Guided Constrained Policy Optimization for Dynamic Quadrupedal Robot Locomotion

Siddhant Gangapurwala, Alexander Mitchell and Ioannis Havoutis

Abstract—Deep reinforcement learning (RL) uses model-free techniques to optimize task-specific control policies. Despite having emerged as a promising approach for complex problems, RL is still hard to use reliably for real-world applications. Apart from challenges such as precise reward function tuning, inaccurate sensing and actuation, and non-deterministic response, existing RL methods do not guarantee behavior within required safety constraints that are crucial for real robot scenarios. In this regard, we develop a problem formulation based upon the policy optimization RL framework to train a control policy for quadrupedal locomotion which tracks user-generated reference base velocities with considerations for model-based control constraints. We introduce schemes which encourage state recovery into constrained regions in case of constraint violations. We present experimental results of our training method and test in a physically realistic simulation environment and on the real ANYmal quadruped robot. We compare our approach against the unconstrained RL method and show that guided constrained RL offers convergence close to the desired optima resulting in an optimal yet physically feasible robotic control behavior without the need for precise reward function tuning.

I. INTRODUCTION

Legged locomotion has been an active area of robotics research over the past few decades. Despite our best efforts, achieving extraordinarily dynamic robotic behavior still remains an open problem. Most of the existing work has focused on the use of traditional model-based control techniques, such as offline trajectory optimization (TO) \cite{1} and online model predictive control (MPC) \cite{2} which, due to their mathematical complexity, are often based on simplified models of the systems. This simplification results in control solutions that are mechanically limiting and inefficient thereby inhibiting the agility of the robotic systems in development.

Considering robotic locomotion as a reinforcement learning (RL) problem \cite{3} offers a model-free data-driven alternative to model-based control. Although RL has witnessed significant contributions from researchers to address issues such as sample inefficiency \cite{4}, hyperparameter tuning \cite{5} and convergence delays \cite{6}, it still faces significant challenges to be used for real-world robotic locomotion applications mainly due to no hard guarantees on safety-critical constraints, violations of which may result in considerable damages to the controlled system and its environment of operation.

In this work, we develop an RL problem formulation that introduces constraints based on optimal control techniques. We test our control policy, trained using proximal policy optimization (PPO) \cite{5} for tracking user-generated reference base velocities, on the ANYmal \cite{7} quadruped, a 33kg legged robot developed for operating in challenging environments. We experimentally validate its performance in comparison with unconstrained training procedures in a physically realistic simulation environment and on the real ANYmal robot as shown in Fig. 1.

A. Related Work

Control architectures for robotic quadrupedal locomotion have seen various forms. One of the common approaches leverages mathematical optimization techniques \cite{8} to generate reference trajectories by solving an optimal control (OC) \cite{9} problem with objectives such as minimization of energy consumption, and constraints that consider the dynamics of the robotic systems. Authors of \cite{10} presented a TO formulation for legged locomotion that automatically generates reference motions without requiring any prior footstep planning. We have explored some of the constraints introduced for the TO formulation in our work. These have been described in section III.

Extending upon OC, some of the work has focused on formulating locomotion as multiple tasks \cite{11}, such as maintaining robot stability and tracking desired limb motions, solved by prioritizing each individual task using quadratic programming \cite{12} solvers. This principle of sub-dividing tasks into simpler problems has also been followed in some of the deep RL research \cite{13}.

This work was supported by the UKRI/EPSRC RAIN and ORCA Hubs [EP/R026084/1, EP/R026173/1], Robust Legged Locomotion [EP/S002383/1] and the EU H2020 Projects MEMMO and THING. It was conducted as part of ANYmal Research, a community to advance legged robotics. The authors are with the Dynamic Robots Systems Group, Oxford Robotics Institute, University of Oxford, UK. Email: {siddhant, ioannis}@robots.ox.ac.uk.

Fig. 1: The ANYmal quadruped robot tracking user-defined velocity commands using our trained controller. The full clip is available in the accompanying video.
Model-based control methods, such as the ones described above, often use simplified mechanical models to ease on the mathematical complexity of the problem. For example, most formulations consider the robotic system as a point mass with massless limbs. These approximations result in control solutions that cannot exploit the full range of capabilities of the systems. Moreover, such approaches often require hand-tuned costs by human-experts which are specific to each system’s task. This limits the generality of the methods.

