# Linear Policies are Sufficient to Enable Low-Cost Quadrupedal Robots to Traverse Rough Terrain

Maurice Rahme[1] Ian Abraham[2] Matthew L. Elwin[1] Todd D. Murphey[1]

***Abstract*— The availability of inexpensive 3D-printed quadrupedal robots motivates the development of learning-based methods compatible with low-cost embedded processors and position-controlled hobby servos. In this work, we show that a linear policy is sufficient to modulate an open-loop trajectory generator, enabling a quadruped to walk over rough, unknown terrain, with limited sensing. The policy is trained in simulation using randomized terrain and dynamics and directly deployed on the robot. We show that the resulting controller can be implemented on resource-constrained systems. We demonstrate the results by deploying the policy on the OpenQuadruped, an open-source 3D-printed robot equipped with hobby servos and an embedded microprocessor.**

Fig. 1. **Legged locomotion over complex terrain.** Example of a trajectory taken by the OpenQuadruped robot using sim-to-real transfer of a linear policy on difficult terrain.

## I. Introduction

Current high-end legged robots (e.g., [1]–[4]) are cost-prohibitive for many potential users such as schools and small businesses. Low-cost hobbyist robots such as Stanford Pupper [5] and OpenQuadruped [6], increase accessibility but have limited features because of their 3D printed parts, low-powered microcontrollers, and hobby servo motors. Unfortunately, algorithmic development for these quadrupedal robots often requires powerful computers and graphics cards to both train and deploy the controllers and deep neural networks that enable robots to exhibit complex behaviors like walking over rough and uneven terrain or recovering from falls [7]–[10]. Generally, the graphics cards used to execute complex algorithms and train neural networks (e.g., a GeForce RTX 2080 Ti retailing for $1200) exceed the cost of a typical hobbyist robot (e.g., the OpenQuadruped at $600) [6], [11]. These robotic systems become even more inaccessible if we consider the required additional sensing and computation needed to map terrains [12] and execute high-frequency torque control [13].

If we want robotic systems to be more accessible, it is necessary to acknowledge that access to arbitrary amounts of compute is not always available, and develop light-weight algorithms that explicitly consider the constraints and limitations found with low-cost hobbyist robots [14]. Thus, this paper focuses on illustrating that learning simple linear policies can augment and improve locomotion skills for a class of low-cost robotic systems subject to sensing and control limitations. We show that such policies lead to sufficient locomotion over unknown rough and uneven terrain in a variety of simulated and experimental evaluations.

The current state-of-the-art in the field of quadrupedal walking robots has demonstrated the ability to traverse rough terrain using only proprioception and position-controlled legs [7]. Using the Policies Modulating Trajectory Generators (PMTG) architecture [15] to combine open-loop leg trajectories with feedback from a neural network, [7] showed that the ANYmal robot is capable of traversing a wide variety of difficult terrain without explicitly sensing it. Although the 12-layer deep neural network from [7]—trained in simulation using a multi-stage process with an observation history of 100 time-steps—can be successfully deployed on the more than $100,000 ANYmal robot, it is not conducive to implementation on a sub $1000 robot with only a single embedded microprocessor controlling 3D printed, hobby-servo actuated legs. Not only do these inexpensive systems have limited memory and processing power, but the hobby-servo motors used in these systems often lack the position feedback required by the network used in [7].

To bridge the gap between effective but resource-intensive walking methods and inexpensive hardware, we adopt an overall architecture similar to that of [7]. However, we show that Bezier curve gaits [3] can be used as a lower-order open-loop gait model while a learned linear policy (instead of a deep neural network) modulates the gaits for improved locomotion on rough terrain. The policy can be efficiently trained on a CPU and be deployed on the low-powered embedded systems commonly used with the low-cost robots (e.g. [16], [17]). Similar techniques have been shown to work

*This work is partly supported by the Northwestern Robotics program. Also, this material is based upon work supported by the National Science Foundation under Grant CNS 1837515. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the aforementioned institutions.

[1] Department of Mechanical Engineering, Northwestern University, Evanston, IL 60208

[2] Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213

Corresponding Email: mauricerahme2020@u.northwestern.edu, ia@andrew.cmu.edu

Multimedia can be found at https://sites.google.com/view/drgmbc/home.

on sloped terrain in simulation [18].

The primary contribution of this work is to show that walking over rough terrain using only inertial measurements is possible with a linear policy obtained through a direct policy search method [19] with domain randomization [20] on a CPU that can be readily deployed in the real world. By randomizing the terrain and robot dynamics during training, the policies can generalize to simulation errors, enabling direct transfer of policies to the real robot. Although such simple policies cannot be expected to match the performance of existing work [7], [9], they significantly improve the ability of inexpensive robots to traverse rough terrain without expensive computation, sensing, or actuation. Because these linear policies are effective and can be implemented on existing low-cost robotic platforms they are crucial to the overall development and accessibility of low-cost walking robots.

We also provide open-source plans for the OpenQuadruped robot used in our experiments [6], and an open-source simulation environment [21].

## II. Problem Statement

We pose the problem as a partially observable Markov decision process (POMDP) where we have a set of uncertain observations of the robot's state $o_t \in \mathcal{O}$, a reward function $r_t = \mathcal{R}(s_t, a_t)$ defining the quality of the locomotion task as a function of the state $s_t \in \mathcal{S}$, and an action space $a_t \in \mathcal{A}$ containing the set of control inputs to the system. A policy $a_t = \pi(o_t, \theta) = \theta^\top o_t$ parameterized by $\theta$ maps $o_t$ to $a_t$.[1] Given this formulation, the goal is to find the policy parameters $\theta$ that maximize the reward over a finite time horizon $T$ using only partial observations of the state $o_t$. More precisely,

$$\theta^\star = \arg\max_\theta \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right], \tag{1}$$

where $0 < \gamma \leq 1$ is a discount factor, $\theta^\star$ is the optimal policy parameters, and $\mathbb{E}$ is the expected value over the randomized parameters described in Section III.

