convergence coefficient and convergence exponent respectively. k_i is analogous to a gain common to both e_i and $\dot{e}_i$. $0.5 < \alpha_i < 1$ ensures finite time convergence to the "sliding surface", i.e., where there the generalized error, σ_i, is zero. Note that if we had $\alpha_i = 1$, this would be a PD controller. The controller parameters obtained after a short trial and error process are given in Table 2.

Table 2: Controller Parameters

i	1	2	3	4
α_i	0.7	0.7	0.7	0.7
τ_i	1/10	1/10	1/20	1/5
k_i	50	100	75	10

We will design controllers using the same controller parameters, but different references all in the following form

$$q_c^{ref} = \begin{cases} [\theta_2^{ref1} & q_3^{ref} & q_4^{ref1} & q_5^{ref}]^T, & condition, \\ [\theta_2^{ref2} & q_3^{ref} & q_4^{ref2} & q_5^{ref}]^T, & \text{otherwise}, \end{cases} \tag{14}$$

Table 3: Controller References

Controller	θ_2^{ref1}	θ_2^{ref2}	q_3^{ref}	q_4^{ref1}	q_4^{ref2}	q_5^{ref}
ζ_0	225°	204°	−60°	−21°	0°	0°
ζ_1	230°	210°	−45°	−25°	0°	−15°

where we selected *condition*, above, to be $\theta_1 := q_1 + q_5 < \pi$. Note that the references are piecewise constant and time-invariant. Two discrete sets, obtained by trial and error, are as given in Table 3.

As we will see in Section 7.1, controller ζ_0 and ζ_1 work best on downhill and uphill terrain respectively. In addition, we may easily obtain new controllers as linear combinations of rows in Table 3. Since controller parameters are the same for all, we will abuse the notation to note 'reference of ζ_i' simply by ζ_i. Then, we define

$$\zeta_i := i\zeta_0 + (1-i)\zeta_1, \qquad i \in (0,1)\,. \tag{15}$$

So, $\zeta_{0.5}$ would use the average of two references given in Table 3. Figure 4 illustrates the steady-state walking gaits on flat terrain for $\zeta_{0.5}$. We could of course optimize controller parameters or references; however, this is not a requirement to apply our methods. For much of this paper, we use just three of these controllers, so the resulting controller set Z will then be given by

$$Z := \{\zeta_0,\ \zeta_{0.5},\ \zeta_1\}. \tag{16}$$

In case we have 5 controllers we will mean having $Z = \{\zeta_0,\ \zeta_{0.25},\ \zeta_{0.5},\ \zeta_{0.75},\ \zeta_1\}$.

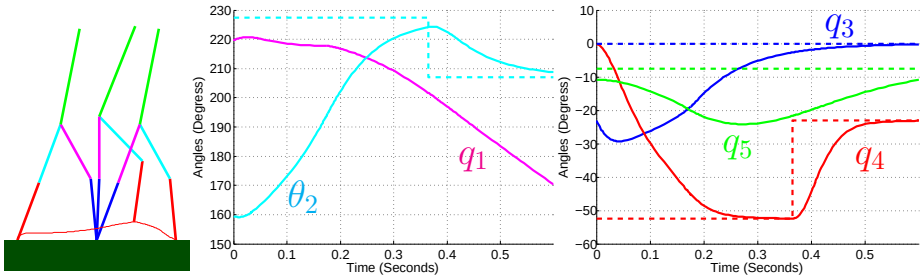

Figure 4: Steady-state walking gait (limit cycle) on flat terrain for $\zeta_{0.5}$. Angles over time are shown with solid lines. Dashed lines represent the references.

5 Meshing

5.1 The Methodology

We will assume the terrain profile is angular, i.e., that it consists of slopes, noted by γ. The slope ahead only changes at impacts; i.e., it remains constant until the next step. This terrain assumption captures the fact that to calculate the pre-impact state, the terrain for each step can simply be interpreted as a ramp with the appropriate slope. Our general method is still applicable to more complicated terrain models, and we note briefly that the most important modeling detail for future work is to consider vertical (tripping) obstacles in between the footholds. Moreover, the robot will not change controllers within a step. Then, we define a Poincaré section at the transition from single support phase to double support phase. For the rest of the paper, we abuse notation, and refer to $x \in IS$ simply as x. In this setting, the next state of the robot, $x[n+1]$, is a function of the current state $x[n]$, the slope ahead $\gamma[n]$, and the controller used $\zeta[n]$, i.e.

$$x[n+1] = h^t(x[n], \gamma[n], \zeta[n]). \tag{17}$$

Next, we will mesh to obtain a Markov Decision Process model of the system. Remember we already determined a controller set as in 16. We also need a wide-enough slope set S. Having a wider than needed slope range has no disadvantage, but too narrow a range would cause inaccuracy. The robot should *not* be able to walk at the boundaries of the slope set, otherwise we extend the range. For the controller set we have, -8 to 8 degrees is appropriate. Then, the slope set is

$$S = \left\{ k^\circ \ : \ \frac{k}{d_s} \in \mathbb{Z}, \ -8 \leq k \leq 8 \right\}, \tag{18}$$

where d_s determines the density, which just like range, can be chosen depending on the controllers' performance, and on the robot. Also, the slope set does not have to be evenly spaced in general, it may be denser around slopes of particular interest. As we increase the density of the slope set, we are able to capture the dynamics more accurately at the expense of higher computational costs. We discuss mesh accuracy in more detail in Section 7.2. We will typically use $d_s = 1$ in this paper.

Two key goals in meshing are: First, we want to have a set of states, Y, which well covers the (reachable) part of the state space the robot visits. Secondly, we want to learn what $h^t(y, s, \zeta)$ is for all $y \in Y$, $s \in S$, and $\zeta \in Z$. In our meshing algorithm, an initial mesh, Y_i, must be chosen. In this study, we use an initial mesh consisting of only two points. One of these points (y_1) represents all (conservatively defined) failure states, no matter how the robot failed, e.g. a foot slipped, or the torso touched the ground. The other point (y_2) is the stable fixed point of the robot model when the terrain is flat and only ζ_0 was used.

Then, our algorithm explores the reachable space deterministically. We initially start by setting $\overline{Y} = \{y \in Y, \ y \neq y_1\}$, which corresponds to all the states

that are not yet fully "expanded" via simulation to determine their reachable next states, given terrain stochasticity and our finite controller set. Then we start the following iteration: As long as there is a state $y \in \overline{Y}$, simulate to find all possible $h^t(y[n], s[n], \zeta[n])$ and remove y from $\overline{Y}$. For the points newly found, check their distance to other states in Y. Points exceeding a given distance metric from all other states in Y are then added to Y and $\overline{Y}$.

The crucial question is what should this distance metric be so that the resulting Y has a tractable number of states while accurately covering the reachable state space? We found using the standardized (normalized) Euclidean distance to be extremely useful. When a is a vector, and B is a set of vectors (growing in size, during meshing) each with the same dimension as a, the distance of a from B is calculated as

$$d(a, B) := \min_{b \in B} \left\{ \sqrt{\sum_i \left(\frac{a_i - b_i}{r_i} \right)^2} \right\}, \tag{19}$$

where r_i is the standard deviation of b_i elements. In addition, the closest point in B to a is given by

$$c(a, B) := \operatorname*{argmin}_{b \in B} \left\{ \sum_i \left(\frac{a_i - b_i}{r_i} \right)^2 \right\}. \tag{20}$$

We are now ready to present the pseudocode in Algorithm 1. Two important tricks to make the algorithm run faster are as follows. First, the slope set allows a natural cluster analysis. The distance comparison for a new point can be made only with the points that are associated with the same slope. This might result in more points in the final mesh, but it speeds up the meshing and later calculations significantly. Secondly, fix a controller ζ and a state y. Then we can simulate $h^t(y, min(S), \zeta)$ and then extract $h^t(y, s, \zeta)$ for all $s \in S$. In other words, in order robot to experience an impact at -8 degree, it has to pass through all the possible impact points with higher degrees, and we can extract all impact cases from a single simulation.

5.2 Dimension of the Mesh

While meshing uniformly makes a lot of sense for several dimensional systems, it becomes unfeasible as dimension increases. This is due to the exponentially growing number of points required to mesh for a fixed density or accuracy. To illustrate, say we fix distance between samples to be 1. Then covering $[1, 10]$ requires 10 points, whereas covering $[1, 10]^{10}$ requires 10^{10} points! This is not feasible for the methods we will apply in the following sections. The goal of our meshing method is to capture the reachable spaces for high dimensional systems while accurately approximating the dynamics governing there with a small number of points. This can be well-achieved for some systems, including the 5-Link Walker (10D) with a passive toe, presented here.

