Trajectory based Deep Policy Search for Quadrupedal Walking

Shishir Kolathaya, Ashish Joglekar, Suhan Shetty, Dhaivat Dholakiya, Abhimanyu, Aditya Sagi, Shounak Bhattacharya, Abhik Singla, Shalabh Bhatnagar, Ashitava Ghosal, Bharadwaj Amrutur

***Abstract*— In this paper, we explore a specific form of deep reinforcement learning (D-RL) technique for quadrupedal walking—trajectory based policy search via deep policy networks. Existing approaches determine optimal policies for each time step, whereas we propose to determine an optimal policy for each walking step. We justify our approach based on the fact that animals including humans use "low" dimensional trajectories at the joint level to realize walking. We will construct these trajectories by using Bézier polynomials, with the coefficients being determined by a parameterized policy. In order to maintain smoothness of the trajectories during step transitions, hybrid invariance conditions are also applied. The action is computed at the beginning of every step, and a linear PD control law is applied to track at the individual joints. After each step, reward is computed, which is then used to update the new policy parameters for the next step. After learning an optimal policy, i.e., an optimal walking gait for each step, we then successfully play them in a custom built quadruped robot, Stoch 2, thereby validating our approach.**

Keywords: *Deep-RL, Quadruped*

I. Introduction

Reinforcement learning has seen a lot success in a variety of robotic systems such as ping pong playing robots [1], autonomous helicopters [2], to name a few. Deep reinforcement learning (D-RL) is a more recent trend, which was successfully implemented to realize quadrupedal walking in Minitaur [3], Stoch [4], and bipedal walking in Cassie [5]. This success is mainly attributed to the policy search based methods [6], [7], [8] which are very effective for infinite dimensional state and action spaces.

Deep Neural networks (DNNs) are a popular choice for parameterizing both the policy and value networks for reinforcement learning. Furthermore, due to the availability of powerful tools for computation of policy and policy gradients, namely TensorFlow, PyTorch, it is easier and faster to deploy the learning algorithms both in simulation and hardware today. Reference [3] demonstrated walking in the quadrupedal robot Minitaur by using 30 parallel agents [9]. Actions are computed from the policy network and then applied on the robot in the form of joint angle commands during every time step. The incremental reward obtained, e.g., distance travelled, is observed and the policy parameters are updated accordingly. This methodology was also followed in the bipedal robot Cassie [5], where the policy parameters were used to improve upon an existing reference trajectory in every time step. The resulting policy learned is on par with the existing classical methods used for Cassie.

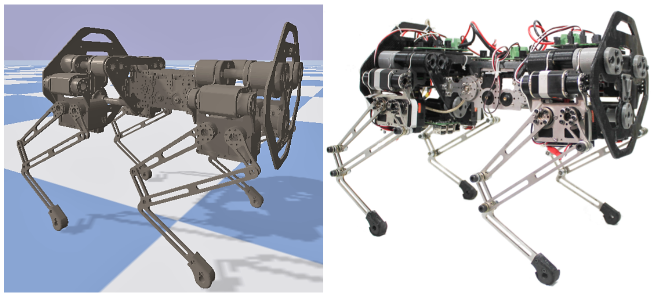
Fig. 1: Figure showing the custom built quadruped robot, Stoch 2. Simulated version is shown in the left, and the actual hardware is shown in the right.

With a view towards a more rigorous analysis of the policies obtained from D-RL, it is important to revisit some of the traditional methodologies followed to realize walking in both bipeds and quadrupeds. In fact, traditional RL algorithms used parameterized functions of time that were played in a loop from start to finish [10], [11]. RL was used to determine the coefficients in these functions, and then policy gradients were used to update these coefficients. It is also worth noting that [4] focused on realizing gaits in simulation via deep policy networks, but the experimental implementation did not require the use of these networks. In fact, with the help of principal component analysis (PCA), kinematic motion primitives were extracted from the simulated gaits and were deployed in the quadrupedal robot, Stoch, resulting in stable walking. Moreover, multiple types of locomotion, namely, trot, bound, gallop and walking gaits were realized by one instance of training. This result demonstrated that despite the complexity in the policies, the resulting trajectories that yielded stable walking were "low" dimensional.