In simulation, we have access to the full robot state, which we use to construct the reward function for training; however, full state information is unavailable on the real robot. Therefore, we train our policy in simulation using a partial observation of the state $o_t$ to mimic on-robot sensing.

### A. Reinforcement Learning for Gait Modulation

The problem statement (1) serves as a template for developing and learning policies that adapt open-loop gaits for legged locomotion. During each episode, the policy $\pi$ is applied to modulate and augment a gait generator. This gait generator (described subsequently) outputs body-relative foot placements which the robot follows using position control.

Let $\ell$ be a label for the quadruped's legs: FL (front-left), FR (front-right), BL (back-left), BR (back-right). An open loop gait is a one-dimensional closed parametric curve $\Gamma_\ell(S(t), \zeta, \beta)$ embedded in $\mathbb{R}^3$ and specifying the position of a foot in the frame of its corresponding hip. Here, $S(t) : \mathbb{R} \to [0, 2]$ is a cyclical phase variable, $\zeta \in \mathbb{R}^n$ are the directional control inputs, and $\beta \in \mathbb{R}^m$ contains the gait parameters (in this paper, $m = 2$). The control inputs $\zeta$ and gait parameters $\beta$ determine the shape of the gait trajectory. The controls enable the robot to move in any lateral direction and rotate about its central axis. Control inputs, parameters, and gaits are discussed in Section IV.

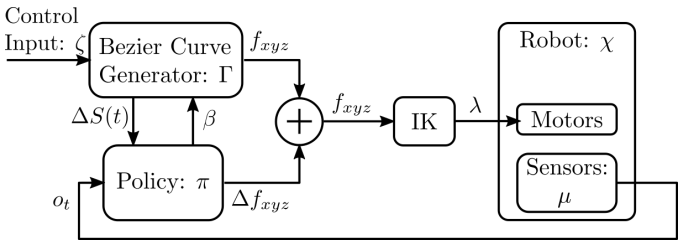


Fig. 2. **System diagram.** All controllers (including policy updates) run at 600 Hz. The gait generation and policy modulation are calculated simultaneously allowing fast generation of foot poses that are robust to limited sensing and rough terrain.

The policy $(\Delta f_{xyz}, \beta) = \pi(o_t, \theta)$ augments the foot positions via additive foot position residual $\Delta f_{xyz}$ and modifies the open-loop gait generator such that (1) is maximized using only partial observations $o_t$. Here $\Delta f_{xyz} = \begin{bmatrix}\Delta f_{xyz}^{FL} & \Delta f_{xyz}^{FR} & \Delta f_{xyz}^{BL} & \Delta f_{xyz}^{BR}\end{bmatrix}$ augments each foot position, with $\Delta f_{xyz}^{\ell} \in \mathbb{R}^3$, and $\beta = \{\psi, \delta\}$ where $\psi$ is the clearance height of the foot above the ground and $\delta$ is a virtual ground penetration depth.

The final foot positions are computed as a combination of the gait generator output and the policy residual:

$$f_{xyz} = \Gamma(S(t), \zeta, \beta) + \Delta f_{xyz}, \tag{2}$$

where $f_{xyz} \in \mathbb{R}^{12}$ is the stacked vector of each three-dimensional foot position that the robot tracks and $\Gamma$ contains $\Gamma_\ell$ for each leg. Given the foot positions, the robot computes the inverse kinematics to move its leg joints to the appropriate angles as shown in Fig. 2.

The challenge is to solve (1) in simulation using only the observations $o_t$ such that the resulting policy is suitable for use on a real robot subject to rough terrain and uncertain physical parameters. We present Gait Modulation with Bezier Curves (GMBC), summarized in Algorithm 1, which uses the Bezier curve gait generator for $\Gamma$ described in Section IV as a solution to (1). GMBC uses a simple policy search method to directly solve (1). We add domain randomization during the simulated training (see Section III) to improve the performance of linear policies over rough terrain and sim-to-real transfer.

## III. Domain Randomization

We employ two techniques that adapt (1) for improving the performance of sim-to-real transfer of gait modulating policies. Specifically, this approach merges the ideas

[1] In this work the actions map to inputs to a Bezier curve gait generator and robot foot position residuals.

**Algorithm 1** Gait Modulation with Bezier Curves (GMBC)

**Given:** Policy $\pi$ with parameter $\theta$, (Bezier) Curve Generator $\Gamma$, External motion command $\zeta$, robot sensor observations $o_t$, Leg phase $S(t)$

1: obtain gait modulation from $\pi$ with learned parameter $\theta$
2: $\Delta f_{xyz}, \beta = \pi(o_t, \theta)$
3: calculate (Bezier) gait foot placement
4: $f_{xyz} = \Gamma(S(t), \zeta, \beta)$
5: **return** $f_{xyz} + \Delta f_{xyz}$ to robot for IK joint control

from [20] to randomize not only the dynamics parameters of the simulated robot, but also the terrain that it traverses. We then solve (1) using a policy search method (augmented random search (ARS) [19]) to learn a linear policy as a function of observations subject to sampled variation in the dynamics and domain of the simulated episodes.[2] We describe the domain randomization procedure below.