Algorithm 1 Meshing algorithm

Input: Initial set of states Y_i, Slope set S, Controller set Z and threshold distance d_{thr}

Output: Final set of states Y, and state-transition map

$\overline{Y} \leftarrow Y_i$ (except y_1)
$Y \leftarrow Y_i$
while $\overline{Y}$ is non-empty **do**
    $\overline{Y}_2 \leftarrow \overline{Y}$
    empty $\overline{Y}$
    for each state $\overline{y} \in \overline{Y}_2$ **do**
        for each slope $s \in S$ **do**
            for each controller $\zeta \in Z$ **do**
                Simulate a single step to get the final state x,
                when initial state is $\overline{y}$, slope ahead is s,
                and controller ζ is used. Store this information
                in the state-transition map
                if robot did not fall and $d(x, Y) > d_{thr}$ **then**
                    add x to $\overline{Y}$
                    add x to Y
                end if
            end for
        end for
    end for
end while
return Y, state-transition map

Note that after taking enough steps on flat terrain with a stable fixed controller, the walker converges toward a single point in the Poincaré map, corresponding to a limit cycle. However, as we have changing slopes and multiple controllers, we end up with a reachable region instead. This is illustrated in Figures 5 and 6, where we show the mesh obtained using two controllers, $d_s = 1$ and $d_{thr} = 1$. We picked x-axis to be q_1 and y-axis to be $\dot{q}_4$. The states which used ζ_0 (ζ_1) in the last step are shown with blue (red) points. We observe that each controller forms a surface that is clearly lower dimensional. So, we only need to mesh these manifolds, which can be done with a small number of points.

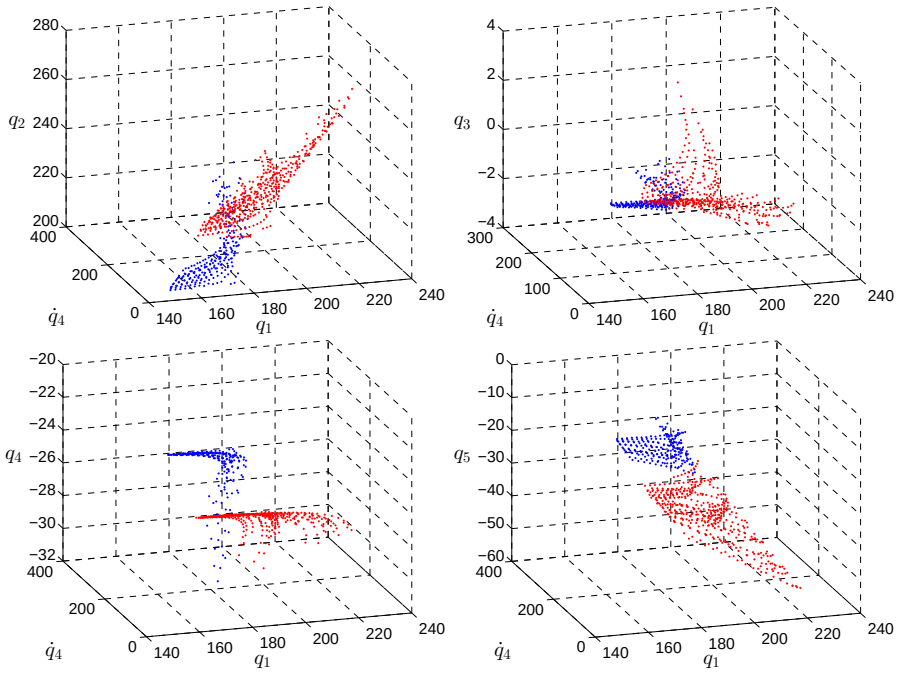

Figure 5: Angles (on the Poincaré map) for the mesh obtained with two controllers, $d_s = 1$ and $d_{thr} = 1$. Blue (red) dots shows states which resulted from using ζ_0 (ζ_1) in the last step. Angles are in degrees and $\dot{q}_4$ is in deg/sec.

To estimate the intrinsic dimension of the reachable space, we do Principal Component Analysis (PCA) (Pearson (1901)). We first shift the data to have zero mean at each dimension and then calculate the covariance matrix. We find the eigenvalues of this matrix and scale them so that they sum to one. We order and name them as $\lambda_1^{pca} \geq \lambda_2^{pca} \geq ....$ It is a rule of thumb to pick as many eigenvalues as needed to cover 80-90%, which will give information on the intrinsic dimension. Table 4 illustrates the results for various meshes. It is clear that a single eigenvalue is not enough (between 71% and 77%), but the first two are arguably sufficient (between 92% and 94%). Remember our main concern is the number of points in a final mesh, which is also given in the table. Our

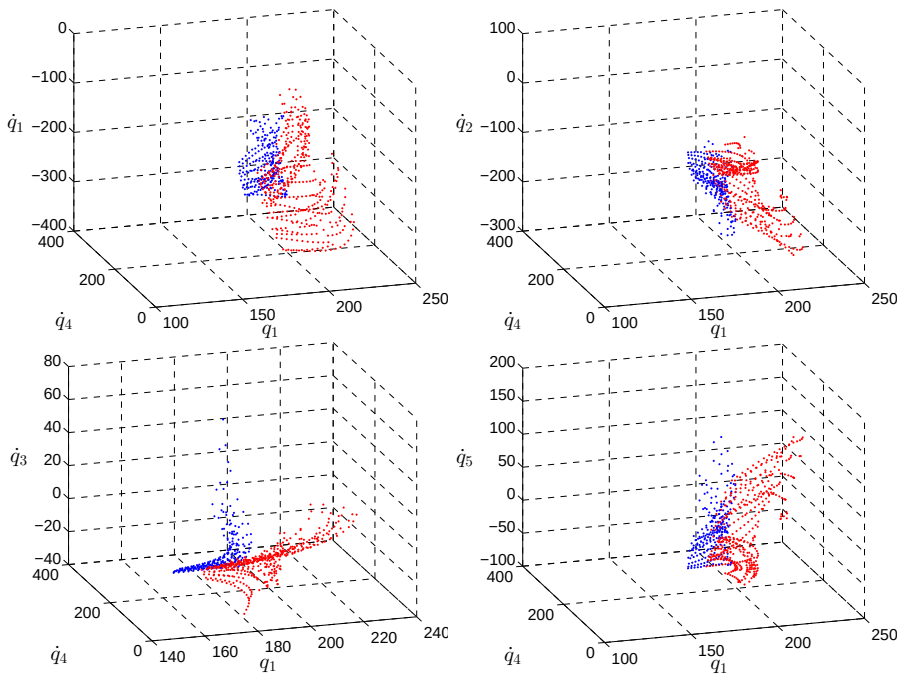

Figure 6: Velocities (on the Poincaré map) for the mesh obtained with two controllers, $d_s = 1$ and $d_{thr} = 1$. Blue (red) dots shows states which resulted from using ζ_0 (ζ_1) in the last step. q_1 is in degrees and velocities are in deg/sec.

conclusion is that although 5-Link biped is 10 dimensional, dynamics converge to quasi-2D manifolds.

Table 4: Dimension Statistics

i	d_s	d_{thr}	Points	λ_1^{pca}	λ_2^{pca}	λ_3^{pca}	$\lambda_1^{pca}+\lambda_2^{pca}$
1	1	1	304	0.7367	0.1921	0.0644	0.9288
		0.5	1530	0.7690	0.1556	0.0689	0.9246
	0.25	1	1685	0.7135	0.2113	0.0683	0.9248
		0.5	7191	0.7571	0.1671	0.0695	0.9242
2	1	1	692	0.7394	0.1858	0.0411	0.9251
		0.5	2540	0.7357	0.1926	0.0403	0.9284
	0.25	1	3204	0.7588	0.1703	0.0381	0.9291
		0.5	12480	0.7624	0.1687	0.0379	0.9311
5	1	1	2714	0.7267	0.2089	0.0355	0.9356
		0.5	14583	0.7220	0.2143	0.0375	0.9363
	0.25	1	12991	0.7371	0.1965	0.0370	0.9337
		0.5	70360	0.7450	0.1907	0.0374	0.9357

5.3 Obtaining a Markov Chain

Now thanks to the meshing (state-transition map), we know $h^t(y, s, \zeta)$ for all $y \in Y$, $s \in S$, and $\zeta \in Z$. We define

$$h^a(x[n], \gamma[n], \zeta[n]) := c(h^t(x[n], \gamma[n], \zeta[n]), Y), \tag{21}$$

where (20) is used, and the superscript a stands for approximation. Then we write the approximate step-to-step dynamics as

$$y[n+1] = h^a(y[n], s[n], \zeta[n]). \tag{22}$$

After that, the deterministic state transition matrix can be written as

$$T_{ij}^d(s, \zeta) = \begin{cases} 1, & \text{if } y_j = h^a(y_i, s, \zeta) \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

Note that (23) is the result of our basic, nearest-neighbor approximation, which appears to work well in practice. More sophisticated approximations results in transition matrices not just having one or zero elements, but also fractional values in between. This increases memory and computational costs while, to our experience, not providing much increase in accuracy.