The presence of low dimensional trajectories is not only true in the gaits obtained from D-RL, but also prevalent in biology. It was shown in [12] that the walking in quadrupeds (like horses) inherently contained some basic patterns that

*This work is supported by DST INSPIRE Fellowship IFA17-ENG212, and Robert Bosch Center for Cyber Physical Systems, IISc, Bengaluru.

S. Kolathaya is an INSPIRE Faculty Fellow and S. Bhatnagar, A. Ghosal, B. Amrutur are with the faculty of the Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bengaluru, India. email: {shishirk,amrutur,shalabh,asitava} at iisc.ac.in

A. Joglekar, S. Shetty, D. Dholakiya, Abhimanyu, A. Sagi, S. Bhattacharya, A. Singla are with the technical staff of the Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bengaluru, India. {ashishj,dhaivatd,shounakb,abhiksingla} at iisc.ac.in, {suhan.n.shetty,aditya.sagi13} at gmail.com

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.

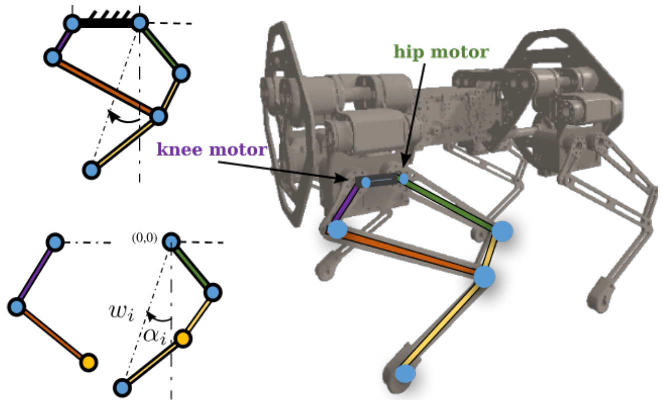

Fig. 2: Figure on the right shows the highlighted leg, the top left figure shows the five-bar linkage, and the bottom left figure shows five-bar linkage divided into two serial 2-R linkages. The polar coordinates are also shown.

were consistent across different gaits. Oscillators located in the spinal cord generated these patterns that resulted in a variety of locomotion gaits. A similar study was conducted in human walking in [13], wherein it was shown that the hip and knee angle trajectories had characteristics of an excited spring-mass-damper system. This observation lead to the realization of *canonical walking functions* in a series of robots AMBER1 [14], DURUS [15], and DURUS-2D [16]. This shows that in order to realize optimal gaits in hardware it is sufficient to search for low dimensional functions of time that yield a stable gait. Therefore, in this paper, we present a trajectory based policy search method for realizing walking gaits for the custom built quadruped robot Stoch 2 (Fig. 1).

We will construct the parameterized functions of time for Stoch 2 by using Bézier polynomials. A similar methodology was followed in [17], wherein a Bézier based optimization was used for passive neural control of all the four legs. Other choices are also available, for example, trigonometric functions, which are equally powerful. The policy network observes the current pose from the robot and determines the coefficients of the Bézier polynomials. These polynomials are played from start to finish and tracked by an appropriate control law to obtain the rewards, which are then used to update the policy parameters. This procedure not only reduces the frequency of update of the policy network, but also yields a jerk-free trajectory that satisfies positional and velocity limits in each joint. Towards the end we will show the effectiveness of these trajectories by demonstrating in the hardware platform Stoch 2.

The paper is structured as follows. In Section II, we will present Stoch 2 model. In Section III, we will present the D-RL framework that describes the entire procedure for realizing walking gaits in Stoch 2. Finally, in Section IV, we will provide the main results along with the video.

II. Robot Description

Stoch 2 is a quadrupedal robot designed and developed in-house at the Indian Institute of Science (IISc), Bangalore, India. It is the second generation robot in the *Stoch* series [4], [18]. In this section, we will briefly provide the hardware description of Stoch 2.

