Feedback motion planning of legged robots by composing orbital Lyapunov functions using rapidly-exploring random trees

Ali Zamani, Joseph D. Galloway II, Pranav A. Bhounsule

***Abstract*— We present a sampling-based framework for feedback motion planning of legged robots. Our framework is based on switching between limit cycles at a fixed instance of motion, the Poincaré section (e.g., apex or touchdown), by finding overlaps between the regions of attraction (ROA) of two limit cycles. First, we assume a candidate orbital Lyapunov function (OLF) and define a ROA at the Poincaré section. Next, we solve multiple trajectory optimization problems, one for each sampled initial condition on the ROA to minimize an energy metric and subject to the exponential convergence of the OLF between two steps. The result is a table of control actions and the corresponding initial conditions at the Poincaré section. Then we develop a control policy for each control action as a function of the initial condition using deep learning neural networks. The control policy is validated by testing on initial conditions sampled on ROA of randomly chosen limit cycles. Finally, the rapidly-exploring random tree algorithm is adopted to plan transitions between the limit cycles using the ROAs. The approach is demonstrated on a hopper model to achieve velocity and height transitions between steps.**

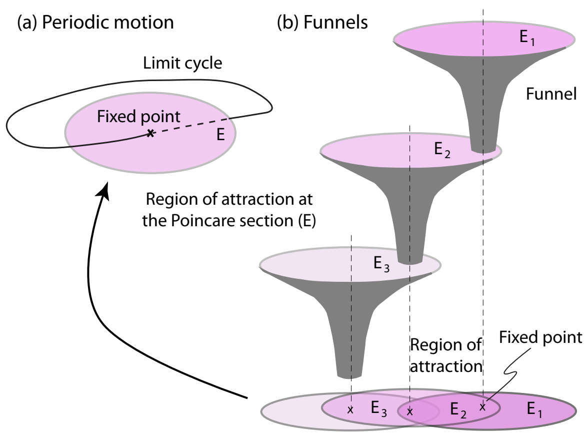

Fig. 1: Relation between funnels and limit cycles: (a) A steady state solution or limit cycle and the corresponding region of attraction at the Poincaré section. (b) Feedback motion planning by composing regions of attraction based on common overlapping areas to funnel the systems from one limit cycle to another. Note that the system switches from one limit cycle to another at the Poincaré section.

I. Introduction

For legged robots to achieve mainstream applications, motion planning or planning movement over multiple steps is essential. However, the complex dynamics (e.g., unstable modes, hybrid dynamics) and the under-actuation (more degrees of freedom than actuators) of legged robots make motion planning conceptually and computationally quite challenging. In this context, one approach is to have a hierarchical control: first, multiple low-level dynamic controllers are developed for within-a-step balance control and then, the low-level controllers are combined to achieve high-level behaviors over multiple steps. Our control framework is based on the same idea: we use an orbital Lyapunov function (a Lyapunov function defined at the Poincaré section and is indicative of the step-to-step or orbital stability) to achieve stability of periodic motions and then combine these periodic motions using rapidly-exploring random trees for planning.

II. Background and Related Work

One of the earliest techniques for motion planning was by Raibert and Wimberly [1] who used a computer simulation to generate a table of control actions as a function of the robot states. But such a table was prohibitively large for storage and search purposes, so they approximated the table with high order polynomials and demonstrated their approach on a hopping robot. Similarly, Da et al. [2] used a computer simulation to create controllers as a function of robot states and terrain height. The data was then input into a supervised learning framework to learn a control policy, which was then implemented on a bipedal robot. One issue with these past works are that they do not explicitly minimize an objective function.

Dynamic programming (DP) is a method for optimal motion planning. The technique relies on the discretization of the state space and defining a suitable cost function. The prime issue with this method is that the quality of the results depends on the fineness of discretization, but for a fine discretization and high degrees of freedom system, the control synthesis becomes quickly intractable due to high computation and storage cost. The common techniques for tackling this issue is to make simplifications to the system. For example, Whitman [3] simplified a humanoid robot into three simple independent models; a sagittal, a lateral, and a frontal model. Controllers were developed for the individual models and combined during run-time using time as the phase variable. Mandersloot et al. [4] discretized the system from one-step to another using the Poincaré section and used this step-to-step discretization within the dynamic programming framework. Because of the extensive computations, DP is mostly an offline method.

Model predictive control (MPC) is an online optimization

Robotics and Motion Lab., Dept. of Mechanical Engineering, The University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA. Corresponding author: `pranav.bhounsule@utsa.edu`.
This work was supported by NSF grants 1566463 and 1816925 to PAB.

technique that relies on repeatedly computing the optimal control policy for a given planning time, then implementing only a part of the planned control policy, followed by replanning as new information becomes available. Park et al. [5] considered the problem of a quadruped jumping over randomly placed obstacles. They first created stable low-level bounding controllers for the quadruped. Then using the MPC framework, the robot planned apex velocity and step length based on terrain information, which was then input to the low-level controller. Rutschmann et al. [6] considered MPC but in the context of tracking a foothold position while maintaining balance on rough terrain for a hopper model.

