

Robust Feedback Motion Policy Design Using Reinforcement Learning on a 3D Digit Bipedal Robot

Guillermo A. Castillo¹, Bowen Weng¹, Wei Zhang², and Ayonga Hereid³

Abstract—In this paper, a hierarchical and robust framework for learning bipedal locomotion is presented and successfully implemented on the 3D biped robot Digit built by Agility Robotics. We propose a cascade-structure controller that combines the learning process with intuitive feedback regulations. This design allows the framework to realize robust and stable walking with a reduced-dimensional state and action spaces of the policy, significantly simplifying the design and increasing the sampling efficiency of the learning method. The inclusion of feedback regulation into the framework improves the robustness of the learned walking gait and ensures the success of the sim-to-real transfer of the proposed controller with minimal tuning. We specifically present a learning pipeline that considers hardware-feasible initial poses of the robot within the learning process to ensure the initial state of the learning is replicated as close as possible to the initial state of the robot in hardware experiments. Finally, we demonstrate the feasibility of our method by successfully transferring the learned policy in simulation to the Digit robot hardware, realizing sustained walking gaits under external force disturbances and challenging terrains not incurred during the training process. To the best of our knowledge, this is the first time a learning-based policy is transferred successfully to the Digit robot in hardware experiments.

I. INTRODUCTION

For the bipedal locomotion problem, policy *robustness* is a critical characteristic and remains one of the biggest challenges in the field. In practice, the robustness of the control policy can be presented as (i) the capability of handling various external disturbances (e.g., push recovery), (ii) maintaining stable gaits while operating under various terrain conditions, and (iii) accomplishing the sim-to-real transfer with as little effort as possible. In this paper, we present a successful application of designing a feedback motion policy on Digit, a challenging 3D bipedal robot, and demonstrate robust performance among all the aspects mentioned above.

In general, a robust bipedal locomotion policy can be obtained through (i) the model-based approach, (ii) the learning-based approach, or (iii) a combination of both. In the model-based regime, existing research typically relies on a simplified template model [1], [2]. Some have also explored robust control formulations of Control Lyapunov Function

*This work was supported in part by the OSU M&MS Discovery Theme Initiative, and the National Natural Science Foundation of China under Grant No. 62073159.

¹Electrical and Computer Engineering, Ohio State University, Columbus, OH, USA; {castillomartinez.2, weng.172}@osu.edu.

²SUSTech Institute of Robotics, Southern University of Science and Technology (SUSTech), China; zhangw3@sustech.edu.cn.

³Mechanical and Aerospace Engineering, Ohio State University, Columbus, OH, USA. hereid.1@osu.edu.

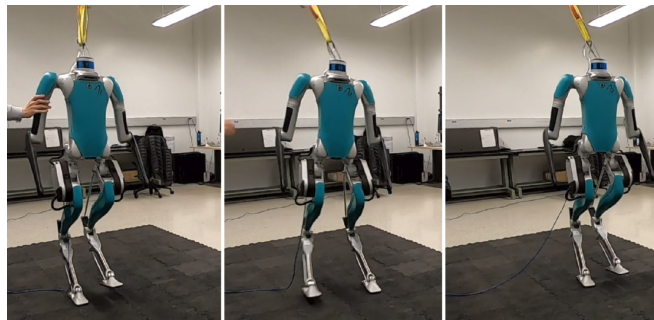


Fig. 1: Digit recovering from an external force disturbance applied in the lateral direction.

Control Barrier Functions (CLF-CBF) [3], and Hybrid Zero Dynamics (HZD) inspired solutions [4]. In general, the derived policy from the simplified model requires further tuning of low-level control heuristics. On the other hand, a typical learning-based solution involves supervised learning, reinforcement learning, and imitation learning. The policy is typically constructed in an end-to-end manner [5], directly working with the robot’s full-body dynamics. Some existing work has shown remarkable progress in push-recovery [6] and operating in various terrain conditions [7]. While model-based methods often rely on a simplified model and require extensive control gain tuning, most learning-based solutions require a large amount of data.

A locomotion policy combining the learning-based and the model-based solution is typically formulated in a cascade structure. The high-level controller computes a reference trajectory of a selected anchor point (e.g., the center of mass position). The lower-level controller then seeks to track such a learned reference through basic model information such as kinematics. Morimoto et.al. [8], [9] learned the Poincaré map of the periodic walking pattern and applied the method to two 2D bipedal robots. Some recent work has proposed to learn the joint-level trajectory for each joint as the reference motion through supervised learning [10] or using reinforcement learning [11]–[13]. This approach simplifies the design of the lower-level tracking, which can be as simple as a PD controller.

With the aforementioned approaches, the empirical successes are mostly evaluated in simulations. Some recent work has sought to tackle the sim-to-real problem through dynamics randomization [14], system identification [15], and periodic reward composition [16]. However, many of these methods still suffer from poor sampling efficiency. Moreover,

the robustness is often partially validated in real hardware experiments. To the best of our knowledge, it remains a challenge to efficiently obtain a locomotion policy that handles external disturbances and uneven terrains with little effort sim-to-real transfer. The challenge further escalates as one considers a 3D, underactuated, and highly nonlinear bipedal system such as the Digit robot (see Fig. 2).

Motivated by the existing challenges, this paper enhances the hybrid zero dynamics inspired reinforcement learning framework presented in our previous work [11], [13], emphasizing the policy robustness and sim-to-real transfer. Built upon [13], this paper adds important considerations to the learning framework for the effective transfer of the policy learned in simulation to the real robot. In particular, we add a balancing controller into the learning pipeline that ensures the training starts from a pool of feasible initial states. Moreover, we provide a detailed implementation in hardware to couple the high-level learning-based controller with the low-level model-based controller. Finally, we show results from exhaustive testing of the learned policy under challenging terrains and external disturbances.