The robot is divided in three modules: two body modules and one central module. The body modules are connected via the central module. The overall size and form factor of the robot is similar to *Stoch* [18]. Each body module is composed of two legs, and each leg contains three actuators—hip flexion/extension, hip abduction and knee flexion/extension. However, the simulation model only uses the hip flexion/extension and knee flexion/extension motions while keeping the abduction locked in position. This enables the leg to follow a given trajectory in a plane (see Fig. 2 and Table I). The central module is rigidly connecting the front and back body modules. The URDF model used in the simulator was created directly from the SolidWorks assembly (see Fig. 1). Overall, the robot simulation model consists of 6 base degrees-of-freedom and 8 actuated degrees-of-freedom.

A. Kinematic Description

Each leg comprises of a parallel five-bar linkage mechanism where two of the links are actuated as shown in Fig. 2. This enables the end foot point to follow the given trajectory in a plane. The two actuators which control the motion of upper hip and knee linkages are mounted on a fixed link. These actuated linkages in turn connect to the lower linkages via revolute joints. The key specifications of the robot leg are summarized in Table I.

In this paper, we focus on realizing trajectories of the feet in polar coordinates (more details in Section III). Hence, given the trajectory, we obtain the desired motor angles by using a custom inverse kinematics solver. As seen from Fig 2, the five-bar linkage is divided into two serial 2-R linkages and solved for each branch. The details of the equations for a serial 2-R linkage can be found in [18]. Safety limits are also included in the inverse kinematic solver to avoid singular positions.

B. Feedback Control

The base position $p_b \in \mathbb{R}^3$, base orientation $o_b \in \mathbb{SO}(3)$, and the 4×2 joint motor angles $q \in \mathbb{R}^8$ form the configuration of the robot. The control $u \in \mathbb{R}^8$ power the motors via a servo drive.

We consider one step to be either the protraction (stance) or retraction (swing) of the leg. Therefore, we employ the

Parameter	Value
total leg length	245 mm
upper hip link length	120 mm
lower hip link length	145 mm
upper knee link length	40 mm
lower hip link length	155 mm
min./max. hip joint	-45°/ 45°
min./max. knee joint	-70°/ 70°
min./max. spine front joint	-15°/ 15°
min./max. spine back	-15°/ 15°
max hip and knee torque	67 kg-cm
actuator power (servo)	72 W

TABLE I: Robot leg specifications.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.

following trajectory tracking control law for the motors for each step:

$$u = -K_p(q - q_d(t)) - K_d(\dot{q} - \dot{q}_d(t)), \qquad (1)$$

where K_p, K_d are the gain matrices, and $q_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^8$ is vector of desired motor angles. In order to switch from stance to swing and vice versa, the desired trajectories are swapped between the left and right legs at the end of every step. These desired values are mapped from the polar coordinates, which are in turn mapped from a parameterized function of time, i.e., the Bézier polynomials. These polynomials are learned via RL, which is explained next.

III. Reinforcement Learning Framework for Quadrupedal Walking

In this section, we will study the deep reinforcement learning (D-RL) framework used for realizing quadrupedal walking in simulation. We will be mainly focusing on realizing trotting gaits in this paper, although this methodology can be directly extended to other types of gaits like bound, canter and walk.

A. Definitions

We formulate the problem of locomotion as a Markov Decision Process (MDP). An MDP is represented by a tuple $\{S, A, \mathcal{P}, r, \gamma\}$. Here $S \subset \mathbb{R}^n$ is the set of robot states of dimension n, and $A \subset \mathbb{R}^m$ is the set of feasible actions of dimension m. $\mathcal{P} : S \times A \times S \to [0,1]$ is the transition probability function that models the evolution of states based on actions, and $r : S \times A \to \mathbb{R}$ is the reinforcement or the reward obtained for the given state and action. γ is called the discount factor defined in the range $(0,1)$.

The goal in RL is to achieve a policy that maximizes the agent's expected return, which is the cumulative sum of the rewards. Since the state and action space is uncountably infinite, a parametrized policy is used, denoted as $\pi_\theta : S \times A \to [0,1]$, with the parameters θ. The optimal policy yields the maximum return. The goal is to obtain the optimal policy π_{θ^*} that yields the maximum return:

$$\theta^* = \arg\max_{\theta} \sum_{k=0}^{M} \mathbb{E}_{(s_k,a_k)\sim p_{\pi_\theta}(s_k,a_k)}[R(s_k,a_k)], \qquad (2)$$

where p_{π_θ} is the marginalization of the state-action pair given the set of all possible trajectories $(s_0, a_0, s_1, a_1, ..., s_M)$ following policy π_θ. The optimal parameters θ^* are evaluated iteratively by taking gradient ascent steps in the direction of increasing cumulative reward.