Our method capitalizes on combining multiple steady state gaits to create motion plans as illustrated next. Fig. 1 (a) shows a limit cycle. Associated with the limit cycle is a fixed point, an initial condition picked at the Poincaré section (an instance of motion such as apex or touchdown) that maps onto itself at the Poincaré section of the next step. Let the pink ellipse denote the region of attraction (ROA) of the fixed point. The ROA represents all the initial conditions at the Poincaré section that will converge to the fixed point after one or more steps. Fig. 1 (b) shows how multiple limit cycles may be combined to create motion plans. For example, the fixed point of the ROA E_1 is inside the ROA E_2. Similarly, the fixed point of ROA E_2 is inside the ROA of E_3. When the system is near the fixed point of E_1, switching the controller to that of E_2 will cause the system to move towards the fixed point of E_2. Thus, switching the controllers based on ROA's is a straightforward method to create motion plans. This technique was used by Veer et al. [7], [8], [9] to achieve variable speed walking on a bipedal robot and by Cao et al. [10] to achieve gait transition for a quadruped robot. In both works, extensive numerical simulations were used to estimate the ROA at the Poincaré section. Sampling-based methods may be used with ROAs to achieve feedback motion planning. For example, Tedrake [11] used a linear quadratic regulator (LQR) to stabilize limit cycles and to estimate the ROA of the LQR controller. For a random point chosen on the state space, LQR-trees inspired by rapidly-exploring random trees (RRT) were grown towards the limit cycle. The ROAs were estimated using the sum of squares optimization [12].

In this paper, we switch controllers at the Poincaré section as done before. However, unlike previous works, we use an orbital Lyapunov function [13], [14] to assume a candidate ROA and use trajectory optimization to guarantee exponential convergence of all initial conditions with the ROA. Our previous paper explored composing limit cycles using heuristics to achieve motion planning [15]. This paper presents two improvements over our past approach: (1) use of deep neural networks to learn control policies for fixed points and their ROAs, and then generalize across multiple fixed points without additional computations, and (2) adopt the RRT algorithm to reason with ROAs to achieve feedback motion planning.

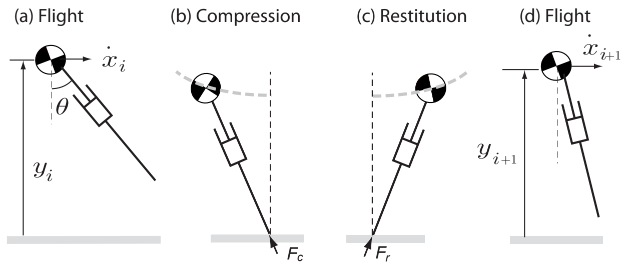

Fig. 2: Different phases of motion for the model. The model has a prismatic actuator for generating inline force along the stance $F = P + k(\ell_0 - \ell)$ along the stance leg and a hip actuator for foot placement at an angle θ with respect to the vertical. See [16] for more details including hopper parameter values.

III. Model

Figure 2 shows the model of the hopper. It consists of a point mass body of mass m and a massless leg with a maximum leg length ℓ_0. Gravity is denoted as g and points downwards. There is a prismatic actuator that can generate an axial force (F) and a hip actuator for foot placement 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. The model starts at the apex where the state vector with respect to the world frame is $\{0, \dot{x}_k, y_k, 0\}$. The model then moves under gravity given by equations, $\ddot{x} = 0$ and $\ddot{y} = -g$, until ground contact, which 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 equations, $m\ddot{x} = F\frac{x}{\ell}$ and $m\ddot{y} = F\frac{y}{\ell} - mg$, where x and y are taken relative to the contact point, $F = P + k(\ell_0 - \ell)$ is the force on the stance leg, $P > 0$ is a constant force, k is a constant, and $\ell = \sqrt{x^2 + y^2}$ is the instantaneous leg length. For the first half of the stance phase from touchdown to mid-stance (defined by $\dot{y} = 0$ [1]) called the compression phase, we assume that the constant force is $P = P_c$. For the second half of the stance phase from mid-stance to take-off called the restitution phase, we assume that the constant force is $P = P_r$. We assume there is no foot slipping during ground interaction. The model takes off from the ground when the leg is fully extended, that is, $\ell_0 - \ell = 0$. Thereafter, the mass has a flight phase to end at the next apex with respect to world frame, $\{x_{k+1}, \dot{x}_{k+1}, y, 0\}$.

IV. Preliminaries

A. *Poincaré map and limit cycle*

We define the Poincaré map, $\mathbb{F}$, at the apex. The apex is defined by the condition, $\dot{y} = 0$. Given the state at the apex at step k, $\mathbf{x}_k = \{\dot{x}, y\}$, and the control actions, $\mathbf{u}_k = \{\theta, P_c, P_r\}$ we compute the state at the next step,

$$\mathbf{x}_{k+1} = \mathbb{F}(\mathbf{x}_k, \mathbf{u}_k). \tag{1}$$

[1]Note that the event $\dot{y} = 0$ is different from the event corresponding to full leg compression, which is given by $\dot{\ell} = 0$.

