Force control for Robust Quadruped Locomotion: A Linear Policy Approach

Aditya Shirwatkar*,[1], Vamshi Kumar Kurva*,[2], Devaraju Vinoda[2], Aman Singh[1], Aditya Sagi[1], Himanshu Lodha[1], Bhavya Giri Goswami[1], Shivam Sood[4], Ketan Nehete[1], Shishir Kolathaya[3]

***Abstract*— This work presents a simple linear policy for direct force control for quadrupedal robot locomotion. The motivation is that force control is essential for highly dynamic and agile motions. We learn a linear policy to generate end-foot trajectory parameters and a centroidal wrench, which is then distributed among the legs based on the foot contact information using a quadratic program (QP) to get the desired ground reaction forces. Unlike the majority of the existing works that use complex nonlinear function approximators to represent the RL policy or model predictive control (MPC) methods with many optimization variables in the order of hundred, our controller uses a simple linear function approximator to represent policy along with only a twelve variable QP for the force distribution. A centroidal dynamics-based MPC method is used to generate reference trajectory data, and then the linear policy is trained using imitation learning to minimize the deviations from the reference trajectory. We demonstrate this compute-efficient controller on our robot Stoch3 in simulation and real-world experiments on indoor and outdoor terrains with push recovery.**

Keywords: *Quadruped Robots, Reinforcement Learning, Model Predictive Control (MPC), Linear policy*

I. INTRODUCTION

The domain of quadrupedal robot locomotion, despite having a high degree of underactuation, has reached a significant level of maturity today with several methodologies being successfully developed and deployed [1], [2], [3], [4]. There are classical control-based methods on one side of the spectrum and end-to-end learning-based methods on the other side. Classical control methodologies like Model Predictive Control (MPC) use system dynamics to get an optimal control policy for a broad class of locomotion behaviors [1], [5], [6]. Typically, MPC solves an optimization problem over some finite horizon subject to constraints imposed by dynamics and actuator limits to calculate the push forces for the legs. Quadratic programming (QP) based solvers are a popular choice for solving this optimization problem, however, some of the recent works extensively use nonlinear solvers [7], [4].

* These authors have contributed equally. This work is support by the SERB core research grant: CRG/2021/008115, and Pratiksha Trust Young Investigator Fellowship.

[1]A. Shirwatkar, A. Singh, A. Sagi, H. Lodha, B. Goswami, K. Nehete and S. Kolathaya are with the Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bengaluru.

[2]D. Vinoda, V. Kurva is with the Department of Computer Science & Automation, Indian Institute of Science, Bengaluru.

[3]S. Kolathaya is with the Robert Bosch Center for Cyber Physical Systems and the Department of Computer Science & Automation, Indian Institute of Science, Bengaluru.

[4]S. Sood is with the Indian Institute of Technology, Kharagpur.
(email: stochlab@iisc.ac.in)

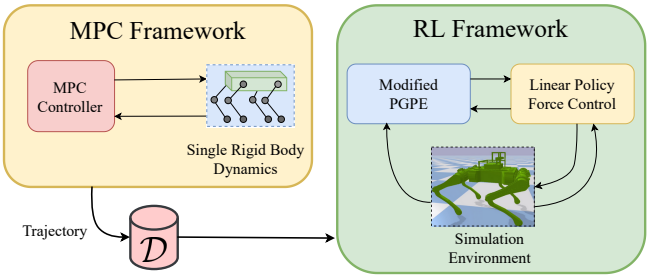

Fig. 1: High-level overview of the proposed learning pipeline

Learning-based methods, on the other hand, achieve locomotion by reinforcement learning (RL), i.e., learn a policy that maximizes some reward criterion by continuously interacting with the environment. It has been shown that RL control policies can learn to tackle unexpected contacts and slippage by exploring different types of actions with the environment [2], [3], [8], [9]. Also, a robust policy can be trained by adding adversarial disturbances like different friction conditions and perturbing forces. These will account for external disturbances or model mismatches during the deployment phase. Nevertheless, most of these methods use complex and hierarchical neural network based policies that are computationally costly. Also, these policies are sometimes hard to train due to the problems of multiple local minima and the tedious tuning of hyper-parameters.