The learned policy is able to achieve robust bipedal locomotion on a challenging 3D Digit robot in experiments to accomplish the following tasks.

Learn disturbance resistance without explicitly experiencing adversarial attacks in training.

Adapt to various terrain conditions with the policy learning process only limited to flat ground.

Transfer the learned policy between different simulation environments and from simulation to the real robot without randomized dynamics in training or extensive hyper-parameter tuning.

It is worth emphasizing that the policy is learned from scratch and does not rely on demonstrations. To the best of our knowledge, this is the first successful sim-to-real transfer of a learning-based policy to Digit hardware. We further emphasize the following features that contribute to the aforementioned robust performance of the proposed policy:

The high-level learning-based policy propagates reference motion trajectories in real-time at a lower frequency compatible with real hardware implementation. This encourages the stability of the walking gait by modifying the reference trajectories accordingly to the reduced-dimensional state of the robot.

The low-level model-based regulation applies compensation to the reference trajectories at a higher frequency based on instantaneous state feedback. This feedback-based trajectory regulation compensates for uncertainty in the environment, and based on our experiment observations, it is one of the key factors leading to the effortless sim-to-real transfer (as illustrated in Fig. 6).

A model-based balancing controller is implemented that induces a set of feasible initialization states for operations in the simulator and with the real robot. While a feasible initialization is typically not of interest in simulation given that the state can be set arbitrarily, it is crucial to the observed

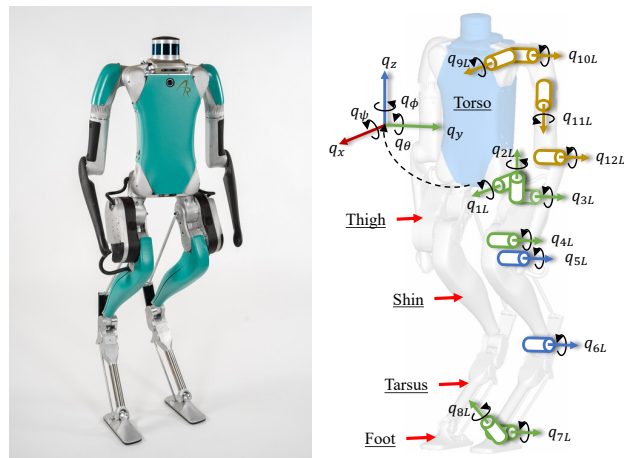


Fig. 2: Digit robot (left) and its kinematic tree structure (right). The floating base coordinate is located at the pelvis link. Only left leg and left arm joints are shown. Green represents actuated leg joints, blue represents passive leg joints, and orange represents arm joints.

success of implementing the proposed locomotion policy to work with Digit in real-world experiments.

The remainder of the paper is organized as follows. Section II presents details of the Digit robot. Section III introduces the proposed framework, including the high-level learning module, the low-level model-based controllers, the training, and the execution of the overall framework. Section IV includes details about the controller implementation on hardware. A series of experiments are included in Section V, emphasizing the locomotion robustness in terms of disturbance rejection, stable walking gaits on uneven terrains, and effortless sim-to-real transfer. Section VI concludes the paper and discusses future work.

II. KINEMATIC MODEL OF DIGIT ROBOT

This section will briefly introduce the Digit robot and its kinematic model and notations used in the following sections.

A. Robot Hardware

Digit is a versatile bipedal robot designed and built by Agility Robotics [17]. The robot has an integrated perception system, 20 actuated joints, 1 IMU, and 30 degrees of freedom (DoF) that allows robust and dynamic locomotion. The robot design is based on its predecessor Cassie, whose legs morphology is derived from an Ostrich-like bird. Hence, the location of the knee and ankle is not immediately evident from the robot's appearance. Fig. 2 shows the Digit robot with a description of the body links and its kinematic structure. The total weight of the robot is 48 kg. The robot's torso weighs 15 kg and contains Digit's onboard computer, power source, and vision sensors. Vision sensors include a LiDAR, an RGB camera, three monochrome depth cameras, and an RGB-depth camera.

B. Kinematic Model and Notations

Each arm of Digit has four joints for basic manipulation tasks. Each leg has eight joints, from which four are directly

actuated by electrical motors (hip yaw, hip roll, hip pitch, knee), two are indirectly actuated by electrical motors (toeA, toeB) and correspond to the actuated joints of the foot (foot pitch, foot roll). The addition of the foot pitch and foot roll joints improves the ability to balance stably on a wide variety of terrains. The remaining two joints (ankle and shin) are passive, and they are connected via specially designed leaf-spring four-bar linkages for additional compliance.