We first modify the physical dynamic parameters that typically differ between simulation and reality, including the mass of each of the robot's links and the friction between the robot's foot and the ground. This distribution for which we train the gait modulating policy is defined as $\sigma_{\text{dyn}} \sim \mathbb{P}_{\text{dyn}}$, where $\sigma$ is the vector of randomized dynamics parameters and $\mathbb{P}_{\text{dyn}}$ is a probability distribution that treats each parameter independently (see Table I). At each training epoch, we sample from $\mathbb{P}_{\text{dyn}}$ and run a training iteration using the sampled dynamics parameters.

TABLE I
DOMAIN RANDOMIZED PARAMETERS

| **Randomized Parameter** $\sigma$ | **Range** |
|---|---|
| Base Mass (Gaussian) | 1.1kg $\pm$20% |
| Leg Link Masses (Gaussian) | 0.15kg $\pm$20% |
| Foot Friction (Uniform) | 0.8 to 1.5 |
| XYZ Mesh Magnitude (Uniform) | 0m to 0.08m |

We additionally randomize the terrain through which the legged robot moves (see Fig 3). We parameterize the terrain as a mesh of points sampled from $\sigma_{\text{terr}} \sim \mathbb{P}_{\text{terr}}$, where $\sigma_{\text{terr}}$ is the displacement on the uniform mesh grid, and $\mathbb{P}_{\text{terr}}$ is a bounded uniform distribution for which we vary the grid (see Table I). As with dynamics randomization, we sample terrain geometry from $\mathbb{P}_{\text{terr}}$ and train an iteration of ARS on the fixed sampled terrain.

We combine sources of domain randomization by leting $\sigma \sim \mathbb{P}$ be the joint distribution of $\mathbb{P}_{\text{dyn}}$ and $\mathbb{P}_{\text{terr}}$ over which we take the expectation in (1), where $\sigma$ is the joint domain sample. The training details are described in Algorithm 2 using ARS and GMBC from Algorithm 1. During training, the external commands $\zeta$ are held fixed and defined by the task (e.g., move forward at a fixed velocity).

In the next section, we describe the specifics of the Bezier curve gait generator $\Gamma$ used throughout this work.

[2]An episode being a single $T$ step run of the simulation with a fixed initial (randomly sampled) state.

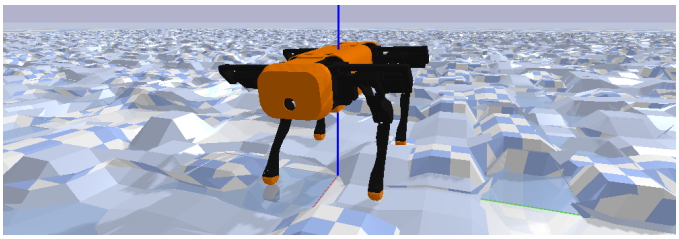

Fig. 3. **Simulation environment.** Illustration of the domain randomized terrain used for training in simulation. The terrain height varies up to 40% of robot height shown in the image [21], [22].

**Algorithm 2** RL Simulation training for DR-GMBC using Augmented Random Search [19]

**Initialize:** policy parameters $\theta_0$, domain distribution $\mathbb{P}$, reward function $\mathcal{R}$, GMBC (Algorithm 1), iteration number $k = 0$, construct ARS.

1: **while** training not converged **do**
2: $\sigma \sim \mathbb{P}$ sample domain parameters
3: ARS step of (1) with domain randomization+GMBC
4: $\theta_{k+1} \leftarrow \text{ARS}(\pi, \theta_k, \mathcal{R}, \sigma, k)$
5: $k \leftarrow k + 1$
6: **end while**
7: **return** $\theta_k$

## IV. EXTENDED BEZIER CURVES WITH MOTION COMMANDS

We use an extended and open-loop version of the Bezier curve gaits developed in [3], combining multiple 2D gaits into a single 3D gait that enables transverse, lateral, and yaw motion. The policy then modulates the gait parameters to adapt to uneven terrain and removes the need to sense impacts and forces at the foot.

A gait trajectory is a closed curve that a foot follows to execute a desired locomotion skill. The gait trajectory consists of two phases: swing and stance. During swing, the foot moves through the air to its next position. During stance, the foot contacts the ground and moves the robot using ground reaction forces. A gait is parameterized by a phase $S(t) \in [0, 2)$, which determines the foot's location along the trajectory. The leg is in stance for $S(t) \in [0, 1)$ and in swing for $S(t) \in [1, 2)$. A trajectory generator $\Upsilon(S(t), \tau)$ then maps phase and step length $\tau$ to a set of trajectories in $\mathbb{R}^2$. We use a trajectory consisting of a Bezier curve during swing and a sinusoidal curve during stance, see Section VI-A for more details.

The gait has as input three control inputs which direct the movement of the robot: $\zeta = \begin{bmatrix} \rho & \bar{\omega} & L_{\text{span}} \end{bmatrix}$, where $\rho \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ is the trajectory's rotation angle relative to the robot's forward direction, $\bar{\omega}$ is the robot's yaw velocity, and $L_{\text{span}}$ is half the stride length. Locomotion consists of a planar translation $f_{qz}^{tr} = \Upsilon(S(t), L_{\text{span}})$ and yaw trajectory $f_{qz}^{yaw} = \Upsilon(S(t), \bar{\omega})$. These trajectories also depend on curve parameters $\beta = \begin{bmatrix} \psi & \delta \end{bmatrix}$ (see Fig. 4 and Section VI-A for illustration).