Despite being more computationally efficient than the deep RL frameworks, MPC frameworks require extensive modeling and construction of the optimization problem in every time step. On the other hand, Deep RL frameworks have the ability to automatically realize walking without the need for extensive modeling, but suffer from sim-to-real transfer problems. Therefore, several works have tried to combine these two methods to achieve the best of both over the years. GLiDE [10] uses the centroidal model to develop a Deep RL policy that predicts the desired accelerations, which are transformed into ground reaction forces using a QP. MPC-net [11] learns a mixture-of-networks (MEN) policy guided by an MPC controller using Imitation learning. Since these works use neural networks (multiple in case of [11]), on-board computation becomes expensive at run time. Hence, in this work, we would like to answer a fundamental question: *Can we identify a different class of policies that are simpler than the MPC, RL frameworks in terms of hardware deployment, and yet be effective enough for robust locomotion in quadrupeds?*

Our main motivation for pursuing a simpler class of policies is the classical Raibert's controllers [12], which were instrumental in realizing robust locomotion on monopeds, bipeds and quadrupeds. They were very simple and yet very effective for robust locomotion. In a similar vein, linear policies were developed in [13], [14], [15], with the idea of shaping the end-foot trajectories based on sensory feedback from motors and IMU. It was observed that learning a simple linear policy was sufficient to achieve stable locomotion on sloped terrains in both bipeds and quadrupeds [16], [14]. However, the agility and robustness shown in these works are still not comparable with what have been achieved with the more advanced MPC; hopping, jumping [17] and back-flips [6] have been shown in quadrupeds. Therefore, a primary goal of this paper is to develop a learning based framework that can incorporate MPC based force controllers, but with a significantly lower computational load. In other words, instead of generating force controllers from a model based framework like the MPC, we learn a linear policy that generates the force, torque and trajectory commands directly, yielding a light-weight and a robust control framework for quadrupeds. Thus our contributions can be summarized below:

- We introduce a novel action space consisting of foot trajectory parameters and the desired wrench on the body. The wrench is used in the QP to convert and distribute the desired ground reaction forces among the legs, thus resulting in a computationally lightweight controller.
- We use a modified Policy Gradients with Parameter Exploration (PGPE), a simple evolutionary gradient-free algorithm for imitation learning of the linear policy, instead of complex gradient-based RL algorithms. The training time was < 30 minutes.

An overview of our proposed framework can be seen in Fig. 1 and Fig. 3. We use data generated from a single rigid body dynamics based MPC to train the linear policy. The data includes states of the rigid body like positions and velocities and forces. Reinforcement learning is then used to imitate this trajectory in the custom quadruped, Stoch3, in a simulator (pybullet). The linear policies are trained using PGPE and directly transferred to hardware. We provide both simulation and experimental videos in the supplement.

The paper's organization is as follows: Section II will go through the robot model, notations, and hardware specifications. Section III describes the linear policy and the walking controller. Section IV describes the training and evaluation methods used. Section V summarizes the simulation results, assessments, and hardware tests. Finally, Section VI concludes our work.

II. Robot Description

Stoch-3, as shown in Fig 2, is a custom-built dynamic quadruped robot developed for rapid prototyping of learning-based controllers. This section will briefly describe the accompanying actuators, sensors, and kinematics framework.

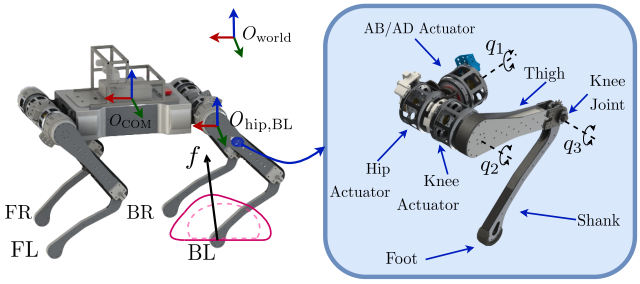

Fig. 2: An illustration of Stoch3 hardware platform. The symbols FL, FR, BL, BR, represent the front-left, front-right, back-left, and back-right legs respectively. The dashed curve shows the ideal end-foot trajectory generated by the trajectory generator, and bold curve in pink shows trajectory modified by the linear policy

1) Actuation: Overall, the robot model consists of 6 floating and 12 actuated Degrees of Freedom (DoF). The actuators are custom built with a high torque density motor and a 1:6 gear reduction. The actuators can measure joint angles and torques via joint encoders and current sensors. The nominal torque provided by the motors is $18\ \mathrm{N\,m}$.

2) Onboard Computation: We use a Raspberry-Pi 4b (RPi) microcomputer for running all the high-level controllers. The mjbots pi3hat board is used for low-level communication between the RPi and the actuators using Control Area Network (CAN) communication.

3) Sensors: We have encoders connected to each motor providing the angle and velocity information of the actuated joints. For inertial measurement, we use an Xsens MTi-610, which provides calibrated data on the 3-D orientation, angular velocities and acceleration.

