Task-level control and Poincaré map-based sim-to-real transfer for effective command following of quadrupedal trot gait

Pranav A. Bhounsule[1], Daniel Torres[1], Ernesto Hernandez Hinojosa[1], Adel Alaeddini[2]

***Abstract*— The ability of quadrupedal robots to follow commanded velocities is important for navigating in constrained environments such as homes and warehouses. This paper presents a simple, scalable approach to realize high fidelity speed regulation and demonstrates its efficacy on a quadrupedal robot. Using analytical inverse kinematics and gravity compensation, a task-level controller calculates joint torques based on the prescribed motion of the torso. Due to filtering and feedback gains in this controller, there is an error in tracking the velocity. To ensure scalability, these errors are corrected at the time scale of a step using a Poincaré map (a mapping of states and control between consecutive steps). A data-driven approach is used to identify a decoupled Poincaré map, and to correct for the tracking error in simulation. However, due to model imperfections, the simulation-derived Poincaré map-based controller leads to tracking errors on hardware. Three modeling approaches – a polynomial, a Gaussian process, and a neural network – are used to identify a correction to the simulation-based Poincaré map and to reduce the tracking error on hardware. The advantages of our approach are the computational simplicity of the task-level controller (uses analytical computations and avoids numerical searches) and scalability of the sim-to-real transfer (use of low-dimensional Poincaré map for sim-to-real transfer). A video is here `http://tiny.cc/humanoids23`.**

I. Introduction

In recent years, quadrupedal robots such as Boston Dynamics' Spot and Unitree's A1/Go1 have been readily available for industrial applications and research labs [1], [2]. The access of reliable hardware besides basic features of quadrupedal robots such as low center of gravity, large support base, and light legs makes them ideal for stable and effective navigation in uneven (outdoors) and structured (indoor) environments.

One of the important basic ability of quadrupeds is to follow commands such as prescribed linear and rotational speed with high fidelity. This enables their use in more complex scenarios, such as navigation in constrained environments, in homes and industrial settings, in the presence of obstacles and clutter. The main challenges are to do faithful command following with minimal computations so that the extra computing resources can be used for more complex tasks (e.g., task and motion planning, computer vision). Another challenge is the disparity between simulations and hardware, known as the sim-to-real gap, which makes it challenging to transfer simulated controllers to hardware. This paper presents a scalable approach to do high fidelity command following and faithful sim-to-real transfer and demonstrates the result on a Unitree A1 robot.

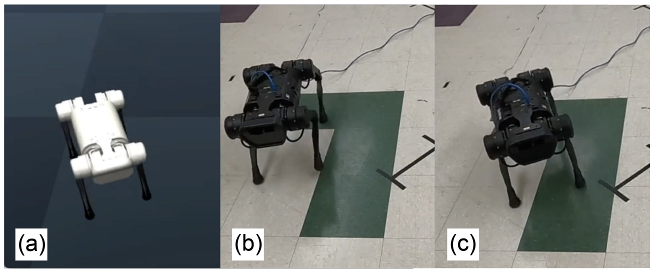

Fig. 1. Overview of the final result: (a) Final pose in simulation (intended motion); (b) Final pose obtained in hardware without sim-to-real transfer; (c) Final pose obtained in hardware using Poincaré map-based sim-to-real-transfer matches the intended motion in (a).

The background and related work focusses on current approaches to do low-level (continuous) and high-level (step-level) control and sim-to-real transfer for quadrupedal robots.

A. Low-level (continuous) control

The low-level control approaches are divided into model-based and model-free or a combination of the two.

The model-based control approach uses a model to compute joint torques based on the prescribed reference motion of the torso. The most successful approach has been to use model predictive control using the single rigid body model that assumes massless legs [3], [4]. The model is used to compute the ground reaction forces and foot stepping needed to achieve the desired linear and angular speed of the torso over a finite time horizon. The forces are converted to joint torques using the Jacobian from the foot to the torso. There have been variants of this approach such as, using automatic differentiation to compute gradients [5], using exponential coordinates to better approximate the dynamics [6], and use of full nonlinear dynamics with approximations of the first and second derivatives [7]. The limitation of this approach is that it relies on extensive onboard computations.

The model-free control approach involves optimizing the control policy by minimizing a suitable cost function. The most successful approach has been to use reinforcement learning. The control policy is a neural network that takes in as inputs the position and velocity of the legs and torso, and returns outputs of either the joint torques [8] or joint reference angles [9]. But reinforcement learning is not sample efficient. To overcome this, one can seed the learning

[1] Dept. of Mechanical and Industrial Engineering, University of Illinois at Chicago, 842 W Taylor St, Chicago, IL 60607, USA. Email: `pranav@uic.edu`, `dtorre38@uic.edu`, `eherna95@uic.edu` [2] Dept. of Mechanical Engineering, The University of Texas at San Antonio, 1 UTSA Circle, San Antonio, TX 78249, USA Email: `adel.alaeddini@utsa.edu` The work at UIC was supported by NSF grant 2128568.

with samples from a motion capture system [10] or learn in lower dimensional space such as in the Cartesian space [11].

Hybrid approaches that combine model-based and model-free methods can overcome the limitation of either approach. One approach is to use reinforcement learning to generate an initial population of solutions for the model predictive controller [12]. This approach has shown to generalize over a series of legged robots with minimal modification. Another approach is to use experimental data to improve the model, which then updates the model-predictive controller [13].

B. High-level (step-level) control

It is known that humans and animals control movement over the time scale of a step, also known as step-level control [14], [15]. The step-level control enables humans to use discrete footholds for efficient navigation [16].

The Poincaré section and map is one approach to do effective step-level control [17]. The Poincaré section is an instant in the locomotion cycle (e.g., foot strike). The Poincaré map is a function that maps the state and control from one Poincaré section to the state at the next Poincaré section [18]. Using the Poincaré map, it has been shown that a quadrupedal model with springy legs and no external actuation can demonstrate walking, trotting, and tolting gaits [19]. Using the eigenvalues of the linearized Poincaré map, it is shown that these passive gaits are stable [20].

The Poincaré map has been used to compute ground reaction forces to ensure periodicity in the bounding gait [21], improve the robot's ability to handle friction constraints [22], and enable gait transitions by finding common stable regions of two gaits at the Poincaré section [23].

