Learning Families of Behaviors for Legged Locomotion using Model-Free Deep Reinforcement Learning

Raunaq Mahesh Bhirangi

CMU-RI-TR-20-40
August 2020

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Robotics

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisors, Matthew Travers and Howie Choset, for their support and the freedom they afforded me in defining my academic journey as a master's student. I have covered a lot of ground in learning to be an independent researcher, and it would not have been possible without their guidance and unwavering belief in my ability.

I would also like to thank everybody at the Biorobotics Laboratory for creating an environment conducive to research and intellectual discourse. In particular, I would like to thank Benjamin Freed, Hans Kumar, Naman Gupta and Julian Whitman for all the stimulating discussions, and for making the lab a place I looked forward to going back to every day.

My friends and family have been encouraging and understanding through my journey at CMU, and this thesis would not have been possible without their support and patience. I would particularly like to thank Tejas Kotwal and Roshail Gerard who have been constant sources of inspiration and motivation throughout college and grad school, while also being constant sinks for my frustration. A big shout-out to my housemate of the last two years, Tejas Srinivasan, for being a reliable friend and for making our apartment a homely place to come back to after a tiring day of work.

Last but not the least, I would like to thank BJ Fecich and Peggy Martin for being the most approachable staff members I have known at CMU, and for all that they do behind the scenes to allow for a seamless research experience.

ABSTRACT

Successful deployment of complex systems, such as articulated legged robots, require general solutions to planning and control because the high dimensional state spaces associated with such systems make it impractical to search for a new solution each time the robot encounters a different challenge. To achieve some level of generality, in this work, we present a framework for end to end learning of a set of parameterized families of behaviors that can be modulated by low dimensional sets of control parameters. We draw inspiration from Central Pattern Generators (CPGs), which use networks of oscillators, to form expressive low-dimensional parameterizations of locomotive behaviors. We do not directly use CPGs because their design requires significant domain knowledge and hand tuning. Instead, we turn to model-free deep reinforcement learning (RL) which offers a framework for learning behavioral policies by interacting with the environment, with minimal to no domain knowledge about the robot or its environment. The results from RL are still restrictive because they represent a single behavior whose characteristics cannot be easily modified. Therefore, this work presents a framework that brings together ideas from CPGs and model-free deep RL to enable expressive parameterizations of behaviors to be learned end-to-end by interaction with the environment.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Overview

This thesis addresses the problem of generality in controlling legged robots that have high dimensional state spaces using expressive low dimensional parameterized behaviors. Our approach brings together contemporary ideas in deep reinforcement learning and expressive low-dimensional parameterizations of behavior families called Central Pattern Generators [1] in developing a novel model architecture that is capable of learning expressive families of locomotive behaviors from scratch. We demonstrate the efficacy of our approach by using it to learn families of behavior for two simulated robot environments.

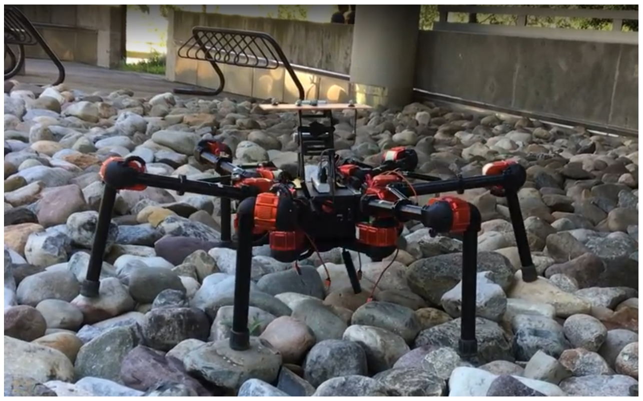


Figure 1.1: The Matt-6 walking on rocks

Our approach is inspired by CPGs, which are often modelled as coupled systems of oscillators and yield stable controllers for robotic systems. Fig. 1.2 shows a schematic representation of a CPG controller for a legged robot. The system of oscillators is mapped to the joint space of the robot, and oscillations result in a coordinated control policy for the legged robot. A policy, based on CPGs, can adapt to its changing environment by modulating parameters of the oscillator system, such as the frequency, amplitude and coupling, which directly correspond to interpretable characteristics of the robot's motion. For instance, the oscillator frequency dictates the robot's gait frequency, the oscillator amplitude may be correlated with the step height or stride length, while the coupling generally corresponds to the robot's footfall pattern.

[1]Biologically, CPGs are neural networks responsible for generating rhythmic signals that are used in a number of biological functions like breathing, chewing and locomotion.