A Markov Chain can be represented by a stochastic state-transition matrix T^s is defined by

$$T^s_{ij} := Pr(y[n+1] = y_j \mid y[n] = y_i). \tag{24}$$

To calculate this matrix, we first need to assume a distribution over slope set, P_S, given by

$$P_S(s) := Pr(s[n] = s). \tag{25}$$

In this paper, we will assume a normal distribution for P_S, with mean μ_s, and standard deviation σ_s, i.e.,

$$s[n] \sim \mathcal{N}(\mu_s, \sigma_s^2). \tag{26}$$

After distributing s values, we end up with a Markov Decision Process model. The final step needed to describe a Markov Chain is to decide on a policy for selecting which controller to use. We will study this in detail, but for now let's assume only one of the controllers, say ζ_i is used (no switching). Then, T^s can be calculated as

$$T^s = \sum_{s \in S} P_S(s)\ T^d(s, \zeta_i). \tag{27}$$

The definition of T^s will always remain the same, but its calculation will be updated when we consider switching between controllers.

As we decrease d_{thr} and d_s, we have more accurate representations of the full dynamics at the expense of a higher number of states in the final mesh. However, we claim the accuracy converges rapidly, as we will show in section 7.2.

6 Mean First Passage Time (MFPT)

We study bipedal walking as a metastable system process (Byl and Tedrake (2009)). In the presence of noise (e.g., rough terrain), the robot will eventually fall, but the number of steps before doing so might be very high (millions or more). This section serves as a summary on how we estimate this number. We invite interested reader to Saglam and Byl (2014*a*).

The eigenvalues of T^s cannot have magnitude larger than one. However, the largest eigenvalue will be 1 because we model the failure as absorbing. Let λ_2 be the second largest eigenvalue. λ_2 will be non-negative and real. For metastable (rarely-falling) walking, it will be close to but smaller than 1.

If a metastable robot does not fall within several steps, then it converges to so called metastable distribution, ϕ, across state space (probabilistically). The fascinating fact is what happens when the robot takes a single step after that. With probability $1-\lambda_2$, the robot is going to fall during the next step. If it does not fall, then the pdf describing the robot state remains unchanged. Then, the probability of taking n steps only, equivalently falling at the nth step is simply

$$Pr(y[n] = y_1,\ y[n-1] \neq y_1) = \lambda_2^{n-1}(1-\lambda_2). \tag{28}$$

For $\lambda_2 < 1$, realize that as $n \to \infty$, the right hand side goes to zero, i.e., the system will eventually fail. Note that we also count the step which ended up

falling as a step. This can be verified considering "falling down" at the first step (taking 1 step only). When $n = 1$ is substituted, we get $1 - \lambda_2$ as expected. Then, the expected number of steps can be then calculated as

$$\begin{aligned} MFPT &= E[FPT] \\ &= \sum_{n=1}^{\infty} n \; Pr(y[n] = y_1, \; y[n-1] \neq y_1) \\ &= \sum_{n=1}^{\infty} n\lambda_2^{n-1}(1-\lambda_2) = \frac{1}{1-\lambda_2}, \end{aligned} \tag{29}$$

where we used the fact that $\lambda_2 < 1$. As a result, MFPT can be calculated using

$$M = \begin{cases} \infty & \lambda_2 = 1, \\ \frac{1}{1-\lambda_2} & \lambda_2 < 1. \end{cases} \tag{30}$$

Note that being stable corresponds to $\lambda_2 = 1$, but we will introduce enough roughness so that we always have $\lambda_2 < 1$. This will be achieved with a wide-enough slope set and a high enough σ_s. To illustrate, we look at the mesh obtained with 3 controllers, $d_{thr} = 0.5$ and $d_s = 1$. We consider when $\sigma_s = 1.5$ and $\mu_s = 0$ deg. Then the MFPT is calculated to be 1195 for $\zeta[n] = \zeta_0$, 656 for $\zeta[n] = \zeta_{0.5}$, and 39759 steps for $\zeta[n] = \zeta_1$. Thus, we are done with the first goal mentioned in Section 2: Estimating the average number of steps before failure.

We are also interested in obtaining the MFPT vector, m, which gives the MFPT for each state. It is defined as

$$m_i := \begin{cases} 0 & i = 1, \\ 1 + \sum_j T_{ij}^s m_j & \text{otherwise.} \end{cases} \tag{31}$$

The equation above says that the robot will take zero steps to fall if it has failed already. Otherwise, the number of steps until failure is 1 less after a step is taken. We also note here the equality $M = m'\phi$ (Saglam and Byl (2014*a*)). We present Figure 7 to show the MFPT vector of the system summarized in the previous paragraph. Note that the majority of states are either destined to fall (0 steps), or are as stable as the system-wide MFPT for the respective controller.

We note that Monte Carlo simulations are not a computationally practical means of verifying MFPT when it is very high, which has motivated our testing methodology throughout. However, a Monte Carlo study was presented in Saglam and Byl (2013*a*) for smaller numbers of steps and fixed controllers.

7 High-Level Control

Remembering Figure 1, the second goal in Section 2 was to obtain robust, near-optimal control policies for low-level controllers. In general form, a policy will

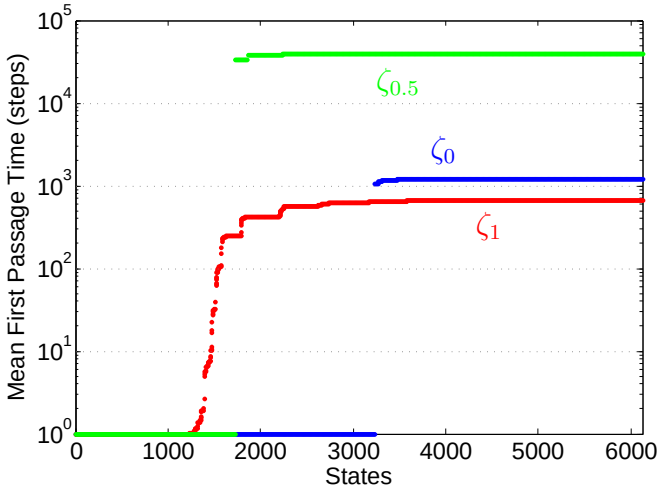

Figure 7: MFPT vectors for $\sigma_s = 1.5$ deg, $\mu_s = 0$ deg, and three possible controllers. The mesh obtained with $d_{thr} = 0.5$ and $d_s = 1$.

be a function of the noisy slope information and the current state. However, we will proceed step by step. More simplified policies will be special cases of our general framework. We will start with three controllers, i.e., $Z = \{\zeta_0,\ \zeta_{0.5},\ \zeta_1\}$.

7.1 Fixed Controllers

The simplest policy is to use just one controller, i.e.,

$$\zeta[n] = \zeta_i \tag{32}$$

for fixed i. In this case T^s is calculated as in (27) for a given μ_s and σ_s. We then get the second largest eigenvalue and calculate MFPT as in (30). So for a mesh, the MFPT will be a function of (1) the policy, (2) μ_s, and (3) σ_s.

We will look at Figure 8 in more detail soon, but for now let's just focus on the solid lines and understand how to interpret the data in the figure. We will typically fix $\sigma_s = 1.5$ deg, μ_s will be the x-axis, the y-axis will show the MFPT, and we will distinguish different policies by labels. A particular location on the x-axis represents a particular long-term mean slope case. Once we have the mesh (including T^d), it is fast to calculate these figures, which allows us to evaluate each controller for different terrain conditions on its own and relative to others. Roughly, the figure shows that controller ζ_0, $\zeta_{0.5}$, and ζ_1 each perform best when $\gamma < -0.5$ (downhill), $-0.5 \leq \gamma \leq 0.5$ (flat), and $0.5 < \gamma$ (uphill) respectively.

A good portion of research on bipedal walking has concentrated on finding the best ζ_i. We just showed our way of evaluating different controllers' performance. On the other hand, a particular ζ_i might be optimal only for a small

region in the slope set, meaning local terrain features can be better negotiated through switching control. In addition, the optimal ζ_i is different for different cost definitions, e.g., including energetics or other aspects in addition to MFPT.

7.2 Convergence of the Mesh

Note that the number of points in the final mesh, and the accuracy obtained from (22) with this mesh, are inversely related to parameters d_{thr} and d_s. We claim that, as $d_s \to 0$ and $d_{thr} \to 0$, the mesh captures the true, hybrid dynamic system dynamics. In addition we argue that, for a dense enough slope set, as $d_{thr} \to 0$, the accuracy of the mesh, and as a result, the MFPT of the controllers, converges. For $d_s = 1$, we illustrate convergence numerically by first fixing $\sigma_s = 1.5$ (deg) and plotting six independently obtained meshes with $d_{thr} = \{0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6\}$. In all plots that follow, each location on the x-axis assumes a different long-term mean in slope, resulting in a different stochastic transition matrix, from (24).