Deep RL methods attempt to address some of these limitations of model-based control by employing model-free techniques which optimize over a control policy, a neural network which maps states into actions, so as to maximize a task-specific reward signal by means of trial and error. Such methods have been investigated for legged locomotion tasks [14], [15] but have mainly been demonstrated in simulations using unrealistic robot models, e.g. ideal torque sources, infinite velocity/torque ranges. Moreover, RL techniques usually require large amounts of data making training on a real robot infeasible. This necessitates the use of physics simulators. However, policies trained using simulations often do not transfer for real world tasks since the reality gap between simulations and the physical world are strongly pertinent. These issues of RL have been tackled using techniques such as actuator modeling [16], implementing a modular training approach [17], and adding noise to observations and actions in the training environments. Authors of [16] have demonstrated the use of a deep RL approach to complex legged locomotion tasks. Our work, extends upon their training methods to a constrained learning approach as discussed in section III.

Despite the success of deep RL approaches on real-world robotic applications, one of the main challenges in solving an RL problem is precise reward function tuning. As a solution, an inverse reinforcement learning (IRL) [18] problem can be characterized as - given expert trajectories of an agent in various circumstances, determine the reward function to be minimized. This reward function is then used to solve an RL problem. The authors of [19] and [20] have successfully implemented IRL methods for perception and control tasks, however, the need for the extra step of solving an RL problem adds to training delays. Instead, designing the problem as that of imitation learning (IL) [21] gets rid of the reward recovery step, and directly optimizes over a policy given expert demonstrations. Along similar lines, guided policy search (GPS) [22] techniques can be used to reduce training times by directing policy learning in turn avoiding poor local optima.

Model-based RL [23] techniques, which require a knowledge of system dynamics, have also been proposed as an approach to boost convergence along desired optima. In the pursuit of making RL methods desirable for use in safety critical systems, methods such as constrained policy optimization (CPO) [24] have also been investigated to ensure that an RL control policy obeys the necessary safety constraints during operation.

**B. Contributions**

Our work extends upon the above research to realize an RL problem formulation that considers the constraints required to guarantee the stability of a quadrupedal robot system. Furthermore, our problem formulation introduces constraints such as end-effector boundaries, swing phase times and joint velocity limits, that direct policy learning towards a desired quadrupedal locomotion behavior. This constrained formulation, coupled with techniques such as GPS, further results in the reduction of training time, while also eliminating the need for precise reward function tuning.

We introduce several schemes that make the quadrupedal system more robust and therefore better suited for use in real-world applications. Since we do not use approximations required for model-based control, our learnt policies better utilize system dynamics to generate efficient locomotion behavior. We further provide evidence of dynamic behavior of the learnt control policy by testing its response after changing the physical properties of the real system, and also continuously varying control step times.

**II. APPROACH**

In this section we describe the RL methods we use for the task of quadrupedal locomotion. We also detail upon the training procedure we incorporate to realize an RL policy for the ANYmal robot that takes user-defined base velocity commands and accordingly generates limb motions to track these commands.

**A. Algorithm**

Based on the framework of Markov decision processes (MDPs) [25], a constrained Markov decision process (CMDP) [26] is defined as a tuple \((S,A,R,C,P,d,\mu)\), where \(S\) is the set of states, \(A\) is the set of actions, \(R: S \times A \times S \to \mathbb{R}\) is the reward function, \(P: S \times A \times S \to [0,1]\) is the state transition probability, and \(\mu\) is the starting state distribution as characterized in the MDP tuple. CMDPs augment the MDP with a set \(C\) of cost functions, \(C_1, \ldots, C_m\), with \(C_i: S \times A \times S \to \mathbb{R}\), and limits \(d_1, \ldots, d_m\) as described in [24]. Being consistent with the definitions and notation used by the authors of [24], a stationary policy \(\pi: S \to \mathcal{P}(A)\) is defined as a function mapping states to probability distributions over actions. The set of stationary policies is defined as \(\Pi.\) \(\pi(a|s)\) denotes the probability of selecting action \(a\) in state \(s\).

Given a performance measure,

\[
J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right],
\]

where \(\gamma \in [0,1)\) is the discount factor, \(\tau\) denotes a trajectory dependent on \(\pi\), we aim to select a policy \(\pi\) which maximizes \(J(\pi)\). Using \(C_i(\tau)\) to denote the cumulative discounted return of a trajectory \(\tau\), and analogous to \(V^\pi\), \(Q^\pi\) and \(A^\pi\) in an MDP, for a CMDP value function, \(V^\pi_C(s) \triangleq \mathbb{E}_{\tau \sim \pi} [C_i(\tau)|s_0 = s]\), action-value function \(Q^\pi_C(s,a) \triangleq \mathbb{E}_{\tau \sim \pi} [C_i(\tau)|s_0 = s, a_0 = a]\), advantage function \(A^\pi_C(s,a) \triangleq V^\pi_C(s,a) - V^\pi_C(s)\), and discounted future state distribution \(d^\pi(s) \triangleq (1-\gamma) \sum_{t \sim \pi} P(s_t = s|\pi)\).
The difference in performance between two policies $\pi'$ and $\pi$ is given as