4) Kinematics: We treat each leg independently to derive analytical relations for forward and inverse kinematics. Here q_1, q_2, and q_3 represent abduction, hip, and knee joints and form a serial-3R kinematic chain as shown in Fig. 2.

III. Control Architecture

This section provides an overview of the control architecture. We use a linear policy as a high level controller to provide high-level commands like the desired wrench and foot shifts. Similarly, as a low level controller, we use a Quadratic Program (QP) to distribute the forces among the contact legs, and a trajectory generator to generate the foot positions. Below the low level controller is a motor-level controller that combines the feed-forward torque with a PD tracking controller resulting in locomotion. Fig. 3 describes this control architecture.

A. High Level controller

On a high-level, we treat the locomotion as an RL problem. The RL policy parameterizes the end-foot trajectories and desired CoM wrench to speed up training. Our method is similar to [13], [14], which uses a feedback mechanism to alter the foot trajectories based on the current state. However, those works only rely on PD tracking, whereas we also include a force control component.

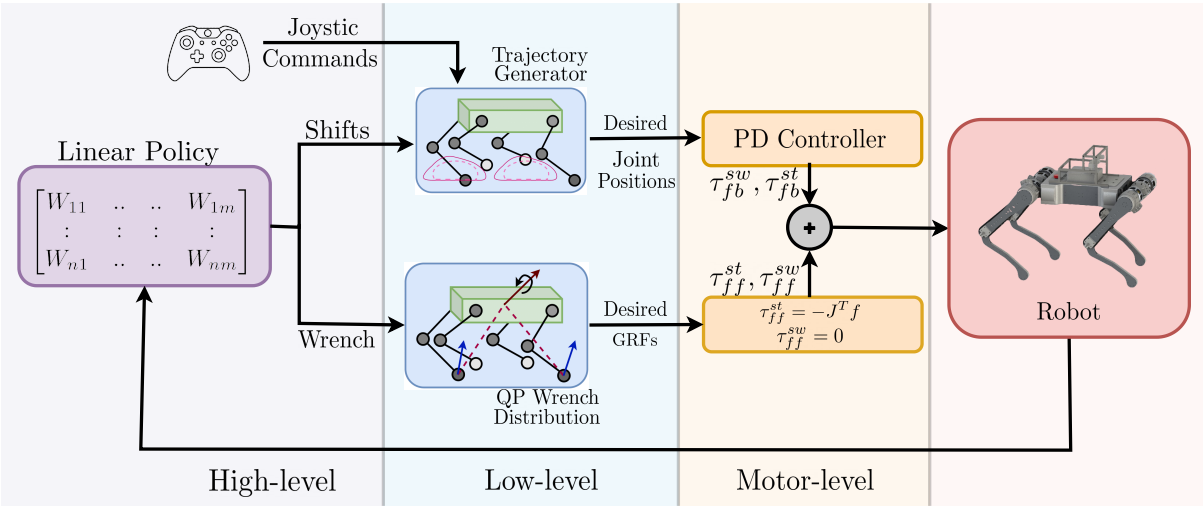

Fig. 3: Overview of the proposed control architecture

1) Observation Space: The observation space $\mathcal{S} \subset \mathbb{R}^{28}$ consists of robot states such as walking height h, body orientation in roll α, pitch β, yaw γ, the CoM twist $\mathcal{V} \in \mathbb{R}^6$, and foot contact boolean vector for each leg η, along with a one-time-step history of all the mentioned states.

2) Action Space: The action space $\mathcal{A} \subset \mathbb{R}^{18}$ represents the instantaneous shifts $\xi \in \mathbb{R}^{12}$ and desired CoM wrench $\mathcal{F}_b \in \mathbb{R}^6$ in body frame. The shifts are translational transforms for the leg trajectories of the robot in the body frame. These shifts and the desired wrench allow us to incorporate reactive and robust force control behaviors.

3) Linear policy: We choose the our policy to be $\pi(s) := M(\theta)s$, where $M \in \mathbb{R}^{18\times 28}$ is a linear mapping from observation space to action space, $s \in \mathcal{S}$ and θ represents the parameters. To make our policies robust in stochastic environments, we sample θ from a normal distribution $\mathcal{N}(\mu, \sigma)$ after each episode. Thus the training algorithm learns the μ and σ for the distribution. More details on the training are in Section IV.

