Realizing Learned Quadruped Locomotion Behaviors through Kinematic Motion Primitives

Abhik Singla, Shounak Bhattacharya, Dhaivat Dholakiya,
Shalabh Bhatnagar, Ashitava Ghosal, Bharadwaj Amrutur and Shishir Kolathaya

***Abstract*— Humans and animals are believed to use a very minimal set of trajectories to perform a wide variety of tasks including walking. Our main objective in this paper is two fold 1) Obtain an effective tool to realize these basic motion patterns for quadrupedal walking, called the kinematic motion primitives (kMPs), via trajectories learned from deep reinforcement learning (D-RL) and 2) Realize a set of behaviors, namely trot, walk, gallop and bound from these kinematic motion primitives in our custom four legged robot, called the "Stoch". D-RL is a data driven approach, which has been shown to be very effective for realizing all kinds of robust locomotion behaviors, both in simulation and in experiment. On the other hand, kMPs are known to capture the underlying structure of walking and yield a set of derived behaviors. We first generate walking gaits from D-RL, which uses policy gradient based approaches. We then analyze the resulting walking by using principal component analysis. We observe that the kMPs extracted from PCA followed a similar pattern irrespective of the type of gaits generated. Leveraging on this underlying structure, we then realize walking in Stoch by a straightforward reconstruction of joint trajectories from kMPs. This type of methodology improves the transferability of these gaits to real hardware, lowers the computational overhead on-board, and also avoids multiple training iterations by generating a set of derived behaviors from a single learned gait.**

I. Introduction

Legged locomotion has been well studied for more than six decades, starting from the inception of the GE walking truck in 1965 [1]. A variety of approaches, starting from inverted pendulum models [2], zero moment points [3], passivity based control [4], [5], capture points [6], hybrid zero dynamics [7] to even machine learning based approaches [8], [9] have been explored. A more recent trend has been the use of deep reinforcement learning (D-RL) methods [10], [11] to determine optimal policies for efficient and robust walking behaviors in both bipeds and quadrupeds. D-RL was successfully used to achieve walking in the quadrupedal robot Minitaur [10], where the control policy was trained via one of the well known policy gradient based approaches called the proximal policy optimization (PPO) [12].

This work is supported by the Robert Bosch Center for Cyber Physical Systems, Bangalore, India

Abhik Singla, Shounak Bhattacharya and Dhaivat Dholakiya are with the Robert Bosch Centre for Cyber-Physical Systems, IISc, Bangalore, India. E-mail: {abhiksingla, shounakb, dhaivatd}@iisc.ac.in.

Shalabh Bhatnagar is with the Faculty of Computer Science and Automation, Ashitava Ghosal is with the Faculty of Mechanical Engineering, Bharadwaj Amrutur is with the Faculty of Electrical & Computer Engineering, and Shishir Kolathaya is an INSPIRE Faculty Fellow at the Robert Bosch Center for Cyber Physical Systems, IISc, Bengaluru, India {shalabh,asitava,amrutur,shishirk}@iisc.ac.in

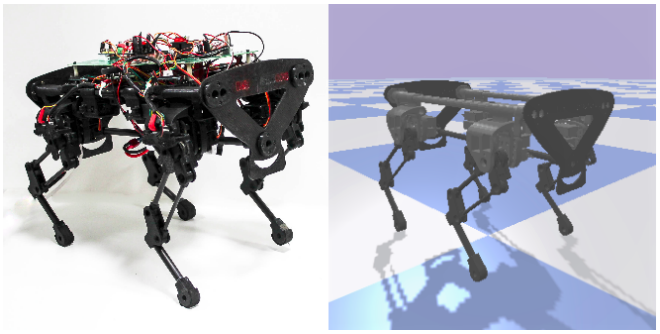


Fig. 1: Figure showing our custom built quadrupedal robot Stoch, and its simulation model shown in Pybullet simulator.