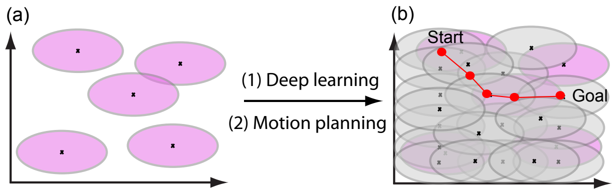

Fig. 3: Our control framework: (a) generate controllers to create regions of attraction for multiple fixed points using trajectory optimization; (b) use deep learning to develop control policies for other fixed points (grey ellipses) to fill the state space and then use overlaps between ROAs for motion planning from start point to goal point.

There is no closed form for the map $\mathbb{F}$. We use numerical integration to obtain the map. The ith limit cycle is found by fixing $\mathbf{x}_{k+1} = \mathbf{x}_k = \mathbf{x}_i$ and searching for $\mathbf{u}_k = \mathbf{u}_i = \{\theta, P_c = 0, P_r = 0\}$ such that

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

B. Orbital Lyapunov Function (OLF)

We define a Lyapunov function for the ith limit cycle at the Poincaré section ($\mathbb{F}$) as follows

$$V(\Delta\mathbf{x}_k^i) = (\Delta\mathbf{x}_k^i)^T\mathbf{S}\Delta\mathbf{x}_k^i = (\mathbf{x}_k - \mathbf{x}_i)^T\mathbf{S}(\mathbf{x}_k - \mathbf{x}_i) \tag{3}$$

where the positive definite matrix $\mathbf{S} = diag\{\frac{1}{s_1^2}, \frac{1}{s_2^2}\}$. The resulting Lyapunov function is a 2-dimensional ellipse that has its major and minor axes along the horizontal apex velocity and apex height axes respectively. We use the diagonal Lyapunov function in this paper because we want to maximize the velocity and height switch, but other non-diagonal or non-quadratic functions could be used, provided that the Lyapunov function is positive definite. We impose an exponentially decaying condition on the Lyapunov function

$$V(\Delta\mathbf{x}_{k+1}^i) - V(\Delta\mathbf{x}_k^i) \leqslant -\alpha V(\Delta\mathbf{x}_k^i), \tag{4}$$

where $0 < \alpha \leqslant 1$ is the rate of decay of the Lyapunov function between steps.

C. Region Of Attraction (ROA)

The Region Of Attraction (ROA), $\mathcal{R}$, of the controller is the set of all initial conditions $\mathbf{x}_k$ that would converge to the corresponding limit cycle $\mathbf{x}_i$ over one or more steps. The region of attraction is defined by the level set c from the equation $(\mathbf{x}_k - \mathbf{x}_i)^T\mathbf{S}(\mathbf{x}_k - \mathbf{x}_i) - c =$. In general, the value of c is constrained by actuator limits or kinematic limits. We choose the constant $c = 1$ (a design choice) to ensure that the biggest switch in velocity is $2c = 2$ m/s.

V. Methods

A. Trajectory optimization for a given initial condition

For a given initial condition $\mathbf{x}_k \neq \mathbf{x}_i$ at the Poincaré section, we solve the following trajectory optimization problem

$$\underset{\mathbf{u}_k}{\text{minimize}} \quad \text{MCOT} \tag{5}$$

$$\text{subject to:} \quad \mathbf{x}_{k+1} = \mathbb{F}(\mathbf{x}_k, \mathbf{u}_k) \tag{6}$$

$$V(\Delta\mathbf{x}_{k+1}^i) - (1-\alpha)V(\Delta\mathbf{x}_k^i) \leqslant 0 \tag{7}$$

where MCOT is the Mechanical Cost Of Transport and is defined as the energy used in a step (E_{step}) per unit weight (mg) per unit distance traveled in one step (D)

$$\begin{aligned}\text{MCOT} &= \frac{E_{step}}{mgD}\\ E_{step} &= E_k + E_{P_c} + E_{P_r}\\ &= \int_{\text{step}} \Big(|k(\ell_0 - \ell)\dot{\ell}| + |P_c\dot{\ell}| + |P_r\dot{\ell}|\Big)dt. \end{aligned} \tag{8}$$

where E_k, E_{P_c}, and E_{P_r} are mechanical work done by the axial actuator to simulate a springy leg, constant compression force, and restitution force respectively, $|x|$ is the absolute value of x, and $\dot{\ell} = \frac{x\dot{x}+y\dot{y}}{\ell}$. In order to use gradient-based optimization methods, we smooth the absolute value (it has a kink at x=0) using square root smoothing [17]. That is, $|x| = \sqrt{x^2+\epsilon^2}$ where ϵ is a small number (we use $\epsilon = 0.01$). We solve the optimization problem defined by Eqns. 5-7 using single shooting method [18], [19]. One caveat in the optimization problem formulation is that there is no formal guarantee that the optimization problem will converge to a feasible solution, but our experience has been that when the control actions (e.g., foot placement, brake force, thrust force) have a fairly independent effect on the system state, which is a measure of the controllability of the system in sense, the optimization problem generally converges to a feasible solution without any issues.