As shown in Fig. 2, the variables q_x, q_y, q_z denote the Cartesian position of the robot’s pelvis, while the torso orientation is represented by the x-y-z Euler angles (roll, pitch, yaw) as q_y, q_q, q_f . Then, the coordinate of the floating base coordinate system at the pelvis is denoted by

$$\mathbf{q}^b = [q_x, q_y, q_z, q_y, q_q, q_f]^T. \quad (1)$$

The actuated joints are denoted by the vector given as,

$$\mathbf{q}^a = [q_{1L}, q_{2L}, q_{3L}, q_{4L}, q_{7L}, q_{8L}, q_{9L}, q_{10L}, q_{11L}, q_{12L}, q_{1L}, q_{2L}, q_{3L}, q_{4L}, q_{7L}, q_{8L}, q_{9L}, q_{10L}, q_{11L}, q_{12L}]^T, \quad (2)$$

where the joints q_1 - q_4 correspond to the leg’s actuated joints (i.e. hip roll, hip yaw, hip pitch, knee), joints q_7, q_8 correspond to the foot pitch and foot roll, and joints q_9 - q_{12} correspond to the arm’s actuated joints (i.e. shoulder roll, shoulder pitch, shoulder yaw, elbow). The passive joints are denoted by the vector:

$$\mathbf{q}^p = [q_{5L}, q_{6L}, q_{5R}, q_{6R}]^T, \quad (3)$$

where q_5, q_6 correspond to the shin and tarsus joints. Hence, the generalized coordinates of Digit robot are denoted by:

$$\mathbf{q} = [\mathbf{q}^b, \mathbf{q}^a, \mathbf{q}^p]^T. \quad (4)$$

In addition, the robot has four end-effectors, which are the left/right feet and fists. We define the Cartesian position of any of these end-effectors as:

$$\mathbf{x}_{ee}(\mathbf{q}) = [x_{ee}, y_{ee}, z_{ee}]^T. \quad (5)$$

The cartesian position of the center of mass (CoM) is denoted by:

$$\mathbf{x}_{CoM}(\mathbf{q}) = [x_{CoM}, y_{CoM}, z_{CoM}]^T. \quad (6)$$

The current position of the robot’s feet and CoM can be determined by applying Forward Kinematics (FK). In particular, we use the open-source package FROST [18] to obtain symbolic expressions for $\mathbf{x}_{footL}(\mathbf{q})$, $\mathbf{x}_{footR}(\mathbf{q})$, and $\mathbf{x}_{CoM}(\mathbf{q})$. We created a URDF model of Digit based on the XML model provided by Agility Robotics.

III. LEARNING APPROACH

In this section, we build upon our previous work proposed in [11], [13] to implement a cascade-structure learning framework that realizes stable and robust walking gaits for real 3D bipedal robots. The specific design ensures the successful transfer of learned policies in simulation to robot hardware with minimal turning.

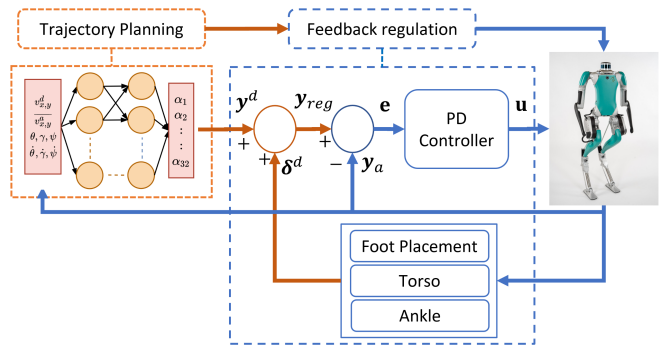


Fig. 3: Overall structure of the proposed Trajectory-based RL framework. The trajectory planning phase is done by the neural network policy, while the feedback regulation block uses the robot’s sensor information to improve the stability of the walking gait and the velocity tracking performance.

A. Overview of the Learning Framework

As shown in Fig. 3, the proposed framework presents a cascade decoupled structure to tackle the 3D walking problem through its two main components: neural-network trajectory planning and feedback regulation.

The neural network trajectory planner uses a reduced-dimensional representation of the robot states to compute a set of coefficients \mathcal{A} that parameterize the actuated joints’ reference trajectories \mathbf{y}^d through 5th-order Bézier Polynomials. The detailed structure of the neural network is provided in Section III-C. Since the forward propagation of the neural network could take significant time when compared with the running frequency of the low-level controller (1 kHz), the trajectory planning phase runs at a lower frequency (250 Hz). This consideration is especially relevant for the implementation of the learned policy on the real robot. More importantly, it opens the door to the real-time implementation of new learning structures that may be computationally expensive in both simulation and hardware.

The feedback regulation uses the robot’s pelvis velocities and torso orientation to modify the joint reference trajectories through d^d . It compensates for the uncertainty in the robot’s model and the environment, improving the robustness of the walking gait. To track the compensated reference joint trajectories \mathbf{y}^{reg} , the joint-level PD controllers compute the torque \mathbf{u} , which is applied directly to the actuated joints of the robot. The compensations applied during the feedback regulation are discussed in detail in Section IV-B. The low-level feedback regulation is a key component of our control structure to close the sim-to-real gap, enabling the successful transfer of the learned policy to hardware without the need for exhaustive tuning, dynamic randomization, or curriculum learning.

B. Dimensionality Reduction of the State and Action Space

By applying physical insights from the walking motion, we can significantly reduce the number of inputs and outputs of the NN. Unlike other end-to-end RL frameworks, we

do not use all the available states of the robot to feed the NN. Instead, we select states that provide insightful information such as the torso orientation (q_y, q_q, q_f) , pelvis linear velocity $(\dot{q}_x, \dot{q}_y, \dot{q}_z)$, torso angular velocity $(\dot{q}_y, \dot{q}_q, \dot{q}_f)$, and desired walking direction, which is encoded by the desired forward and lateral walking velocity $(\dot{q}_x^d, \dot{q}_y^d)$.