While all the approaches described above were applicable to many kinds of legged robots, there was also an independent group of researchers focusing entirely on capturing symmetries in quadrupedal walking, and then utilize these symmetries to obtain a low dimensional representation for walking [13]. In essence, the focus here was more on trajectory reconstruction for each of the joints of the robot from a minimal number of basis functions, called the *kinematic motion primitives* (kMPs) [14]. It was shown in [15] that only four-five primitives are enough to represent the various locomotion patterns like walking, trotting, and canter. Comparisons were even made with kMPs derived from horse [13], showing a similarity of the kMPs irrespective of the gait used. This type of minimal representation of walking point to the fact that the joint angle trajectories do not contain "high frequency" modulations. Alternative representations of these motion primitives by using canonical walking functions [16], and 6^{th} order Bezier polynomials [7] also indicate this low dimensional behavior of walking. These observations strongly indicate that a small number of basic trajectories (or basis functions) are sufficient to realize periodic trajectories in the individual joints of the robot.

The walking community is currently divided on deciding which approaches are best for achieving legged locomotion. Some use formal methods, which are model based and determine the underlying principles of walking, and then develop controllers that yield stable locomotion [4], [5], [7]. Others use model-free methods like RL (and also D-RL) to derive robust locomotion behaviors in both bipeds and quadrupeds [8], [9], [10], [17]. This methodology not only eliminates the difficult and often unpleasant task of identification and modeling of the robot, but also enables the computer to continuously adapt to changing environments. However, it

Fig. 2: Figure showing a graphical representation of our methodology for realizing quadrupedal walking in Stoch. We use the D-RL based controller for obtaining stable walking gaits in simulation, and then translate them into the quadruped Stoch, experimentally via kinematic motion primitives (kMPs).

is important to note that the controller learned is model-free, but the training is normally done in simulation which is often not realistic. This reality gap is also a well known problem faced by traditional model based optimization tools for walking [18], [19], [20].

One of the main goals of this paper is to approach this problem of reality gap by using the tools used in formal methods. Formal methods usually employ the optimization tools only for generating the reference gaits (trajectories), and then these trajectories are tracked at the joint level by model free methods (e.g. PD control) [18], [21], [22]. Therefore, we would like to view D-RL based training as a trajectory generator (optimization tool) more than the on-board compute platform for walking. This is similar to the method followed by [17], where RL was used to find the optimal set of parameters of the foot trajectory. We will obtain a gait from D-RL, extract kMPs from this gait, and then realize a set of derived behaviors, namely trot, walk, gallop and bound in Stoch experimentally. Trot and gallop were achieved in [10] by separate instances of training, while we were able to realize more than four behaviors by one instance of training. A pictorial representation of our approach is given in Fig. 2. This hybrid approach allows for a mechanistic way of generating gaits in complex legged locomotion—that combines the data driven D-RL based gait synthesis in simulation leading to extraction of kMPs which are then used for gait tracking in hardware with minimum computational overhead.

The paper is organized as follows: Section II introduces the Deep Reinforcement Learning (D-RL) framework for walking. Section III has a detailed description on kinematic motion primitives (kMPs) and the associated kMP analysis of D-RL based gaits. Finally Section IV describes the experimental results of Stoch walking.

II. 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 also discuss the problems with transferability to real hardware.

A. Definitions

We formulate the problem of locomotion as a Markov Decision Process (MDP). An MDP is represented by a tuple (S, A, P, R, γ). Here $S \in \mathbb{R}^n$ is the set of robot states referred to as state space, and $A \in \mathbb{R}^m$ is the set of feasible actions referred to as the action space. $P : S \times A \times S \to [0, 1]$ 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 expected cumulative reward over time. A policy, denoted as $\pi : S \times A \to [0, 1]$, is the probability distribution of the actions over the states. When dealing with huge state and action spaces, a parametrized policy is used, denoted π_θ, as with the parameters θ. We use the following to obtain the optimal policy π_θ:

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)}[R(s_t, a_t)], \quad (1)$$

where p_θ is the marginalization of the state-action pair (s_t, a_t) given the set of all possible trajectories $(s_0, a_0, s_1, a_1, \ldots, s_T)$ following policy π_θ. The optimal parameters θ^* are evaluated iteratively by taking gradient ascent steps in the direction of increasing cumulative reward. More details on the algorithm, and the associated optimization technique are provided in Section II-C.