B. Overview of the feedback motion planning approach

1) Trajectory optimizations for sampled initial conditions within the ROA for multiple limit cycles: First, we solve the trajectory optimization problem defined in Sec. V-A for a given limit cycle $\mathbf{x}_i$ and for multiple initial conditions chosen within the ROA $(\mathbf{x}_k-\mathbf{x}_i)^T\mathbf{S}(\mathbf{x}_k-\mathbf{x}_i)-1 < 0$. We also store the fixed point of the limit cycle $\mathbf{x}_i$, the initial conditions $\mathbf{x}_k$, and the corresponding control actions $\mathbf{u}_k$. This process is repeated to generate control actions for ROAs for multiple limit cycles as shown in Fig. 3 (a).

2) Filling state space with regions of attraction and associated stabilizing controllers followed by feedback motion planning using rapidly-exploring random trees: We use the stored data to find a control policy for each control action as a function of the fixed point and initial conditions, $u_j = f_j(\mathbf{x}_i, \mathbf{x}_k)$ where $j = 1, 2, 3$ correspond to the three control actions for the hopper. We use deep neural networks to estimate the functions f_j. Our previous work informs the choice of neural networks for the f_j (see [14]). We can use the functions f_j to predict the control actions for a randomly chosen fixed point at the Poincaré section. The resulting function f_j is validated for ROAs of multiple fixed points as shown by grey ellipses in Fig. 3 (b). Next, we plan transitions using rapidly-exploring random trees (RRT). The key idea is to switch at the Poincaré section (i.e., at the apex for the hopper model) by reasoning with the overlap between the ROA of two limit cycles.

We present the modified RRT algorithm [20] in Algo. 1 and a pictorial depiction is in Fig. 4. We restrict the algorithm to the hopper model and to the region of attraction described

Algorithm 1 GENERATE_RRT($\mathbf{x}_{init}, \mathbf{x}_{goal}, \delta\dot{x}, \delta y, N_P$)

```
1:  T.init($\mathbf{x}_{init}$);
2:  for $p = 1$ to $N_P$ do
3:      $\mathbf{x}_{rand} \leftarrow$ RANDOM_STATE();
4:      $\mathbf{x}_{near} \leftarrow$ NEAREST_NEIGHBOR($\mathbf{x}_{rand}, T$);
          {NOTE: $\mathbf{x}_{near} = \mathbf{x}_k$}
5:      $\mathbf{x}_{fixed} \leftarrow$ NEW_FIXED_POINT($\mathbf{x}_{near}, \mathbf{x}_{rand}, \delta\dot{x}, \delta y$);
          {NOTE: $\mathbf{x}_{fixed} = \mathbf{x}_i$}
6:      $\mathbf{u}_{near} \leftarrow$ SELECT_INPUT($\mathbf{x}_{fixed}, \mathbf{x}_{near}$)
          {NOTE: $\mathbf{u}_{near} = \mathbf{u}_k$, SELECT_INPUT is$f_j$,
            $u_j = f_j(\mathbf{x}_{fixed}, \mathbf{x}_{near})$, $j = 1, 2, 3.$}
7:      $\mathbf{x}_{new} \leftarrow$ SIMULATION(model,$\mathbf{x}_{near}$,$\mathbf{u}_{near}$)
          {NOTE: $\mathbf{x}_{new} = \mathbf{x}_{k+1}$}
8:      T.add_vertex($\mathbf{x}_{new}$);
9:      T.add_edge($\mathbf{x}_{near}$,$\mathbf{x}_{new}$);
10:     if $|\mathbf{x}_{new} - \mathbf{x}_{goal}| < \min(\delta\dot{x}, \delta y)$ then
11:         break;
12:     end if
13: end for
```