$$J(\pi') - J(\pi) = \frac{1}{1 - \gamma} \left( E_{s,a,i} [A^\pi(s,a)] - E_{s,a,i} [A^{\pi'}(s,a)] \right)$$  \hspace{1cm} (1)$$
as proved in [27]. For a CMDP, the expected discount cost return $J_C(\pi) \doteq E_{s,a,i} \left[ \sum_{t=0}^{\infty} \gamma^t C_i(s_{1:t}, a_{1:t}, s_{1:t+1}) \right]$ for a policy $\pi$ with cost function $C_i$. The set of feasible stationary policies

$$\Pi_C = \{ \pi \in \Pi : \forall i, J_C(\pi) \leq d_i \}.$$  \hspace{1cm} (2)$$

The RL problem is then expressed as

$$\pi^* = \arg \max_{\pi \in \Pi_C} J(\pi).$$

For a policy $\pi_\theta$ parameterized with $\theta$, most policy optimization strategies iteratively update the base policy using local policy search methods [28] by maximizing $J(\pi)$ over a trust region [29]. For a CMDP, policy iteration using trust regions [24] can be expressed as

$$\pi_{k+1} = \arg \max_{\pi \in \Pi_k} \left( E_{s,a,i} [A^{\pi_\theta}(s,a)] \right)$$

subject to $J_C(\pi_k) + \frac{1}{1 - \gamma} \left( E_{s,a,i} [A^\pi(s,a)] \right) \leq d_i \ \forall i$

$$D_{KL}(\pi_k || \pi_{k+1}) \leq \delta.$$  \hspace{1cm} (3)$$

where $D_{KL} = E_{s,a,i} [D_{KL}(\pi || \pi_k)]$, and $\delta > 0$ is a step size. $D_{KL}$ refers to the Kullback-Leibler divergence. Authors of [24] show that developing CPO as a trust region method implies CPO inherits the performance guarantee given by the lower bound on policy performance difference as

$$J(\pi_{k+1}) - J(\pi_k) \geq -\frac{\sqrt{2\delta \gamma} \epsilon_{k+1}}{(1 - \gamma)^2}.$$  \hspace{1cm} (4)$$

The worst-case upper bound on the cumulative discounted return for the CMDP [24] is given as

$$J_C(\pi_{k+1}) \leq d_i + \frac{\sqrt{2\delta \gamma} \epsilon_{k+1}}{(1 - \gamma)^2}.$$  \hspace{1cm} (5)$$

where $\epsilon_{k+1} = \max_s E_{a \sim \pi_{k+1}} [A^{\pi_\theta}(s,a)]$. The trust region policy optimization (TRPO) [29] method optimizes a surrogate objective [5]

$$L_i^{CPM}(\theta) = E_i \left[ \pi_\theta(a_i|s_i) \pi_{old}(a_i|s_i) A_i \right] = E_i \left[ r_i(\theta) A_i \right].$$  \hspace{1cm} (6)$$

As an extension to TRPO, the proximal policy optimization (PPO) [5] technique introduces a clipped objective function

$$L_i^{CLIP}(\theta) = E_i \left[ \min \left( r_i(\theta) A_i, \text{clip} \left( r_i(\theta), 1 - \epsilon, 1 + \epsilon \right) A_i \right) \right]$$

where $\epsilon$ is a hyperparameter. In our implementation, we introduce an approximation of the constraint expressed in (2) to the above objective, and rewrite it as

$$L_i^{Cclip}(\theta) = L_i^{CLIP}(\theta) - \sum_i \zeta_i J_C(\pi_\theta),$$  \hspace{1cm} (7)$$

where $\zeta$ is an experimentally tuned hyperparameter. The objective $L_i^{CLIP}$ is often augmented to include a value function loss term $L_i^{VF}$ and an entropy term $S$ [5]. The objective function, with coefficients $c_1$ and $c_2$ is then

$$L_i^{Cclip+VF} = E_i \left[ r_i(\theta) - c_1 L_i^{VF}(\theta) + c_2 S(\pi_\theta) \right].$$  \hspace{1cm} (8)$$

We implement a constrained proximal policy optimization (CPO) algorithm along with generalized advantage estimate (GAE) [30], and optimize over the loss function represented in (7).