B. State and Action Space

Compared to Stoch [4], the choice for the state and action spaces are different. Fig. 3 has a detailed representation of these spaces, and are described next.

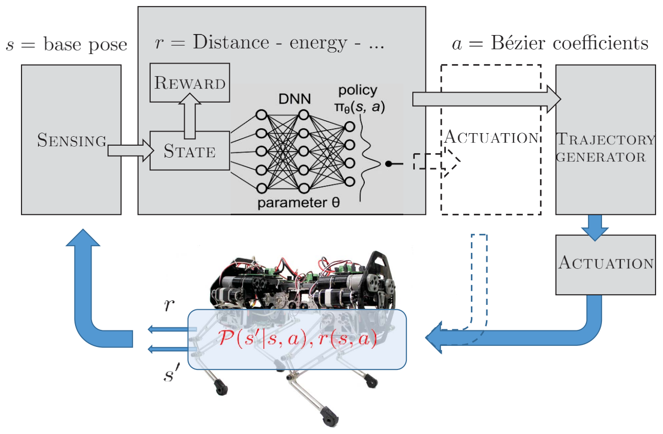
s = base pose
r = Distance - energy - ...
a = Bézier coefficients
Reward
DNN
policy $\pi_\theta(s, a)$
Sensing
State
Actuation
Trajectory Generator
parameter θ
Actuation
r
s'
$\mathcal{P}(s'|s,a), r(s,a)$

Fig. 3: Reinforcement learning framework used to realize walking in Stoch 2. This is a deviation from the standard approach, which maps the actions directly to motor angles (as shown by the dotted boxes). In this paper, we propose to include the *trajectory generator* block before the *actuation* block.

1) State Space: The state is represented by body center of mass (base) position and orientation. Note that the previous version of Stoch [4] consisted of angles, velocities, torques of the active joints as a part of the statespace. We chose to omit them for two reasons 1) The policy update is after every walking step, 2) Calculating velocities and accelerations are noisy on a physical system, which leads to instability. See Fig. 3 for a pictorial representation of RL framework.

2) Action Space: We chose to learn the end-point trajectory of the legs in polar coordinates, which are denoted by $w_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$, $\alpha_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$, where $i \in \{0,1,2,3\}$. As shown by Fig. 2, w_i is distance of the end foot w.r.t. the hip, and α_i is the angle w.r.t. the vertical. Each value of i corresponds to one leg of the robot i.e., front left, front right, back left, and back right respectively. Earlier approaches of determining the trajectories for w_i, α_i involved using the policy network directly i.e., for each time step t_k, the policy network π_θ determined the values for w_i, α_i. See Fig. 3, which shows the inclusion of a new block, the *trajectory generator.*

For each i, w_i is determined via a Bézier polynomial:

$$w_i(\tau) = \Sigma_{j=0}^{N} a_{ij}^{w} (1-\tau)^{(N-j)} \tau^j + w_{\text{off}}, \qquad (3)$$

where N is the order of the polynomial, w_{off} is the offset, and $a_{ij}^w \in \mathbb{R}$, $j \in \{0,1,2,\dots,N\}$ are the coefficients. The polynomials for α_i's are also obtained accordingly, which will be denoted by a_{ij}^{α}'s and α_{off}. For Stoch 2, we chose N to be 3, and also restricted the values for a_{ij}'s in the set $[-1,1]$. $\tau : \mathbb{R}_{\geq 0} \to [0,1]$ is the phase variable that varies from 0 to 1. As is standard practice [19], we will mark $\tau = 0$ to be the beginning of the step, and $\tau = 1$ to be the end of the step. By setting a desired period $T > 0$ seconds, we obtain the phase as a function of time: $\tau(t) = \frac{t}{T}$. After the completion of every step we reset t to zero.