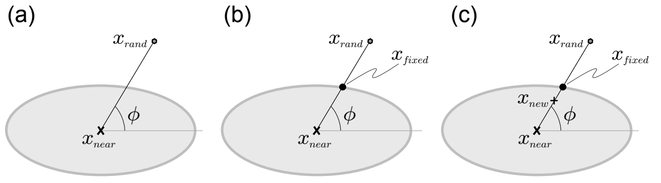

Fig. 4: Pictorial depiction of the RRT algorithm

by the major and minor axes on the x and y directions, but is easy to adapt to other cases. The inputs to the models are: the initial state $\mathbf{x}_{init}$, the goal state $\mathbf{x}_{goal}$, the major and minor axes of the ellipse as given by $\delta\dot{x} = s_1\sqrt{c}$ and $\delta y = s_2\sqrt{c}$, and maximum number of nodes allowed N_p. The tree is denoted by T. At item number 3, we generate a random state $\mathbf{x}_{rand}$ using the function RANDOM_STATE(). At item number 4, we use the function NEAREST_NEIGHBOR() to search for the nearest state $\mathbf{x}_{near}$ on the existing tree using the Euclidean distance. We compute the direction from the nearest point on the tree to the random point as $\phi = \tan^{-1}((y_{rand} - y_{near})/(\dot{x}_{rand} - \dot{x}_{near}))$ (see Fig. 4 (a)). Next at item number 5, the function NEW_FIXED_POINT() is used to choose a fixed point $\mathbf{x}_{fixed} = \mathbf{x}_{near} + \Delta\mathbf{x}_{near}$ where $\Delta\mathbf{x}_{near} = \{\delta\dot{x}\cos(\phi), \delta y\sin(\phi)\}$ (see Fig. 4 (b)). Next at item number 6, we use the SELECT_INPUT() function, which is the same as the neural networks identified in Sec. V-B.2 to choose control actions $\mathbf{u}_{near}$. Next at item number 7, we use the function SIMULATION(), which is a forward simulation of the model, to grow the tree to the point $\mathbf{x}_{new}$ (see Fig. 4 (c)). At item numbers 8 and 9, we add $\mathbf{x}_{new}$ as a vertex and connect it to $\mathbf{x}_{near}$ to form an edge on the tree T. We terminate the algorithm if the $\mathbf{x}_{goal}$ is within the region of attraction of $\mathbf{x}_{new}$ as shown at item number 10. In summary, the key modifications to the original RRT algorithm are: (1) at item number 5 where the randomly chosen direction is used to choose a fixed point, and (2) at item number 7 where forward simulation is used to update $\mathbf{x}_{new}$. Unlike the static RRT, popular for path planning without dynamics, we do feedback motion planning because $\mathbf{x}_{new}$ is obtained from $\mathbf{x}_{near}$ (feedback) and using the system dynamics. Once the tree T is built, we search it backwards from the goal to the start to find a feasible path.

VI. Results

We present results for the hopper model described in Sec. III. As mentioned earlier, the Poincaré section is at the apex. There are two state variables at the apex $\mathbf{x}_k = \{\dot{x}, y\}$ and three control actions $\mathbf{u}_k = \{\theta, P_c, P_r\}$. The parameters for the orbital Lyapunov function are $s_1 = 1$ and $s_2 = 0.3$, and the rate of convergence $\alpha = 0.9$.

A. Trajectory optimizations for ROAs of multiple limit cycles

We choose the following 7 limit cycles characterized by fixed points $\mathbf{x}_i$s; $\{2, 1.3\}$, $\{2, 1.6\}$, $\{3, 1.5\}$, $\{4, 1.4\}$, $\{4, 1.6\}$, $\{5, 1.4\}$, and $\{5, 1.6\}$. We chose 105 initial conditions in the ROA for each limit cycle, totaling $105 \times 7 = 735$ points. For each point, we solve the optimization problem specified in Sec. V-A. We use the non-linear optimization software SNOPT [21] and single shooting method using *ode113* in MATLAB. On average each optimization took about 15 seconds on a laptop (circa 2012). Clearly, the optimization speed is too slow for real-time implementation. This necessitates offline computation of the control policy as given next.

B. Training the control policy

We use the data from the 7 limit cycles to train three neural networks, each corresponding to a control action (θ, P_c, and P_r). Each neural network has 12 hidden layers and is trained using Levenberg-Marquardt algorithm with mean squared error as the performance criterion. The inputs to the neural networks are the limit cycles $\{\dot{x}_i, y_i\}$ $(i = 1, 2, 3, ..., 7)$, the k perturbed initial conditions at the Poincaré section $\{\dot{x}_k, y_k\}$, and the corresponding control actions $\mathbf{u}_k$, where $k = 1, 2, 3, ..., 735$. For more details on the specifics of the control policy, see [14]. Note that we could have also used a single neural network to map the three control actions (outputs) to the two states (inputs).

C. Testing the control policy

We describe how we test the control policy using the forward simulation. We choose a random limit cycle $\mathbf{x}_i$ and 451 random initial conditions $\mathbf{x}_k$ within the ROA for that limit cycle. Then, using the control law found earlier, we simulate the system for a single step starting from each initial condition. Figure 5 shows the results for 3 limit cycles as a histogram of the percentage of points (total of 451) that lie within the various level sets of the orbital Lyapunov function after one step. Note that only (a) is within our training range but (b) and (c) are outside the training range, yet the test shows that all initial conditions are reduced to within $V(\mathbf{x}_k) < 0.3$. We also check that none of the initial conditions lead to an increase in V at the subsequent steps. This verifies (using limited samples) that the chosen control policy is able to perform satisfactorily.