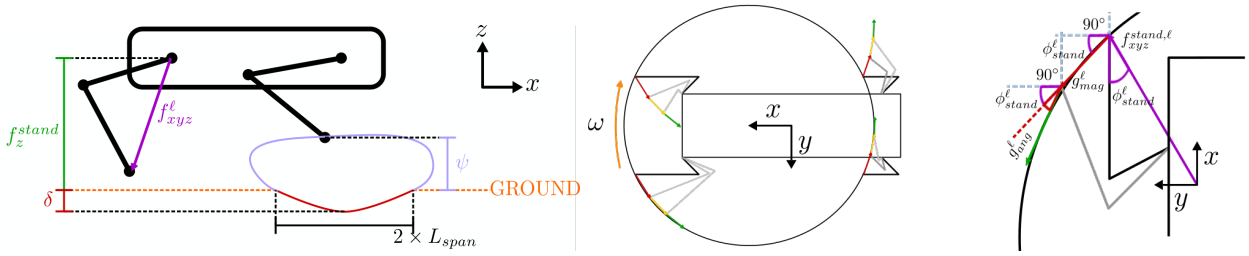


Fig. 4. **Bezir gait generator schematic.** Schematic of foot placement based on Bezier gait generator: $f_{xyz}$. Desired directional and velocity inputs are passed to the gait generator to modify the foot position based on a phase close $S(t)$ which executes the desired movements. The modulating policy augments the gait generator and its subsequent parameters in response to on-board inertial sensing to improve locomotion over unknown terrain.

Using the control inputs, the planar trajectories $f_{qz}^{tr}$ and $f_{qz}^{yaw}$ are converted into 3D foot-position trajectories $f_{xyz}^{tr}$ and $f_{xyz}^{yaw,\ell}$, where each leg $\ell$ has the same translational velocity but different yaw velocity. Finally, we transform the yaw and translational curves into a frame relative to each leg's rest position $f_{xyz}^{\text{stand}}$ to get the final foot trajectory for leg $\ell$:

$$f_{xyz}^{\ell} = f_{xyz}^{tr} + f_{xyz,\ell}^{yaw} + f_{xyz}^{\text{stand}}. \tag{3}$$

This scheme enables movements encompassing forward, lateral, and yaw commands, and straight-line motion which can extend to more complex motions. Here, $(f_q^{tr}, f_z^{tr})$ and $(f_q^{yaw}, f_z^{yaw})$ are the coordinates of $f_{qz}^{tr} \in \mathbb{R}^2$ and $f_{qz}^{yaw} \in \mathbb{R}^2$ respectively. Rotating planar trajectory $f_{qz}^{tr}$ by $\rho$ yields the 3D foot trajectory:

$$f_{xyz}^{tr} = \begin{bmatrix} f_q^{tr}\cos\rho & f_q^{tr}\sin\rho & f_z^{tr} \end{bmatrix}. \tag{4}$$

Yaw control of the legged robot is obtained by a four-wheel steered car model [23]. To trace a circular path, each foot path's angle must remain tangent to the rotational circle, as shown in Fig. 4. To make the robot yaw (i.e., rotate about its z-axis) each foot must move to position $g_{xyz}^{\ell}(t)$, where

$$g_{xyz}^{\ell}(t) = f_{xyz}(t-1) - f_{xyz}^{\text{stand}}. \tag{5}$$

The distance and angle of this step in the $xy$ plane are:

$$g_{\text{mag}}^{\ell} = \sqrt{(g_x^{\ell})^2 + (g_y^{\ell})^2} \text{ and } g_{\text{ang}}^{\ell} = \arctan\left(\frac{g_y^{\ell}}{g_x^{\ell}}\right),$$

where $g_x^{\ell}$ and $g_y^{\ell}$ are the $x$ and $y$ components of $g_{xyz}^{\ell}$. Each leg then translates at an angle

$$\phi_{\text{arc}}^{\ell} = g_{\text{ang}}^{\ell} + \phi_{\text{stand}}^{\ell} + \frac{\pi}{2}, \tag{6}$$

for a given yaw motion with the standing phase being equal to the following

$$\phi_{\text{stand}}^{\ell} = \begin{cases} \arctan\frac{f_y^{\text{stand}}}{f_x^{\text{stand}}} & \ell = \text{FR, BL} \\ -\arctan\frac{f_y^{\text{stand}}}{f_x^{\text{stand}}} & \ell = \text{FL, BR} \end{cases} \tag{7}$$

where $f_x^{\text{stand}}$ and $f_y^{\text{stand}}$ are the $x$ and $y$ components of $f_{xyz}^{\text{stand}}$. The leg rotation angle $\phi_{arc}^{\ell}$ provides the final yaw trajectory:

$$f_{xyz,\ell}^{yaw} = \begin{bmatrix} f_q^{yaw}\cos\phi_{arc}^{\ell} & f_q\sin\phi_{arc}^{\ell} & f_z^{yaw} \end{bmatrix} \tag{8}$$

The description of the gait generation procedure is outlined in Algorithm 3, and the generated Bezier curve is shown in Fig. 2.