B. Low Level Controller

Low-level controller consists of trajectory generator for generating the foot trajectories for the swing and stance legs, and a QP solver to distribute the wrench among stance legs. Swing/Stance state of the leg is determined using the foot contact estimation based on the torque feedback from the actuators.

1) Trajectory generator: Trajectory generator takes the commands (desired linear and angular velocity command v_d, ω_d) from the joystick along with gait parameters (gait type, swing time T_{sw}, stance time T_{st}, swing height h_{sw}, walking height h_0) to calculate the touch-down point of the swing leg. For simplicity, we will show the trajectory generation for a single i^{th} leg ($i \in \{\text{FL, FR, BL, BR}\}$), which can be extended to other legs easily. The foot placement for the swing leg is determined using the Raibert heuristic [12], and the instantaneous shifts ξ coming from the policy,

$$\rho_{xy} = \frac{v_d T_{st}}{2} + \xi_{xy}^i \tag{1}$$

where ρ_{xy} is the desired step location on ground plane, ξ_{xy}^i are the x, y components of the shifts for the i^{th} leg. From the classical viewpoint, shifts are similar to the capture-point feedback term [18] used for push recovery. However, this capture-point requires certain model assumptions and only depends on velocity feedback, which may not always hold in all scenarios; hence the motivation to learn the shifts that depend on multiple robot states. The leg trajectories for every time step are generated as follows,

$$\begin{aligned} v_{cmd} &= v_d + \omega_d \times r_{xy} \\ \Delta r_{xy} &= \begin{cases} \frac{\rho_{xy} - r_{xy}}{T_{sw}} \frac{\pi}{\pi - \phi} \Delta t & \text{, when in swing} \\ -v_{cmd}\, \Delta t & \text{, when in stance} \end{cases} \\ r_{xy,d} &= r_{xy} + \Delta r_{xy} \\ r_{z,d} &= \begin{cases} h_{sw} \sin\phi + h_0 + \xi_z^i & \text{, when in swing} \\ h_0 + \xi_z^i & \text{, when in stance} \end{cases} \end{aligned} \tag{2}$$

where $\phi \in [0, \pi]$ is the i^{th} leg phase indicating the percentage completion of swing time, Δt is the control time-step, ξ_z^i is z component of the shifts for the i^{th} leg, r_{xy} represent the xy components of the current foot position, and $r_{xy,d}$ and $r_{z,d}$ represent the desired foot positions for the xy components and z component respectively.

2) Wrench distribution using QP: When the leg is in contact with the ground, we use the linear relationship between Ground Reaction Forces (GRFs) $f \in \mathbb{R}^{12}$ and desired wrench $\mathcal{F}_b$ acting on the body which is commonly used in the classical control approaches [17], [19], [20]:

$$\underbrace{\begin{bmatrix} \mathcal{I}_3 & \mathcal{I}_3 & \mathcal{I}_3 & \mathcal{I}_3 \\ \hat{r}^{\text{FL}} & \hat{r}^{\text{FR}} & \hat{r}^{\text{BL}} & \hat{r}^{\text{BR}} \end{bmatrix}}_{\mathcal{A}} f = \underbrace{\begin{bmatrix} mg \\ 0_3 \end{bmatrix}}_{b} + \mathcal{F}_b, \tag{3}$$

where $\mathcal{I}_3$ is the 3×3 identity matrix, $\hat{r}^i$ is the skew-symmetric matrix representing the cross product $r^i \times f$, $[mg,\ 0_3]^T$ is the wrench due to gravity. The desired wrench on CoM $\mathcal{F}_b$ after being inferred from the linear policy is thus used to optimally distribute f using the following QP problem:

$$\begin{aligned} f_* = \underset{f \in \mathbb{R}^{12}}{\arg\min} \quad & (\mathcal{A}f - b)^T \mathcal{H} (\mathcal{A}f - b) + f^T \mathcal{P} f \\ \text{s.t.} \quad & \mathcal{C}f \geq d \end{aligned} \tag{4}$$

where $\mathcal{H}$ is the weight matrix for determining the relative importance between the translational and rotational dynamics, and $\mathcal{P}$ is for enforcing force normalization. The constraints $\mathcal{C}f \geq d$ are applied to ensure the GRFs lie inside the friction pyramid to avoid slippage and to ensure foot contact conditions are satisfied. We use an open-source solver qpSWIFT [21] for solving the QP, as it has been shown to be computationally lightweight and efficient.