In our work, we introduce three degrees of constraints and handle them accordingly:

1) Soft ($\rho$) constraints are included as part of the reward function. These need not be critical for safe operations. Instead these are introduced in order to direct policy search towards a desired behavior. In our case, for example, adding a cost term for joint velocity exceeding a limit during locomotion pushes policy search in the direction of smoother locomotion patterns.

2) Hard ($\kappa$) constraints cannot be violated and are included in the set of constrained cost functions $C$. These are directly included during policy updates as shown in (6).

3) No-go ($\eta$) constraints are introduced during training such that when $\kappa$-constraints are violated beyond a certain threshold, the training episode (or a part of it) is disregarded during updates. This is to prevent exploration along regions that violate the critical constraints since the bounds expressed in (3) and (4) are loosened due to approximations introduced in (6).

In order to accelerate training, we combine principles from behavioral cloning (BC) [31] and GPS, and utilize trajectories generated by model-based whole-body controller for the ANYmal quadruped to direct policy exploration. We initialize the policy parameters using supervised learning, then perform policy optimization through exploration, and progressively alternate between reinforcement learning and supervised learning as described in algorithm 1.

B. Simulation

Most RL algorithms are sample inefficient and require significant amount of trials to learn a desirable control policy. Instead of training on a physical platform, which is slow and unsafe, we train the locomotion policy on a significantly faster simulation environment. However, policies trained in simulations often do not perform well in real-world systems. This is mostly due to the reality gap associated with simulations which do not perfectly model the physical world. Moreover, while performing experiments we realized different actuator models for ANYmal resulted in considerably different behaviors for the same training parameters.
We tested ANYmal simulations in RaiSim [32], PyBullet [33] and MuJoCo [34] for a simple task of moving forward with maximum feasible base velocity in order to compare the generated behaviors and training simulation times. The input to the control policy (34-dimensional state vector) consisted of \{base\text{\_}height, base\text{\_}orientation, base\text{\_}twist, joint\text{\_}states\}, and the output (12-dimensional action vector) consisted of \{joint\text{\_}positions\} desired for the next state. We trained the policies using PPO with the same hyperparameters, and on the same device, using the reward function
\[
0.3 \times \text{base\text{\_}forward\text{\_}vel} - 4e-5 \times \|\text{joint\text{\_}torque}\|^2.
\]

We trained the policies for up to 10M time steps with each iteration comprising of 76.8k episodic step samples. Using a discount factor \(\gamma = 0.998\), and maximum episode length of 6.4k simulation steps we achieved the results represented in Fig. 2.

The discounted reward curves represented in Fig. 2 cannot be used to qualitatively evaluate performances of each of the simulators. Instead, we performed experiments using different simulators and actuator models to validate our point that learnt behavior significantly depends on the setup of the RL environment, further substantiated by running experiments in RaiSim using different feed-forward torque and damping parameters for the actuators, as represented in Fig. 2. However, the higher returns for actuators with higher torque output and medium damping does not imply that this control policy would perform better in real world. In fact, in our experiments, actuator model with lower torque output and low damping performed better in Gazebo [35] simulation of ANYmal shipped as part of the ANYmal development framework. Yet, none of the trained policies were observed to be inherently stable, necessitating the use of a good actuator model. The series elastic actuators [36] on the ANYmal robot cannot be accurately modeled to be used in simulations. Hence, we used the approach followed by authors of [16] to approximate the actuator model using a neural network trained through supervised learning.

Moreover, due to sample inefficiency in most RL algorithms, it is important to consider time required for each simulation step. In our experiments, we observed that for same number of parallel executions, RaiSim was faster than both PyBullet and MuJoCo as shown in table I. With several more parallel executions possible across multiple threads, we managed to execute 1B simulation steps in less than 3 hours using RaiSim on a PC.

<table>
<thead>
<tr>
<th>Training Time (seconds)</th>
<th>RaiSim</th>
<th>PyBullet</th>
<th>MuJoCo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1031.6403</td>
<td>2043.9825</td>
<td>1820.8244</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I: Training time required for executing 10M simulation steps using 12 parallel environment runs tested on a PC housing an Intel i7-8700K and an Nvidia RTX 2080Ti.
C. Environment Setup

Authors of [16] and [13] provide competitive baselines. We extend their approach to constrained policy optimization utilizing some of the hyperparameters used in their work. In this section we describe the setup of the ANYmal RL environment for the task of tracking user-generated reference base velocity commands.