**Algorithm 3** Bezier Curve Generator $\Gamma$ - Per Leg $\ell$

**Inputs:** $\zeta$

1: Map $t$ to foot phase $S_\ell(t)$ using (14)
2: $(f_q^{tr}, f_z^{tr}) = \Upsilon(S_\ell(t), L_{\text{span}})$
3: $(f_q^{yaw}, f_z^{yaw}) = \Upsilon(S_\ell(t), \bar{\omega})$
4: $f_{xyz}^{tr} = \begin{bmatrix} f_q^{tr}\cos\rho & f_q^{tr}\sin\rho & f_z^{tr} \end{bmatrix}$
5: $\phi_{\text{arc}}^{\ell} = g_{\text{ang}}^{\ell} + \phi_{\text{stand}}^{\ell} + \frac{\pi}{2}$
6: $f_{xyz,\ell}^{yaw} = \begin{bmatrix} f_q^{yaw}\cos\phi_{arc}^{\ell} & f_q\sin\phi_{arc}^{\ell} & f_z^{yaw} \end{bmatrix}$
7: $f_{xyz}^{\ell} = f_{xyz}^{tr} + f_{xyz,\ell}^{yaw} + f_{xyz}^{\text{stand}}$
8: **return** $f_{xyz}^{\ell}$ to the robot for joint actuation

## V. Results

We present several experiments to evaluate our approach for improving legged locomotion with a simple linear policy. We first describe the simulated training in more detail. We then evaluate the learned linear policies based on:

1) Generalization to randomized dynamics and terrain.
2) Improvement over open-loop gait generators with and without domain randomization.
3) Sim-to-real transfer performance on a real robot.

### A. Simulated Training

As mentioned in Algorithm 2, we use the augmented random search (ARS) method to train a policy to modulate GMBC (Algorithm 1) using the objective function defined in (1). To match the sensors on the real robot, we train the policy using an observation comprised of body roll $r$ and pitch $p$ angles relative to the gravity vector, the body 3-axis angular velocity $\omega$, the body 3-axis linear acceleration $\dot{v}$, and the internal phase of each foot $S_\ell(t)$, making $o_t = [r, p, \omega, \dot{v}, S(t)]^\top \in \mathbb{R}^{12}$. Training took 12 hours on a laptop with an Intel Core i7-8565U CPU.

A linear policy is chosen so that it can run in real-time on inexpensive hardware while improving the open-loop gaits as much as possible. The policy is defined as

$$a_t = \pi(o_t, \theta) = \theta^\top o_t,$$

where $\theta \in \mathbb{R}^{12\times 14}$. Here, the policy outputs the nominal clearance height and virtual ground penetration depth of the Bezier curve and residual foot displacements that are added to the output of the Bezier curve.

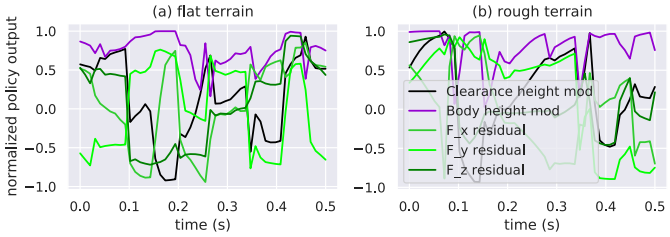


Fig. 5. **Policy output.** Example policy output during testing for a single leg. The simple linear policy successfully modulates robot using only inertial measurements.

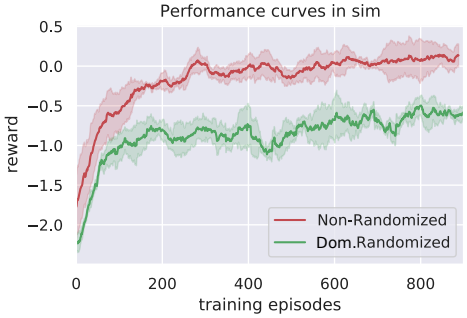


Fig. 6. **Performance curves in simulation.** Evaluated reward curves for simulation training with and without domain randomization. Despite the improved training performance without randomization, policies trained with domain randomization transfer better to unforeseen dynamics and terrain as shown in Table II.

To prevent infeasible foot positions and Bezier curve parameters, we center the policy output within the domain $\pi(o_t, \theta) \in [-1, 1]^{14}$. The output is then remapped to the acceptable range of Bezier curve parameters and a bounded domain of foot residuals. We use this combination of Bezier curve modulations and foot residuals to allow for large trajectory deviations while maintaining a continuous and feasible desired path. For each simulated example, the high-level motion commands are fixed at $L_{span} = 0.035$m, and $\rho = 0$. A proportional controller sets the yaw rate $\bar{\omega}$ to keep the robot's heading at zero. Fig. 5 provides an example of the output of the learned linear policy for a single leg.

Augmented random search (ARS) randomly searches for policy parameters that maximize the cumulative returns. For each ARS optimization step, we run 16 episodes with randomly sampled policy parameters with a parameter learning rate of 0.03 and parameter exploration noise of 0.05. That is, each of the 16 episodes samples a new parameter $\bar{\theta} = \theta + \Delta\theta$ where $\Delta\theta \sim \mathcal{N}(\mathbf{0}, 0.05)$ where $\mathcal{N}$ is a normal distribution with mean $\mathbf{0} \in \mathbb{R}^{12\times 14}$ and variance $0.05$. Each episode lasts $T = 5000$ steps (50 seconds). The reward function is

$$r_t = \Delta x - 10\,(|r| + |p|) - 0.03 \sum |\omega|, \qquad (9)$$

where $\Delta x$ is the global distance traveled by the robot in the horizontal $x$-direction in one time step. We found that dividing the final episode reward by the number of time steps improves the policy's learning due to a reduction in penalization for sudden falls after an otherwise successful run. The reward function ultimately encourages survivability. For domain randomized training, we resample a new set of domain parameters (see Sec. III) at each training episode of ARS.

### B. Effects of Domain Randomization and Linear Policy

To study the effectiveness of domain randomization, we train linear policies with and without randomization and benchmark them on unseen terrain and dynamics. We measure the distance the robot travels using each method before it fails. We compare these results to the open-loop Bezier curve gait generator by running 1000 trials with random dynamics and terrain. We count survivability (how many times the robot did not fall or exceeded a roll and pitch of 60°) within the simulation time of 50,000 steps (500 seconds).