C. Sim-to-real transfer

It is a common practice to tune control policies in simulation before they are deployed to hardware. However, model deficiencies may prevent these control policies from working well on hardware. This well known problem is known as the reality gap [24], [25]. This necessitates the use of techniques to close the reality gap to enable effective sim-to-real transfer.

Dynamic randomization is a popular method to enable sim-to-real transfer [26]. Here, a robust control policy is learned in simulation by randomizing the model with the expectation that the randomization would be effective in capturing the hardware and the physical environment [27]. This method can be generalized by training on many quadrupedal robots to make the controller robust to the hardware randomization [28]. One variant of this approach is to train two networks, one for the controller and the other for the variabilities in the environment. During run time, the second neural network predicts the correct environment variables, which is then used by the first network for control [29], [30]. However, one of the challenges in these approaches is that they rely on extensive simulations for deployment on hardware.

D. Our method and contribution

In this paper, we are interested in high-fidelity command following of a quadrupedal robot. To achieve this, we present three levels of control. (1) Low-level control: A task-level controller that uses inputs of prescribed torso orientation and velocity and outputs the ground reaction forces and the foot stepping location. These forces and foot stepping locations are then converted to joint torques using the Jacobian of the foot to the torso and the joint-level servo. (2) High-level control: A Poincaré map is used to correct for command tracking. The Poincaré map is obtained in simulation using a data-driven approach. (3) Sim-to-real transfer: A correction to the Poincaré map is done to account for the discrepancies between simulation and hardware. We investigate three methods for improving the Poincaré map, a low-order polynomial, a Gaussian process model, and a shallow neural network. Note that robot stability is not the focus of this paper, only the ability to follow a prescribed velocity command. In our testing for trotting on flat tiled and carpeted flooring, we found that the robot trot gait was stable enough that we did not have any falls.

The main contributions of this work are: (1) a computationally simple task-level balance controller that relies on an analytical inversion of matrices rather than iterative solutions of nonlinear equations in previous model-based approaches (e.g., [3], [4]); (2) the demonstration that the use of reference velocities (fore-aft, lateral, and yaw) leads to simple, decoupled, linear Poincaré maps that can be identified with a small set of experiments; and (3) the use of three approaches (polynomial, Gaussian process, and neural networks) to model the reality-gap to improve the sim-to-real transfer. Figure 1 summarizes the results obtained using the Poincaré map-based sim-to-real transfer. Some related papers are [31] where we used only Gaussian Process regression for sim-to-real transfer for a hopping robot, [32] compares neural network, gaussian process, and polynomial for approximating the Poincaré map for a hopping model, and [33] where we used Gaussian Process to model the Poincaré map for bipedal control.

The paper is organized as follows. The task-level controller is discussed in Sec. II followed by the high-level controller and sim-to-real transfer in Sec. III. The results of the approaches are presented in Sec. IV, followed by the discussion in Sec. V, and conclusion and future work in Sec. VI.

II. Continuous control (Low-level)

A. Hardware platform

The hardware platform used in this work is the quadruped A1 by Unitree Robotics [1]. The robot has four legs, each with 3 degrees of freedom. Each leg has a mass of 2 kg and the mass of the torso is 4.75 kg. There is an encoder at each joint, an inertial measurement unit on the torso, and a touch sensor on each foot. The minimum and maximum torques on each joint are -33.5 and 33.5 Nm respectively. There are sensors to measure the motor currents which are calibrated to compute the joint torques. There are two computers for

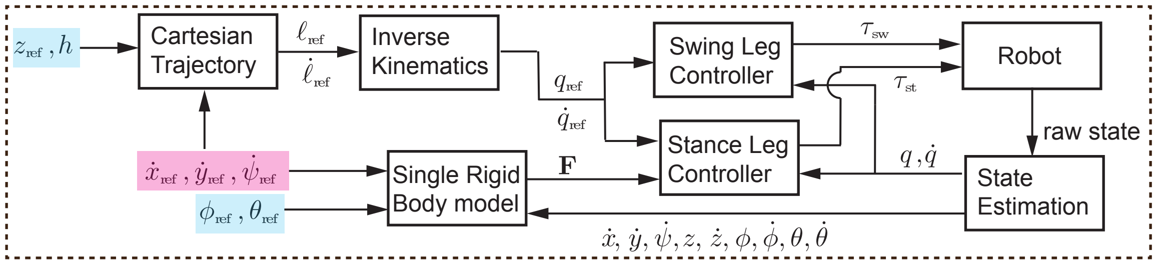

Fig. 2. Block diagram of the robot controller. A state machine is used to program the robot controller logic. The inputs to the state machine are shown in the blue/pink highlighted blocks. The inputs in the blue block, z_{ref} leg height, h foot clearance, ϕ_{ref} and θ_{ref} roll and pitch angle of the torso respectively are kept fixed in this paper. The inputs in the pink block $\dot{x}_{\text{ref}}$, $\dot{y}_{\text{ref}}$, $\dot{\psi}_{\text{ref}}$, the fore-aft, lateral and yaw reference velocities are set once per step. These blocks are explained in the text.

control execution: an external computer to set torques and reference motion, and a proprietary motor controller for joint control. The external computer sends commands at a rate of 1 kHz to the motor controller. Five commands can be sent and they include torque, position gain, velocity gain, position set-point, and velocity set-point. The motor controller on the A1 proprietary controller runs at 10 KHz and it is not accessible. The robot uses Lightweight Communications and Marshalling (LCM) and User Datagram Protocol (UDP) for communication.

B. State machine and its inputs

Figure 2 shows a block diagram of the continuous (low-level) controller. The continuous-level controller uses a finite state machine and it executes at 1 kHz. The legs are labelled as Front Right (FR), Front Left (FL), Rear Right (RR), and Rear Left (RL). For the trot gait, the combinations of FR-RL and FL-RR move together. The pair of legs cycle between stance and swing legs at a fixed time sequence given by the step time. The step time is fixed at 0.2 sec throughout this work.

The inputs to the state machine are the reference height of the legs z_{ref}, the ground clearance h, the reference roll and pitch of the torso ϕ_{ref} and θ_{ref} respectively. The other inputs are fore-aft speed, the lateral speed, and the yaw speed $\dot{x}_{\text{ref}}$, $\dot{y}_{\text{ref}}$, and $\dot{\psi}_{\text{ref}}$ respectively are set once per step by the step-level controller (see Sec. III).