1) Observation Space: In order to be extendable to the physical robot, the observation space chosen for the ANYmal environment needs to be accessible through on-board sensors and state-estimators. In this regard, the 109-dimensional state vector for the RL environment is defined as \( \{ b_h, O, v_{\text{base}}, \theta_{\text{base}}, J, J_{\text{des}}, J_{\text{des}}^{\text{ang}}, J_{\text{des}}^{\text{lin}} \} \) where \( b_h \) is the robot base height, \( O \) is the base orientation, \( v_{\text{base}} \) is the linear velocity in base frame, \( \theta_{\text{base}} \) is the angular velocity in base frame, \( J \) is the joint position at time \( t \), \( J_{\text{des}} \) is the policy output at time \( t \), \( \dot{J} \) is the joint velocity at time \( t \) and \( V_{\text{base}} \) is the user-generated desired base velocity expressed in base frame.

2) Action Space: The control policy outputs a 12-dimensional action vector comprising of \( \{ \dot{J}_{\text{des}} \} \). The desired joint positions are forwarded as an input to the approximated actuator network which outputs the torques for each of the joints for the ANYmal quadruped. These torques, clipped between \((-35 \text{Nm}, 35 \text{Nm})\), are then directly applied to the joints.

3) Network Architecture: Since our work focuses on developing a constrained optimization technique, and not on the comparative analysis of control policy models, we used the network architecture implemented in [16] consisting of two hidden layers of size \( \{256, 128\} \) with \( \tanh \) activation as our policy network.

4) Reward Terms: For the case of the MDP formulation, we introduce some of the reward terms similar to those used in [13]. These are shown in table II. Each of these terms, computed at every simulation step, \( t \), is multiplied by a corresponding coefficient which scales the rewards based on the desired behavior. These coefficients are scaled further to increase the difficulty for the RL agent as training progresses, a concept taken from the curriculum learning [37] theory.

5) \( \rho \)-Constraint Costs: Augmenting the formulation to a CMDP, we introduce three degree-based constraint costs for the RL environment. We employ some of the constraints used for trajectory optimization in order to direct policy optimization, and also to ensure that the safety critical constraints are obeyed to a necessary extent. We use the zero-moment point (ZMP) [38] as a stability constraint such that, the ZMP must always lie within the region of support polygon formed with vertices given by leg contacts as shown in Fig. 3. We also define a set of feasible end-effector regions for safe operations which further helps limit policy exploration.

For \( \rho \)-constraints, the costs are directly added to the reward terms. For our RL environment, we consider the cost terms shown in table III.

6) \( \kappa \)-Constraint Costs: The hard constraints that cannot be violated are directly introduced in the CPPO loss function as part of the expected discount cost return \( J_{\text{r}} \) for cost term \( i \). We introduce cost terms that account for stability constraints such as ZMP that have been used extensively for optimal control problem formulations. The \( \kappa \)-constraint cost terms are shown in table IV.

In order to encourage recovery into a stable state upon violation of \( \kappa \)-constraints, we introduce an additive reward term for each of the constraints in case the robot state shifts back to obeying these constraints upon violations.

7) \( \eta \)-Constraint Costs: For cases when the control policy executes actions that cause the robot to land in unstable and unrecoverable states, we introduce \( \eta \)-constraints. Upon violations of these constraints we terminate the training episode, and disregard the training steps that may have been explored between \( \kappa \)-constraint and \( \eta \)-constraint violations and add a negative terminal reward to the last training step in the reformatted episode samples. The intuition behind this was to limit updates through explorations in regions that, apart from violating constraints, do not contribute to learning a feasible and desired locomotion behavior. The constraint violations for \( \rho \)-constraints, \( J_{\text{des}}^{\text{lin}} \) refers to the joint speed limit for \( \rho \)-constraints, \( J_{\text{des}}^{\text{ang}} \) is the joint acceleration limit for \( \rho \)-constraints, \( f_{\text{des}} \) is the height of end-effector \( i \), \( f_{\text{v}} \) is the velocity of the end-effector \( i \), \( v_{\text{ref}} \) is the corresponding feasible end-effector region for \( \rho \)-constraints, and \( f_{\text{b}} \) is the base position for end-effector \( i \) for a given joint configuration.