Having defined the polynomial trajectories for each leg,

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.

the corresponding coefficients are obtained from the policy network, π_θ. Therefore, a_{ij}'s form the action space of the robot. Since there are four leg modules, the dimension of a_{ij}'s is 32 (this dimension will be reduced further below). Accordingly, the goal is to learn an optimal set of a_{ij}'s that yield walking in Stoch 2. More details on the learning algorithm are provided in Section III-D.

C. Hybrid Invariance Conditions

In order to ensure continuity of the trajectories, the desired values for w_i, α_i of the current step at $\tau = 1$ should match with corresponding desired values of the next step at $\tau = 0$. Moreover, for a trotting gait, the trajectories of diagonally opposite legs are identical, and are swapped between the left and right legs after every step. Therefore, we require that the following are satisfied:

$$\begin{aligned} \text{Continuity}: w_0(0) = w_1(1), \quad & w_0(1) = w_1(0) \\ \text{Trot}: w_2(\tau) = w_1(\tau), \quad & w_3(\tau) = w_0(\tau), \forall \tau, \end{aligned} \tag{4}$$

which yield us the constraints on a_{ij}^w's. These constraints are easy to ensure via Bézier polynomials. For example, $w_0(0) = a_{00}^w + w_{\text{off}}$, and $w_0(1) = a_{03}^w + w_{\text{off}}$.

In order to realize smooth trajectories, we also ensure continuity of the derivatives of w_i, α_i during switching. Accordingly, we have constraints similar to (4) yielding the following:

$$\begin{aligned} a_{01}^w &= 6a_{13}^w - a_{12}^w, & a_{11}^w &= 6a_{03}^w - a_{02}^w, \\ a_{21}^w &= a_{11}^w, & a_{31}^w &= a_{01}^w. \end{aligned} \tag{5}$$

A similar procedure is followed for obtaining a_{ij}^α's. This implementation reduces the dimension of the action space to 8. It is worth noting that these hybrid invariance conditions are a well known concept in the walking community [19], wherein stable periodic orbits are realized in a low dimensional manifold through these conditions. We have used a simpler form of these conditions by ignoring the impacts.

The polar coordinates for the four legs collectively provide an 8 dimensional space. For each w_i, α_i, we can obtain the corresponding joint angles (both hip and knee of leg i) via an inverse kinematics solver described in Section II-A.

D. Network and Learning Algorithm

Since the states and actions are continuous, we chose to use policy gradient algorithms for determining the optimal policy for the robot. Proximal Policy Optimization (PPO) [20] have been very successful for obtaining optimal continuous action values [21]. It is an on-policy, model-free algorithm based on actor-critic learning framework. We used the open source implementation of PPO2 by Stable Baselines [22]. The implementation is run on GPUs and performs gradient ascent updates using Adam [23] optimizer. More details on the learning algorithm are provided in [4].

E. Simulation Framework

We used MuJoCo simulator, for simulating the robot. A three-dimensional computer-aided-design (CAD) model is developed using SolidWorks to capture the kinematics and inertial properties of the robot. This model is transferred to the simulator by using a Universal Robot Description Format (URDF) exporter. In addition, actual mass of all the links, actuator force limits, joint limits and kinematic-loop constraints of the flexural joints are measured and manually updated in the URDF file for a more realistic simulation.

F. Reward Function

The agent receives a scalar reward after each action update according to the reward function

$$r = W_{vel} \cdot \Delta x - W_E \cdot \Delta E + W_C \cdot C_{\text{ref}} - p. \tag{6}$$

Here Δx is the difference between the current and the previous base position along the x-axis. W_{vel}, W_E and W_C are weights corresponding to each of the terms in (6). ΔE is the energy spent by the actuators for the current step. It is computed as

$$\Delta E = \Sigma_{k=0}^{M} \Sigma_{i=1}^{8} \max\{0, \Gamma_i^T(t_k)\omega_i(t_k)\}\Delta t, \tag{7}$$

where M is the number of time samples in the time interval $[0, T]$. Γ is the vector of motor torques, and ω is the vector of motor velocities. C_{ref} is given as follows:

$$C_{\text{ref}} = e^{-c\Sigma_{i=0}^{1}\Sigma_{k=0}^{M}\left[(x_i(t_k)-x_i^*(t_k))^2+(y_i(t_k)-y_i^*(t_k))^2\right]}, \tag{8}$$

where (x_i, y_i)'s form the cartesian positions of the feet w.r.t. the hip obtained from the polar coordinates w_i, α_i:

$$\begin{aligned} x_i(t) &= w_i(t)\sin(\alpha_i(t)) \\ y_i(t) &= -w_i(t)\cos(\alpha_i(t)), \end{aligned} \tag{9}$$