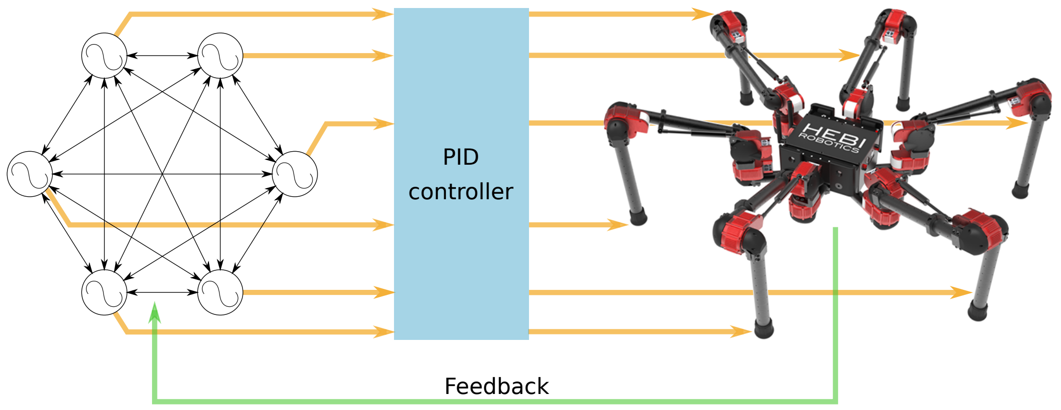


Figure 1.2: Schematic diagram of a CPG for a hexapod robot. The six circles with connecting arrows represent the coupled oscillator system which produces reference signals, indicated by the yellow arrows flowing into the PID controller. These signals ultimately produce motor commands, also indicated by yellow arrows, for the hexapod robot. The CPG controller is a closed loop system as indicated by the green arrow going back from the robot to the system of oscillators.

Despite these advantages, a major drawback of CPG-based approaches is a lack of principled methodologies to design controllers based on them. For example, there exists no clear choice of mapping between the system of oscillators and the actuator space of the robot, making the design of these controllers a tedious, time-consuming process that requires significant domain knowledge and expertise.

Some recent work [29, 19, 30] has attempted to create a general set of tools to devise this oscillator-actuator mapping, but such tools continue to be a work in progress. Therefore, we look beyond the CPG framework and create a novel architecture that can learn families of behaviors that are "CPG-like". This means that we inherit the benefits of the CPG, i.e., robot behavior can be modulated by a low dimensional set of control parameters, while requiring minimal to no domain knowledge on the part of the designer. Our framework retains the oscillators used by CPG models and introduce a learned control policy that maps from the space of oscillators to the actuator space of the robot. This policy can be modulated by the parameters of the oscillator to result in a change in the robot's behavior.

Our approach inherits another advantage of the CPG framework – the ability to damp out perturbations by mapping state feedback to the oscillator [30, 17, 18]. To incorporate feedback from the robot, we introduce a representation model that maps from the state space of the robot to the oscillators. Fig. 1.3 shows a general outline of the different components of the proposed architecture. We seek to formulate this as an end-to-end learning problem to enable the different components of the architecture to be learned without the domain knowledge and expertise required to design a CPG controller.

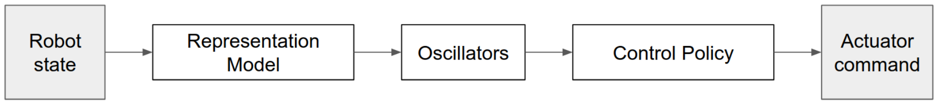


Figure 1.3: An overview of the proposed model. A representation model maps the robot's state to the oscillators. A control policy then maps the output of the oscillators to an actuator command for the robot.

In order to enable end-to-end learning of the proposed model, we use deep reinforcement learning (RL) as the learning framework. Model-free deep RL has been shown to learn policies for legged locomotion for simulated robots [34, 12, 24], making minimal to no assumptions about the robot and its environment. The environment provides the agent with *rewards* at each time step, which depend on the robot's forward progress, power usage etc. Deep RL algorithms enable the robot to learn a policy based only on its interactions with the environment by maximizing the cumulative sum of these rewards. This ability of model-free deep RL to enable end-to-end learning of behavioral policies, comes at the cost of the lack of an expressive parameterization of the policy. The agent learns a single policy whose characteristics cannot be easily modulated. Our proposed architecture provides a framework that enables deep RL algorithms to learn a family of multiple behaviors, that can be modulated by a small set of parameters once trained, in a single pass of the algorithm.