C. Motor level control

The command torques for the motors are generated differently for the swing and stance legs. We still treat the legs to be massless during the swing phase, which results in the feedforward torque $\tau_{ff}^{sw} = 0$. To obtain feedback torques for trajectory tracking, we use an inverse kinematics relation to convert the desired foot positions to desired joint angles $q_d \in \mathbb{R}^3$, and a PD controller with appropriate gains K_p^{fb}, K_d^{fb} as follows,

$$\tau_{fb}^{sw} = K_p^{fb}(q_d - q) + K_d^{fb}(\dot{q}_d - \dot{q}) \tag{5}$$

where, $q \in \mathbb{R}^3$ is the vector of joint angles, $\dot{q} \in \mathbb{R}^3$ is the vector of joint velocities, and $\dot{q}_d \in \mathbb{R}^3$ is the vector of desired joint velocities, which is obtained by numerically differentiating q_d. This is done for each leg i. Thus, the command torques for the swing leg is obtained as $\tau^{sw} = \tau_{fb}^{sw}$.

For the stance legs, we solve for feedforward torques needed to achieve the GRF using $\tau_{ff}^{st} = -J(q)^T f_*^i$, where J is the leg jacobian and f_*^i is the desired GRF for the i^{th} leg. We still include PD trajectory tracking for stance legs with low gains for safety. Thus, the final command torques for stance leg τ^{st} is given by

$$\begin{aligned} \tau_{fb}^{st} &= 0.1[K_p^{fb}(q_d - q) + K_d^{fb}(\dot{q}_d - \dot{q})] \\ \tau^{st} &= \tau_{ff}^{st} + \tau_{fb}^{st}. \end{aligned} \tag{6}$$

IV. Training Framework

This section provides an overview of the training architecture, i.e., learning the linear policy using imitation learning. The main objective of this work is to enable the RL policy to distill useful information from the dynamics model of the robot to achieve robust locomotion.

A. Trajectory Generation using MPC

In this work, we train the RL agent to minimize the deviations from the reference trajectory generated by the MPC controller [5] that uses the 3-D Single Rigid Body (SRB) Dynamics model of the following form $\dot{x} := Ax + Bu$. The state vector for this MPC is given by $x = \begin{bmatrix} p & \dot{p} & R & w \end{bmatrix}$, where p, $\dot{p}$ are the linear position and velocity of the torso of the robot and R, w are the angular orientation (represented as rotation matrix) and angular velocities respectively. The action vector for MPC is given by $u = \begin{bmatrix} f^{\text{FL}} & f^{\text{FR}} & f^{\text{BL}} & f^{\text{BR}} \end{bmatrix}$, where f^i is the desired ground reaction force at i^{th} foot contact. We generate the trajectory data $\mathcal{D}$ for an episode which consists of (x_t, u_t) for every time-step (t). These are the optimal state-action pairs solved by MPC to achieve a commanded trajectory which is accelerating at $1\ \mathrm{m/s^2}$ until reaching a velocity of $0.8\ \mathrm{m/s}$. MPC uses a horizon length of 6 and solves a QP problem in 144-variables to track the desired trajectory.

B. Policy Training using PGPE

We use Policy Gradient with Parameter-Based Exploration (PGPE) algorithm to train the control policy to maximize the expected return. PGPE estimates the gradients by directly sampling from the parameter space and is shown to have lower variance in estimates compared to other standard policy gradient methods [22]. In addition to original vanilla PGPE implementation, we have made some enhancements based on current literature such as,

- Solutions were fitness ranked linearly in the range of -0.5 to 0.5, and their ranks were used for gradient computation rather than raw reward values [23].
- σ updates that are more than 20% of the original values are clipped to keep them stable. An idea similar to Proximal Policy Optimization [24], where parameter clipping is used to keep new policy within safe range from old policy.
- Using Adam [25] optimizer to update the gradient of μ.

The policy learnt stable and robust locomotion from reference trajectories within 60 iterations, corresponding to roughly 30 minutes. We used 128 parallel processes, two (one in positive and the other in negative) corresponding to each exploration direction. The maximum episode length was 1000 time steps in all experiments for training the policy. Since the policy is linear, the training time was much less than the latest works in this field, which usually lasts for days and require millions of samples.

C. Domain Randomization

To increase the robustness we employ certain domain randomization techniques, a common practice to make the controller robust. We spawn the robot at random orientations uniformly for torso roll, pitch, yaw in $[0, \pi/6]$ at the start of each episode. Friction parameters of the ground contact is also sampled uniformly in $[0.8, 1.2]$. Lastly to account for sudden undulations in the terrain, we fine-tune the learnt policy on rough terrain with a varying height-field of $\pm 5\mathrm{cm}$. The performance of the learnt policy after enabling these domain randomization methods can be seen in the simulation videos provided.