$c > 0$ is a constant, and x_i^*, y_i^* are reference trajectories in the Cartesian space[1]. See green plots in Fig. 5. Specific values for the weights used for the training are shown in Table II. p is the penalty that penalizes with a high value whenever the robot falls. The fall condition is determined when either the base position is too low or the body orientation exceeds a certain limit. The episode is terminated when the robot falls.

IV. Results

We will start with the training and simulation results, and then focus on the experimental results of Stoch 2 walking.

A. Training and Simulation Results

The actor and critic network in the learning algorithm consists of two fully connected layers with the first and second layers consisting of 64 nodes each. Activation units in the actor and critic networks are ReLU $\rightarrow$ ReLU $\rightarrow$ $\tanh$, and ReLU $\rightarrow$ ReLU $\rightarrow$ linear respectively. Other hyper-parameters are mentioned in Table. II. Six simulation

[1]These reference trajectories for x_i, y_i were, in fact, obtained from the experimental results of a previous version, Stoch [4]. A similar procedure was followed in [24], where a reference gait was used to determine optimal policies faster.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.

Entity	Value
p, W_E, W_{vel}, W_C, c	$0, 0.05, 1, 2, 2$
Discount factor (γ)	0.99
Learning rate (actor, critic)	$2.5 \times 10^{-4}, 2.5 \times 10^{-4}$
Batch size	128×6
Hidden layers (actor)	$[64, 64]$
End-point position limit (θ)	$[-60^\circ, 60^\circ]$
End-point position limit (r)	[14cm, 25cm]
Step period (T)	0.5s
Offsets $w_{\text{off}}, \alpha_{\text{off}}$	23cm, 0°

TABLE II: Hyper-parameter values of the learning algorithm are given here.

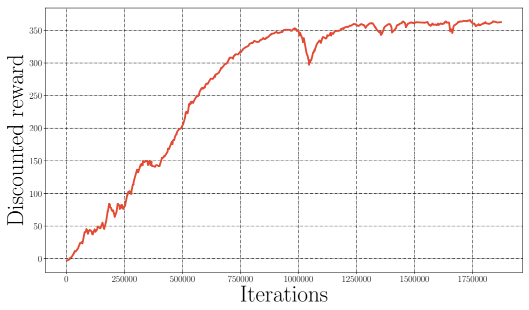

Fig. 4: Figure showing the discounted reward vs. the number of iterations. The reward saturates after approximately 1 million iterations.

environments are used in parallel for faster collection of samples. In order to reduce the training time, we have added offsets (denoted by $a^{*w}, a^{*\alpha}$) to the coefficients obtained from the policy network:

$$\begin{aligned} a^{*w}_{02} &= 0.6478, & a^{*w}_{03} &= -2.7844, \\ a^{*\alpha}_{02} &= -5.7351, & a^{*\alpha}_{03} &= -0.5798, \\ a^{*w}_{12} &= -1.4153, & a^{*w}_{13} &= -0.6887, \\ a^{*\alpha}_{12} &= 4.1526, & a^{*\alpha}_{13} &= 0.8775. \end{aligned}$$

These offsets were manually obtained (which were, in fact, learned in the first version of Stoch), and were fixed for all the training instances. The training starts yielding positive rewards in $20,000$ (8 minutes) iterations and saturates after approximately 1 million iterations. Fig. 4 shows the plot of the discounted reward vs. the number of iterations. The observed training time was 7 hours after 1 million iterations on a six core Intel CPU i7 @3.7Ghz processor with 32 GB RAM, and a GeForce RTX 2080 Ti GPU.