To summarize, this thesis presents:

(a) an algorithm for learning CPG-like families of legged locomotion behaviors, eliminating the domain knowledge and hand tuning required to design a CPG controller.

(b) a technique that expands the capability of deep reinforcement learning to learn a family of locomotive behaviors, instead of a single behavior, that can be modulated by extrinsic parameters at runtime.

We demonstrate an ability to learn a family of "CPG-like" cyclical gait behaviors in a single pass of the deep RL algorithm. The learned policy can be modulated by a single control parameter to change the frequency of the gait executed by the robot. We intend this work to be a proof of concept for the idea of learning a

CPG-like controller for legged robots. We use this architecture as a starting point for learning a model that allows the locomotion policy to be modulated by a larger set of parameters like coupling, swing and stance frequencies, and step height.

1.2 Outline

The rest of this thesis is organized as follows:

In chapter 2, we briefly discuss existing ideas in central pattern generator models, deep reinforcement learning and variational inference relevant to this work. We also review existing techniques related to this work, and distinguish our contribution. In chapter 3, we give an overview of the proposed model, present a novel neural network architecture, and elaborate on the methodology used in training the model. In chapter 4, we apply this technique to a number of different legged locomotion environments, and demonstrate the efficacy of the approach. In chapter 5, we conclude and present directions for future work.

Chapter 2

BACKGROUND AND RELATED WORK

We present a novel policy learning architecture that combines the flexibility of model-free deep RL with the low-dimensional parameterizations CPGs offer. In this chapter, we discuss CPGs in the context of legged robotics, which motivate the presented architecture. We then move on to deep RL which provides the framework for learning behaviors using the proposed architecture. Finally, we review variational autoencoders, which this architecture uses as a representation model.

2.1 Central Pattern Generators

CPGs in animals are biological neural circuits that produce rhythmic outputs and are instrumental in numerous biological functions such as digestion, locomotion and respiration in a wide variety of animals [17]. In robotics, they have gained popularity as rhythmic pattern generators that can be used to produce cylical locomotive trajectories for a number of highly articulated robots. They have been used to generate swimming gaits for lamprey and eel-like robots [3, 8], slithering gaits for snake robots [6, 18] and walking gaits for a wide variety of legged robots [11, 30, 2].

Typically, CPGs encode trajectories with a small number of control parameters which can be used to modulate the behavior of the robot. This reduction in dimensionality allows control of the robot using simpler high-level control signals like gait frequency and coupling, as opposed to controlling each degree of freedom. Additionally, CPGs also exhibit limit cycle behavior, i.e. they produce stable periodic signals. This means that any transients or perturbations to the nominal behavior prescribed by the limit cycle are naturally and quickly damped out. These stability properties, along with the reduction in dimensionality of the control problem, make CPGs an attractive choice of controller for legged locomotion.

A major drawback of CPG-based approaches, however, is a lack of design methodologies for building these controllers [30]. While there has been some recent work in developing a generic set of tools to design CPG models [19, 29], they usually need to be handcrafted for specific robots and associated gait behaviors. In most cases, each limb of the legged robot is associated with an oscillator [29, 17, 31]. The mapping between the oscillator and the actuator space of the robot is crucial to the performance of the controller and currently requires significant domain knowledge and expertise to design.

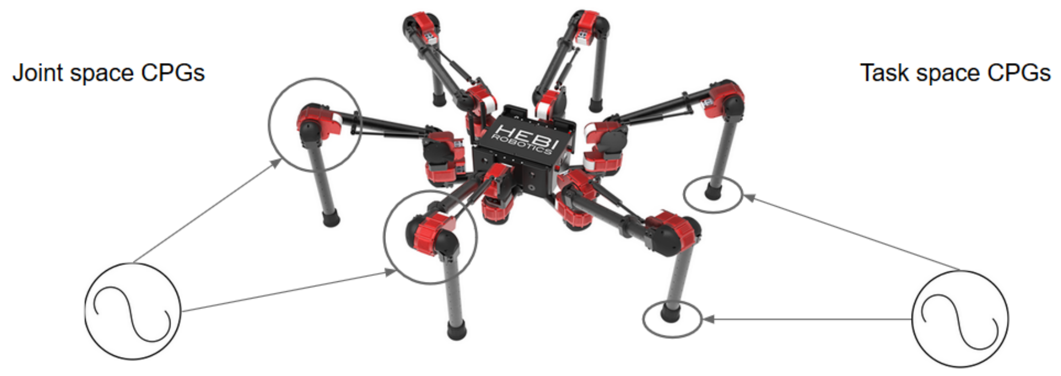


Figure 2.1: Example mappings between CPG oscillators and the actuator space of the robot. The joint space CPG, on the left, maps the output of an oscillator to positions of the joints for each limb. The task space CPG, on the right, maps the output of an oscillator to the motion of the foot of the robots.