D. Reward Function Formulation

The reward function is designed in such a way that the the RL policy tracks the reference trajectory as close as possible at every time step. Any deviation away from the reference trajectory is penalised. We measure the deviation from reference variables using a Gaussian kernel defined as $G_{w,k}(x, y) = w \exp(-k\|x-y\|^2)$, where w and k are scalar weights. Since the states, actions used in MPC model and our RL policy are different, we resort to appropriate conversions whenever needed. The reward at any time-step is a linear combination of multiple terms. The expression and weight for each type of reward is mentioned in Table I. The variables with superscript *ref* indicate the reference trajectory state and actions, where as the ones without superscript are the current state and actions.

TABLE I: Reward terms and weights

Reward term	Expression	(w, k)
linear position	$G_{w,k}(p^{ref}, p)$	$(1, 10)$
linear velocity	$G_{w,k}(\dot{p}^{ref}, \dot{p})$	$(1, 50)$
orientation	$G_{w,k}(R^{ref}, R)$	(1, 50)
angular velocity	$G_{w,k}(\omega^{ref}, \omega)$	(1, 50)
GRF	$G_{w,k}(F^{ref}, F)$	(30, 100)

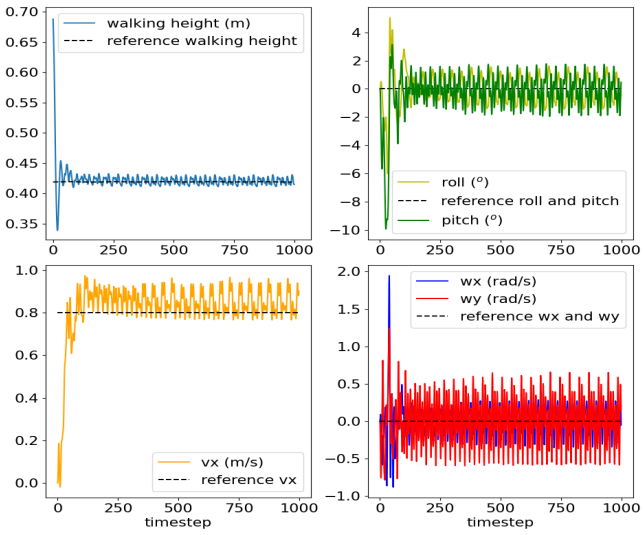

Fig. 4: Graphs for walking height, orientation, linear-x velocity tracking and angular velocities on flat terrain in simulation

V. RESULTS

This section presents the simulation, hardware results and behavioral analysis for stability and robustness. For training purposes, we used a custom gym environment of Stoch3 based on the PyBullet physics simulator [26]. The current work only considers the trot gait for walking, and we believe this should be easily extendable to other gaits.

A. Simulation results

Fig. 4 shows the tracking performance of the learnt policy on flat terrain in simulation. One can see that the linear policy, along with 12-variable QP could extract optimal behaviors from an MPC whose number of optimization variables is well beyond 100. We speculate that the linear policy as a high-level controller, worked well because it enabled the motor-level controller to handle the non-linearities associated with leg kinematics and dynamics of the robot. We further show that the proposed controller could handle outdoor terrains and considerable external disturbances without explicit MPC demonstrations for such scenarios.

B. Comparision with alternatives

We evaluated two policy types (linear vs. fully-connected neural network) and two output types (wrench vs. GRFs) from each type. The results indicated that using QP to distribute the wrench was more effective than predicting GRFs directly from the policy. We believe this is because QP makes it easier to enforce friction-cone constraints. We found that a linear policy was adequate for predicting the desired wrench, making using a neural network policy unnecessary. In addition, the linear policy's simple structure made it easy to transfer the learned policy onto the hardware with minimal changes.

C. Hardware results

We successfully deployed the proposed controller on hardware with minimal changes to the policy trained in simulation. The results presented below are after clipping foot shifts to avoid the legs going out of the kinematic workspace. The wrench values were also bounded for safety. To show the effectiveness of our controller, we performed various experiments like walking on outdoor terrains with external pushes and disturbances. All these experiments can be seen in the video provided.