C. Cartesian and Joint trajectory

The cartesian trajectory block takes in the torso height (z_{ref}), foot clearance (h), forward speed in body frame ($\dot{x}_{\text{ref}}$), lateral speed in body frame ($\dot{y}_{\text{ref}}$), and rate of turning ($\dot{\psi}_{\text{ref}}$) and produces the foot position as a function of time, ℓ_{ref} and using the fixed step time, it generates the linear speed $\dot{\ell}_{\text{ref}}$. Here $\ell = \{\ell_x, \ell_y, \ell_z\}$ are the positions of the foot with respect to the shoulder joint in the body frame. A fifth order polynomial is used generate a reference profile for the foot. The coefficients of the polynomial are obtained based on the 6 inputs: the initial and final position are known, the initial and final velocity are known, and the initial and final acceleration is zero.

The inverse kinematics blocks takes in the reference position and velocity of the foot, ℓ_{ref} and $\dot{\ell}_{\text{ref}}$ and produces

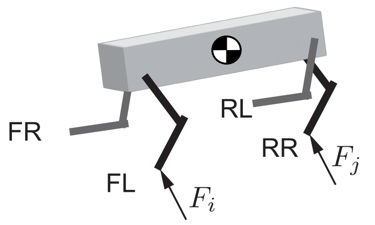

Fig. 3. Single Rigid Body (SRB) model assumes entire mass is located on the torso and the legs are massless. The model is used to compute ground forces for the prescribed torso motion.

the angular position (q_{ref}) and angular velocity ($\dot{q}_{\text{ref}}$) of the joints of the leg. Here $\mathbf{q} = \{q_0, q_1, q_2\}$ where q_0, q_1, q_2 are the hip roll angle, hip pitch angle, and knee pitch angle. The forward kinematics ($\mathbf{f}$) can be written as $\ell = \mathbf{f}(\mathbf{q})$. To compute the inverse kinematics, we have $\mathbf{q} = \mathbf{f}^{-1}(\ell)$. It is possible to compute an analytical inverse of $\mathbf{f}$ which makes computation fast (see [34] for details). The angular velocity can be found as $\dot{\mathbf{q}} = \mathbf{J}^{-1}\dot{\ell}$ where the Jacobian is given by $\mathbf{J} = \frac{\partial \ell}{\partial q}$.

D. Single Rigid Body Model

The single rigid body model (SRB) shown in Fig. 3 is used to compute the ground reaction force on the legs that are in the contact with the ground. These forces are then converted to joint torques (see Sec. II-E)

The SRB model assumes that the mass and inertia are concentrated in the torso and the legs are massless. Since we are interested in trotting, any two diagonal legs are assumed to be in contact with the ground. These contacting legs interact with the ground through ground reaction forces $\mathbf{F}_i$ and $\mathbf{F}_j$ where i and j are the two legs. For each force there is an x-, a y-, and a z- component. Using the free-body diagram, one can write

$$\mathbf{M}_{\text{tot}}\ddot{P} = \mathbf{F}_i + \mathbf{F}_j - m_{\text{tot}}\mathbf{g} \quad (1)$$

$$\mathbf{I}_{\text{tot}}\ddot{\Omega} = \mathbf{r}_i \times \mathbf{F}_i + \mathbf{r}_j \times \mathbf{F}_j = [\mathbf{r}_i]_\times \mathbf{F}_i + [\mathbf{r}_j]_\times \mathbf{F}_j \quad (2)$$

where $[\mathbf{v}]_\times$ is the vector $\mathbf{v}$ written as a skew-symmetric matrix. The total mass is m_{tot}, the corresponding mass matrix is $\mathbf{M}_{\text{tot}}$, the inertia about the center of mass is $\mathbf{I}_{\text{tot}}$, and gravity vector $\mathbf{g}$. In the last equation, we have ignored the term $\Omega \times (\mathbf{I}_{\text{tot}}\Omega)$. This is justified as the roll and pitch angular velocities are small and hence their coupling with yaw motion is negligible even for a large yaw rate (turning).

Using the SRB model, we do a task-based control of the torso using the stance legs. Our goal is to regulate the torso as follows

$$\ddot{x} = C_x(\dot{x}_{\text{ref}} - \dot{x}), \tag{3}$$
$$\ddot{y} = C_y(\dot{y}_{\text{ref}} - \dot{y}), \tag{4}$$
$$\ddot{z} = K_z(z_{\text{ref}} - z) - C_z\dot{z} \tag{5}$$
$$\ddot{\phi} = K_\phi(\phi_{\text{ref}} - \phi) - C_\phi\dot{\phi}, \tag{6}$$
$$\ddot{\theta} = K_\theta(\theta_{\text{ref}} - \theta) - C_\theta\dot{\theta}, \tag{7}$$
$$\ddot{\psi} = C_\psi(\dot{\psi}_{\text{ref}} - \dot{\psi}) \tag{8}$$

where K_* and C_* are position and damping gains that are hand-tuned in simulation. We substitute Eqns 3-5 in Eqn. 1 where $\ddot{P} = \begin{bmatrix}\ddot{x} & \ddot{y} & \ddot{z}\end{bmatrix}$ and substitute Eqns 6-8 in Eqn. 2 where $\ddot{\Omega} = \begin{bmatrix}\ddot{\phi} & \ddot{\theta} & \ddot{\psi}.\end{bmatrix}$.