As shown in Table. II, robots trained with domain randomization (DR-GMBC) that survive travel farther than those trained without randomization (i.e., with fixed dynamics and terrain). Out of 1000 trials, 146 out of 305 (45%) DR-GMBC survivals traveled past 90m (we cut off trials at 100m due to simulation time constraints), and only 26 out of 327 (8%) GMBC survivals did the same, showing a 5.6× improvement. Both methods outperformed the open-loop gait trials, which never made it past 5m and did not survive a single run.

The training curves for the domain randomized linear policy and non-randomized trained policies in Fig. 6 predict that the non-randomized policy is expected to perform significantly better than domain randomized policies, in contrast to the simulation results. This discrepancy between the training and actual performance highlights the potential for simple linear policies to perform well in legged locomotion tasks and provides evidence that training performance is not a useful indicator for testing policy-based locomotion skills in sim-to-real settings.

TABLE II

EACH METHOD WAS TESTED FOR 1000 TRIALS, EACH LASTING 50,000 TIMESTEPS. OVERALL, ROBOTS TRAINED WITH DR-GMBC AND SURVIVE TRAVEL FARTHER THAN THOSE TRAINED WITH GMBC.

| | DR-GMBC | | GMBC | | Open-loop | |
|---|---|---|---|---|---|---|
| **Distance** | **# Died** | **# Lived** | **# Died** | **# Lived** | **# Died** | **# Lived** |
| **≤ 5m** | 488 | 64 | 450 | 121 | 1000 | 0 |
| **5m to 90m** | 207 | 95 | 222 | 180 | N/A | N/A |
| **≥ 90m** | 0 | 146 | 1 | 26 | N/A | N/A |

### C. Sim-to-Real Transfer

The previous simulations illustrate the generalization capabilities and improved performance of DR-GMBC linear policies. We now describe three experiments conducted on OpenQuadruped [6], an inexpensive open-source robot. The 600Hz policy update and execution uses approximately 2% CPU on a Raspberry Pi 4 B.

The first experiment tests DR-GMBC on a robot whose task is to traverse the 2.2 m track shown in Fig. 7 (a) covered with loose stones whose heights range between 10 mm to 60

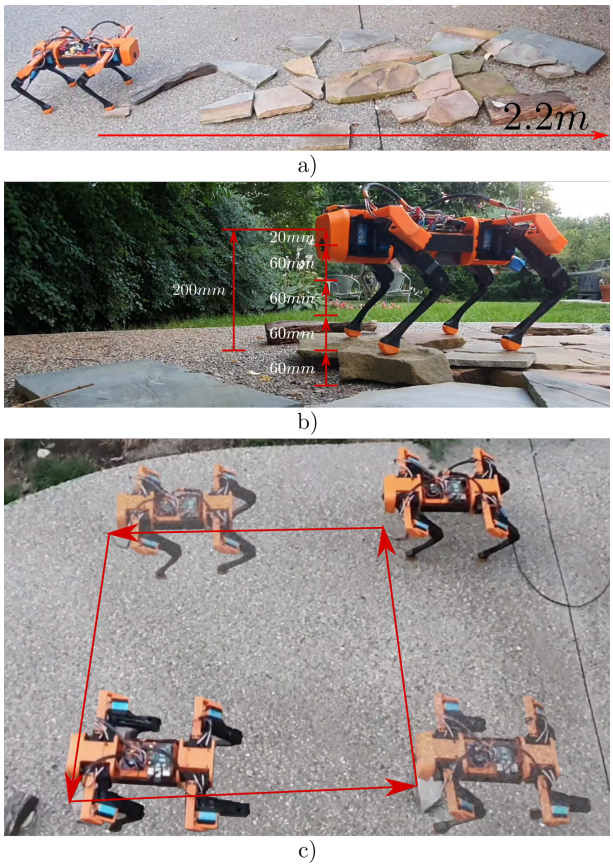


Fig. 7. **Experiment designs.** Illustration of experimental testbeds: a) Experiment 1: Rocky Test Track (2.2m), b) Experiment 2: $60mm$ Loose Stone Descent, c) Experiment 3: Omnidirectional Performance on Flat Ground.

mm (roughly $30\%$ of the robot's standing height). The second experiment evaluates the robot's ability to descend from the peak of loose stones at the maximum 60 mm height onto flat ground shown in Fig. 7 (b). The goal of the last experiment is to show the generalization capabilities of DR-GMBC linear policies trained in simulation for following unseen high-level motion commands $\zeta$ by having the robot follow the $1 \times 1$ m square shown in Fig. 7(c).

**Experiment 1: Traversing Unknown Terrain** In this experiment, we test the policy on terrain that was not seen in training. Additionally, the stones are loose, making the terrain non-stationary and potentially difficult to traverse. Due to the lack of a global odometry, a human operator provided high-level yaw rates to steer the robot as it traversed the stones. To prevent human bias, the robot randomly selected with a 50% probability the DR-GMBC gait or a benchmark open-loop Bezier curve gait. The experiment continued until both methods were used for at least 10 trials.

As shown in Table III, we observed a $1.408\times$ increase in average traversed distance using DR-GMBC compared to the open-loop gait. Although the track was only 2.2 m long, DR-GMBC improved the survivability by $4.28\times$ compared to the open-loop Bezier gait. Furthermore, our approach does not require sensing leg joint positions or foot contact.