To reduce the dimension of the action space, we will keep the arms joints fixed during the walking motion, the stance foot passive, and the swing foot parallel to the ground during the walking gait. Without the need for determining reference trajectories for arm and foot joints, we reduce the number of reference trajectories needed to be computed by the NN to eight, with each leg having three hip joints and one knee joint. In addition, we impose a symmetry condition between the right and left stance during the walking gait. Therefore, given the set of coefficients for the right stance $a^R \in \mathbb{R}^{(M+1) \times N}$, where $M = 5$ is the degree of the Bézier polynomials and $N = 8$ is the number of reference trajectories, we can obtain the set of coefficients for the left stance $a^L \in \mathbb{R}^{(M+1) \times N}$ by the symmetry condition:

$$a^L = \mathbf{T}a^R, \quad (7)$$

where $\mathbf{T} \in \mathbb{R}^{N \times N}$ is a sparse transformation matrix that represents the symmetry between the joints of the right and left legs of the robot.

To further reduce the action space and encourage the smoothness of the control actions after the ground impact, we enforce that at the beginning of every step, the initial point of the Bézier polynomial coincides with the current position of the robot's joints. That is, for each joint i with Bézier coefficients $a_i^R \in \mathbb{R}^{M+1}$, we have

$$a_i^R[0] = q_i(t(0)), \quad t \in [0, 1], \quad (8)$$

where t is the time-based phase variable used to parameterize the Bézier Polynomials, denoted by $t(t) = \frac{t-t}{T_{step}}$, where t is the time at the beginning of the step, and $T_{step} = 0.5[s]$ is the time duration of one walking step. In this work, the T_{step} is kept fixed during the training and evaluation of the policy.

Finally, we enforce the position of the actuated joints to be the same at the end of the right stance and the beginning of the left stance. This encourages continuity in the joint position trajectories after switching the stance foot. Given the properties of the Bézier polynomials, this condition can be enforced through $a_i^R[M] = a_i^L[0]$.

This means two out of the six Bézier coefficients for each joint can be obtained through the above conditions. Therefore, we only need to find the remaining four coefficients for each of the eight reference trajectories, which results in an action space of dimension 32.

C. Neural Network Structure

Given the considerations presented in Section III-B, the number of inputs for the NN is 10, and the number of outputs is 32. We choose the number of hidden layers of the NN to be 4, each with 32 neurons. The activation function for the hidden layers is ReLU, and the activation function for the output layer is sigmoid. Finally, we scale the output of the

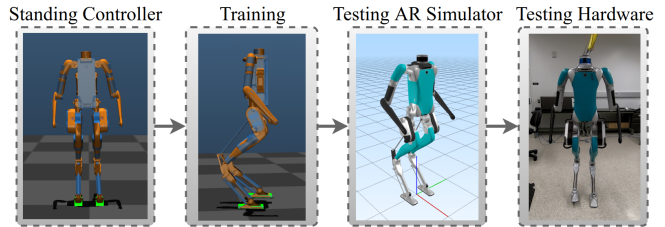


Fig. 4: Pipeline of the proposed learning framework. A standing controller is included within the pipeline of the learning process to obtain a feasible set of initial states for the policy training using a customized Mujoco environment. Then, the trained policy is tested in a more realistic real-time simulator. Finally, it is tested on hardware.

NN within a range of admissible motion for the robot's joints. In particular, we use the convex hull property of Bézier polynomials to translate the joint limits to the corresponding coefficients limits. Given the number of inputs and outputs of the NN, the number of trainable parameters is 4576. To the best of our knowledge, this is the smallest NN implemented in hardware to realize 3D walking locomotion.

D. Walking Policy Learning Pipeline

Since the main purpose of this work is to implement a learning framework that realizes a policy that is transferable to the actual robot, we need to create a pipeline for the training process in simulation that renders working conditions as close as possible to the real hardware. A diagram of the training pipeline is shown in Fig. 4.

The initial state for each training episode is chosen randomly from a pool of initial states whose kinematics and dynamics are feasible to be implemented on the real robot. To achieve this, we implement a standing controller that allows the robot to start up from an arbitrary resting position when the robot is hanging up from the lab's crane. We tested the controller successfully in hardware and used it to replicate the standing process in simulation. More details about the standing controller are provided in Section IV-A.

After the standing controller is activated and the robot is able to stand and balance steadily in simulation, we capture the state of the robot and save it in a pool of initial states. We repeat the process 40 times to create a diverse enough set of initial states that we can use for training our controller. It is important to denote that this process does not provide a fixed initial state for the walking gait but a whole set of different initial configurations that encourage the randomness of the initial state during the training while providing feasible and safe starting conditions for the walking gait. Moreover, this process does not provide any reference trajectories for the walking gait that could bias the learning process to specific or predefined walking gaits. Different from other RL approaches to bipedal locomotion, our method learns walking gaits from scratch without the need for predefined reference trajectories or imitation learning.

Given the pool of initial states, we use a customized

environment using MuJoCo [19] to start the training process using the Evolution Strategies (ES) algorithm. Our choice of ES over other gradient-based algorithms is inspired by the results in [20], where it is shown that the performance of ES can be comparable with gradient-based RL algorithms, especially when the number of time steps in an episode is long, the actions have long-lasting effects, or if no good value function estimate is available. All these conditions are present in the task of learning bipedal locomotion gaits from scratch. However, we denote that the proposed learning pipeline is very general, and any other learning algorithm that handles continuous action spaces could be used.

When the training is finished, we test the trained policy in the Agility Robotics proprietary simulation software. This simulation software provides a more realistic representation of the robot’s dynamic behavior since it runs in real-time and shares the same features with the hardware, such as low-level API, communication delay, and dynamic parameters of the robot.

Finally, once we verify the trained policy is deployed safely in the AR simulation, we proceed to test it in the hardware and verify its performance to do additional tuning of the low-level controller gains if needed. A sequence of the whole learning process and transference to real hardware can be seen in the accompanying video submission.