The equations can then be simplified to the following form.

$$\mathbf{AF} = \mathbf{b} \tag{9}$$

where $\mathbf{A}$ and $\mathbf{b}$ are functions of mass, gravity and location of the center of mass from the feet and $\mathbf{F} = \begin{bmatrix}\mathbf{F}_i, \mathbf{F}_j\end{bmatrix}$ is the 6×1 unknown force vector. Using the pseudo inverse, $\mathbf{A}^+$, to solve for, $\mathbf{F} = \mathbf{A}^+\mathbf{b}$

E. Stance leg controller

The stance leg controller block uses the ground reaction forces from the SRB model and computes the joint torques on the stance legs. The stance leg torque is computed as $\tau_{st} = \mathbf{J}^T\mathbf{F}$ where $\mathbf{J}$ is the Jacobian of the foot with respect to joint angles $\mathbf{J} = \frac{\partial \ell}{\partial q}$. This controller is task-based since it uses the feedback from the task space (or Cartesian space) to obtain the torques. This controller can become unstable for large tracking errors. To create stable control, additional feedback from joint angles is needed,

$$\tau_{st} = \mathbf{J}^T\mathbf{F} + \mathbf{K}_q(\mathbf{q}_{\text{ref}} - \mathbf{q}) + \mathbf{C}_q(\dot{\mathbf{q}}_{\text{ref}} - \dot{\mathbf{q}}) \tag{10}$$

where the joint angles $\mathbf{q} = \begin{bmatrix}q_0 & q_1 & q_2\end{bmatrix}$, the proportional gain at the joint is $\mathbf{K}_q$ and damping gain at the joint is $\mathbf{C}_q$.

F. Swing phase controller

The swing phase controller uses the joint level control using reference joint angles/rates and actual joint angles/rates. The torque is given by

$$\tau_{sw} = \mathbf{J}^T\mathbf{F}_g + \mathbf{K}_q(\mathbf{q}_{\text{ref}} - \mathbf{q}) + \mathbf{C}_q(\dot{\mathbf{q}}_{\text{ref}} - \dot{\mathbf{q}}) \tag{11}$$

where $\mathbf{F}_g$ is the torque due to gravity. Since the legs are lightweight, we set the $\mathbf{F}_g = \mathbf{0}$ to simplify the computation. This had an insignificant effect on the tracking as the legs are light weight; the feedback controller makes up for this modeling error.

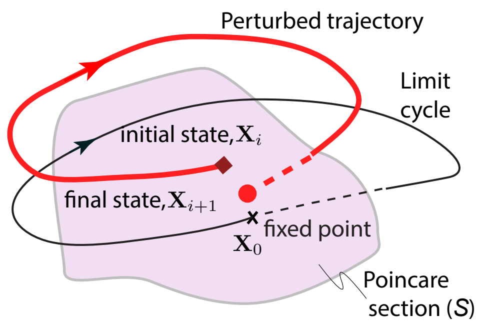

Fig. 4. Poincaré map and section

G. State Estimation

The state estimation block takes inputs from the joint angle sensors and inertial measurement unit and returns the estimated state. The joint angular velocity was also made available by the robot manufacturer. This was probably obtained by differentiation and filtering. The sensors provided by the software were not filtered or post-processed as they were found to be adequate for control. However, we estimated the height of the leg z using forward kinematics and sensor joint angles. We also estimated the linear velocities $\dot{x}, \dot{y}, \dot{z}$ using the derivative of the forward kinematic terms and the measured joint angles. We had to use an exponential filter to smooth out the estimated linear velocities.

III. Step-level control (High-level)

The low-level control discussed in Sec. II achieves the desired motion of the torso using the torques on the stance and swing legs. The torso has 3 positions, 3 angular positions, and their respective rates leading to $N = 12$ states. However, to control the overall progression of the robot, only a subset of these states needs to be regulated. We choose three states, $N_u = 3$. These are: the body frame torso speed in the fore-aft direction $\dot{x}$, the torso speed in the lateral direction $\dot{y}$, and the yaw velocity $\dot{\psi}$. These are compactly written as $\mathbf{\Theta}_u = \{\dot{x}, \dot{y}, \dot{\psi}\}$. The Poincaré section and map is an efficient way of controlling this low-dimensional set of states and is discussed in the next section.

To control these, we use three parameters from our low-level controller discussed Sec. II and shown in Fig. 2. These are fore-aft reference velocity $\dot{x}_{\text{ref}}$, lateral reference velocity $\dot{y}_{\text{ref}}$, and the yaw velocity $\dot{\psi}_{\text{ref}}$. These are compactly written as $\mathbf{U} = \{\dot{x}_{\text{ref}}, \dot{y}_{\text{ref}}, \dot{\psi}_{\text{ref}}\}$. We now resort to a Poincaré map to achieve the desired regulation.

A. Poincaré section and map

Figure 4 shows a pictorial depiction of the Poincaré section and map [17]. Consider an instant in the gait cycle (e.g., an instant when any two diagonal feet contact the ground). This is our chosen Poincaré section. Let the reduced state at the Poincaré map at the current step be $\mathbf{\Theta}_u^i$. This is shown with the red diamond. Our low-level control parameterization chose the control $\mathbf{U}^i$ which then takes the reduced state to $\mathbf{\Theta}_u^{i+1}$ at the Poincaré section at the next step. We can find a function $\mathbf{P}$, known as the Poincaré map, that maps the reduced state from one step to the next. This is given by

$$\mathbf{\Theta}_u^{i+1} = \mathbf{P}(\mathbf{\Theta}_u^i, \mathbf{U}^i) \tag{12}$$

Note that the control $\mathbf{U}^i$ is set once per step and kept constant during the step.

B. High-level control problem

The high-level control problem is stated as follows. Given the state at the current step, $\mathbf{\Theta}_u^i$, compute the control at the current step, $\mathbf{U}^i$ such that the state at the next step is $\mathbf{\Theta}_u^{\text{des}}$. From Eqn. 12 we can write

$$\mathbf{\Theta}_u^{\text{des}} = \mathbf{P}(\mathbf{\Theta}_u^i, \mathbf{U}^i) \tag{13}$$

For a generic Poincaré map, the control $\mathbf{U}^i$ can be found by root finding, which might involve a finite search. However, as shown next, for our chosen task-based controller, the Poincaré map given by Eqn. 12 is linear and decoupled, thus simplifying the control.