1) Direction controlled walking: We commanded the robot to walk in multiple directions (forwards, backward, and lateral) and at heading angles using joystick. This enabled safe navigation in real-world scenarios. The proposed controller exhibited zero-shot generalization to desired commanded velocities. This was accomplished by designing the linear policy as a feedback controller to ensure stability (through the desired wrench) and using the trajectory generator to guarantee robust command tracking (through instantaneous shifts). We show that our controller can generate a stable walking gait by empirically analyzing the phase portrait of the joints. Fig. 5 shows that the gait tends to converge to a stable walking limit cycle on flat indoor terrains, and that the foot forces applied by the controller has a stable periodic pattern.

2) Disturbance rejection: Our experiments also included pushing and kicking the robot in different directions multiple times. Our controller recovered to stable configurations as shown in Fig. 6. Our robot could handle pitch variations of up to $\pm 20^\circ$ and roll variation of up to $\pm 15^\circ$, thereby validating the robustness.

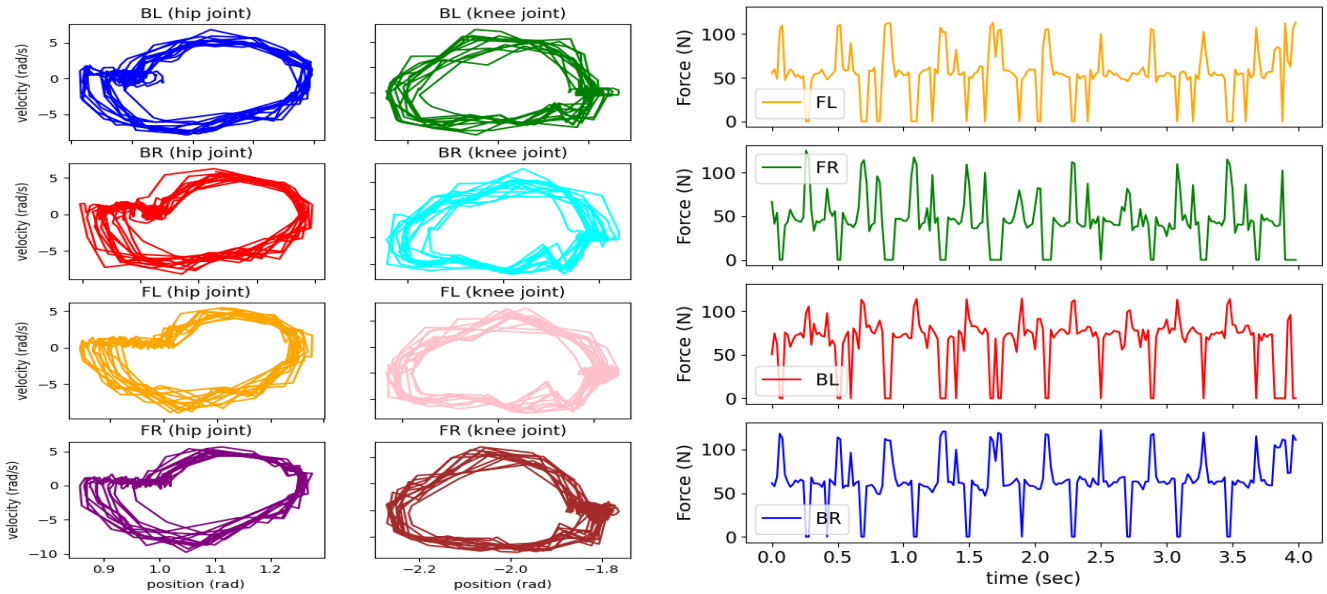

Fig. 5: Phase portrait of joints (left), and force-magnitude profile (right) for the desired GRFs during trot

Fig. 6: Stoch3 recovering from external pushes (left), and the corresponding roll-pitch deviations (right)

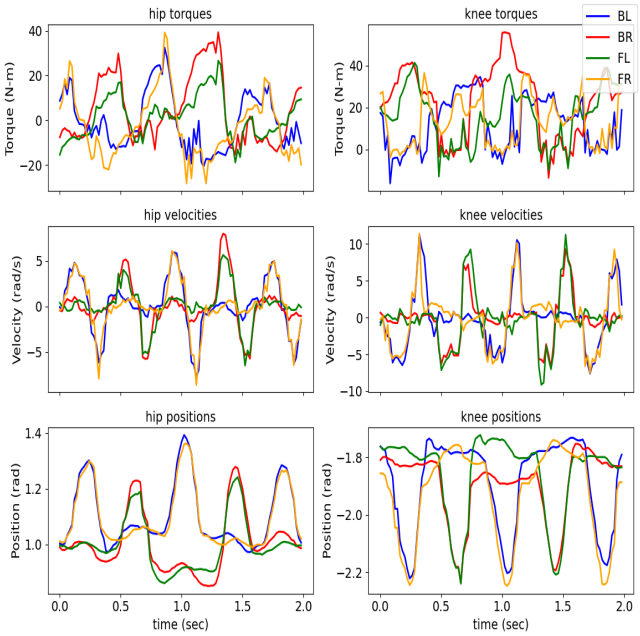

Fig. 7: Joint data for torques, velocities and position for uneven terrain testing

3) Walking on outdoor terrain: We showed that our controller can handle outdoor surfaces, which are not necessarily flat and controlled. Fig. 7 shows the joint data (positions, velocity and torques), which has consistent variations w.r.t. time. This shows that the proposed framework is robust in real-world environments.