<table>
<thead>
<tr>
<th>Term</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_{\text{lin}} )</td>
<td>( K \left( v_{\text{lin}} - v_{\text{lin}}^{\text{des}} \right) )</td>
</tr>
<tr>
<td>Torque</td>
<td>( K \left( \theta_{\text{base}} - \theta_{\text{base}}^{\text{des}} \right) )</td>
</tr>
<tr>
<td>Angular Velocity</td>
<td>( | \dot{J} - \dot{J}_{\text{des}} |^{2} )</td>
</tr>
<tr>
<td>Foot Acceleration</td>
<td>( | \ddot{J} - \ddot{J}_{\text{des}} |^{2} )</td>
</tr>
<tr>
<td>Foot Slip</td>
<td>( | \dot{J}<em>{\text{des}}^{\text{lin}} - \dot{J}</em>{\text{des}}^{\text{ang}} |^{2} )</td>
</tr>
<tr>
<td>Smoothness</td>
<td>( | \dot{J} - \dot{J}_{\text{des}} |^{2} )</td>
</tr>
<tr>
<td>Orientation</td>
<td>( | \theta_{\text{base}}^{\text{des}} - {0, 0, 0} |^{2} )</td>
</tr>
</tbody>
</table>

**TABLE II:** Reward terms for the MDP formulation. Here \( K \) refers to the logistic kernel, \( v_{\text{lin}}^{\text{des}} \) is the desired linear velocity in base frame, \( \tau \) is the joint torque, \( v_{\text{base}}^{\text{des}} \) is the desired angular velocity in base frame, \( v_{\text{foot}}^{\text{des}} \) is the foot velocity in world frame at time \( t \), and \( \theta_{\text{base}}^{\text{des}}_{y,z} \) is the base orientation along the \( x,y,z \) axes.

<table>
<thead>
<tr>
<th>Term</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Speed</td>
<td>( \max \left( J_{\text{des}}^{\text{limit}} -</td>
</tr>
<tr>
<td>Joint Acceleration</td>
<td>( \max \left( J_{\text{des}}^{\text{limit}} -</td>
</tr>
<tr>
<td>Foot Clearance</td>
<td>( \sum (f_{\text{des}} - f_{\text{v}})^{2} )</td>
</tr>
<tr>
<td>Foot Eligible Region</td>
<td>( \text{boon} \left( f_{\text{v}} \notin R_{f}^{\rho} \right) \times | f_{\text{v}} - f_{\text{b}} |_{2}^{2} )</td>
</tr>
</tbody>
</table>

**TABLE III:** Cost terms for \( \rho \)-constraints. Here \( J_{\text{des}}^{\text{limit}} \) refers to the joint speed limit for \( \rho \)-constraints, \( J_{\text{des}}^{\text{ang}} \) is the joint acceleration limit for \( \rho \)-constraints, \( f_{\text{des}} \) is the height of end-effector \( i \), \( f_{\text{v}} \) is the velocity of the end-effector \( i \), \( v_{\text{ref}} \) is the corresponding feasible end-effector region for \( \rho \)-constraints, and \( f_{\text{b}} \) is the base position for end-effector \( i \) for a given joint configuration.
supervised learning, else the policy search still preferred some of the samples from the memory buffer are used along discarded. During successive supervised learning sessions, are added to a memory buffer while the other samples are discarded. Some of the samples used for training during each session are evaluated to true, the training episode is terminated.

<table>
<thead>
<tr>
<th>Term</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Speed</td>
<td>[\max(J_{\text{limit}} -</td>
</tr>
<tr>
<td>Joint Acceleration</td>
<td>[\max(J_{\text{limit}} -</td>
</tr>
<tr>
<td>Foot Eligible Region</td>
<td>[\text{bool}(f_i \notin R^f_i) \times</td>
</tr>
<tr>
<td>ZMP</td>
<td>[\text{bool}(u \notin S) \times |u - C|^2]</td>
</tr>
</tbody>
</table>
| Foot Contacts   | \[\text{bool}\left(\sum(F_i > 0) < 3 \& ((F_{f,j} > 0) \lor \left(F_{f,a} > 0 \& F_{f,a} > 0\right))\right)\]

TABLE IV: Cost terms for \(\kappa\)-constraints. Here \(J_{\text{limit}}\) refers to the joint speed limit for \(\kappa\)-constraints, \(J_{\text{limit}}\) is the joint acceleration limit for \(\kappa\)-constraints, \(R^f_i\) is the corresponding feasible end-effector region for \(\kappa\)-constraints, \(u\) is the ZMP, \(S\) is the region of support polygon with vertices given by the feet in contact with the ground, \(C\) is the center of mass of the quadruped, and \(F\) is the contact force at foot \(i\). The foot contacts cost term ensures that if 2 feet are in contact with the ground, they are not on the same side.