C. Poincaré map simplification

The generic expression for the Poincaré map given by Eqn. 12 has 3 outputs in $\mathbf{\Theta}_u^{i+1}$ and 6 inputs in $\mathbf{\Theta}_u^i$ and $\mathbf{U}^i$. This would make the Poincaré map, $\mathbf{P}$, high dimensional. The task-level controller achieves decoupling of the dynamics. This is evident from Eqns. 3 - 8. Since the velocities in the x-, y-, and ψ direction are decoupled, we can write three decoupled equations for the Poincaré map as follows

$$\dot{x}^{i+1} = \mathbf{P}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) \tag{14}$$
$$\dot{y}^{i+1} = \mathbf{P}_y(\dot{y}^i, \dot{y}_{\text{ref}}^i) \tag{15}$$
$$\dot{\psi}^{i+1} = \mathbf{P}_\psi(\dot{\psi}^i, \dot{\psi}_{\text{ref}}^i) \tag{16}$$

The above equations are an assumed simplification that needs to be checked with data. The checks were done as follows in the simulation. We varied $\dot{x}_{\text{ref}}$ and measured the state $\dot{x}$. Then we fit the Eqn. 14 using the simplest polynomial expression. We repeated this for Eqns. 15 and 16. This completed the training of the equations. Next, we test the equations by running randomly chosen inputs $\dot{x}_{\text{ref}}$, $\dot{y}_{\text{ref}}$, and $\dot{\psi}_{\text{ref}}$ and found that the above equations can explain the test data.

D. Sim-to-real transfer

The simplified, decoupled Poincaré map equations in Sec. III-C are fit in simulation. When tested on hardware, they may not lead to perfect tracking. To improve tracking, we improve the Poincaré map as follows. Let the measured (true) state be $\dot{x}_{\text{true}}^{i+1}, \dot{y}_{\text{true}}^{i+1}, \dot{\psi}_{\text{true}}^{i+1}$. We fit the Poincaré map errors as follows

$$\dot{x}_{\text{true}}^{i+1} - \mathbf{P}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) = \mathbf{\Delta}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) \tag{17}$$
$$\dot{y}_{\text{true}}^{i+1} - \mathbf{P}_y(\dot{y}^i, \dot{y}_{\text{ref}}^i) = \mathbf{\Delta}_y(\dot{y}^i, \dot{y}_{\text{ref}}^i) \tag{18}$$
$$\dot{\psi}_{\text{true}}^{i+1} - \mathbf{P}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) = \mathbf{\Delta}_\psi(\dot{\psi}^i, \dot{\psi}_{\text{ref}}^i) \tag{19}$$

The function $\mathbf{\Delta_x}$, $\mathbf{\Delta_y}$, and $\mathbf{\Delta}_\psi$ are assumed to be functions of state and control as shown. We used three methods to fit these functions: (1) polynomial regression, (2) Gaussian process regression, and (3) neural network-based regression. Once these functions are fitted, the control problem is to compute the control $\dot{x}_{\text{ref}}^i$, $\dot{y}_{\text{ref}}^i$, $\dot{\psi}_{\text{ref}}^i$ given the state at the current step $\dot{x}^i$, $\dot{y}^i$, $\dot{\psi}^i$ and the desired state $\dot{x}^{\text{des}}$, $\dot{y}^{\text{des}}$, and $\dot{\psi}^{\text{des}}$

$$\dot{x}^{\text{des}} = \dot{x}^{i+1} = \mathbf{P}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) + \mathbf{\Delta}_x(\dot{x}^i, \dot{x}_{\text{ref}}^i) \tag{20}$$
$$\dot{y}^{\text{des}} = \dot{y}^{i+1} = \mathbf{P}_y(\dot{y}^i, \dot{y}_{\text{ref}}^i) + \mathbf{\Delta}_y(\dot{y}^i, \dot{y}_{\text{ref}}^i) \tag{21}$$
$$\dot{\psi}^{\text{des}} = \dot{\psi}^{i+1} = \mathbf{P}_\psi(\dot{\psi}^i, \dot{\psi}_{\text{ref}}^i) + \mathbf{\Delta}_\psi(\dot{\psi}^i, \dot{\psi}_{\text{ref}}^i) \tag{22}$$

To solve for the control, we need to do numerical root finding as the equations are nonlinear and cannot be solved analytically.

IV. Results

The low-level controller discussed in Sec. II was programmed using the C interface of MuJoCo [35]. The low-level gains of the joints, $\mathbf{K}_q$ and $\mathbf{C}_q$ (see Eqns. 10 and 11) were manually tuned in simulation and then verified that they give reasonable tracking in hardware. The gains of the task-level controller, C_x, C_y, K_z, C_z, K_ϕ, C_ϕ, K_θ, C_θ, and C_ψ (see Eqns. 3 to 8), were tuned in a similar fashion. We set the robot height $z_{\text{ref}} = 0.275$ m, foot clearance height $h = 0.1$ m, and step time $t_s = 0.2$ sec. The worst-case computation time for the controller was 0.1 ms. Since the controller runs at 1 kHz (1 ms control loop), this is adequate.

Figure 5 shows the performance of the low-level controller when the inputs $\dot{x}_{\text{ref}}$, $\dot{y}_{\text{ref}}$, $\dot{\psi}_{\text{ref}}$, and θ_{ref} are varied. These reference values are shown using a red dotted line and abbreviated as ‘reference’. The output is the raw sensor data that is shown in black and can be seen to be noisy. The torso fore-aft speed, lateral speed, and yaw speed show the most noise and lead to unstable behavior. These values were filtered using an exponential filter. The ‘filtered’ values are shown as a blue solid line and have acceptable values for control. The exponential filtering and task-level gains cause tracking errors between ‘reference’ and ‘filtered’ values for the three speeds. These are improved using Poincaré map, as discussed next.

Our goal is to tune the ‘reference’ such that the ‘filtered’ values track the desired velocities given by $\dot{x}^{\text{des}}$, $\dot{y}^{\text{des}}$, $\dot{\psi}^{\text{des}}$. To do this, we fit the Poincaré map given by Eqns. 14 - 16 in a data-driven fashion. We run three experimental runs in the simulation. In each experiment, each of the three state/command pairs (e.g,. $\dot{x}^i$ and $\dot{x}_{\text{ref}}^i$) are varied. Then fit the simplest polynomial expression for $\mathbf{P}_x$, $\mathbf{P}_y$ and $\mathbf{P}_\psi$ using the simulation data. We found that a first-order polynomial could adequately fit the experimental data. Our fit was

$$\dot{x}^{i+1} = 0.471\dot{x}^i + 0.488\dot{x}_{\text{ref}}^i \tag{23}$$
$$\dot{y}^{i+1} = 0.443\dot{y}^i + 0.511\dot{y}_{\text{ref}}^i \tag{24}$$
$$\dot{\psi}^{i+1} = -0.298\dot{\psi}^i + 1.205\dot{\psi}_{\text{ref}}^i \tag{25}$$

The R^2 for these fits (in the given order) were 0.998, 0.999, and 0.999 respectively and the root mean square error, RMSE, was 0.015 m/s, 0.008 m/s, and 0.012 rad/s respectively. This indicates that the assumed decoupled linear form for the Poincaré map is an adequate approximation of the Poincaré map.