Figure 8 shows the difference in results for $d_{thr} = 0.1$ (more refined) versus $d_{thr} = 0.5$ (more coarse). Here, solid lines are associated with $d_{thr} = 0.1$. Figure 9 has $d_{thr} = 0.2$ instead of $d_{thr} = 0.5$. Comparing the two figures we observe the convergence, which is also shown in Table 5. The second row gives the number of points in the mesh, while the third row shows the total area of the difference with $d_{thr} = 0.1$ plot.

Table 5: Mesh Convergence

d_{thr}	0.6	0.5	0.4	0.3	0.2	0.1
Size	4.154	6,126	10,531	21,726	65,066	394,420
err	14.4813	11.6452	7.9576	5.1959	3.3078	0

Figure 10 has $d_{thr} = 0.2$ fixed and compares $d_s = 1$ mesh (65,066 points) with $d_s = 1/3$ mesh (226,489 points). As we expected, there is a little difference.

For the following sections, the $d_s = 1$ & $d_{thr} = 0.5$ mesh, which has 6,126 points, will be used to solve for policies.

7.3 Visual Walking

After seeing that the performance of the individual controllers depends significantly on the mean slope ahead, we consider policies using only one-step lookahead ($\gamma[n]$) information, which are in the form of

$$\zeta[n] = \pi(\gamma[n]). \tag{33}$$

The approximate dynamics in (22) become

$$y[n+1] = h^a(y[n], s[n], \pi(s[n])), \tag{34}$$

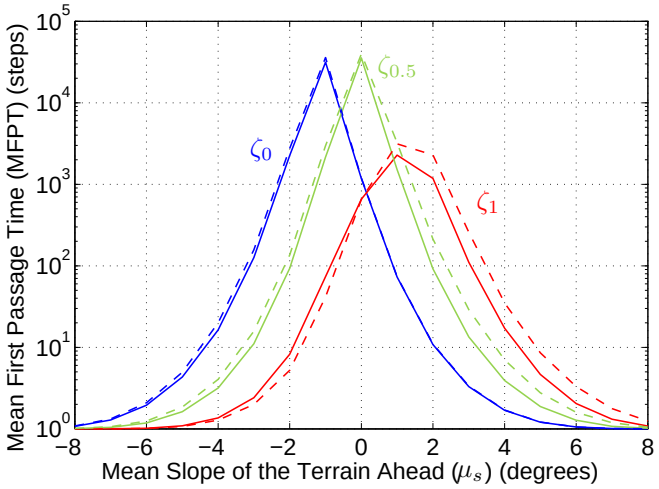

Figure 8: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Figure shows average number of steps before falling calculated using (30) versus μ_s for two independently obtained meshes. Solid lines represent the mesh with 394,420 states (obtained using Algorithm 1 with $d_{thr} = 0.1$), whereas the dashed lines are result of choosing $d_{thr} = 0.5$, which results in 6,126 states. $d_s = 1$ was fixed.

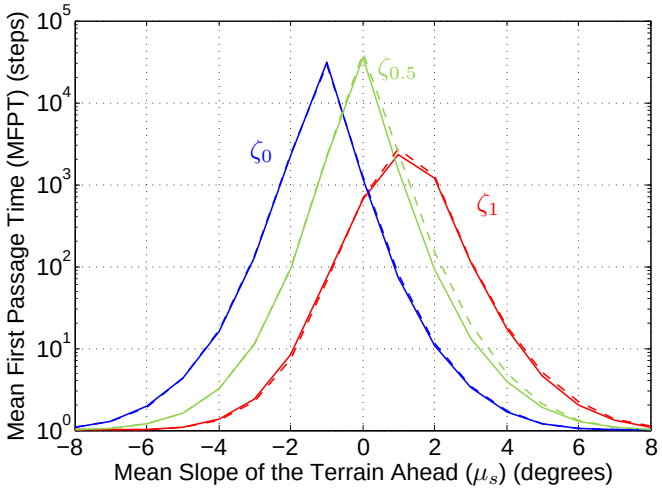

Figure 9: Same as Figure 8, except the dashed line now represents $d_{thr} = 0.2$, which results in 65,066 states.

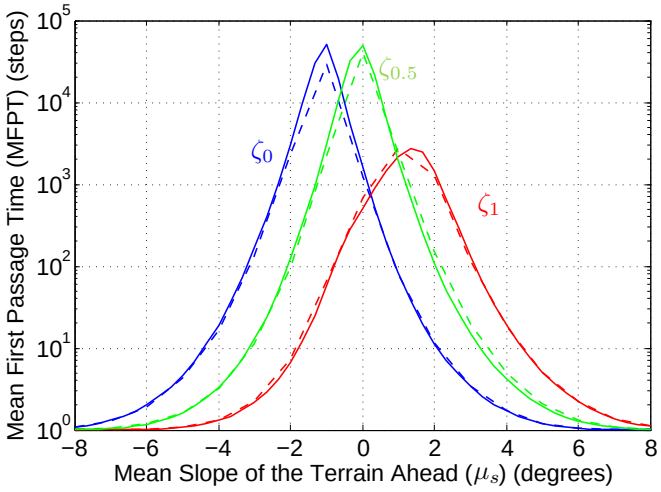

Figure 10: Same as Figure 9, except solid lines represent $d_s = 1/3$, which results in 226,489 states.

and T^s is updated as

$$T^s = \sum_{s \in S} P_S(s)\ T^d(s, \pi(s)). \tag{35}$$

We present three such policies in Figure 11. The "trivial" one uses ζ_0, $\zeta_{0.5}$, and ζ_1 when $\gamma \leq -1$ (downhill), $-1 < \gamma \leq 1$ (flat), and $1 < \gamma$ (uphill) respectively. It is surprising to see how bad this intuitive policy turns out to be! After trying all possibilities to maximize MFPT at zero mean, we tuned to get the "Visual" policy (i.e., based on terrain slope, but not on robot state): Use ζ_0, $\zeta_{0.5}$, and ζ_1 when $\gamma \leq -3$ (downhill), $-3 < \gamma < 5$ (flat), and $5 \leq \gamma$ (uphill) respectively. We do the same again by assuming $\zeta_{0.5}$ is not available and get the "Visual with 2 controllers" policy. We conclude that $\zeta_{0.5}$ is very useful for visual walking, improving MFPT on zero-mean slope by more than 36 times, as shown in Figure 11.

7.4 Blind (to the terrain) Walking

It is typical to assume policy to be a function of the state in Markov Decision Processes, i.e.,

$$\zeta[n] = \pi(y[n]). \tag{36}$$

When this policy is applied, the approximate dynamics in (22) will become

$$y[n+1] = h^a(y[n], s[n], \pi(y[n])). \tag{37}$$

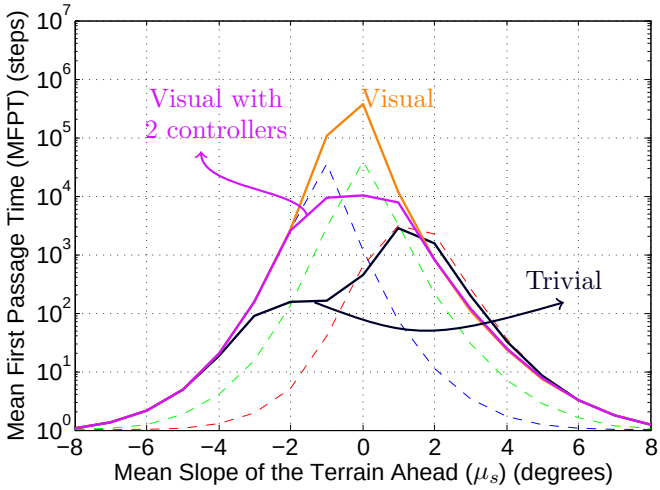

Figure 11: Visual walking, i.e., using only one-step lookahead ($\gamma[n]$) information. Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Figure shows average number of steps before falling calculated using (30) versus μ_s for three different such policies. Fixed controllers ($\zeta_0,\ \zeta_{0.5},\ \zeta_1$) are shown for reference with dashed lines.

In this case, (27) should be also updated as

$$T^s_{ij} = \sum_{s \in S} P_S(s)\ T^d_{ij}(s, \pi(y_i)). \tag{38}$$

We then use value iteration (Bellman (1957)) to get the optimal policy. The algorithm works by recursively calculating

$$V(i) := \max_{\zeta} \left\{ \sum_j P_{ij}(\zeta)\ (R(j) + \alpha\ V(j)) \right\}, \tag{39}$$

where V is the value, $P_{ij}(\zeta)$ is the probability of transitioning from y_i to y_j when ζ is used, $R(j)$ is the reward for transitioning to y_j, and α is the discount factor, which is chosen to be 0.9. This equation is iterated until convergence to get the optimal policy. Remember that the failure state is y_1. The value of the failure state will initially be zero, i.e.,

$$V(1) = 0, \tag{40}$$

and it will always stay as zero, because the reward for taking a successful step is one, while falling is zero, i.e.,

$$R(j) = \begin{cases} 0, & j = 1, \\ 1, & \text{otherwise.} \end{cases} \tag{41}$$

Note that the reward function we use does not depend on the controller, slope ahead, or current state. Use of more sophisticated reward functions (e.g., considering energy, speed, step width) is a topic of Saglam and Byl (2014*b*). However, our focus is on stability in this paper. Substituting (40) and (41) into (39), we obtain

$$V(i) := \max_{\zeta} \left\{ \sum_{j \neq 1} P_{ij}(\zeta) \; (1 + \alpha \; V(j)) \right\}. \tag{42}$$

The probability term is

$$P_{ij}(\zeta) = \sum_{s \in S} P_S(s) \; T_{ij}^{d}(s, \zeta). \tag{43}$$

Optimization results for $\mu_s = 0$ and $\sigma_s = 1.5$ deg are shown in Figure 12. In the light of this figure, we again conclude that use of the third controller, $\zeta_{0.5}$, is very helpful for blind walking.