<table>
<thead>
<tr>
<th>Term</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Speed</td>
<td>[\text{bool}(\dot{J} &gt; J_{\text{limit}})]</td>
</tr>
<tr>
<td>Joint Acceleration</td>
<td>[\text{bool}(\ddot{J} &gt; J_{\text{limit}})]</td>
</tr>
<tr>
<td>Foot Eligible Region</td>
<td>[\text{bool}(f_i \notin R^f_i)]</td>
</tr>
<tr>
<td>ZMP</td>
<td>[\text{bool}(|u - C| &gt; u^R_i)]</td>
</tr>
<tr>
<td>Foot Contacts</td>
<td>[\text{bool}(\sum(F_i &gt; 0) &lt; 2)]</td>
</tr>
</tbody>
</table>

TABLE V: \(\eta\)-constraint terms. Here \(J_{\text{limit}}\) refers to the joint speed limit for \(\eta\)-constraints, \(J_{\text{limit}}\) is the joint acceleration limit for \(\eta\)-constraints, \(R^f_i\) is the corresponding feasible end-effector region for \(\eta\)-constraints and \(u^R_i\) is the maximum allowed ZMP distance from the center of mass.

terms are shown in table V. When any of the expressions evaluate to true, the training episode is terminated.

III. TRAINING

An overview of the training and verification process employed for our RL task is represented in Fig. 4.

A. Generation of Expert Trajectories

We used a whole-body trotting controller to generate expert trajectories sampled at 400Hz using the Gazebo simulator. The trajectories were represented as state-action \((s, a^*)\) pairs with the state samples corresponding to the state vector included in the observation space of the RL problem and the action samples corresponding to the desired joint positions for the next step.

B. Guided Policy Updates

Our training method alternates between supervised learning and reinforcement learning as presented in algorithm 1. The policy updates through supervised learning ensure that the policy search is directed towards a desired behavior. We trained the policy using the mean-squared-error loss between \(a^*_s\) and \(\pi(a|s)\) minimized using the Adam [39] optimizer. Some of the samples used for training during each session are added to a memory buffer while the other samples are discarded. During successive supervised learning sessions, some of the samples from the memory buffer are used along with new samples from the generated expert trajectories.

While performing experiments, we observed that the policy action entropy had to be reduced after each session of supervised learning, else the policy search still preferred exploration. In fact, during some training experiments we found that, without precise reward function tuning, not reducing entropy caused the RL agent to converge at a local minima. We empirically determined the reduction in entropy after each successive update.

C. Constrained Policy Optimization

During policy exploration and optimization, we use the reward and cost functions described in the previous section to train the RL agent. In order to make the RL control policy more robust to unaccountable external factors, we implement several schemes during training as described below.

1) Adding Noise to Observations and Actions: We add Gaussian noise to the state and action vector to account for sensor noise and inaccurate actuation. The standard deviation vector for the observation space is given as \(s_c = \{0.02, 0.1, 0.05\}, [0.07, 0.02, 0.12], [0.0, 0.04, 0.05], [0.05, 0.0, 0.3\}, and for the action space, it is given as \(s_a = \{0.04, 0.12\}\), where \(s_c \in [0, 1]\) is a scaling term increased over the training period.

2) Changing Gravity: We change the gravity in simulation to emulate inertial scaling. The gravity is initially set with a mean value of \(g = 9.81\), and is randomly sampled between \([0.95g, 1.05g]\).

3) Actuator Torque Scaling: To account for differences in actuators between the real actuators and the approximated model, we use actuator network output torque scaling to ensure that the policy is still usable even if the torque outputs are changed. We randomly scale the output torque with the scaling coefficient \(s_t \in [0.5, 2.0]\).

4) Changing Link Mass and Size: We change the mass and size of the links of the robot randomly during training. This ensures that the training does not converge to a local optima where the policy may result in an undesired locomotion behavior. The mass and size of each of the links...
is scaled by the scaling coefficients $s^{\text{link}}_m \in [0.93, 1.07]$, and $s^{\text{link}}_i \in [0.97, 1.05]$ respectively.

5) Adding Actuator Damping: We use a complementary filter for the control policy output to emulate actuator damping where the desired joint position (input to the actuator network) is given as

$$
\tau_i^{\text{des}} = K_{\text{damp}} \tau_i^{\text{des}} + (1 - K_{\text{damp}}) \tau_i^{\text{des}} - 1
$$

where the gain $K_{\text{damp}}$ is randomized between $[1 - (s_c / 4), 1]$.

6) Changing Simulation Step Time: The control embedded system on the real robot need not obey hard real-time constraints. In this regard, during training, we change the step times between [2.25, 2.75] ms. During experiments we observed that the control policy even worked when the control frequency was changed from 400Hz (corresponding to a step time of 2.5ms) to 200Hz.