Fig. 2.1 shows two examples of such mappings – a task space CPG [21] and a joint space CPG [29, 17, 31]. In the context of legged locomotion, a task space CPG implementation directly maps the state of each oscillator to the position of the corresponding end-effector (foot). This allows for straightforward specification of desired foot trajectories when designing the controller by modulating the shape of the limit cycle [21]. The control inputs to the robot however, must be specified as a trajectory in the joint space of the robot, which can only be obtained by solving the inverse kinematics problem to compute leg joint angles from foot locations. This makes it cumbersome to ensure continuity of the specified trajectory in joint space and avoid joint limits.

A joint space CPG implementation eliminates these problems by directly mapping the oscillator state to the joint outputs of each limb, allowing the designer to directly prescribe joint space trajectories. But since the trajectories of the feet now result from the (often complicated) forward kinematic transformation of the joint space trajectories, it becomes difficult to specify desirable characteristics of the robot's gait such as the step height and stride length.

Furthermore, most gaits observed in legged animals consist of a stance phase, where the foot is grounded, and a swing phase where the foot is in the air. Often, the duration of the swing phase is much shorter than the duration of the stance phase [30]. While there has been some work directed at defining mappings between oscillators and the robot's joint space that can capture such behaviors, regardless of the joint space or task space perspective, they often result in undesirable characteristics such as slipping feet when implemented on a robot.

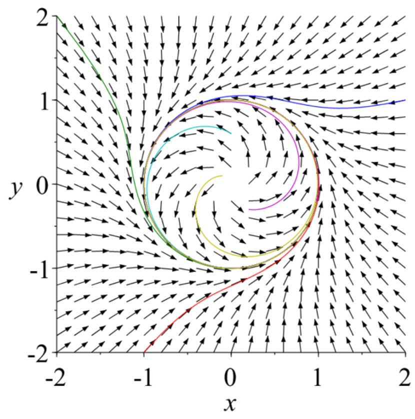


Figure 2.2: Vector field described by Eq. 2.1. Some examples trajectories of evolution along the vector field are shown in different colors. From any initial state, the trajectories converge to a stable limit cycle.

The choice of this oscillator-actuator mapping is therefore crucial to the effectiveness of a CPG controller. Since there are no readily available tools to effectively design the mapping, we steer away from conventional CPGs. We present an architecture that learns this mapping using deep neural networks while optimizing performance on a task objective. Performance is defined in terms of a reward function which in our case incentivizes forward progress in a given direction, while penalizing foot slipping and hitting joint limits. This enables our approach to avoid the difficulties in designing this mapping described above.

Inspired by the CPGs, we use a vector field containing a stable limit cycle as our oscillator model. Drawing from [29], in this work we employ a weakly nonlinear dynamical system that exhibits a stable, circular limit cycle as shown in Fig. 2.2.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{\omega}{r} \begin{bmatrix} -y \\ x \end{bmatrix} + \frac{\alpha}{r} \left(r_0^2 - r^2 \right) \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.1}$$

where $[x, y]^T$ is the current position of the oscillator in $\mathbb{R}^2$, $r = \sqrt{x^2 + y^2}$, ω is the angular frequency of oscillation along the limit cycle, r_0 is the radius of the circular limit cycle, and α corresponds to the strength of forcing towards the limit cycle. Fig. 2.3 shows the effect of varying the parameters, ω and α, on the vector field.