B. State and Action Space

We will briefly describe the quadrupedal model of Stoch. There are four identical legs with two actuators per leg. Each leg has a hip and a knee joint, and the actuators provide flexion-extension in each joint. Fig. 1 shows both the experimental and the simulated robot model. Details on the experimental model are in Section IV and [23].

1) State Space: The state is represented by angles, velocities, torques of the active joints, and body orientation (in quaternions). The combined representation yields a 28D state space. Note that the motor torques are typically treated as states in RL framework.

2) Action Space: Reference [24] demonstrated the effectiveness of choosing actuator joint angles as action space. The planar four bar linkage configuration of the legs, however, lead to self-colliding regions in the motor space. This creates a non-convex search space and degrades the learning efficiency. Hence, we chose to learn the legs' end-point position in polar coordinates represented as $\{r_i, \theta_i\}$ where $i \in \{1, 2, 3, 4\}$. The polar coordinates for the four legs collectively provide an 8 dimensional action space. For each $\{r_i, \theta_i\}$, we can obtain the joint angles (both hip and knee of leg i) via an inverse kinematics solver. We will denote the angles obtained from the inverse kinematics solver as $\{\phi_i^1, \phi_i^2\}$. The polar coordinates can be restricted to a bounding box, thereby indirectly imposing angle limits on the joint angles.

C. Network and Learning Algorithm

Since the states and actions are continuous, we seek for algorithms that enable us to construct optimal policies that yield action values in the continuous space. Proximal Policy Optimization (PPO) [12] have proved to be very successful for obtaining optimal continuous action values [25]. It is an on-policy, model-free algorithm based on actor-critic learning framework. An actor (also called policy) decides an action given a state and a critic aims to improve the policy by evaluating the action value function. Both actor and critic networks are parameterized by neural networks with parameter and respectively.

The learning algorithm samples episodes of maximum length $T = 1000$ using the current policy. The experience tuples consisting of $\{s_t, a_t, s_{t+1}, r_t\}$, with $t \in [0, T]$, are stored and used for on-policy update of the parameters. Policy gradient method [26] gives the objective function to optimize the policy parameters given as:

$$L^{PG}(\theta) = \mathbb{E}\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right] \quad (2)$$

where $\hat{A}_t$ is called the estimated Advantage function. It is given by $\hat{A}_t = r(s_t, a_t) + V(s_{t+1}) - V(s_t)$, where V is the value function at any state. Furthermore, in order to account for the disparity between the behavioral policy, π_{old}, and the current updating policy , the advantage estimation is multiplied with the importance sampling ratio given by

$$L(\theta) = \mathbb{E}_{\pi_{old}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t\right]. \quad (3)$$

Lastly, in order to avoid huge and sudden policy variations during training, PPO penalizes on the high KL-divergence and solves an unconstrained optimization. Hence the final objective for the PPO algorithm is given by

$$L^{KL}(\theta) = \mathbb{E}_{\pi_{old}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t\right] - \beta_{KL}\,\mathbb{E}_{\pi_{old}}\left[D_{KL}(\pi_{old}(a_t|s_t)\,||\,\pi_\theta(a_t|s_t))\right] \quad (4)$$

where β_{KL} is an adaptive penalty coefficient. Given a target KL-divergence, , β_{KL} for the next policy update is calculated as:

if $D_{KL} < = 1.5$, $\beta_{KL} = \beta_{KL}/2$

if $D_{KL} > = 1.5$, $\beta_{KL} = \beta_{KL} \cdot 2$

We used the open source implementation of PPO by Tensor ow Agents [27] that creates the symbolic representation of the computation graph. The implementation is highly parallelized and performs full-batch gradient ascent updates, using Adam [28] optimizer, on the batch of data collected from multiple environment instances.