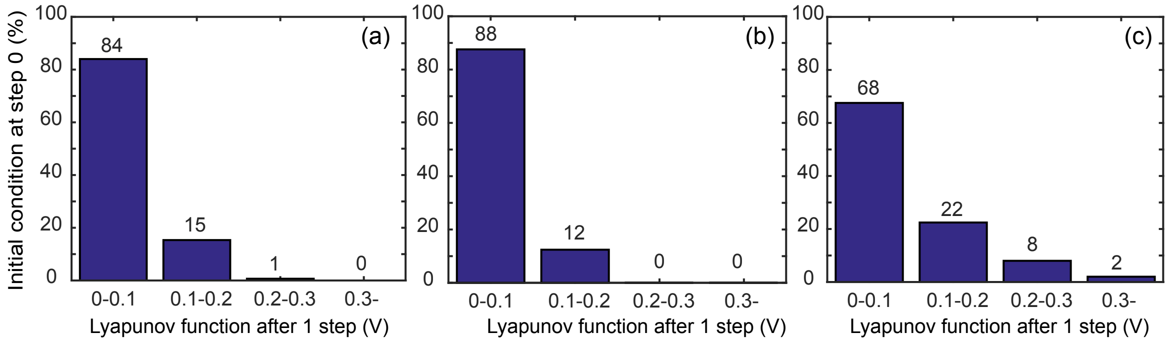

Fig. 5: Testing the neural networks on 451 initial conditions inside the region of attraction for randomly chosen limit cycles. The histogram shows the percentage of the initial conditions that are within the ellipse after one step $V(x_k) = c$, (a) $\{3.5, 1.3\}$ (b) $\{2, 1.7\}$, (c) $\{7, 1.3\}$.

D. Feedback motion planning

We present the results for the feedback motion planning using the RRT discussed in Algo. 1. We considered two scenarios: changing only the velocity as shown in Fig. 6 and changing the velocity and the height as shown in Fig. 7. Animation video is in reference [22]. We discuss these next.

First, we consider the problems of keeping the start and goal apex heights constant, but changing the start and goal apex horizontal velocities as shown in Fig. 6. The cross denotes the start state and the solid circle indicates the goal state. The algorithm terminates when $V(\mathbf{x}_k) < 0.1$ for the fixed point at the goal state. Both these scenarios take 5 steps for the transition. The corresponding control actions $\mathbf{u}_k$ are shown in Fig. 6 (c) and (d) with forces P_r and P_c multiplied by 0.001. To increase the velocity, the restitution force P_r is non-zero while the compression force P_c is zero. This is expected because to increase the velocity, energy needs to be supplied to the system, which comes from providing a non-zero restitution force. The roles of the forces are swapped for decreasing the velocity as expected by following the reverse logic. The foot placement angle shows a steady increase for increasing apex forward velocity (c). This can have the effect of increasing the apex height, while reducing the apex forward velocity without changing the total energy of the system [23]. However, from (a) it can be seen that the apex height is decreasing. This suggests that the foot placement angle control is compensating for the increased restitution force, which may increase both the apex horizontal velocity as well as the apex height. A decreased foot placement angle in (d) can now be reasoned by reversing the logic.

Second, we consider the problems of changing both the start and goal apex heights and horizontal velocities. In Fig. 7 (a), the objective is to increase the apex horizontal velocity and apex height while in Fig. 7 (b), the objective is to decrease the apex horizontal velocity and increase apex height. The first objective is achieved in 5 steps, while the second in 6 steps as shown. The trends in forces are similar to those in Fig. 6 and follow similar logic as before. The increase in foot placement angle in (c) provides a means to increase the apex height, but the decrease in foot placement angle in (d) is a mechanism to compensate for the increased compressive force P_c as noted earlier too.

VII. Discussion

We have presented a framework for feedback motion planning of legged robots by composing periodic gaits using regions of attraction and rapidly-exploring random trees. The efficacy of the approach was demonstrated using a hopper model in tasks involving increasing/decreasing the apex horizontal velocity and apex height.

The key benefit of our sampling-based approach for motion control is that we start by assuming a Lyapunov function and region of attraction and then find a control policy that validates our assumption. This is in sharp contrast to existing approaches that start with a control policy, usually a linear one (e.g., linear quadratic regulator [11]), and then estimate the largest region of attraction. Depending on the assumed control policy and the nonlinearity in the dynamics, the existing methods may lead to a small region of attraction, which may significantly affect the motion planning. Furthermore, we are also able to ensure quick transitions by enforcing an exponential convergence condition on the orbital Lyapunov function.

Existing approaches of using a moving Poincaré section and estimating the region of attraction along the trajectory [24] solve a trajectory tracking problem. In contrast, we are able to convert the traditional trajectory tracking problem into a regulation problem by doing step-to-step control using the Poincaré section. The latter is computationally cheaper and simpler than the former.

The average time taken by the trajectory optimization is 15 seconds per initial condition. This is too slow for online optimization. However, by using offline optimization for the control actions over multiple regions of attraction followed by regression to find a control policy, we are able to create a compact policy that is easy to store and use on hardware.