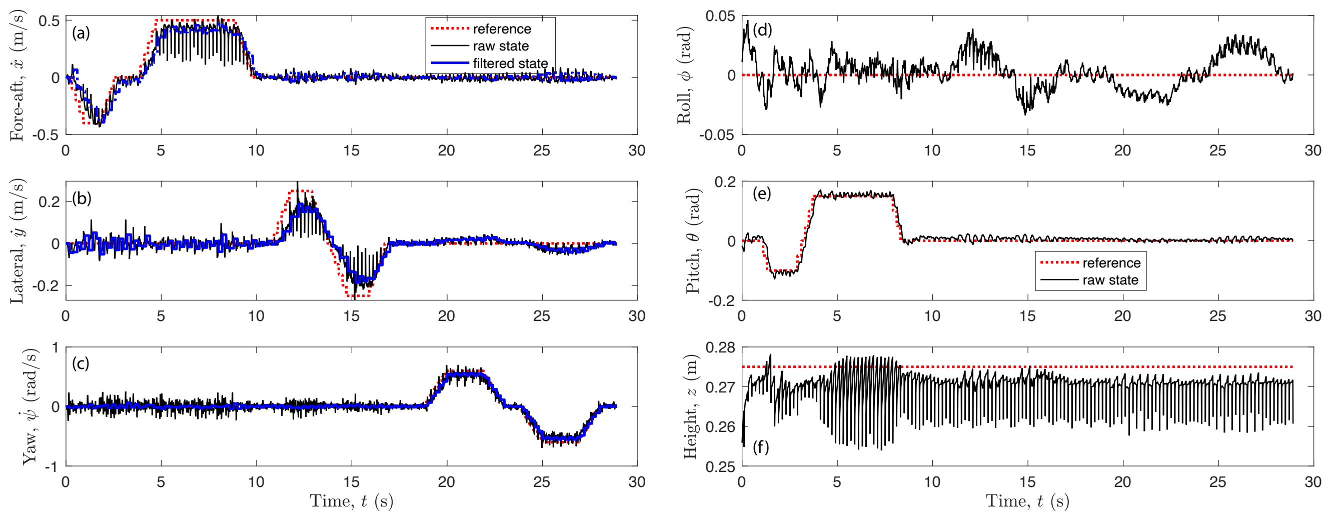

Fig. 5. Low-level control on hardware: Tracking for (a) fore-aft speed $\dot{x}$ (b) lateral speed $\dot{y}$, and (c) yaw speed $\dot{\psi}$, (d) torso roll ϕ, (e) torso pitch θ, and (f) torso height z versus time. The black indicates raw sensor values (for roll, pitch, yaw rate) and estimates from raw sensor data (for fore-aft and lateral speed and height). The values for speed are filtered, as shown in blue using an exponential filter. Note that the tracking for speed is off, the reference (shown in red) is off from the filtered sensor value (shown in blue).

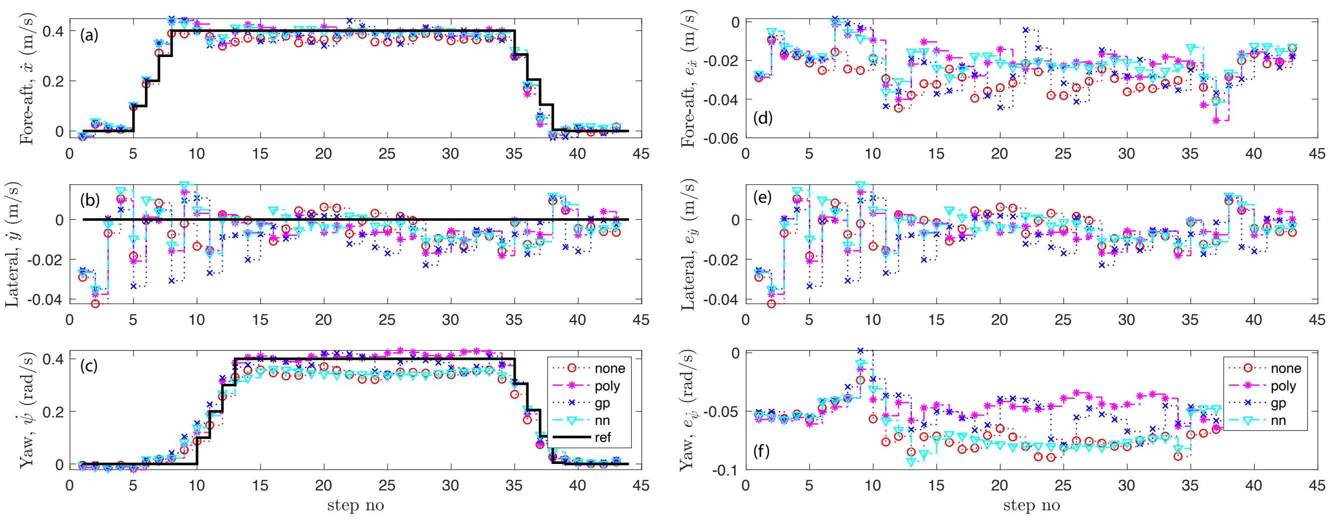

Fig. 6. Sim-to-real transfer for high-level controller: Tracking for (a) fore-aft speed $\dot{x}$, (b) lateral speed $\dot{y}$, and (c) yaw speed $\dot{\psi}$ versus step number. Tracking error for (d) fore-aft speed $e_{\dot{x}}$, (e) lateral speed $e_{\dot{y}}$, and (f) yaw speed $e_{\dot{\psi}}$ versus step number. The 'none' there was no training on this controller, as it was based on simulation. The 'poly', 'gp', 'nn' indicates learning using linear polynomial regression, Gaussian process regression, and neural network-based regression, respectively.