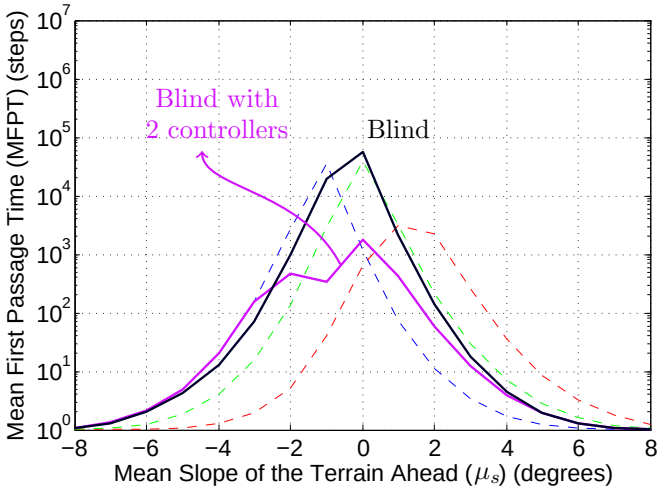

Figure 12: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. In addition to the three fixed controllers (dashed lines), there are two different policies, which walk "blindly", i.e., they have no information about the next slope ahead. The black (top) policy makes use of all three controllers, whereas only ζ_0 and ζ_1 are assumed to be available while getting purple policy.

7.5 Sighted Walking

By "sighted" we mean using both the state information and one-step lookahead to terrain, i.e.

$$\zeta[n] = \pi(y[n], s[n]). \tag{44}$$

In this case, the approximate dynamics in (22) become

$$y[n+1] = h^a(y[n], s[n], \pi(y[n], s[n])), \tag{45}$$

and T^s should also be updated as

$$T^s_{ij} = \sum_{s \in S} P_S(s)\ T^d_{ij}(s, \pi(y_i, s)). \tag{46}$$

To use the one-step lookahead in deriving policy, we will modify the value iteration as

$$V(i) := \sum_{s \in S} \max_{\zeta} \left\{ \sum_{j \neq 1} P_{ij}(\zeta, s)\ (1 + \alpha\ V(j)) \right\}. \tag{47}$$

Instead of modifying the value iteration algorithm, we could define a new 11D state, including the slope in addition. However, (47) makes the analysis of the following parts easier, reduces computational cost, and requires less memory. The probability of 'having s as the slope ahead' and 'transitioning from y_i to y_j when ζ is used' is simply the multiplication of these two probabilities. So, we have

$$P_{ij}(\zeta, s) = P_S(s)\ T^d_{ij}(s, \zeta). \tag{48}$$

We optimize with $\mu_s = 0$ and $\sigma_s = 1.5$ degrees to get Figure 13. Noting the logarithmic y-axis, it is clear that sighted walking is significantly better than visual and blind walking, as one would intuitively expect. The ability to quantify this intuition is a driving goal of our work.

Although Figure 13 is impressive, for this methodology to be applicable to real-life problems, the policies must be robust to uncertainties. So, for the rest of the paper, we will focus on improving robustness of sighted walking.

7.6 Noisy Slope Estimation

We start our study of robustness by considering the addition of noise to slope information. The slope ahead will still be defined by variable s, but the controller will think it is (closest to) $\tilde{s} \in S$, due to the noise $l \in S$. The relationship will be given by

$$\tilde{s} = max(min(S), min(max(S), s + l)). \tag{49}$$

Note that this says $\tilde{s} = s + l$ except at boundaries of the slope set. $P_L(l)$ will be defined by

$$P_L(l) := Pr(l[n] = l), \tag{50}$$

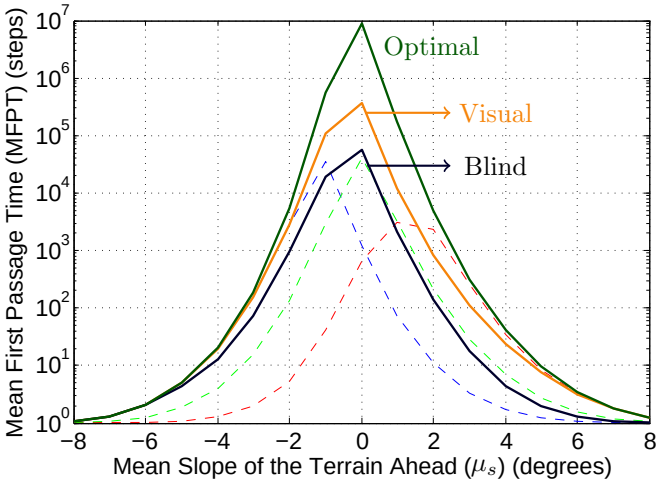

Figure 13: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. Three fixed controllers (dashed) are repeated for reference. Visual policy (orange-middle) from Figure 11 and blind policy (black-bottom) from Figure 12 are plotted alongside the new sighted policy (green-top).

and normally distributed with zero mean and standard deviation σ_l, i.e.,

$$l[n] \sim \mathcal{N}(0, \sigma_l^2). \tag{51}$$

In the presence of noise, the policy will be a function of $\tilde{s}$, not s. Thus, we have

$$\zeta[n] = \pi(y[n], \tilde{s}[n]). \tag{52}$$

Then the approximate dynamics (22) will be

$$y[n+1] = h^a(y[n], s[n], \pi(y[n], \tilde{s}[n])). \tag{53}$$

The stochastic state-transition matrix must also be updated to consider noise as

$$T_{ij}^s = \sum_{s \in S} \sum_{l \in S} P_S(s)\ P_L(l)\ T_{ij}^d(s, \pi(y_i, \tilde{s})). \tag{54}$$

Figure 14 shows what happens to the policy we obtained in the previous section (green-top plot), in the presence of $\sigma_l = 1$ deg noise (purple-bottom plot). We lose almost all the improvements gained by switching! To account for noise in the slope information while optimizing, we first rewrite the modified value iteration algorithm as

$$V(i) := \sum_{\tilde{s} \in S} \max_{\zeta} \left\{ \sum_{j \neq 1} P_{ij}(\zeta, \tilde{s})\ (1 + \alpha\ V(j)) \right\}. \tag{55}$$

The equation is essentially the same as (47), except we made clear $\tilde{s}$ is available to the controller instead of s. The probability of 'thinking $\tilde{s}$ is the slope ahead' and 'transitioning from y_i to y_j when ζ is used' is then

$$P_{ij}(\zeta, \tilde{s}) = \sum_{s \in S} \sum_{l \in S} P_S(s) P_L(l)\ f_S(s, l, \tilde{s})\ T_{ij}^d(s, \zeta), \tag{56}$$

where

$$f_S(s, l, \tilde{s}) = \begin{cases} 1, & \tilde{s} = max(min(S), min(max(S), s + l)) \\ 0, & \text{otherwise.} \end{cases} \tag{57}$$

The blue (middle) plot in Fig. 14 is the policy obtained and plotted assuming $\sigma_l = 1$ deg. Note that the new policy performs almost as well now with noisy slope information as the original policy did using noise-free data. Not surprisingly, as the noise goes down, the new policy performs better. More importantly, as the magnitude of the noise increases, we find performance does not suddenly drop. These data support our hypothesis that accounting for lookahead uncertainty is extremely important and can be done well without a precise noise model.

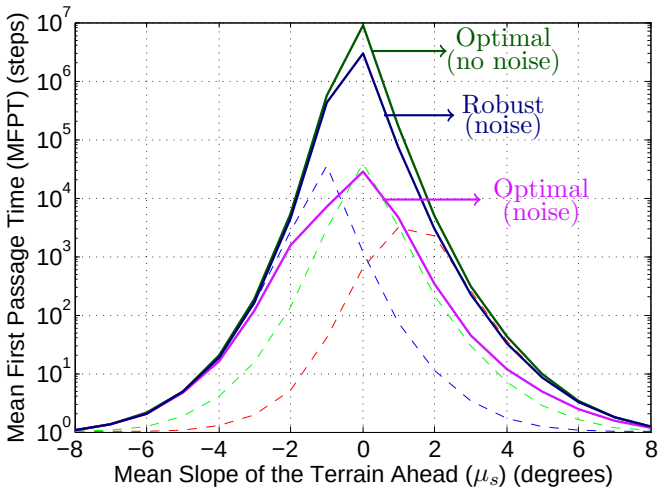

Figure 14: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. Figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. The top plot of Figure 13 is repeated for reference. This policy performs poorly due to the noise (bottom plot). However, as the middle plot shows, we can recover the loss greatly by using (55). Fixed controllers $(\zeta_0,\ \zeta_{0.5},\ \zeta_1)$ are shown for reference.