D. Simulation Framework

We used Pybullet [29] simulator, built on top of Bullet3 physics engine, for a realistic model simulation. A three-dimensional computer-aided-design (CAD) model is developed using SolidWorks [30] to capture the kinematics and inertial properties of the robot. This model is transferred to Pybullet by using a Universal Robot Description Format [31] (URDF) exporter. In addition, actual mass of all the links, actuator force limits, joint limits and kinematic-loop constraints of the exural joints are measured and manually updated in the URDF le for a more realistic simulation.

E. Reward Function

We designed a reward function that gives positive reinforcement with the increasing robot's base speed and simultaneously penalizes high energy consumptions. The agent receives a scalar reward after each action step according to the reward function

$$r_t = w_{vel} \cdot \text{sign}(\Delta x_t) \cdot \max\{|\Delta x_t|, 0.1\} - w_E \cdot E. \quad (5)$$

Here Δx_t is the difference between the current and the previous base position along the x-axis. E is the energy spent by the actuators for the current step, and w_{vel} and w_E are weights corresponding to each term. E is computed as

$$E = \sum_{i=1}^{4} |\tau_i^1(t) \cdot \omega_i^1(t)| + |\tau_i^2(t) \cdot \omega_i^2(t)| \cdot \Delta t, \quad (6)$$

where τ_i^1, τ_i^2 are the motor torques, and ω_i^1, ω_i^2 are the motor velocities of the i^{th} leg respectively.

F. Gait Generation 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 200 and 100 nodes respectively. 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. I. The pybullet simulator is configured to spawn multiple agents for faster collection of samples—in our case 30 agents were used in parallel threads.

Quadrupedal animals typically have eight types of gaits [32], [33]. Reference trajectories were used in [11] open-loop signals were used in [10] to learn a specific gait. However, in our work, we learned everything from scratch. To learn a specific gait type, we leveraged on gait symmetries and placed hard constraints on the foot position. For e.g. in trot, diagonally opposite legs are in sync. Hence, we provide same for $\{r, \theta\}$ to these legs. This also results in reduction in the dimensions of action space from 8 to 4.

Entity	Value
w_E ; w_{vel}	0:05; 1:0
Discount factor( )	0:994
Learning rate (actor, critic)	10^{-4}; 10^{-4}
Training steps per epoch	3000
KL target ( )	0:01
End-point position limit ()	[30 ; 30]
End-point position limit r()	[15cm, 24cm]

TABLE I: Hyper-parameter values of the learning algorithm

The proposed approach yields walking gaits that are efficient, fast and also robust in simulation. The learning algorithm runs for a maximum of 15 million steps and the observed training time is 4:35 hours on a Intel Core i7 @3.5Ghz 12 cores and 32 GB RAM machine.

G. Challenges

There are two main challenges in realizing D-RL based gaits in quadrupeds:

1) Training for model and environment updates:The training framework has a large set of hyper-parameters (a small subset of them are shown in Table I) that require tuning in order to get desirable gaits. These gaits are, indeed, robust and efficient, but any update in the model or change in constraints in the gait requires retuning of these hyper-parameters in order to obtain desirable gaits. Retuning is not intuitive and requires a series of training iterations. This is a common problem even in nonlinear model based walking optimizations [34], where small variations in constraints often lead to infeasible solutions.

2) Transferability on real hardware:Transferability of these learned gaits in the real hardware faces numerous challenges. Some of them are mainly due to uncertainties due to unaccounted compliances, imperfect sensors, and actuator saturations. In addition, inertial and electrical properties of actuators vary from one joint to another. It can be observed that imposing very conservative constraints like tighter velocity and angle limits resulted in longer training times, which often converged to very unrealistic gaits.