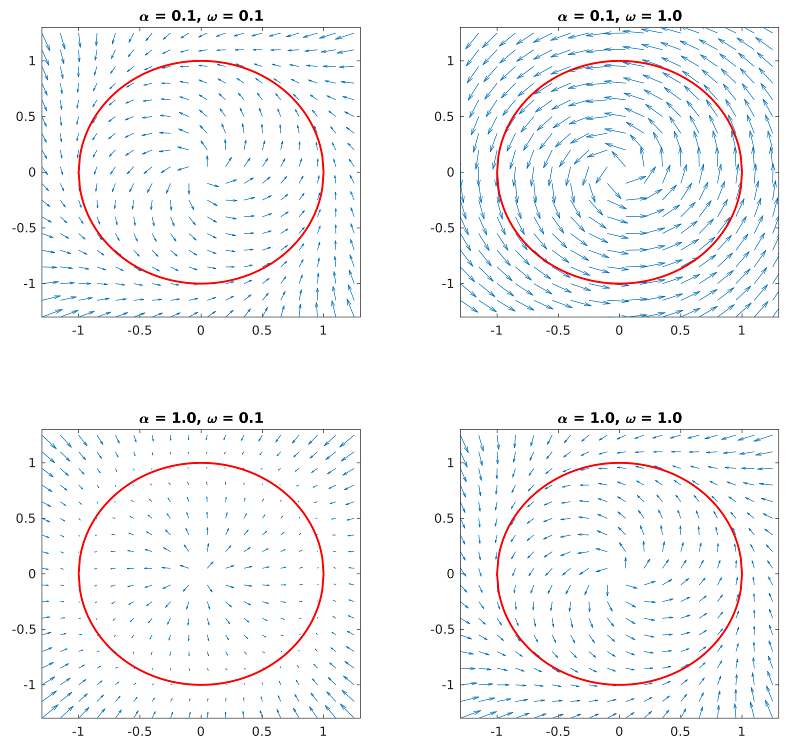


Figure 2.3: Effect of varying ω and α on the vector field described by Eq. 2.1. An increase in the value of ω results in a larger component of the vector along the tangent to the limit cycle, and thus an increasing in the angular frequency of the motion of the oscillator. An increase in the value of α results in a larger component of the vector field normal to the limit cycle, and thus increases forcing onto the limit cycle. Proportionate increases in α and ω do not affect the relative magnitudes of the components of the vector field.

2.2 Deep Reinforcement Learning

In recent years, deep RL has made significant progress in solving a number of benchmark robotic simulation tasks demonstrating its applicability to continuous control tasks [34, 24, 12]. Standard reinforcement learning problems are posed as an agent interacting with a Markov Decision Process described by $(\mathcal{O}, \mathcal{S}, \mathcal{A}, \gamma, P, r)$ [38]. At every time step t, the agent receives an observation, $o_t \in \mathcal{O}$, estimates the state, $s_t \in \mathcal{S}$, and performs an action $a_t \in \mathcal{A}$ based on its policy, $\pi_\theta(a_t|s_t)$. The environment then provides the agent with a reward $r(s_t, a_t)$ and the agent transitions to the next state, s_{t+1} according to the environment transition probability $P(s_{t+1}|s_t, a_t)$. In this work, we restrict ourselves to fully observable environments, i.e., the state, s_t, can be fully determined given the observation, o_t. We restrict our attention to infinite-horizon tasks, i.e., tasks without a fixed time limit. The goal of the agent is to maximize expected cumulative return,

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \tag{2.2}$$

with respect to the parameters of the policy, θ. The discount factor, γ, is used to weigh immediate rewards higher than future rewards.

A number of techniques have been developed over the years to maximize the expected sum of rewards described in Eq. 2.2 [12, 34, 24]. Methods that use a stochastic estimate of the gradient of the objective with respect to the parameters of the policy (the policy gradient) in optimizing the objective are called *policy gradient methods* [38]. Actor-critic methods are a type of policy gradient methods that learn an approximation to the policy (actor) as well as the value function (critic), in order to estimate the policy gradient. The value of a state, $s_t = s$, under a policy π, denoted by $v_\pi(s_t = s)$, refers to the expected return when starting in a state, s at time t, and following policy π for the rest of the episode. The value function with respect to the policy, π, can thus be defined as,

$$v_\pi(s_t = s) = \mathbb{E}_{\pi_\theta}\left[\sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}) \middle| s_t = s\right] \tag{2.3}$$

Actor-critic methods in deep RL use deep neural networks as function approximators for the policy and the value functions [35, 34]. In this work, we use Proximal Policy Optimization (PPO) [34], a type of actor-critic method, for policy learning. Drawing from [26], we use partial-episode bootstrapping where the critic continues to bootstrap when episode termination occurs due to a timeout. We also use generalized advantage estimation (GAE) [33], a technique that provides significant improvements in sample efficiency by reducing the variance of the policy gradient estimates at the cost of some bias.