Our method has several limitations. First, our sampling-based method is computationally expensive and relies on extensive offline optimizations followed by regression to find the control policy. This is especially an issue for complex legged models (e.g., humanoids). Second, our method relies on a number of heuristics such as the choice of Lyapunov

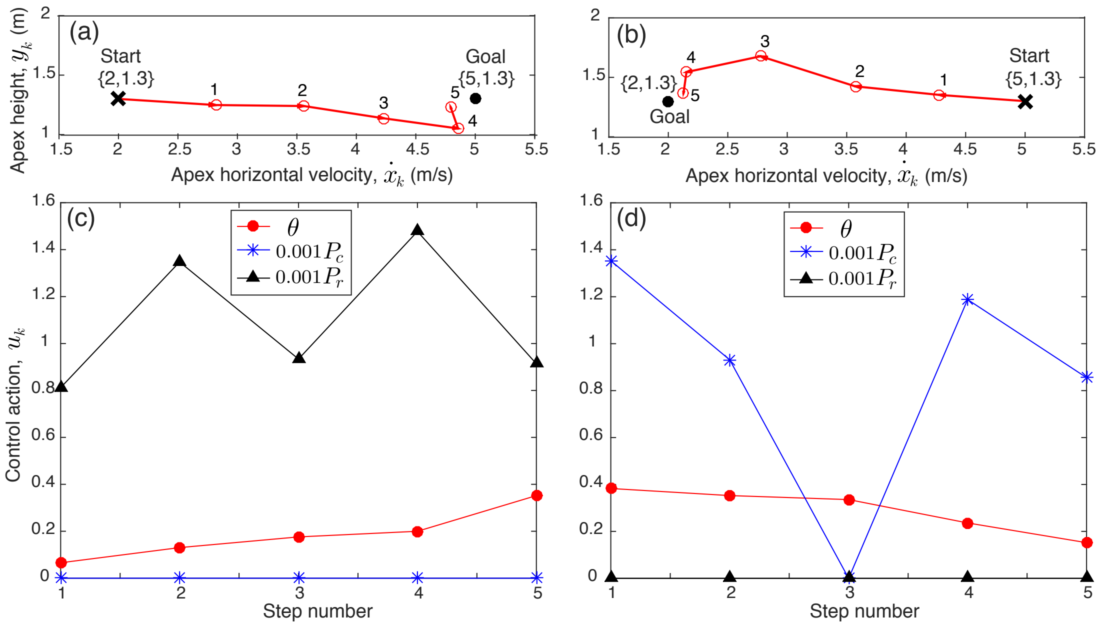

Fig. 6: Results for feedback motion planning: (a,c) increasing apex horizontal velocity and (b,d) decreasing apex horizontal velocity, both for constant start and goal apex heights. Note that the angle is in radians and force is in N.

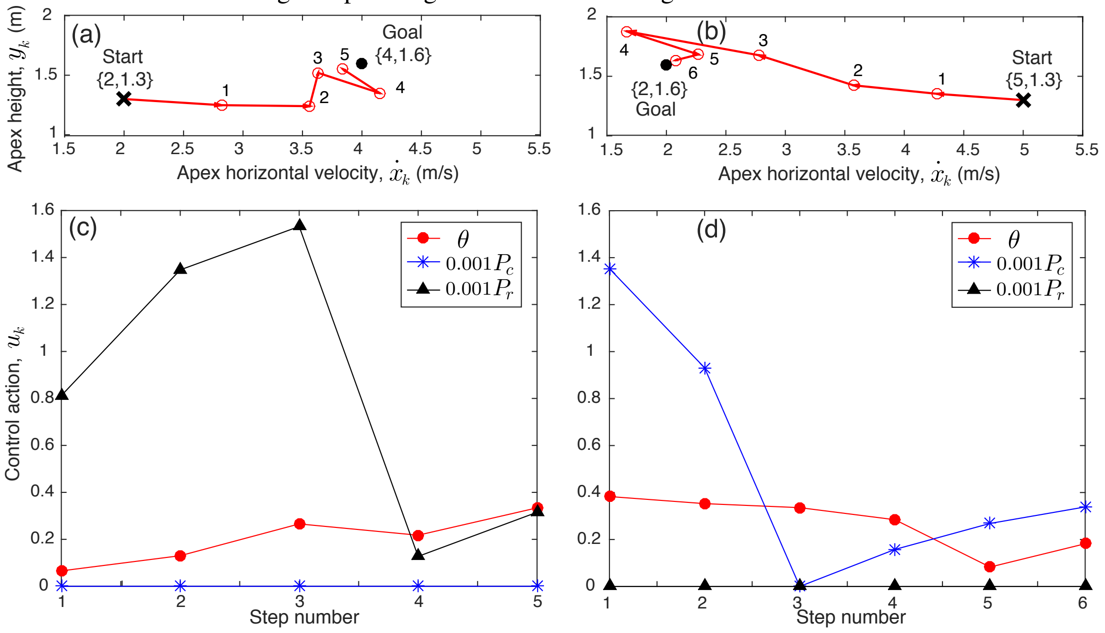

Fig. 7: Results for feedback motion planning: (a,c) increasing apex horizontal velocity and apex height, and (b,d) decreasing apex horizontal velocity and increasing apex heights. Note that the angle is in radians and force is in N.

function, Poincaré section, control actions, and fitting function for the control law. Third, the presence of actuator limits and kinematic limits may lead to difficulties in guaranteeing the validity of the assumed region of attraction. Finally, since we use a single Poincaré section at mid flight, perturbations after mid flight may lead to failure before they can be corrected for in the mid flight of the subsequent step.