To test the fit, we generated a desired state, $\dot{x}^{\text{des}}$, $\dot{y}^{\text{des}}$, $\dot{\psi}^{\text{des}}$. Then we set $\dot{x}^{i+1} = \dot{x}^{\text{des}}$, $\dot{y}^{i+1} = \dot{y}^{\text{des}}$, and $\dot{\psi}^{i+1} = \dot{\psi}^{\text{des}}$ and inverted Eqns. 23 - 25 to compute the input command $\dot{x}^i_{\text{ref}}$, $\dot{y}^i_{\text{ref}}$, $\dot{\psi}^i_{\text{ref}}$ for the measured state, $\dot{x}^i$, $\dot{y}^i$, and $\dot{\psi}^i$. The testing was done on hardware. In the Fig. 6, the red dotted line with a circle shows the fidelity of the tracking. The plots (a), (b), (c) show the reference and tracking, while (d), (e), and (f) show the tracking error. From the latter plots, it can be seen that the tracking errors is within -0.04 to 0, -0.04 to 0, and -0.1 to 0 for $\dot{x}$, $\dot{y}$, and $\dot{\psi}$ respectively.

To further improve the sim-to-real transfer, we used the data from the hardware run to fit the errors between the

Type of sim-to-real	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{\psi}$ (rad/s)
no learning	0.0282	0.0112	0.0446
polynomial	0.0232	0.0111	0.0188
gaussian process	0.0296	0.0141	0.0288
neural network	0.0196	0.0104	0.0447

TABLE I

Root mean square error for methods to do sim-to-real transfer (polynomial, Gaussian process, neural network) again no learning. The latter involved transferring the learned model from the simulation to the hardware.

experimental value and the model given by Eqns. 23 - 25. Our goal is to fit the Δ's in Eqns 17 - 19. We tried three types of regression models, a first-order polynomial, a Gaussian process, and a neural network. For the polynomial, we use the MATLAB function *polyfitn* (available at MATLAB central) to fit a first-order polynomial. For the Gaussian process, we used MATLAB function *fitrgp* with a quadratic basis function and a matern52 kernel. For the neural network, we used MATLAB function *train* with just 2 hidden layers.

The resulting model for the Poincaré map is given by Eqns. 17 to 19 is nonlinear. The resulting control problem given by Eqns. 20 to 22 is solved using a nonlinear root finder. Here inputs are $\dot{x}^i$ and $\dot{x}^{\text{des}}$ and the unknowns are the commands $\dot{x}^i_{\text{ref}}$ and similar to other equations. We used MATLAB function *fsolve* to solve the problem and saved a lookup table. The lookup table was evaluated in real time to compute the required control in hardware. Figure 6 shows the results for each of these fits: magenta dashed line with an asterisk for polynomial (poly), blue dashed line with a cross for Gaussian process (gp), and cyan dashed line with a diamond for neural network (nn). Table IV compares the root mean square error (rmse) for the three fits and with no learning (first line). It can be seen that the neural network does the most improvement to the $\dot{x}$ fit and the Gaussian process does the most improvement to the $\dot{\psi}$ fit while there is almost no improvement for $\dot{y}$ whose desired reference is 0 in the experiment. These results show that the Poincaré map refinement using experimental data enables better sim-to-real transfer.

V. Discussion

The paper presented a task-level balance controller for low-level balance control and a Poincaré map-based high-level controller for command following. The task-level controller takes inputs of reference body orientation and velocities and calculates ground reaction forces using the single rigid body model and stepping locations. These forces are then converted to joint torques using the Jacobian from the foot to the torso and supplemented with joint-level torques from inverse kinematics for increased stability. Then a data-driven approach is used in simulation to fit a Poincaré map between torso velocities at the next step as a function of torso velocities at the current step and reference velocity. The Poincaré map is further refined using experimental data to enable sim-to-real transfer and for effective tracking on the hardware. The approach is validated on a quadrupedal robot.

The prime advantage of the approach is the choice of parameterization used in the task-level control; the reference velocities are in the fore-aft, lateral, and yaw directions. This choice simplifies the Poincaré map to be linear and decoupled. Thus, very little simulated data is needed to fit the Poincaré map. These advantages carry forward to the sim-to-real transfer, where again, very little data is needed to improve the Poincaré map approximation. Another advantage is the computational simplicity of the approach. The task-level control relies on analytical inverse using the equation of the single rigid body model and the Poincaré map is based on 1-dimensional root solving.

The task-level balance control relies on smooth and accurate estimates of the torso's linear velocity (fore-aft and lateral), but this was a major challenge. We used the kinematic model of the stance leg with the joint angle/velocity and torso orientation data to compute the linear velocity of the shoulder joint. We then averaged the velocities of the two stance legs to compute the linear velocity of the torso. This estimate was noisy, and we had to use a slow filter to smoothen the estimate. This caused the estimated velocity to lag the actual velocity even though the controller was stable. One way to solve the problem is to fuse the joint angle/velocity data with the accelerometer data with a model-based filter (e.g., Kalman filter).

The presented work has limitations that need to be addressed in the future. In the trot gait, the quadrupedal is fully actuated at all times (assuming all feet are firmly on the ground). This makes it straightforward to implement task-level control. The task-level control needs to be modified for systems with under-actuation (e.g., bipeds with small feet) and other quadrupedal gaits with phases of underactuation (e.g., bounding). One way of overcoming the under-actuation is to delay stabilization of some degrees of freedom to a longer horizon using the Poincaré map [36]. The computation of feasible ground reaction forces is based on the single rigid body model, which is a good approximation for robots with light legs and heavy torso and at relatively slow speeds. This approximation needs to be thoroughly evaluated for robots with a distributed mass (e.g., humanoids) and at high speeds (e.g., bounding). Our controller is based on fixed step time (as is the standard practice for trot gaits), but this assumption can be problematic when the robot is negotiating rough terrain where the foot contact time is variable. Finally, it is unclear if the simple parameterization of the task-level control would enable simple, decoupled Poincaré map equations for underactuated systems like bipedal robots.

VI. Conclusions and Future Work

We conclude that task-level balance control using a single rigid body model is a promising approach for control of quadrupedal robots for trotting. The approach is computationally simple and leads to simple, decoupled equations for step-level control. The latter enables fitting the Poincaré map with very little data in simulation and hardware, which enables a high fidelity command following in hardware.