The policy network yielded the following a^w_{ij}, a^α_{ij}'s after training:

$$\begin{aligned} a^{w}_{02} &= 0.4471, & a^{w}_{03} &= 1, \\ a^{\alpha}_{02} &= 1, & a^{\alpha}_{03} &= -0.1395, \\ a^{w}_{12} &= -0.5189, & a^{w}_{13} &= 0.0083, \\ a^{\alpha}_{12} &= -1, & a^{\alpha}_{13} &= -0.0225, \end{aligned}$$

and the remaining coefficients of the polynomial are obtained by using the hybrid invariance conditions given by (4) and (5). Fig. 5 shows the trajectories of the feet w.r.t. the hips in cartesian space. There is a nonzero error between the actual and the desired values due to the fact that the PD gains used were small: $K_p = 30, K_d = 0.3$. This is to ensure that the simulation is realistic, although, it must be noted that the learning framework is agnostic to the PD gains, and solely focuses on realizing a walking gait with the maximum return. Fig. 6 shows the walking simulation tiles for one step.

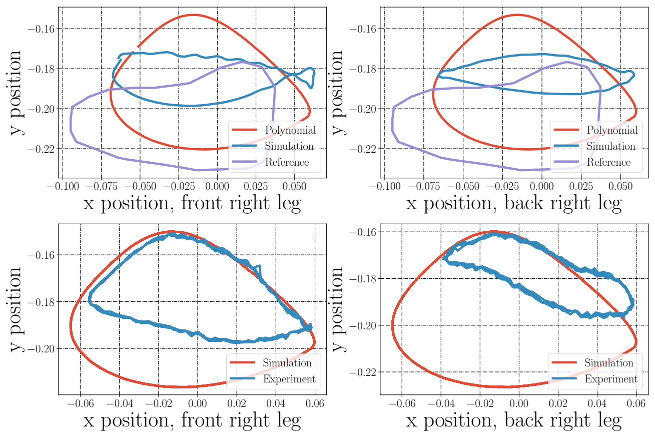

Fig. 5: Figure showing the trajectory of the feet for two complete steps of the robot in cartesian coordinates. There is a large error between the simulated gait and the polynomial (top two figures) values. This is due to the low PD gains used for more realistic simulations. Purple is the reference trajectory. The bottom two figures show the comparison between the desired and actual gait in experiment.

B. Experimental Results

We obtained the desired values for w_i, α_i from the Bézier polynomials. By using an inverse kinematic solver, the desired motor angles are obtained and tracked via a PD control law. The resulting trajectories obtained in experiment are shown in the lower two plots in Fig. 5. Fig. 6 shows the resulting experimental tiles obtained for one step. The following video link also shows the experimental results: `https://youtu.be/aFGM_xWeh3U`

C. Limitations

Since our methodology is based on "step-to-step" learning, i.e., learning the trajectory for the entire step, there is a loss of robustness due to external disturbances within a step. In other words, compared to the walking controllers shown in [3], where the policy update is every time step, our methodology has no notion of recovery *within* the walking step. In addition, we have not incorporated robustifying techniques such as domain randomization, delay adjustment in our learning framework. As a result, minor modifications were required in the experimental implementation. For example, the offset w_{off}, for each leg, was varied by small amounts (≈ 0.01m) to account for the differences in the kinematic model (caused by mechanical wear and tear). It is also worth noting that due to large tracking errors, the gap between the simulation and experimental behaviors were larger. A detailed analysis of this gap will be a main focus of our future work.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.