VIII. Conclusion and Future Work

We conclude that feedback motion planning of legged robots that compose regions of attraction at distinct phases in motion (e.g., apex) while using tools from sampling-based motion planning algorithms is a promising approach. However, the approach depends on a number of heuristics that need to be further explored.

We suggest the following extensions: (1) choose more complex Lyapunov functions depending on the task objectives (e.g., non-diagonal quadratic functions and polytopes); (2) use optimal variants of sampling-based planners (e.g., RRT-star); and (3) explore the space of different control actions, Poincaré section, and robot models (e.g., biped, quadruped),

References

[1] M. H. Raibert and F. C. Wimberly, "Tabular control of balance in a dynamic legged system," *IEEE Transactions on systems, man, and Cybernetics*, no. 2, pp. 334–339, 1984.

[2] X. Da, R. Hartley, and J. W. Grizzle, "Supervised learning for stabilizing underactuated bipedal robot locomotion, with outdoor experiments on the wave field," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3476–3483.

[3] E. C. Whitman, "Coordination of multiple dynamic programming policies for control of bipedalwalking," Ph.D. dissertation, Carnegie Mellon University, 2013.

[4] T. Mandersloot, M. Wisse, and C. G. Atkeson, "Controlling velocity in bipedal walking: A dynamic programming approach," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, 2006, pp. 124–130.

[5] H.-W. Park, P. M. Wensing, S. Kim *et al.*, "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds," 2015.

[6] M. Rutschmann, B. Satzinger, M. Byl, and K. Byl, "Nonlinear model predictive control for rough-terrain robot hopping," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1859–1864.

[7] S. Veer and I. Poulakakis, "Ultimate boundedness for switched systems with multiple equilibria under disturbances," *arXiv preprint arXiv:1809.02750*, 2018.

[8] S. Veer, M. Motahar, and I. Poulakakis, "Generation of and Switching among Limit-Cycle Bipedal Walking Gaits," in *Proceedings of the 56th IEEE Conference on Decision and Control, Melbourne, Australia*, 2017.

[9] S. Veer and I. Poulakakis, "Safe adaptive switching among dynamical movement primitives: Application to 3d limit-cycle walkers," *arXiv preprint arXiv:1810.00527*, 2018.

[10] Q. Cao, A. T. Van Rijn, and I. Poulakakis, "On the control of gait transitions in quadrupedal running," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5136–5141.

[11] R. Tedrake, "LQR-Trees: Feedback motion planning on sparse randomized trees," in *Proceedings of Robotics Science and Systems (RSS), Zaragoza, Spain*, 2009.

[12] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, "Introducing sostools: A general purpose sum of squares programming solver," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 1. IEEE, 2002, pp. 741–746.

[13] P. A. Bhounsule and A. Zamani, "A discrete control lyapunov function for exponential orbital stabilization of the simplest walker," *Journal of Mechanisms and Robotics*, vol. 9, no. 5, p. 051011, 2017.

[14] P. A. Bhounsule, A. Zamani, J. Krause, S. Farra, and J. Pusey, "Control policies for a large region of attraction for dynamically balancing legged robots: a sampling-based approach," *Robotica (submitted)*, 2019.

[15] P. A. Bhounsule, A. Zamani, and J. Pusey, "Switching between limit cycles in a model of running using exponentially stabilizing discrete control lyapunov function," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 3714–3719.

[16] A. Zamani and P. Bhounsule, "Control synergies for rapid stabilization and enlarged region of attraction for a model of hopping," *Biomimetics*, vol. 3, no. 3, p. 25, 2018.

[17] M. Srinivasan, "Why walk and run: energetic costs and energetic optimality in simple mechanics-based models of a bipedal animal," Ph.D. dissertation, Cornell University, 2006.

[18] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. Siam, 2010, vol. 19.

[19] F. Alambeigi, S. Sefati, and M. Armand, "A convex optimization framework for constrained concurrent motion control of a hybrid redundant surgical system," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 1158–1165.

[20] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[21] P. Gill, W. Murray, and M. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12, no. 4, pp. 979–1006, 2002.

[22] P. A. Bhounsule, "Feedback motion planning using rapidly-exploring random trees," https://youtu.be/Ie-WGqAl6-4, September 2018.

[23] J. K. Hodgins and M. Raibert, "Adjusting step length for rough terrain locomotion," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 289–298, 1991.

[24] I. R. Manchester, "Transverse dynamics and regions of stability for nonlinear hybrid limit cycles," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 6285–6290, 2011.