The future work will explore two thrusts. One, extending the control approach to uneven terrain where it is expected that the Poincaré map estimation, state estimation, and sim-to-real transfer would be more challenging. Two, using the step-level control in a motion planning framework for navigation in an environment with obstacles.

Appendix

A video that shows the results can be found on this link `https://youtu.be/OKZ4axbqo44` or this short link `https://tiny.cc/humanoids23`.

Acknowledgment

The authors would like to thank Md. Safwan Mondal for help with the video production.

References

[1] Unitree-Robotics, "A1 more dexterity, more possibilities," https://www.unitree.com/products/a1/, June 2022.

[2] E. Guizzo, "By leaps and bounds: An exclusive look at how boston dynamics is redefining robot agility," *IEEE Spectrum*, vol. 56, no. 12, pp. 34–39, 2019.

[3] Y. Ding, A. Pandala, and H.-W. Park, "Real-time model predictive control for versatile dynamic motions in quadrupedal robots," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8484–8490.

[4] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2018, pp. 1–9.

[5] A. Zheng, S. S. Narayanan, and U. G. Vaidya, "Optimal control for quadruped locomotion using ltv mpc," *arXiv preprint arXiv:2212.05154*, 2022.

[6] W. Chi, X. Jiang, and Y. Zheng, "A linearization of centroidal dynamics for the model-predictive control of quadruped robots," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 4656–4663.

[7] D. Kang, F. De Vincenti, and S. Coros, "Nonlinear model predictive control for quadrupedal locomotion using second-order sensitivity analysis," *arXiv preprint arXiv:2207.10465*, 2022.

[8] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath, "Learning torque control for quadrupedal locomotion," *arXiv preprint arXiv:2203.05194*, 2022.

[9] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[10] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[11] G. Bellegarda and Q. Nguyen, "Robust high-speed running for quadruped robots via deep reinforcement learning," *arXiv preprint arXiv:2103.06484*, 2021.

[12] H. Li, W. Yu, T. Zhang, and P. M. Wensing, "Zero-shot retargeting of learned quadruped locomotion policies using hybrid kinodynamic model predictive control," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 11 971–11 977.

[13] R. T. Fawcett, K. Afsari, A. D. Ames, and K. A. Hamed, "Toward a data-driven template model for quadrupedal locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7636–7643, 2022.

[14] A. E. Patla, "Understanding the roles of vision in the control of human locomotion," *Gait & posture*, vol. 5, no. 1, pp. 54–69, 1997.

[15] I. Beloozerova and M. Sirota, "The role of the motor cortex in the control of accuracy of locomotor movements in the cat." *The Journal of physiology*, vol. 461, no. 1, pp. 1–25, 1993.

[16] M. A. Hollands, D. E. Marple-Horvat, S. Henkes, and A. K. Rowan, "Human eye movements during visually guided stepping," *Journal of motor behavior*, vol. 27, no. 2, pp. 155–163, 1995.

[17] S. H. Strogatz, *Nonlinear dynamics and chaos (with applications to physics, biology, chemistry a*. Perseus Publishing, 2006.

[18] P. A. Bhounsule, J. Cortell, A. Grewal, B. Hendriksen, J. D. Karssen, C. Paul, and A. Ruina, "Low-bandwidth reflex-based control for lower power walking: 65 km on a single battery charge," *International Journal of Robotics Research*, 2014.

[19] Z. Gan, T. Wiestner, M. A. Weishaupt, N. M. Waldern, and C. David Remy, "Passive dynamics explain quadrupedal walking, trotting, and tölting," *Journal of computational and nonlinear dynamics*, vol. 11, no. 2, 2016.

[20] C. D. Remy, K. Buffinton, and R. Siegwart, "Stability analysis of passive dynamic walking of quadrupeds," *The International Journal of Robotics Research*, vol. 29, no. 9, pp. 1173–1185, 2010.

[21] H.-W. Park, S. Park, and S. Kim, "Variable-speed quadrupedal bounding using impulse planning: Untethered high-speed 3d running of mit cheetah 2," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5163–5170.

[22] K. A. Hamed, J. Kim, and A. Pandala, "Quadrupedal locomotion via event-based predictive control and qp-based virtual constraints," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4463–4470, 2020.

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

[24] S. Koos, J.-B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010, pp. 119–126.

[25] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard *et al.*, "Sim2real in robotics and automation: Applications and challenges," *IEEE transactions on automation science and engineering*, vol. 18, no. 2, pp. 398–400, 2021.

[26] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4955–4961.

[27] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[28] G. Feng, H. Zhang, Z. Li, X. B. Peng, B. Basireddy, L. Yue, Z. Song, L. Yang, Y. Liu, K. Sreenath *et al.*, "Genloco: Generalized locomotion controllers for quadrupedal robots," *arXiv preprint arXiv:2209.05309*, 2022.

[29] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[30] S. Choi, G. Ji, J. Park, H. Kim, J. Mun, J. H. Lee, and J. Hwangbo, "Learning quadrupedal locomotion on deformable terrain," *Science Robotics*, vol. 8, no. 74, p. eade2256, 2023.

[31] J. Krause, A. Alaeddini, and P. A. Bhounsule, "Gaussian process regression for sim-to-real transfer of hopping gaits," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2023.

[32] P. A. Bhounsule, M. Kim, and A. Alaeddini, "Approximation of the step-to-step dynamics enables computationally efficient and fast optimal control of legged robots," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 83990. American Society of Mechanical Engineers, 2020, p. V010T10A050.

[33] D. Torres, E. Hernandez Hinojosa, and P. A. Bhounsule, "Control of a bipedal walking using partial feedback linearization and gaussian process regression-based of the step-to-step map," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2023.

[34] P. A. Bhounsule and C.-M. Yang, "A simple controller for omnidirectional trotting of quadrupedal robots: Command following and waypoint tracking," *Robotics*, vol. 12, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2218-6581/12/2/35

[35] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[36] P. A. Bhounsule, A. Ruina, and G. Stiesberg, "Discrete-decision continuous-actuation control: balance of an inverted pendulum and pumping a pendulum swing," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 5, p. 051012, 2015.