7.7 Small Mesh Policy on Big (Refined) Mesh

Up until now, we optimized policies and plotted resulting performance using the same ($d_{thr} = 0.5$) mesh. In this section, we keep optimizing on the coarse ($d_{thr} = 0.5$) mesh, but we estimate performance using a bigger, more refined ($d_{thr} = 0.1$) mesh, intended to better approximate the true system dynamics. As presented in Table 5, the small mesh has 6,126 points, while the big mesh has 394,420 points, meaning the small mesh requires significantly lower computational cost in meshing process and policy optimization. Thus, quantifying and improving robustness to mesh size are important issues.

As before, we will assume the small ($d_{thr} = 0.5$) mesh used to derive a policy is completely known. However, we will assume the larger, high-resolution mesh is not known during value iteration, thus it cannot be used while finding the policy. The larger mesh will be only used to estimate how well the small-mesh policy would work on the true system, i.e., it approximates how the policy behaves when substituted to (17). First, we need to explain how the small-mesh policy can be applied when the slope ahead, γ, may not be in the slope set, and/or state x may not be in the mesh. In these cases, we will apply the most basic idea: The controller will be picked assuming the slope ahead is $s \in S$ closest to γ and the current state is $y \in Y$ closest to x. So the policy now is

$$\zeta[n] = \pi(c(x[n], Y), c(\tilde{\gamma}[n], S)), \tag{58}$$

where $\tilde{\gamma}[n]$ is the noisy slope ahead information, and function c is as defined in (20). In its general form, let us denote the big mesh by Y_b, which is obtained from a slope set S_b. Using this mesh, we can approximate how (58) would behave in the higher fidelity mesh.

$$\zeta[n] = \pi(c(y_b[n], Y), c(\tilde{s}_b[n], S)), \tag{59}$$

where $y_b \in Y_b$ and $\tilde{s}_b \in S_b$. The definition and calculation of T^s remain the same, but this time Y_b and S_b will be used in obtaining it. Figure 15 shows the policy from the last section for reference (blue-top plot). The magenta (bottom) plot shows what happens when this policy is applied on Y_b, when Y_b is obtained with $S_b = S$, but $d_{thr} = 0.1$. (We consider $S_b \neq S$ later, in Section 7.8.) Comparing the purple and blue curves in the figure we immediately note that there is a huge drop. We must once again refine our algorithm, this time to improve robustness to meshing discretization.

In our approach to fix this issue, we consider the following: While the actual state is y_i, the robot may think it is y_k. To make this clear, we rewrite the value iteration algorithm as

$$V(k) := \sum_{\tilde{s} \in S} \max_{\zeta} \left\{ \sum_{j \neq 1} P_{kj}(\zeta, \tilde{s}) \; (1 + \alpha \; V(j)) \right\}. \tag{60}$$

Note that we only exchanged i with k, but this will make future notation easier to follow. The probability of 'thinking $\tilde{s}$ is the slope ahead', 'thinking the state

is y_k', and 'transitioning to y_j when ζ is used' is

$$P_{kj}(\zeta, \tilde{s}) = \sum_{s \in S} \sum_{l \in S} \sum_{i} P_S(s) P_L(l)\ f_S(s, l, \tilde{s})\ P_{ik}^P\ T_{ij}^d(s, \zeta), \tag{61}$$

where P_{ik}^P is the probability of being at state y_i when robot thinks the state is y_k, i.e.,

$$P_{ik}^P := Pr(y[n] = y_i \mid \tilde{y}[n] = y_k). \tag{62}$$

Finding the best calculation for P_P is an open question. However, it is intuitive that for a given state k (the robot thinks the state is y_k), P_{ik}^P should be smaller for i for which $d(y_i, y_k)$ is larger. Among many other methods, inverse distance weighting as in Shepard (1968) did not give good performance. In Saglam and Byl (2014*c*), we illustrate that an exponential distribution works well. However, selecting right parameterization was not obvious. In this paper, we propose using a more understandable distribution:

$$P_{ik}^P = \frac{\lambda^c}{\sum_c \lambda^c} \approx \lambda^c (1 - \lambda), \tag{63}$$

where y_i is the c^{th} closest state to y_k and $0 < \lambda < 1$ is the distribution parameter. Note that this is just a power distribution scaled to have $\sum_k P_{ik}^P = 1$. As we increase λ, robustness increases but performance drops. Very small λ would mean $P_{ii}^P = 1$ and $P_{ik}^P = 0$ for $i \neq k$, i.e., what we had before this section. $\lambda = 1$ would try to consider all points in the mesh with equal probability no matter what state information is. This is different than the visual policy, and it results in very bad performance.

An intuitive description of this approach is as follows. In Figure 7 we observe that the majority of the states are either destined to fall (have small MFPT) or likely to take many steps. When we look for the right policy for a state, an intuitive idea is applying the following iteratively: If multiple controllers are giving good performance, look at the next closest state. For example, say the closest states to y_7 are y_{12} and y_5. Controller $\zeta_{0.5}$ and ζ_0 are both working well and better than controller ζ_1 for states y_7 and y_{12}. Also $\zeta_{0.5}$ performs better than ζ_0 on y_5. Then we would use $\zeta_{0.5}$ for y_7. At this point please take a moment to understand (31). Due to nature of its definition, m_i values may potentially accumulate errors. This is because if we did not consider that states closest to y_7 might have similar dynamics as in y_{12} and y_5, we might have a wrong estimate on m_7. As a result, all m_i values depending on m_7 would be mistaken and so on. By defining (63) we do not only help choosing the right controller for y_7. More importantly, we let other states know about the possibilities at y_7.

It is important to note that robustness to slope and state information is somewhat similar in the following sense: If the slope ahead or the current state is different from what the robot thinks, it may end up at a different point than it estimated beforehand. So in both cases, the optimization tries to account for ending up in a different state than our approximate discrete model predicts. Since the approach in (63) is only ad hoc, we also try increasing the robustness to

slope information to see whether this actually helps when evaluating MFPT on the big mesh. While optimizing policy, instead of a normal distribution for P_S we assume a uniform distribution. Thus, we will optimize assuming the probability of having -7° as the slope ahead is the same as the probability of having 2°. This idea eliminates the need to choose both μ_s and σ_s for optimization. We also assume sensing noise is $\sigma_l = 3$ (deg) while optimizing to increase robustness. These calculations result in the orange curve in Figure 15.

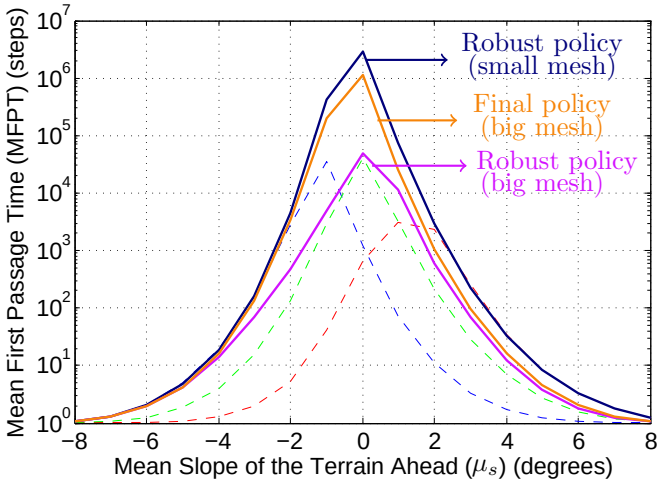

Figure 15: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. All polices are obtained using the mesh via $d_{thr} = 0.5$. However, Figure shows the results when these policies are evaluated on the $d_{thr} = 0.1$ mesh to estimate the actual performance. The blue (top) plot is the middle plot of Figure 14 shown for reference. This policy performs poorly (bottom plot) on the denser mesh. However, as the middle (orange) plot shows, we can recover the loss greatly by using (60).

7.8 Increasing Mesh Resolution for Slope Set, S_b

We then show what happens if the slope set is different for optimization than for the system under evaluation, i.e., $S_b \neq S$. We apply the final policy from Section 7.7 on a new mesh obtained using $d_{thr} = 0.2$ and $d_s = 1/3$, which has 226,489 points. We plot for $\sigma_s = 1.5$ and $\sigma_l = 1$ deg. Figure 16 presents the results, where final policy from Figure 15 is shown for reference. We see that the policy is quite similar when the slope set is denser, which encourages us to think that the final policy would also yield similar performance when simulating the full dynamics (Eq. 1). Figure 16 also shows the robust policy obtained using the method in Section 7.6 for reference. This top (purple) plot can be obtained

only when the mesh is known. It is an upper bound on what we could possibly achieve with the small mesh.