VI. CONCLUSION

In this work, we proposed a computationally efficient control architecture consisting of a linear policy and a QP that can achieve stable and robust locomotion and handle external disturbances. We showed that our policy achieved zero-shot generalization to commanded velocities from the joystick and helped in direction-controlled walking for real-world applications. We learnt the optimal behaviors from a single trajectory of the MPC controller using imitation learning. The training time was less than 30 minutes. We also demonstrated the results on the actual robot by deploying the policy on the hardware with minimal changes.

REFERENCES

[1] 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)*, 2018, pp. 1–9.

[2] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," 2021. [Online]. Available: https://arxiv.org/abs/2107.04034

[3] 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://doi.org/10.1126%2Fscirobotics.abc5986

[4] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model predictive control," 08 2022.

[5] Y. Ding, A. Pandala, C. Li, Y.-H. Shin, and H. won Park, "Representation-free model predictive control for dynamic motions in quadrupeds," *IEEE Transactions on Robotics*, vol. 37, pp. 1154–1171, 2021.

[6] H.-W. Park, P. Wensing, and S. Kim, "High-speed bounding with the mit cheetah 2: Control design and experiments," *The International Journal of Robotics Research*, vol. 36, p. 027836491769424, 03 2017.

[7] S. Hong, J.-H. Kim, and H.-W. Park, "Real-time constrained nonlinear model predictive control on so(3) for dynamic legged locomotion," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 3982–3989.

[8] T. Kwon, Y. Lee, and M. Van De Panne, "Fast and flexible multilegged locomotion using learned centroidal dynamics," *ACM Trans. Graph.*, vol. 39, no. 4, Jul. 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392432

[9] S. Chen, B. Zhang, M. Mueller, A. Rai, and K. Sreenath, "Learning torque control for quadrupedal locomotion," 03 2022.

[10] Z. Xie, X. Da, B. Babich, A. Garg, and M. van de Panne, "Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model," 2021. [Online]. Available: https://arxiv.org/abs/2104.09771

[11] A. Reske, J. Carius, Y. Ma, F. Farshidian, and M. Hutter, "Imitation learning from MPC for quadrupedal multi-gait control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2021. [Online]. Available: https://doi.org/10.1109%2Ficra48506.2021.9561444

[12] M. H. Raibert, *Legged Robots That Balance*. USA: Massachusetts Institute of Technology, 1986.

[13] 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. [Online]. Available: https://arxiv.org/abs/2010.16342

[14] M. Rahme, I. Abraham, M. L. Elwin, and T. D. Murphey, "Dynamics and domain randomized gait modulation with bezier curves for sim-to-real legged locomotion," 2020.

[15] L. Krishna, U. A. Mishra, G. A. Castillo, A. Hereid, and S. Kolathaya, "Learning linear policies for robust bipedal locomotion on terrains with varying slopes," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5159–5164.

[16] L. Krishna, G. A. Castillo, U. A. Mishra, A. Hereid, and S. Kolathaya, "Linear policies are sufficient to realize robust bipedal walking on challenging terrains," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2047–2054, 2022.

[17] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2245–2252.

[18] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 200–207.

[19] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Siegwart, "Control of dynamic gaits for a quadrupedal robot," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3287–3292.

[20] M. Focchi, A. Del Prete, I. Havoutis, R. Featherstone, D. G. Caldwell, and C. Semini, "High-slope terrain locomotion for torque-controlled quadruped robots," *Autonomous Robots*, vol. 41, no. 1, pp. 259–272, 2017.

[21] A. G. Pandala, Y. Ding, and H.-W. Park, "qpswift: A real-time sparse quadratic program solver for robotic applications," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3355–3362, 2019.

[22] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *Artificial Neural Networks - ICANN 2008*, V. Kůrková, R. Neruda, and J. Koutník, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 387–396.

[23] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1703.03864

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[26] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.