In order to put these challenges in context, it is necessary to review the methodology followed in [10]. They proposed to develop accurate robot model, identify the actuator parameters and study the latencies of the system. This procedure, indeed, resulted in more realistic gaits, but required multiple instances of training and is highly sensitive for any update in hardware. In addition, DNNs are computationally intensive (NVIDIA Jetson TX2 processor was used). Therefore, as a deviation from this approach, we primarily want to focus on a methodology that is more robust to hardware changes, and also computationally less intensive. As we add more and more tasks and objectives for a diverse set of environments, it is important to focus on using a compressed set of walking parameters with minimal training. We will discuss this in detail in the next section.

III. Realizing Behaviors through kMPs

The kinematic motion primitives (kMPs) are invariant waveforms, sufficient to explain a wide variety of complex coordinated motions, including discrete (e.g., reaching for a target with one hand) and periodic behaviors like walking and running [15]. This concept of motion primitives arose from biological observations [15], [35], [13]. In essence, it was observed that biological motions can be described to a good extent with the combination of a small number of lower dimensional behaviors [14], [36].

The process of determining kMPs are primarily conducted by Principal Component Analysis (PCA) [37], [35], [13], [38], [39]. PCA is useful when there is extensive data from walking demonstrations, which can be processed to extract the kMPs. These kMPs can then be adapted/transformed to realize walking in robots. In essence, principal components yield vectors in the joint space with the highest variance for the given data. A linear transformation of the joint space to these principal components results in dimensionality reduction. A more detailed exploration of this analysis is done in [37] and [40].

We will first describe the methods used for the data analysis, and then the associated principal components. kMPs for the different gaits, trot, pace and bound, will be extracted and compared. Towards the end of this section, we will also provide a statistical comparison between the different kMPs.

A. Extraction of kMPs

Having obtained walking in simulation based on a trained policy, we record the joint angles and body orientation of the robot for 4800 steps. The recorded data is then processed as follows: First, we remove front and rear 15% of the data set to avoid transient cases. Second, we divide the data into segments by peak detection. All these data segments are then shifted and scaled w.r.t. time in order to align the end points. By polynomial interpolation, we reconstruct trajectories in between the data points. With this reconstruction, we can directly obtain the mean data.

Given the mean data, let N be the total number of points for each joint. Denote this data by $x[t] \in \mathbb{R}^{n_j}$, where n_j is the number of actuated joints (8 for Stoch). We get a matrix of joint angle values as $X = [x[1] \;\; x[2] \;\; \dots \;\; x[N]]^T \in \mathbb{R}^{N \times n_j}$. Let $\hat{X}$ be the zero mean data obtained from X. We compute the covariance as

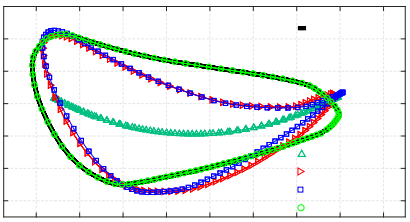
$$\text{cov}(\hat{X}) = \frac{1}{N}\hat{X}^T\hat{X} \tag{7}$$

The principal components are precisely the eigenvectors, $e_1, e_2, \dots, e_{n_j}$, of $\text{cov}(\hat{X})$. Hence, for 8 joint angles, we have 8 principal components. Given the principal components, we have the i^{th} kMP defined as

$$M_{P,i} := \|\hat{X}\cdot e_i\|_1 \hat{X}\cdot e_i, \tag{8}$$

where $\|\cdot\|_1$ (in unity norm) is used to normalize the magnitude. We typically use fewer components (4 for Stoch) for reconstructing the joint angle trajectories from the kMPs, since they account for over 99% of the cumulative percentage variance (see table in Fig. 3).