E. Reward Function Design

Following the work in [13], we use a reward function of the form:

$$\mathbf{r} = \mathbf{w}^T [r_{v_x}, r_{v_y}, r_h, r_{CoM}, r_{ang}, r_{angvel}, r_u, r_{fd}]^T. \quad (9)$$

where \mathbf{w} is a vector of weights corresponding to each component. This reward function encourages forward and lateral velocity tracking (through r_{v_x}, r_{v_y}), safety (through r_h, r_{CoM}, r_{fd}), energy efficiency (through r_u) and natural walking gaits (through r_{ang}, r_{angvel}). The episode length is 10000 simulation steps, which are equivalent to 5 seconds, and it has an early termination if any of the following conditions are violated:

$$\begin{aligned} |jq_y| < 0.5, \quad |jq_x| < 0.5, \quad |jq_f| < 0.5, \\ |\dot{q}_y| < 2, \quad |\dot{q}_x| < 2, \quad |\dot{q}_f| < 2, \\ 0.8 < q_z < 1.2, \quad \Delta_f < 0.05, \end{aligned} \quad (10)$$

where q_z is the height of the robot’s pelvis and Δ_f is the distance between the feet.

IV. CONTROL IMPLEMENTATION ON HARDWARE

In this section, we provide details about the implementation of the high-level and low-level controllers of the control-learning framework introduced in Section III. Moreover, we provide details of the architecture of our controller and its integration with Digit’s low-level API and communication system. This architecture is presented in Fig. 5.

The Agility Robotics low-level API streams data using the UDP protocol and enables the user to access the low-level sensor data and give commands directly to the motor drives. To keep the connection of the low-level API active

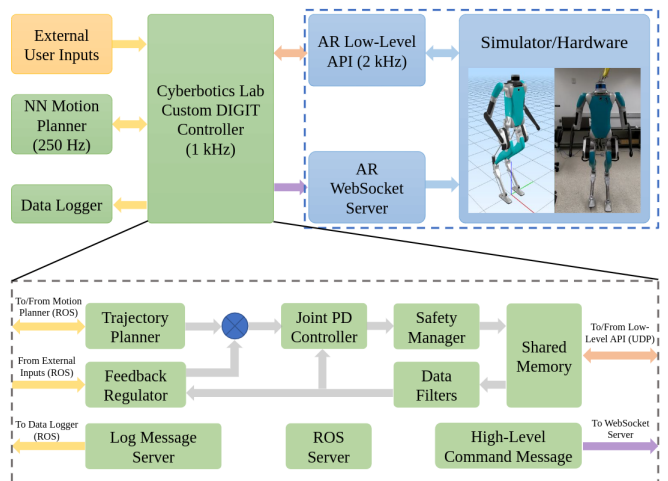


Fig. 5: Our controller architecture allows asynchronous operation of the high-level and low-level controllers. Moreover, it enables fast and reliable communication between different control layers, external user inputs, and the Digit simulator/hardware. Note that the same controller architecture can be used for different tasks, e.g., walking, standing, crouching.

to receive sensor data, commands must be sent periodically. Once the client is connected and sending commands, torque control must be activated by requesting transition to low-level API operation using JSON messages sent through Websocket protocol.

In addition, we use the Robot Operating System (ROS) to manage the communication between different components in our system. This also adds significant flexibility to our controller structure to include nodes for additional tasks such as logging information, capturing external command inputs, and accessing Digit’s perception system.

Since the low-level API needs to run at a very high frequency (2 kHz), we use shared memory to enable fast communication between the low-level API and our custom main control code (1 kHz). The low-level API reads the sensor measurements and writes them into the shared memory. It also reads from the shared memory the torque information and velocity commands written by the main control code and sends the commands to the motors.

On the other hand, the main control code manages the integration and synchronization of all the components in the overall control structure. It reads the sensor information from the shared memory and publishes it in ROS topics to make it available for the other components of the systems, such as the high-level planner. In this work, the high-level planner is the trained NN policy, which takes the information from the corresponding ROS topics to compute the output (coefficients of the Bézier Polynomials) at a frequency of 250Hz. Then, the high-level planner publishes this information in its respective ROS topics to make it available for the main control code.

The main control code reads the Bézier coefficients published by the high-level planner and uses them to compute the

corresponding Bézier Polynomials that become the reference trajectories for the robot's joints. In addition, the main control code uses the sensor feedback information shared by the low-level API to compute the regulations needed to compensate for the model mismatch between the simulation and hardware. The integration of these low-level feedback regulations into the learning framework is a key part that makes the controller structure very robust to uncertainties in the model and makes possible an almost zero effort transfer from the policy learned in simulation into the real hardware. The detailed structure of the regulations used in the low-level controller is discussed in Section IV-B.

A. Standing Controller

The standing controller implemented for Digit is based on the standing controller for Cassie in [21]. To ensure the robot keeps the balance, the standing controller uses intuitive and straightforward regulations based on the torso orientation and the position of the CoM with respect to the feet. That is, keeping the robot CoM within the support polygon while both feet are flat on the ground. The CoM position in the y-axis is adjusted by varying the legs' length, while the CoM in the x-axis is regulated by controlling the pitch angle of the feet.

In our learning framework, the standing controller is used to bring the robot to a stable configuration after initialization in arbitrary standing positions. This process is repeated several times in simulation until we obtain a diverse enough pool of stable configurations that are realistic and feasible to be implemented in hardware. This set of feasible initial configurations is then used to choose the initial states during the training process randomly.