D. Verification

We tested the trained control policy on RaiSim for user-generated base velocity commands. Having obtained a visually stable locomotion behavior, we used the same policy to test locomotion in Gazebo for the ANYmal simulation shipped as part of the ANYmal development framework. This provided simulation consists of an approximated analytical model of the actuators in the ANYmal robot, and is thus suitable for verification of the control policy. With a root-mean-squared velocity tracking error observed in Gazebo being comparable to RaiSim, as shown in table VI, we further tested our control policy on the real robot. The results and observations are detailed in section IV.

IV. Results and Discussion

We performed all our training experiments on a desktop PC housing an Intel i7-8700K and an Nvidia RTX 2080TI. For training a control policy using our method, we used 450M simulation samples for about 240 policy optimization iterations requiring less than 2 hours for a RaiSim based simulation. Though the performance of an RL algorithm significantly relies on the setup of the training environment, definition of the reward function and the hyperparameters used, among several other factors, making it difficult to compare the training approaches, during our experiments with unconstrained learning methods, we required a minimum of about 2B simulation samples with precise reward function tuning to obtain a control policy similar to the one obtained using our constrained learning method.

We observed that the behavior obtained upon training the control policy in the case of unconstrained learning changed considerably with the setup of the environment. For example, applying external forces to the quadruped early on during training caused the policy to converge to a local minima resulting in a behavior which focused more on maintaining stability than following base velocity commands. The robot thus maintained its position and did not respond to velocity inputs. Moreover, we performed considerable reward function tuning to obtain a desired locomotion behavior. Without tuning, we obtained inefficient locomotion strategies such as pronking. Our guided constrained policy optimization method, however, did not necessitate perfectly setting up the training environment or performing significant reward function tuning.

The velocity tracking error observed for RaiSim, Gazebo and the physical system, is shown in table VI. The base and desired velocity plots for the physical quadruped are represented in Fig. 5. The results we obtained with regards to energy consumption and base velocity tracking using our RL method when compared against a model-based controller were similar to those obtained by the authors of [16]. Though this comparison cannot be justified, since, the behavior of the control policy can be altered by re-tuning the reward functions, we can state that the control policy trained using our technique offers a more efficient alternative to model-based control with considerations for safety-critical constraints.

While performing experiments, we tested the recovery behavior of the control policy into constraint-following regions by reducing the actuator position tracking gains while the feet were in contact with the ground. Due to this, the quadruped bent forward, as if being pushed from the back, about to fall. However, the control policy recovered from this state by taking forward steps ensuring the ZMP lied within the region of the support polygon.

We also observed that the control policy managed to track velocity commands even when the step times were changed randomly during verification on Gazebo with the mean of 2.5ms and standard deviation as large as 1.288. We tested the robustness of the control policy by perturbing the robot and observed that our controller followed the defined $\kappa$-constraints while tracking the reference velocity commands and prioritized state-recovery into constrained regions upon external perturbations.

V. Conclusion

We presented an RL training method for quadrupedal locomotion that considers safety-critical constraints in its problem formulation and further encourages system recovery into stable states upon constraint violations. We used a

<table>
<thead>
<tr>
<th>RMS Error</th>
<th>RaiSim</th>
<th>Gazebo</th>
<th>Physical System</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15801</td>
<td>0.24156</td>
<td>0.44132</td>
<td></td>
</tr>
</tbody>
</table>

TABLE VI: Root-mean-squared (RMS) normed velocity tracking error for 6,400 time steps.
guided policy exploration technique that used expert trajectories from a traditional trot controller to optimize policy parameters. One can argue that our approach uses a hand-tuned trot controller, rendering the need for an RL formulation unnecessary, nonetheless our experiments demonstrate that the RL method offers a controller which is significantly different than the expert trajectories, and is more energy efficient and robust compared to model-based hand-tuned controllers. Furthermore, our method does not necessarily need expert trajectories and can be used without the guided updates.

Even though the work presented is currently limited to quadrupedal locomotion on flat ground, this method can be extended to different systems and environments. As part of future research, we aim to demonstrate the applicability of our method to more complex environments, with additional information about the environment available to the RL agent, such as, in the form of a depth map. We also intend to perform comparative analysis of different control policy architectures for their performance.

ACKNOWLEDGEMENT

Siddhant would like to thank Dr. Jemin Hwangbo from the RSL, ETH Zurich for his valuable input, help and discussions that have helped shape this work.

REFERENCES