Having obtained the kMPs, we reconstruct the original joint trajectories as follows. Denote the vector of kMPs as $P := [M_{P,1} \;\; M_{P,2} \;\; \dots \;\; M_{P,n_c}] \in \mathbb{R}^{N \times n_c}$, where n_c is



1^{st}	88.9%
2^{nd}	95.8%
3^{rd}	98.4%
4^{th}	99.9%

Fig. 3: Figure on the left shows end point trajectories of the one of the feet after reconstruction from different numbers of kMPs. The X-axis corresponds to the forward direction, and the Y-axis corresponds to the vertical direction respectively. Table on the right shows the percentage cumulative variance for adding each component at a time.

typically less than n_j (for Stoch $n_c = 4$). We have the final reconstruction of the joint angles as

$$Q = P \quad S + Z; \tag{9}$$

where $Q = R^{N \quad n_j}$ is the matrix of desired joint angle values, $S\ 2\ R^{n_c \quad n_j}$ is coefficient matrix, also called as kMP synergy map, that maps the kMPs to the joint space, and $Z\ 2\ R^{N \quad n_j}$ is the zero mean offset matrix (derived from $\bar{x}$), which is added back to the joint angles.

B. Discussion & Comparison

Fig. 3 shows the result of reconstruction from the kMPs. A table showing the cumulative percent variance of the original trajectory with reconstruction from each kMP is also provided. It can be verified that only four kMPs were sufficient for the final reconstruction . Fig. 4 shows the comparison between the end foot trajectories obtained from the simulation, and from the reconstruction. It must be noted that kMPs yield trajectories that are as good as the time averaged trajectories of the original data (which is eight dimensional). In addition, with kMPs, there is existing literature for obtaining derived behaviors like walk, trot and gallop [15], [38], which will be explained more in the next section.

It was shown in [13] that biological quadrupeds (like horses) inherently contain some basic patterns that were consistent across different gaits and models. Based on this result, we compared the trot data obtained from our simulation with the trot data obtained from the horse, and the results are in Fig. 5. We observed that there was a high correlation not only between the kMPs obtained from the learned and horse gait (Fig. 5-A), but also between the kMPs obtained from different learned gaits (Fig. 5-B). Table III provides more statistical details, where it is shown that cross-covariance w.r.t. the first two kMPs is more than 90%. These observations point to the fact that even the kMPs derived from the RL based simulation followed very similar patterns, irrespective of the type of gaits and the model used.

This observation is true for any continuous trajectory that does not have "high-frequency" oscillations. This property was extensively used to reconstruct walking trajectories by using ellipses in [17] and low-order basis functions in [8].

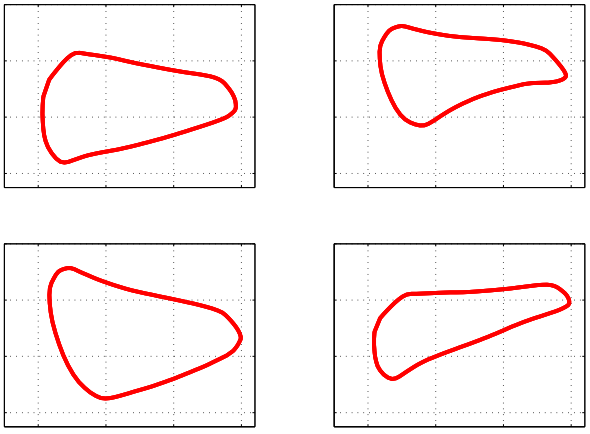


Fig. 4: Figures showing the end point trajectories of all the four feet of the quadruped. The plots in blue correspond to the end point trajectories from simulation data. The plots in red are the reconstruction from the kMPs.

IV. Experimental Results

In this section we discuss the experimental results conducted on the quadrupedal robot Stoch. We first describe the hardware, and then describe the methodologies used to realize walking along with the results.