B. Feedback Regulations

The feedback regulation module in our controller structure is a key component of the framework as it allows the controller to compensate the trajectories obtained from the high-level planner to adapt it to unknown disturbances such as the mismatch between the simulation model and real robot, and environmental factors like external disturbances or challenging terrains that the policy has not experimented in simulation. These regulations are divided into three main groups: foot placement, torso, and foot regulations.

Foot placement regulation controller has been widely used in 3D bipedal walking robots to improve the speed tracking and the stability and robustness of the walking gait [21]–[23]. Longitudinal speed regulation, defined by (13), sets a target offset in the swing hip pitch joint, whereas lateral speed regulation (11) does the same for the swing hip roll angle. Direction regulation (12) adds an offset to the yaw hip angle to keep the torso yaw orientation at the desired angle.

$$d_{q_{1i}} = S_y(t(K_{p_y}(\dot{q}_y \quad \dot{q}_y^d) + K_{d_y}(\dot{q}_y \quad \dot{q}_y^{ls})) + b_y), \quad (11)$$

$$d_{q_{2i}} = t(q_f \quad q_f^d), \quad (12)$$

$$d_{q_{3i}} = (t(K_{p_x}(\dot{q}_x \quad \dot{q}_x^d) + K_{d_x}(\dot{q}_x \quad \dot{q}_x^{ls})) + b_x), \quad (13)$$

where $i \in \{L, R\}$ depends on which foot is the swing foot, $S_y = 1$ if $i = L$, $S_y = -1$ if $i = R$, \dot{q}_x , \dot{q}_y are the longitudinal and lateral speeds of the robot, \dot{q}_x^{ls} , \dot{q}_y^{ls} are the speeds at the end of the previous step, \dot{q}_x^d , \dot{q}_y^d are the reference speeds, and K_{p_x} , K_{d_x} , K_{p_y} , K_{d_y} are the proportional and derivative gains.

The phase variable t is used to smooth the regulation at the beginning of each walking step and reduce torque overshoots. The term b is the output of an additional PI controller used to compensate for the accumulated error in the velocity and prevent the robot from drifting towards a non-desired direction. Based on our experiments, the inclusion of b is key in the successful sim-to-real transfer of our controller.

Torso regulation is applied to keep the torso in an upright position, which is desired for a stable walking gait. Assuming that the robot has a rigid body torso, simple PD controllers defined by (14) and (15) can be applied respectively to the hip roll q_{1i} and hip pitch q_{3i} angle of the stance leg:

$$u_{q_{1i}} = (K_{p_{roll}}(q_y \quad q_y^d) + K_{d_{roll}}(\dot{q}_y \quad \dot{q}_y^d)), \quad (14)$$

$$u_{q_{3i}} = S_q(K_{p_{pitch}}(q_q \quad q_q^d) + K_{d_{pitch}}(\dot{q}_q \quad \dot{q}_q^d)), \quad (15)$$

where $i \in \{L, R\}$ depends on which foot is the stance foot, $S_q = 1$ if $i = L$, $S_q = -1$ if $i = R$, q_f and q_q are the torso roll and pitch angles, and $K_{p_{roll}}$, $K_{d_{roll}}$, $K_{p_{pitch}}$, $K_{d_{pitch}}$ are manually tuned gains.

Foot orientation regulation is applied to keep the swing foot flat during the swinging phase to ensure a proper landing of the foot on the ground. Since Digit has 2-DoF actuated feet, one regulation is needed for each of the roll and pitch angles. By using forward kinematics, the swing foot roll and pitch regulation are determined by

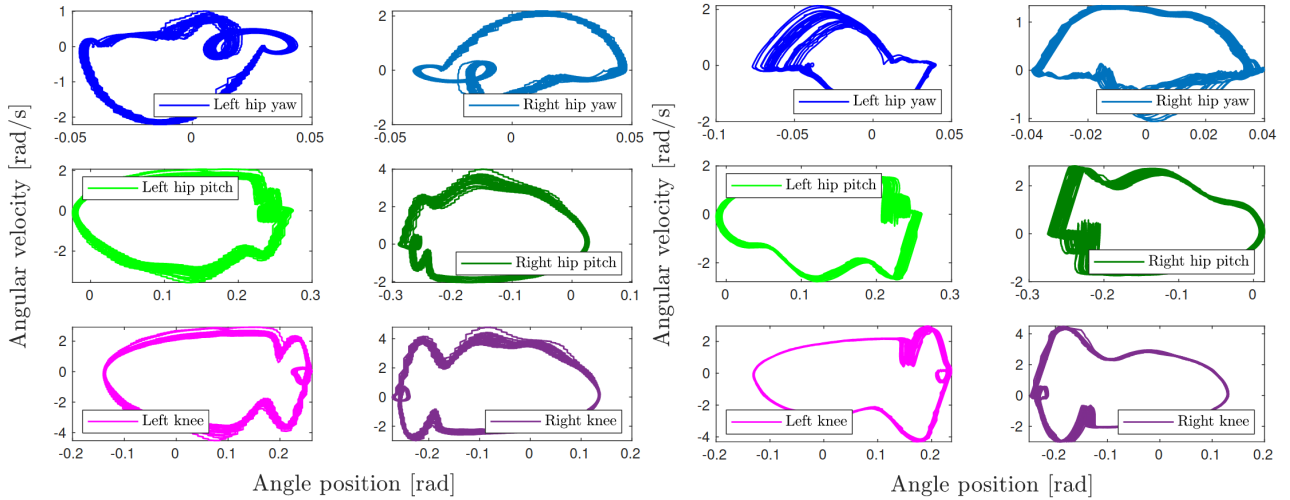
$$q_{7i}^d = q_y + q_{1i} + S_f(21 \text{ deg}) \quad (16)$$

$$q_{8i}^d = q_q + q_{3i} + S_f(6 \text{ deg}), \quad (17)$$

where $i \in \{L, R\}$ depends on which foot is the swing foot, $S_f = 1$ if $i = L$, $S_f = -1$ if $i = R$, and q_{7i}^d , q_{8i}^d are the desired pitch and roll foot angle. Moreover, the stance foot is kept passive during the stance phase, which helps to maintain the stability of the walking gait, especially for soft or irregular surfaces.