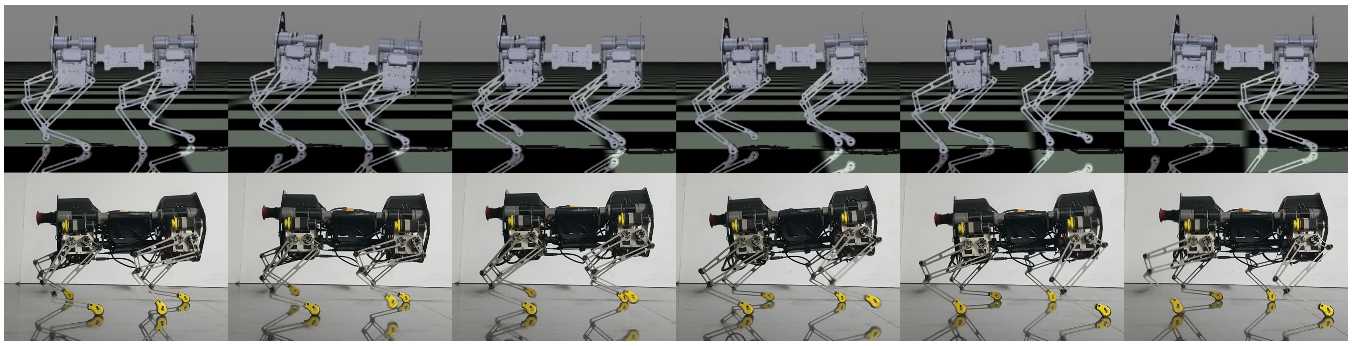

Fig. 6: The walking tiles in simulation (top) and in experiment (bottom) are shown here.

V. Conclusion

In this paper, we demonstrated how trajectory based policy search methods can be utilized to realize stable quadrupedal walking. Motivated by the approach shown in [10], [11], we search for parameterized functions of time, wherein the coefficients of these functions are learned via D-RL. This is a deviation from the approaches followed in [4], [18], wherein the policy network was used to determine the motor angle commands directly. Future work will involve bridging the gap between simulation and experiments, and realization of more complex behaviors like stair climbing, running and bounding.

References

[1] J. Peters, J. Kober, K. Mülling, O. Krämer, and G. Neumann, "Towards robot skill learning: From simple skills to table tennis," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 627–631.

[2] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous helicopter flight via reinforcement learning," in *Advances in neural information processing systems*, 2004, pp. 799–806.

[3] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *CoRR*, vol. abs/1804.10332, 2018. [Online]. Available: http://arxiv.org/abs/1804.10332

[4] A. Singla, S. Bhattacharya, D. Dholakiya, S. Bhatnagar, A. Ghosal, B. Amrutur, and S. Kolathaya, "Realizing learned quadruped locomotion behaviors through kinematic motion primitives," *arXiv preprint arXiv:1810.03842*, 2018.

[5] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne, "Iterative reinforcement learning based design of dynamic locomotion skills for cassie," 2019.

[6] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[7] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[8] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.

[9] D. Hafner, J. Davidson, and V. Vanhoucke, "Tensorflow agents: Efficient batched reinforcement learning in tensorflow," *arXiv preprint arXiv:1709.02878*, 2017.

[10] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, pp. 2849–2854.

[11] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, pp. 2619–2624.

[12] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural networks*, vol. 21, no. 4, pp. 642–653, 2008.

[13] A. D. Ames, "Human-inspired control of bipedal walking robots," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1115–1130, May 2014.

[14] S. N. Yadukumar, M. Pasupuleti, and A. D. Ames, *From Formal Methods to Algorithmic Implementation of Human Inspired Control on Bipedal Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 511–526.

[15] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3d dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5 2016, pp. 1447–1454.

[16] W.-L. Ma, S. Kolathaya, E. R. Ambrose, C. M. Hubicki, and A. D. Ames, "Bipedal robotic running with durus-2d: Bridging the gap between theory and experiment," in *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '17. New York, NY, USA: ACM, 2017, pp. 265–274. [Online]. Available: http://doi.acm.org/10.1145/3049797.3049823

[17] A. A. Saputra, N. N. W. Tay, Y. Toda, J. Botzheim, and N. Kubota, "Bzier curve model for efficient bio-inspired locomotion of low cost four legged robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4443–4448.

[18] D. Dholakiya, S. Bhattacharya, A. Gunalan, A. Singla, S. Bhatnagar, B. Amrutur, A. Ghosal, and S. Kolathaya, "Design, development and experimental realization of a quadrupedal research platform: Stoch," *CoRR*, vol. abs/1901.00697, 2019. [Online]. Available: http://arxiv.org/abs/1901.00697

[19] E. R. Westervelt, J. W. Grizzle, and D. E. Koditschek, "Hybrid zero dynamics of planar biped walkers," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 42–56, Jan 2003.

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

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16669

[22] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[24] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," *CoRR*, vol. abs/1803.05580, 2018. [Online]. Available: http://arxiv.org/abs/1803.05580

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on October 03,2020 at 21:46:32 UTC from IEEE Xplore. Restrictions apply.