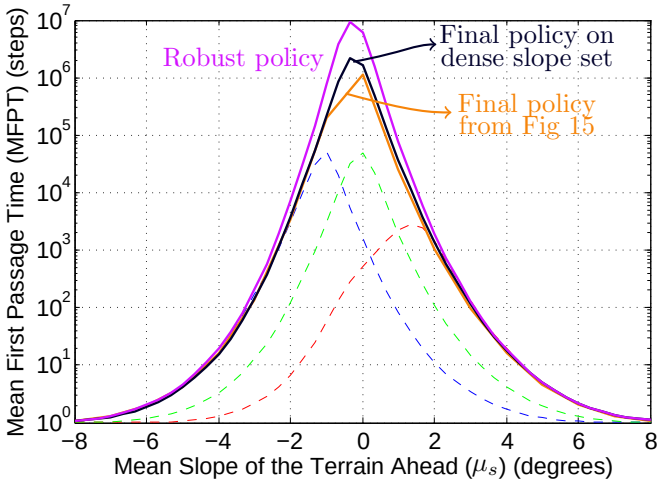

Figure 16: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. The policy was obtained using the mesh via $d_{thr} = 0.5$ and $d_s = 1$. However, this figure shows the results when this policy is evaluated on the mesh obtained with $d_{thr} = 0.2$ and $d_s = 1/3$ to estimate the actual performance. The final policy from Figure 15 is shown for reference. This policy performs even better on an even denser mesh. The robust policy is shown for reference as an upper bound.

7.9 Optimizing for $\mu_s \neq 0$

In the sighted walking case, we found that knowing the long-term terrain mean helps very little, and we correspondingly only presented policies optimized for $\mu_s = 0$. Not needing to know the long-term mean or variance is a desirable result.

7.10 Performance for different σ_s values

Next, we present how this final policy behaves when σ_s is different on the same dense set ($d_{thr} = 0.2$ and $d_s = 1/3$). We present Figure 17 to show the case when $\sigma_s = 1$, and $\sigma_l = 0.5$ deg. Figure 18 illustrates what happens when $\sigma_s = 2$, and $\sigma_l = 1.5$ deg. The final policy is still advantageous compared to fixed ones. For example, when $\mu_s = 0$, $\sigma_s = 1$, and $\sigma_l = 0.5$ deg, the robot takes approximately thousand times more steps compared to most stable fixed controller.

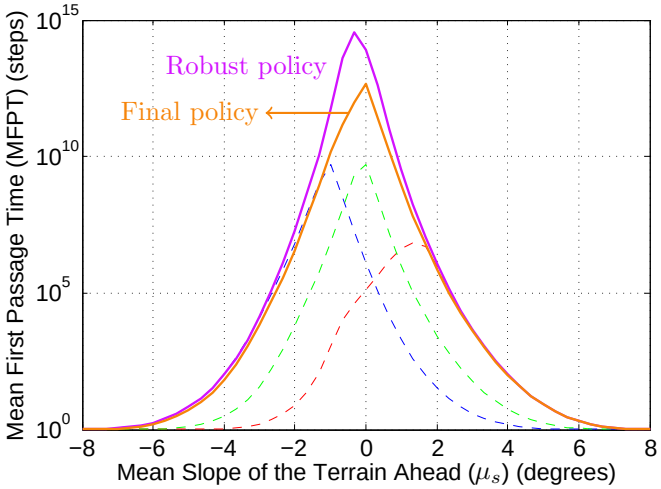

Figure 17: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\underline{\sigma_s = 1}$ deg. Slope information experiences a noise with zero mean and standard deviation $\underline{\sigma_l = 0.5}$ deg. The policy was obtained using the mesh via $d_{thr} = 0.5$ and $d_s = 1$. However, this figure shows the results when this policy is evaluated on the mesh obtained with $d_{thr} = 0.2$ and $d_s = 1/3$ to estimate the actual performance. The robust policy is shown for reference as an upper bound.

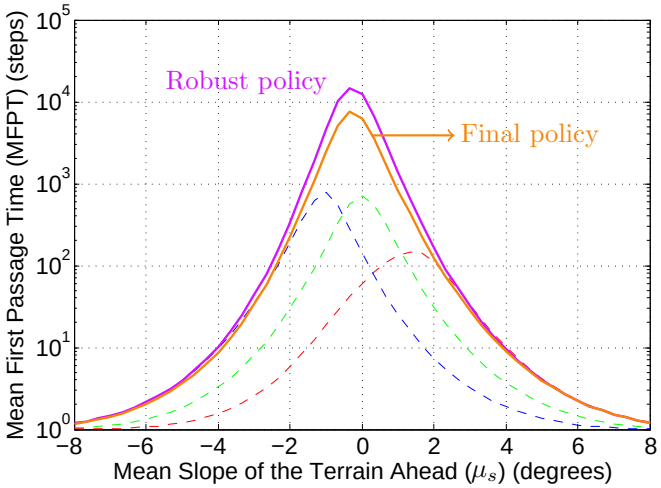

Figure 18: Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\underline{\sigma_s = 2}$ deg. Slope information experiences a noise with zero mean and standard deviation $\underline{\sigma_l = 1.5}$ deg. The policy was obtained using the mesh via $d_{thr} = 0.5$ and $d_s = 1$. However, this figure shows the results when this policy is evaluated on the mesh obtained with $d_{thr} = 0.2$ and $d_s = 1/3$ to estimate the actual performance. The robust policy is shown for reference as an upper bound.

7.11 Higher Number of Controllers

Motivated by how the third controller ($\zeta_{0.5}$) helped for visual and blind walking, we explore increasing the number of controllers. We first fix $d_{thr} = 0.5$ and $d_s = 1$. Then Table 6 illustrates the number of points in the resulting meshes.

Table 6: Number of Controllers versus Mesh Size

Number of Controllers	2	3	5	10
Mesh Size	2,541	6,126	14,583	34,735

In all three figures that follow we will assume $\sigma_l = 1$ deg. Figure 19 illustrates using one-step lookahead only, i.e., visual walking. We see that increasing the number of controllers helps until some point. Notice that in this figure there are μ_s values, where fixed controllers outperform the visual policies, because the latter are optimized for $\mu_s = 0$. For example, ζ_0 is better than the others for μ_s near -1 deg.

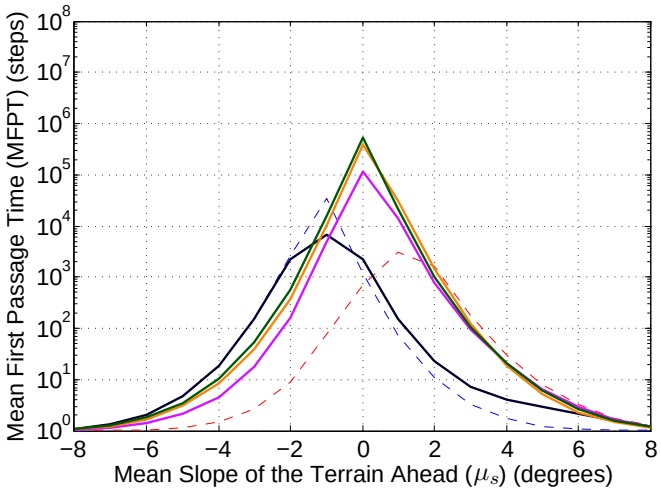

Figure 19: Visual policies for different number of controllers. Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. This figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. The number of controllers increases as we move from bottom to top.

Figure 20 shows blind (to terrain) walking. Again, there is a convergence on the peak for number of controllers. Notice that ζ_1 outperforms blind policies

for $\mu_s > 1$ deg. This is possible because policies are optimized for $\mu_s = 0$.

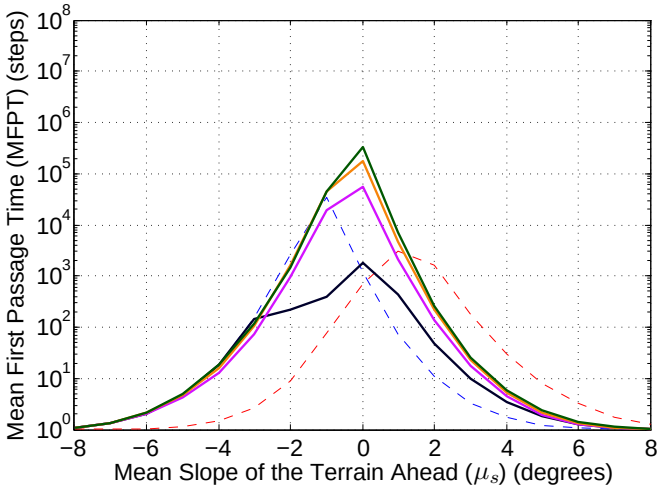

Figure 20: Blind policies for different number of controllers. Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. This figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. The number of controllers increases as we move from bottom to top.