V. EXPERIMENTAL RESULTS

In this section, we validate the learning framework and controller structure presented in this paper through simulation and real hardware experiments on the Digit robot. We prove that our controller structure allows the transference of walking policies learned in simulation to the real hardware with minimal tuning and enhanced robustness. To the best of our knowledge, this is the first time that a policy learned in simulation is successfully transferred to hardware to realize stable and robust dynamic walking gaits for the Digit robot. In addition, we show that the improved controller structure presented in this work is robust enough to mitigate the uncertainty in the robot's dynamics caused by the mismatch between the simulation and real hardware. Finally, by thorough experiments on hardware, we show the robustness of the learned controller against external disturbances applied to



(a) Limit walking cycle of trained policy in simulation.

(b) Limit walking cycle of trained policy in real robot.

Fig. 6: Comparing limit walking cycles between the simulation trails and real-robot tests: (i) For the same joint, simulated motion and real robot motion share similar torque limits. (ii) For all hip joints, the regulation term modifies the actual motion leading to the slight difference of the limit walking cycles. (iii) As the regulation is not involved in any of the knee joints, the limit walking cycles are almost identical.

the robot as well as its capability to adapt to various terrains without the need for training for such challenging scenarios.

A. Sim-to-real Transfer and Stability of the Walking Gait

We transferred the learned policy trained in simulation into the real robot, and we empirically tested the stability of the walking gait by analyzing the walking limit cycle described by the robot’s joints during the walking motion. Fig. 6a shows the limit cycle described by some of the robot’s joints while the robot is walking in place for about 1 minute in simulation, while Fig. 6b shows the limit cycles from the real experiment. The convergence of the walking limit cycle to a stable periodic orbit demonstrates that the walking gait is stable and symmetric, meaning that the policy is transferred successfully from simulation to hardware.

B. Robustness to Disturbances

To evaluate the robustness of the policy against disturbances, external forces are applied to the robot in different directions. Fig. 1 and Fig. 7 show respectively the transition of the robot recovering from a push in the lateral direction and the velocity profile of the robot during the disturbance. In addition, Fig. 8 shows the limit cycles of the robot’s joints when the disturbance is applied, where it can be seen that the controller can recover effectively from the push as the joint limit cycles return to a stable periodic orbit after the disturbance.

C. Robustness on Different Terrains

To further evaluate the robustness of the learned control policy, we make Digit walk in various terrains with different difficulty levels. These terrains include flat vinyl ground, mulch, flat rubber ground, and irregular rubber terrain. The controller is able to adapt to any of these terrains while keeping a stable and robust walking gait. Snapshots of the

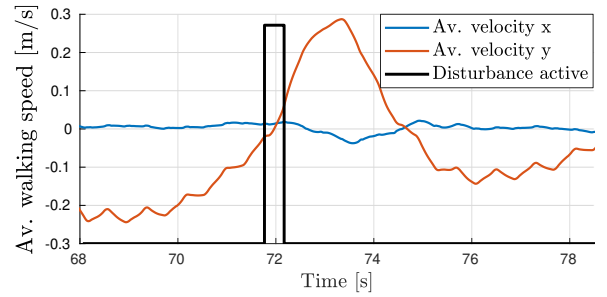


Fig. 7: Digit recovering from an external disturbance

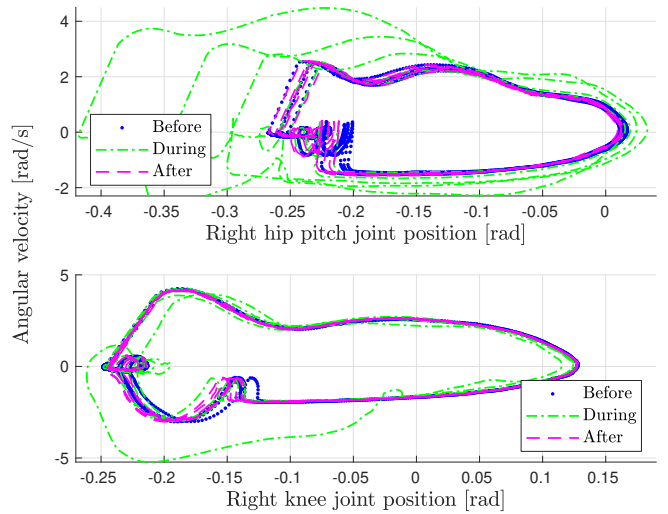


Fig. 8: Disturbance rejection of learned policy. The walking limit cycles described before, during, and after the lateral push show that the controller can effectively handle large external disturbance forces.