TABLE III

EXPERIMENT 1: ROCKY TEST TRACK (2.2M).

| | **DR-GMBC** | **Open-loop Bezier Gait** |
|---|---|---|
| **Distance Mean (of 2.2)** | 1.93 | 1.37 |
| **Std. Dev** | 0.30 | 0.44 |
| **Success Rate (of 1)** | 0.60 | 0.14 |
| **Distance Improvement** | 40.73% | |
| **Success Improvement** | 4.28 × | |

**Experiment 2: Descending from Loose Stones** We set the robot on a raised platform consisting of 60 mm stones to perform a descent test. We record the successful and failed (falling over) descents for both DR-GMBC and open-loop controllers. The operator, unaware of which policy is used, drives the robot forward until it descends or falls. The DR-GMBC agent fell 3 out of 11 times, and was $2.36\times$ more successful than the open-loop controller, which fell 9 out of 13 times.

TABLE IV

EXPERIMENT 3: OMNIDIRECTIONAL SPEED ON FLAT GROUND.

| **Gait** | **FWD (m/s)** | **LEFT (m/s)** | **BWD (m/s)** | **RIGHT (m/s)** |
|---|---|---|---|---|
| DR-GMBC AVG | 0.21 | 0.29 | 0.15 | 0.25 |
| DR-GMBC STD | 0.04 | 0.04 | 0.03 | 0.05 |
| Open-Loop AVG | 0.20 | 0.26 | 0.26 | 0.21 |
| Open-Loop STD | 0.02 | 0.04 | 0.09 | 0.04 |

**Experiment 3: External Command Generalization** This experiment runs the robot on flat terrain to validate the generalization of DR-GMBC to external yaw and lateral commands after the sim-to-real transfer. The experiment additionally provides evidence that improved robustness to rough terrain does not negatively affect performance on flat terrain, even though the policy was trained exclusively for rough terrain.

The operator drives the robot around a $1\text{m} \times 1\text{m}$ track, performing forward, backward and strafing motions, with some yaw commands to correct the robot's heading if necessary. The experiment allowed us to measure locomotion speed by correlating video timestamps with marks on the ground. In our tests, the DR-GMBC policy, compared to the open-loop gait, was $11.5\%$ faster strafing left, $19.1\%$ faster strafing right, and had the same forward speed. Backward speed fell by $57.6\%$. This outlier result was due to the policy pushing the two rear 23kg hobby servos to reach their torque limits, dipping the robot. As a result, the policy, anticipating a fall, would dampen the robot's motion. Simulations indicate that with more powerful motors, backwards walking would experience no performance degradation. Despite this result, the linear policy was still sufficiently able to transfer from sim-to-real and enhance the locomotion performance of the OpenQuadruped over unknown terrain.

## VI. CONCLUSION

In this work, we illustrated that simple linear policies are sufficient for controlling low-cost hobby quadrupedal robots

over uneven unknown terrain. By using an modified open-loop gait generator, we are able to sufficiently modulate the gaits with a train a linear policy on a CPU and deploy it on a low-cost embedded system. We also illustrate that the resulting linear policy can operate on partial observations that are often associated with the limited sensing capabilities of these low-cost robots to generate stable locomotion on unobserved rough terrain. The method is shown to be empirically robust, achieving a $5.6\times$ farther distance ($\geq$90m) on arbitrary terrains through randomized training in fewer than 600 training epochs. Real robot tests show $4.28\times$ higher survivability and farther travel than a robot using solely an open-loop gait, despite the terrain being vastly different from simulation. For future work, we are interested in extending this method to torque-controllable systems to achieve more dynamic behaviors without terrain sensing.

## APPENDIX

### A. 2D Bezier Curve Gait

We discuss the Bezier curve trajectories developed in [3].

*1) Trajectory Generation:* Each foot's trajectory is

$$\Upsilon(S(t),\tau) = \begin{cases} \begin{bmatrix} \tau(1-2S(t)) \\ \delta\cos\frac{\pi\tau(1-2S(t))}{2\tau} \end{bmatrix} & 0 \leq S(t) < 1, \\ \sum_{k=0}^{n} c_k(\tau,\psi) B_k^n(S(t)-1) & 1 \leq S(t) < 2 \end{cases}, \tag{10}$$

a closed parametric curve with

$$B_k^n(S(t)) = \binom{n}{k}(1-S(t))^{(n-k)}S(t). \tag{11}$$

Here $B_k^n(S(t))$ is the Bernstein polynomial [24] of degree $n$ with $n+1$ control points $c_k(\tau,\psi) \in \mathbb{R}^2$. Our Bezier curves use 12 control points (see Table V). The parameter $\tau$ determines the curve's shape and $0 \leq S(t) \leq 2$ determines the position along the curve. The stance phase is when $0 \leq S(t) \leq 1$ and the swing phase is when $1 \leq S(t) < 2$.

TABLE V
BEZIER CURVE CONTROL POINTS. [3]

| Control Point | (q, z) | Control Point | (q, z) |
|---|---|---|---|
| $c_0$ | $(-\tau, 0.0)$ | $c_7$ | $(0.0, 1.1\psi)$ |
| $c_1$ | $(-1.4\tau, 0.0)$ | $c_8, c_9$ | $(-1.5\tau, 1.1\psi)$ |
| $c_2, c_3, c_4$ | $(-1.5\tau, 0.9\psi)$ | $c_{10}$ | $(-1.4\tau, 0.0)$ |
| $c_5, c_6$ | $(0.0, 0.9\psi)$ | $c_{11}$ | $(\tau, 0.0)$ |

*2) Leg Phases:* During locomotion, each foot follows a periodic gait trajectory. The total time for the legs to complete a gait cycle is $T_{\text{stride}} = T_{\text{swing}} + T_{\text{stance}}$, where $T_{\text{swing}}$ is the duration of the swing phase and $T_{\text{stance}}$ is the duration of the stance phase. We determine $T_{\text{swing}}$ empirically (0.2 seconds in our case) and set $T_{stance} = \frac{2L_{\text{span}}}{v_d}$, where $v_d$ is a fixed step velocity and $L_{\text{span}}$ is half of the stride length.