A. Hardware Description of Stoch

Stoch is a quadrupedal robot (see Fig. 1) designed and developed in-house at IISc, Bengaluru [23]. Model and actuator specifications are summarized in Table II. The legs are assembled with lightweight carbon fiber (CF) tubes for the linkages, and 3D printed poly-lactic acid (PLA) parts for the joints. PWM signals to the servo motors are sent via Adafruit 16-Channel 12-bit: PCA968. Main computation platform used is Raspberry Pi 3-B. The electric power to the robot is supplied from Li-Po batteries. More details about the hardware are described in [23].

total leg length	230 mm	min./max. hip joint	-45 / 45
leg segment length	120 mm	min./max. knee joint	-70 / 70
total mass	3.0 kg	max hip/knee speed	461 /s
max hip/knee torque	32 kg-cm	motor power(servo)	16 W

TABLE II: Hardware specifications of Stoch.

		Cross-covariance				Delay			
		1st	2nd	3rd	4th	1st	2nd	3rd	4th
A.	RT-ET	0.97	0.74	0.93	0.79	0	-0.02	0.02	0.04
	RT-HT	0.99	0.89	0.86	0.82	0	-0.02	-0.05	-0.04
	ET-HT	0.99	0.77	0.88	0.87	0	0.15	-0.08	0.03
B.	RT-RP	0.87	0.94	0.91	0.7	-0.03	-0.07	-0.08	0.16
	RT-RB	0.81	0.84	0.72	0.88	0	-0.02	-0.05	-0.04
	RP-RB	0.96	0.92	0.85	0.72	-0.01	0.0	-0.08	0.18

TABLE III: **RT,RB,RP** are trot, bound and pace gaits obtained from D-RL respectively. **ET** is the trot gait obtained from experiment, and **HT** is the trot gait obtained from the horse. **A.** (resp. **B.**) shows the comparison between **RT,ET,HT** (resp. **RT,RB,RP**).

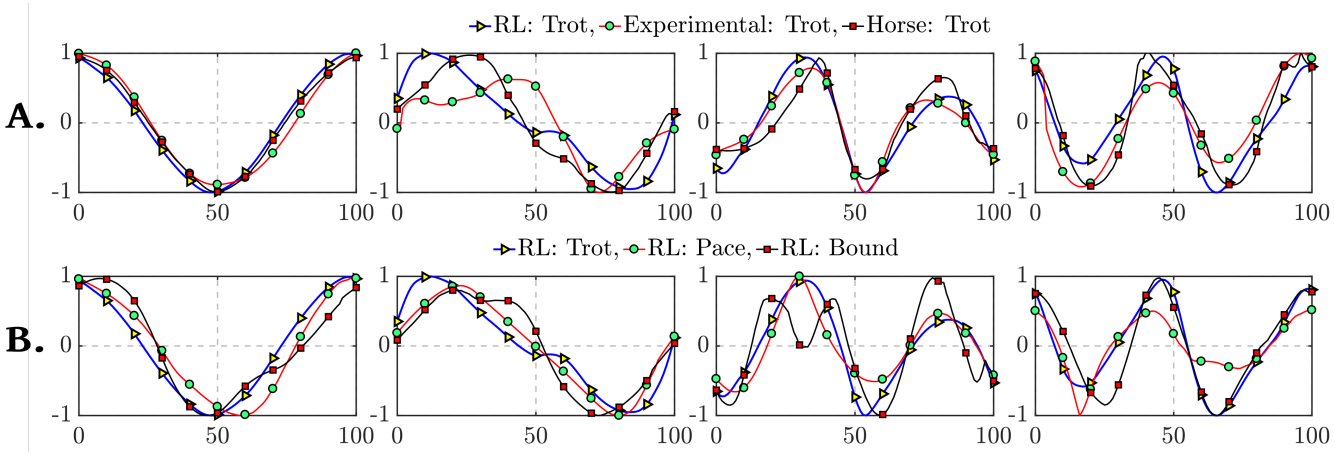


Fig. 5: Here we make two types of comparisons. The top four figures (A) show the comparison between kMPs for the trot gait: hardware trot, simulation trot, and finally horse trot. The bottom four figures (B) show the comparison between kMPs obtained from the learned gaits in simulation: trot, pace and bound. The kMPs (of hardware, simulation or horse) seem to follow a pattern: **VNMW**. Statistical comparison of these patterns are provided in TABLE III