Finally, we show the robust (to slope noise) policies in Figure 21. Note that although increasing number of controllers seems to help (up to some point), the computational cost for both meshing and following operations increases greatly also.

8 Applicability

Our methods have potential to be applicable on real robots because we were able to show two main points. (1) Some high dimensional systems, including bipedal walkers, have much lower intrinsic dimensions. So, meshing with small number of points is possible. (2) We explored the effects of two possible sources of noise, namely on slope and state information. We were able to design policies robust to those.

9 Future Work

Our main goal is testing our methods on an experimental robot. We have not yet studied the dynamics in 3D, so we do not know how much dimension

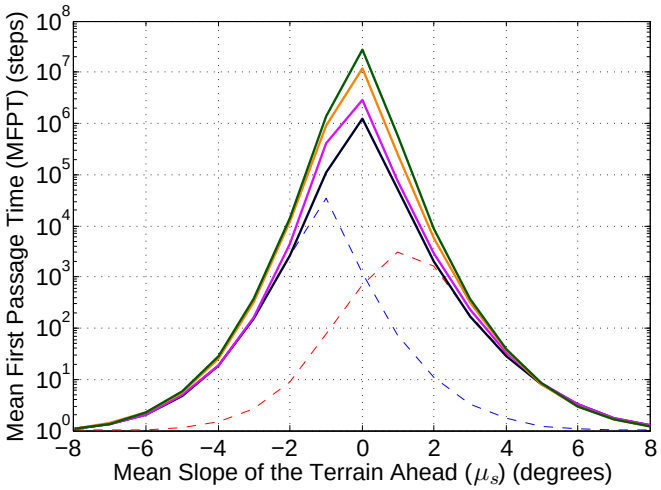

Figure 21: "Robust" policies for different number of controllers. Slopes ahead of the robot are assumed to be normally distributed as noted in (26) with $\sigma_s = 1.5$ deg. Slope information experiences a noise with zero mean and standard deviation $\sigma_l = 1$ deg. This figure shows average number of steps before falling calculated using (30) versus μ_s for the mesh obtained via $d_{thr} = 0.5$. The number of controllers increases as we move from bottom to top.

increase we will experience in such a situation. However, for planar walker, we believe our methods should be capable of providing a meaningful stability metric. The switching faces more potential problems. The main challenge would be robustness towards unhandled sources of noise, e.g., the system modeling errors. If we also would like to use terrain information, the robot should have adequate sensing capability. This can be achieved either with computer vision or a white cane (like the ones visually impaired use).

We also believe that there is still a lot to do in simulation and theory. In Saglam and Byl (2014*b*), we illustrated using more complicated cost functions, which considers energetics in addition to stability. This can be extended for other performance metrics.

We are also interested how much a two-, three- and infinite-step lookahead increases the stability, beyond one-step knowledge. Our guess is that one-step would be mostly enough.

The mesh we present covers a region in 10-dimensional space from which the robot will not escape (except by falling) after entering the region. We hope to cover all initial conditions that lead (presumably quite rapidly) to the same region.

In this paper we use the same control parameters for all controllers. We are interested in finding a mapping from a given reference set to the optimal controller parameter (i.e., gain) set.

The ground profile in this paper was triangular, so each step experienced a constant slope. In future, we plan to use more realistic ground types. The challenging part is meshing. We also want to consider cases when there are bad spots on terrain that we do not want robot to step on.

10 Conclusion

In this paper we studied bipedal locomotion step by step. (1) We presented the simple control structure we adopted. We were able to easily obtain qualitatively different controllers after a quick trial and error. (2) We showed how to mesh the system dynamics. We illustrated that the resulting mesh would be quasi-2D for a 1 DOF underactuated planer biped. (3) We used this mesh to calculate the Mean First Passage Time (MFPT), which corresponds to average number of steps before failure. (4) We studied the great advantages of switching to make use of qualitatively different controllers and maximize number of steps before falling. (5) Finally, we introduced and solved for two primary sources of noise (terrain slope discretization and state discretization) to obtain robust policies.

11 Acknowledgment

This work was supported by the Institute for Collaborative Biotechnologies through grant W911NF-09-0001 from the U.S. Army Research Office. The content of the information does not necessarily reflect the position or the policy of

the Government, and no official endorsement should be inferred.

References

Ames, A. D. (2012), First steps toward automatically generating bipedal robotic walking from human data, *in* K. Kozowski, ed., 'Robot Motion and Control 2011', number 422 *in* 'Lecture Notes in Control and Information Sciences', Springer London, pp. 89–116.

Bellman, R. (1957), 'A markovian decision process', *Indiana University Mathematics Journal* **6**(4), 679–684.

Bhounsule, P. A., Cortell, J. and Ruina, A. (2012), Design and control of ranger: an energy-efficient, dynamic walking robot, *in* 'Proc. of the International Conference on Climbing and Walking Robots'.

Byl, K. and Tedrake, R. (2009), 'Metastable walking machines', *The International Journal of Robotics Research* **28**(8), 1040–1064.

Chen, M.-Y. and Byl, K. (2012), Analysis and control techniques for the compass gait with a torso walking on stochastically rough terrain, *in* 'American Control Conference (ACC), 2012', IEEE, pp. 3451–3458.

Collins, S., Ruina, A., Tedrake, R. and Wisse, M. (2005), 'Efficient bipedal robots based on passive-dynamic walkers', *Science* **307**(5712), 1082–1085.

Erbatur, K., Seven, U., Taskiran, E., Koca, O., Kiziltas, G., Unel, M., Sabanovic, A. and Onat, A. (2008), SURALP-l - the leg module of a new humanoid robot platform, *in* '8th IEEE-RAS International Conference on Humanoid Robots, 2008. Humanoids 2008', pp. 168–173.

Hurmuzlu, Y. and Marghitu, D. (1994), 'Rigid body collisions of planar kinematic chains with multiple contact points', *The International Journal of Robotics Research* **13**(1), 82–92.

Kuo, A. D. (2007), 'Choosing your steps carefully', *Robotics & Automation Magazine, IEEE* **14**(2), 18–29.

McGeer, T. (1990), 'Passive dynamic walking', *The International Journal of Robotics Research* **9**(2), 62–82.

Pearson, K. (1901), 'LIII. on lines and planes of closest fit to systems of points in space', *Philosophical Magazine Series 6* **2**(11), 559–572.

Sabanovic, A. and Ohnishi, K. (2011), Motion control systems, John Wiley & Sons.

Saglam, C. O. and Byl, K. (2013*a*), Stability and gait transition of the five-link biped on stochastically rough terrain using a discrete set of sliding mode controllers, *in* 'IEEE International Conference on Robotics and Automation (ICRA)', pp. 5675–5682.

Saglam, C. O. and Byl, K. (2013*b*), Switching policies for metastable walking, *in* 'Proc. of IEEE Conference on Decision and Control (CDC)', pp. 977–983.

Saglam, C. O. and Byl, K. (2014*a*), Metastable markov chains, *in* 'IEEE Conference on Decision and Control (CDC)'. accepted for publication.

Saglam, C. O. and Byl, K. (2014*b*), Quantifying the trade-offs between stability versus energy use for underactuated biped walking, *in* 'IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)'.

Saglam, C. O. and Byl, K. (2014*c*), Robust policies via meshing for metastable rough terrain walking, *in* 'Proceedings of Robotics: Science and Systems', Berkeley, USA.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N. and Fujimura, K. (2002), The intelligent ASIMO: system overview and integration, *in* 'IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002', Vol. 3, pp. 2478–2483 vol.3.

Shepard, D. (1968), A two-dimensional interpolation function for irregularly-spaced data, *in* 'Proceedings of the 1968 23rd ACM National Conference', ACM '68, ACM, New York, NY, USA, pp. 517–524.

Tucker, V. A. (1975), 'The energetic cost of moving about: Walking and running are extremely inefficient forms of locomotion. much greater efficiency is achieved by birds, fish and bicyclists', *American Scientist* **63**(4), pp. 413–419.

Tzafestas, S., Raibert, M. and Tzafestas, C. (1996), 'Robust sliding-mode control applied to a 5-link biped robot', *Journal of Intelligent and Robotic Systems* **15**(1), 67–133.

Vukobratovic, M. and Borovac, B. (2004), 'Zero-moment point - thirty five years of its life', *International Journal of Humanoid Robotics* **1**(01), 157–173.

Westervelt, E., Chevallereau, C., Morris, B., Grizzle, J. and Ho Choi, J. (2007), *Feedback Control of Dynamic Bipedal Robot Locomotion*, Vol. 26 of *Automation and Control Engineering*, CRC Press.

Westervelt, E., Grizzle, J. W. and Koditschek, D. (2003), 'Hybrid zero dynamics of planar biped walkers', *IEEE Transactions on Automatic Control* **48**(1), 42–56.