The relative timing between the swing and stance phases of each leg determines which legs touch the ground and which swing freely at any given time. Different phase lags between legs correspond to different locomotion types such as walking or galloping. This periodic motion lets us determine each leg's position relative to $t_{\text{FL}}^{\text{elapsed}}$, the time of the most recent front-left leg impact. Since our robot does not sense contacts, we reset $t_{\text{FL}}^{\text{elapsed}}$ to 0 every $T_{stride}$ seconds.

We define the clock for each leg $\ell$ to be

$$t_\ell = t_{\text{FL}}^{elapse} - \Delta S_\ell(t) T_{stride}, \tag{12}$$

where $\Delta S_\ell$ is the phase lag between the front-left leg and $\ell$. We set the relative phase lag $\Delta S_\ell$ to create a trotting gait:

$$\begin{bmatrix} \Delta S_{FL} \\ \Delta S_{FR} \\ \Delta S_{BL} \\ \Delta S_{BR} \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.5 \\ 0.5 \\ 0.0 \end{bmatrix} \tag{13}$$

Next, we normalize each leg's clock, mapping $t_\ell$ to the parameter $S_\ell(t)$ such that the leg is in stance when $0 \leq S_\ell(t) < 1$ and in swing when $1 \leq S_\ell(t) \leq 2$:

$$S_\ell(t) = \begin{cases} \frac{t_\ell}{T_{stance}} & 0 < t_\ell < T_{stance} \\ \frac{t_\ell + T_{stride}}{T_{stance}} & -T_{stride} < t_\ell < -T_{swing} \\ \frac{t_\ell + T_{swing}}{T_{swing}} & -T_{swing} < t_\ell < 0 \\ \frac{t_\ell - T_{stance}}{T_{swing}} & T_{stance} < t_\ell < T_{stride} \end{cases} \tag{14}$$

Legs are in swing for the first two cases in (14) and stance otherwise. We can now compute the foot position for leg $\ell$ using (10). This scheme suffices for forward walking, but we need the methods of Section IV to walk laterally and turn.

## ACKNOWLEDGMENT

Thank you to Adham Elarabawy for co-creating and collaborating on the development of OpenQuadruped.

## REFERENCES

[1] ANYbotics, "ANYmal C," March 2021. [Online]. Available: https://www.anybotics.com/anymal-legged-robot/

[2] B. Dynamics, "Spot," July 2020. [Online]. Available: https://www.bostondynamics.com/spot

[3] D. J. Hyun, S. Seok, J. Lee, and S. Kim, "High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the mit cheetah," *The International Journal of Robotics Research*, vol. 33, no. 11, pp. 1417–1445, 2014. [Online]. Available: https://doi.org/10.1177/0278364914532150

[4] Unitree, "A1," July 2020. [Online]. Available: https://www.unitree.com/products/a1/

[5] S. S. Robotics, "Open source hobbyist quadruped," 2020. [Online]. Available: https://stanfordstudentrobotics.org/pupper

[6] M. Rahme and A. Elarabawy, "Open source hobbyist quadruped," 2020. [Online]. Available: https://github.com/moribots/spot_mini_mini/tree/spot/spot_real

[7] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020. [Online]. Available: https://robotics.sciencemag.org/content/5/47/eabc5986

[8] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/26/eaau5872

[9] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, 2020.

[10] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[11] NVIDIA, "GeForce RTX 2080 Ti," March 2021. [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/

[12] P. Fankhauser, M. Bjelonic, C. Dario Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5761–5768.

[13] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2261–2268, 2018.

[14] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar, "ROBEL: RObotics BEnchmarks for Learning with low-cost robots," in *Conference on Robot Learning (CoRL)*, 2019.

[15] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke, "Policies modulating trajectory generators," in *Conference on Robot Learning*, 2018, pp. 916–926.

[16] R. Pi, "Raspberry pi 4 model b." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

[17] PJRC, "Teensy 4.0." [Online]. Available: https://www.pjrc.com/store/teensy40.html

[18] K. Paigwar, L. Krishna, S. Tirumala, N. Khetan, A. Sagi, A. Joglekar, S. Bhatnagar, A. Ghosal, B. Amrutur, and S. Kolathaya, "Robust quadrupedal locomotion on sloped terrains: A linear policy approach," 2020.

[19] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *arXiv preprint arXiv:1803.07055*, 2018.

[20] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[21] M. Rahme, I. Abraham, M. Elwin, and T. Murphey, "Spotminimini: Pybullet gym environment for gait modulation with bezier curves," 2020. [Online]. Available: https://github.com/moribots/spot_mini_mini

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation in robotics, games and machine learning," 2017. [Online]. Available: www.pybullet.org

[23] J. H. Lee and J. H. Park, "Turning control for quadruped robots in trotting on irregular terrain," in *Proceedings of the 18th International Conference on Circuits Advances in Robotics, Mechatronics and Circuits*, 2014, pp. 303–308.

[24] G. Lorentz, *Bernstein Polynomials*, ser. AMS Chelsea Publishing Series. Chelsea Publishing Company, 1986.