B. Synergy Matrices

The general procedure to obtain the walking gaits for Stoch is as follows. We first reconstruct the trot gait from the kMPs by using the synergy matrix S. If Q_s is the trot data obtained from the simulation, then we obtain S as

$$\mathsf{S} = \mathsf{P}_{\mathbf{inv}} \quad (\mathsf{Q}_s \quad \mathsf{Z}); \tag{10}$$

where $\mathsf{P}_{\mathbf{inv}}$ is the pseudo-inverse of P. This is computed offline. Synergy matrices for other types of gait are obtained by modifying Q_s. For example, to obtain the bound gait, we modify Q_s (by time shifting) in such a way that the front legs are in phase (resp. hind legs). We have also reshaped the end point trajectories in small amounts, and obtained the corresponding Q_s. Since we are manually modifying the data, a better approach would have been to use optimization, which will be explored in future.

C. Results

Having obtained the kMPs, and the synergy matrices for trot, walk, gallop and bound gaits, we reconstruct the joint angle trajectories by using (9) in hardware. These joint angle trajectories are then used as command positions to each of the actuators (see [23, Section IV-B]). Video results showing different types of walking, and robustness tests are shown in https://youtu.be/kiLKSqI4KhE

Due to space constraints, we only show the experimental data for the trot and its derived gaits. We have also provided kMPs extracted directly from the experimental trajectory, and the results are in Fig. 5-A and Table III. It can be verified that experimental kMPs also have a high correlation with the horse and RL kMPs. We have also provided Table IV to compare the speeds of the various gaits obtained.

In order to put these observations in perspective, it is important to discuss the kMPs derived from horse data more. We cannot realize walking by using only horse kMPs. Based on (9), we will need to determine the right synergy matrix **S** that suits the given set of kMPs. It is also important to note that the kMPs derived from horses may not be the optimal choice for our quadruped (there are infinitely many kMPs). On the other hand, due to the fact that we already have (locally) optimal trajectories obtained from training, we focus on computing the synergy matrix from simulation kMPs. With these kMPs we were able to demonstrate not only trotting, but also other derived behaviors like walking, galloping and bounding.

Gait type	Trot	Walk	Gallop	Bound	Modified Trot 1	Modified Trot 2
Speed (m=s)	0.60	0.51	0.51	0.55	0.62	0.59

TABLE IV: Walking speed of Stoch obtained during different gait experiments. The last two columns are the result of reshaping of the end point trajectories of trot.

V. CONCLUSION

We demonstrated walk, trot, gallop and bound gaits in the quadruped robot Stoch experimentally. The key contribution was in the methodology followed in transferring of the D-RL based gaits from simulation to real hardware. Our approaches were mainly motivated by [13] and [15], which pointed to the fact that motion primitives have a very similar underlying structure that are independent of the type of gait and the model. Small variations in gaits and model do not require retraining via D-RL methods from scratch, and kMPs are sufficient for such scenarios. On the other hand, large variations in models and environments require a thorough search, thereby requiring learning based approaches. Since we mainly focused on data driven kMPs in this paper, future work will involve obtaining these motion primitives from continuous functions like polynomials, canonical walking functions or other basis functions that are equally suitable for generating walking gaits.

ACKNOWLEDGMENT

We acknowledge Ashish Joglekar, Balachandra Hegde, Ajay G and Abhimanyu for the PCB design and software development.