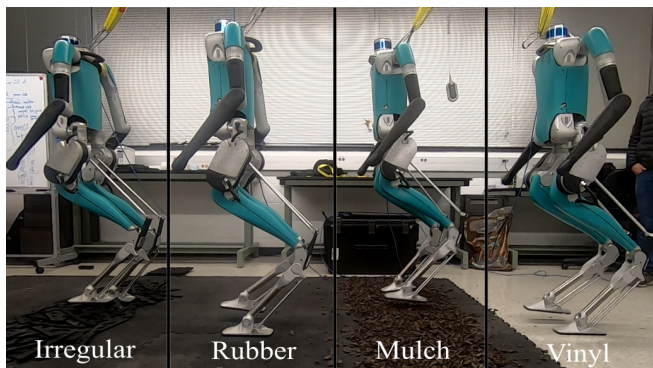


Fig. 9: Digit walking on different terrains. From right to left, we have vinyl, mulch, flat rubber, and irregular rubber terrain. The stance foot is kept passive during the stance phase to allow the foot to adapt easily to different terrains.

walking gait over the different terrains are shown in Fig. 9, and the complete motion for all the tests performed can be found in the accompanying video¹.

VI. CONCLUSIONS

This paper presents a framework for learning robust bipedal locomotion policies that can be transferred to real hardware with minimal tuning. By combining a sample efficient learning structure with intuitive but powerful feedback regulations in a cascade structure, we decouple the learning problem into two stages that work at a different frequency to facilitate the implementation of the controller in the real hardware. While the trajectory planning stage is handled by the neural network to produce reference trajectories for the actuated joints of the robot at a lower frequency (250 Hz), the feedback regulation stage runs at a higher frequency (1 kHz) using the sensor feedback to realize compensations of the reference trajectories that guarantee the stability of the walking limit cycle. The result is policies learned from scratch that are transferred successfully to hardware with minimal tuning. The controller is exhaustively tested in hardware, demonstrating stable walking gaits that are robust to external disturbances and challenging terrain without being trained under those conditions.

Future work will focus on leveraging the versatility of the proposed framework to include arm motion for Digit as part of the learning outcome and testing the learned policy in more challenging and unstructured environments.

REFERENCES

- [1] M. Shafiee-Ashtiani, A. Yousefi-Koma, and M. Shariat-Panahi, "Robust bipedal locomotion control based on model predictive control and divergent component of motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3505–3510.
- [2] H. Chen, B. Wang, Z. Hong, C. Shen, P. M. Wensing, and W. Zhang, "Underactuated motion planning and control for jumping with wheeled-bipedal robots," *IEEE Robotics and Automation Letters*, 2020.
- [3] Q. Nguyen and K. Sreenath, "Optimal robust safety-critical control for dynamic robotics," *arXiv preprint arXiv:2005.07284*, 2020.
- [4] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3D dynamic walking with underactuated humanoid robots: a direct collocation framework for optimizing hybrid zero dynamics," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. Stockholm, Sweden: IEEE, May 2016, pp. 1447–1454.
- [5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
- [6] C. Yang, K. Yuan, W. Merkt, T. Komura, S. Vijayakumar, and Z. Li, "Learning whole-body motor skills for humanoids," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 270–276.
- [7] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne, "Allsteps: Curriculum-driven learning of stepping stone skills," *arXiv preprint arXiv:2005.04323*, 2020.
- [8] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *IEEE International Conference on Robotics and Automation*. IEEE, 2004.
- [9] J. Morimoto, J. Nakanishi, G. Endo, G. Cheng, C. G. Atkeson, and G. Zeglin, "Poincare-map-based reinforcement learning for biped walking," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005.
- [10] X. Da, R. Hartley, and J. W. Grizzle, "Supervised learning for stabilizing underactuated bipedal robot locomotion, with outdoor experiments on the wave field," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3476–3483.
- [11] G. A. Castillo, B. Weng, A. Hereid, Z. Wang, and W. Zhang, "Reinforcement learning meets hybrid zero dynamics: A case study for rabbit," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 284–290.
- [12] T. Li, H. Geyer, C. G. Atkeson, and A. Rai, "Using deep reinforcement learning to learn high-level policies on the atrias biped," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 263–269.
- [13] G. A. Castillo, B. Weng, W. Zhang, and A. Hereid, "Hybrid zero dynamics inspired feedback control policy design for 3d bipedal locomotion using reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8746–8752.
- [14] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, 2020.
- [15] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," *arXiv preprint arXiv:1903.01390*, 2019.
- [16] J. Siekmann, Y. Godse, A. Fern, and J. Hurst, "Sim-to-real learning of all common bipedal gaits via periodic reward composition," *arXiv preprint arXiv:2011.01387*, 2020.
- [17] J. Hurst, "Building Robots That Can Go Where We Go," *IEEE Spectrum: Technology, Engineering, and Science News*, Feb. 2019. [Online]. Available: <https://spectrum.ieee.org/robotics/humanoids/building-robots-that-can-go-where-we-go>
- [18] A. Hereid and A. D. Ames, "FROST: fast robot optimization and simulation toolkit," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*. Vancouver, BC, Canada: IEEE/RSJ, Sept. 2017, pp. 719–726.
- [19] 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*, Oct. 2012, pp. 5026–5033.
- [20] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.
- [21] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle, "Feedback control of a Cassie bipedal robot: walking, standing, and riding a segway," *American Control Conference (ACC)*, 2019.
- [22] X. Da, O. Harib, R. Hartley, B. Griffin, and J. W. Grizzle, "From 2D design of underactuated bipedal gaits to 3D implementation: Walking with speed tracking," *IEEE Access*, vol. 4, pp. 3469–3478, 2016.
- [23] S. Rezazadeh, C. Hubicki, M. Jones, A. Peekema, J. Van Why, A. Abate, and J. Hurst, "Spring-mass walking with Atrias in 3D: robust gait control spanning zero to 4.3 kph on a heavily underactuated bipedal robot," in *ASME 2015 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers, 2015.

¹<https://www.youtube.com/watch?v=j8KbW-a9dbw>