In training our policy, we make use of intrinsic reward signals to aid learning. Intrinsic rewards are rewards that the agent gives itself in addition to the reward obtained from the environment. The intrinsic reward is added to the extrinsic reward obtained from the environment at every time step, and the policy is then trained to maximize the cumulative discounted sum of the intrinsic and extrinsic rewards over time. Intrinsic rewards have previously been used to encourage RL agents to perform actions that improve their ability to predict the consequences of their actions [16, 25, 27]. Our model uses a similar approach to train agents to choose actions based on predictions of the desired next state using the CPG-like vector field embedded in the policy network. The details of this implementation are described in Ch. 3.

Inductive biases refer to incorporating prior knowledge about the world within the design of neural network architectures, so as to influence the space of internal models considered by the machine learning system. CNNs, for instance, use translational invariance in space as an inductive bias and have been shown to be very effective on vision-related machine learning tasks [4]. Most model-free approaches to deep RL learn behavioral policies for robots using generic multilayer perceptrons to represent the policy. There has been limited work in designing neural network architectures to add an inductive bias to the learning procedure so as to enhance certain properties like generalizability, sample efficiency, and robustness [37, 39]. In this work, we aim to add an inductive bias for the periodic nature of legged locomotion by embedding a vector field similar to a CPG within the neural architecture which is used to represent the controller. This enables our model to learn a family of behaviors in a single pass of the learning algorithm, that can be effectively modulated by a small set of parameters.

We presently restrict our scope to policies modulated by the gait frequency; we discuss extension to other gait characteristics generally seen in CPG models, like limb coupling and swing and stance phase durations in Ch. 5. Some recent works [10, 1] present techniques for learning a diverse set of behaviors in a single pass of a deep RL algorithm. The proposed model can be used in conjunction with these methods to replace the policy networks and provide a continuous modulation of the different learned behaviors emerging from these frameworks.

2.3 Variational Autoencoders

A number of recent works in deep RL use variational autoencoders (VAEs) to learn a representation model for the agent's observations [13, 14, 40]. Autoencoders are neural network architectures that consist of two components – an encoder and a decoder – and are used to learn latent encodings of data by minimizing error between the input data and the reconstruction obtained by passing it sequentially through the encoder and the decoder. VAEs reformulate the learning problem as variational inference over a directed graphical model shown in Fig. 2.4. The generative process consists of two steps: (i) The latent variable $\mathbf{z}$ is sampled from a prior $p_\theta(\mathbf{z})$, followed by (ii) the observed variable $\mathbf{x}$, which is sampled from a conditional distribution $p_\theta(\mathbf{x}|z)$. The prior $p_\theta(\mathbf{z})$ and the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ are assumed to come from a parametric family of distributions. In most cases, the distribution parameters θ are unknown, and the posterior distribution, $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable. A parameterized distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is assumed to be an approximation to the posterior $p_\theta(\mathbf{z}|\mathbf{x})$. This allows us to perform efficient inference by sharing parameters to allow for generalization across the posterior estimates for all latent variables [22, 28]. Approximate inference is then performed by optimizing a lower bound on the log likelihood of the observed data, the Evidence Lower Bound (ELBO) given by,

$$\log p_\theta(\mathbf{x}^{(i)}) \geq -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})\|p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right] \tag{2.4}$$

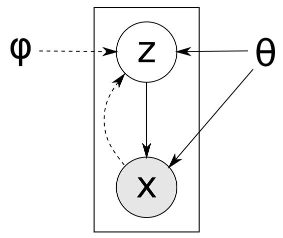


Figure 2.4: Schematic representation of the directed graphical model. Solid lines denote the generative distribution $p_\theta(\mathbf{x}, \mathbf{z})$, and dotted lines denote the variational approximation, $q_\phi(\mathbf{z}|\mathbf{x})$

The parameters θ, ϕ can be optimized to maximize this lower bound by stochastic backpropagation as demonstrated by [22, 28]. In some cases, we would like the components of the generative process to depend on additional observed random variables. The VAE framework can be generalized to incorporate such conditioning on another known random variable, $\mathbf{y}$, to the posited generative model shown in Fig. 2.5. The probability distributions, $p_\theta(\mathbf{x}, \mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$ are now replaced by probability distributions conditioned on $\mathbf{y}$, $p_\theta(\mathbf{x}, \mathbf{z}|\mathbf{y})$ and $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$ respectively. The resulting model is called a conditional VAE [36], and admits a similar lower bound to the conditional log-likelihood of the data, given by,

$$\begin{aligned}\log p_\theta(\mathbf{x}^{(i)}|\mathbf{y}) \geq -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}, \mathbf{y})\|p_\theta(\mathbf{z}|\mathbf{y})) \\ + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)},\mathbf{y})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}, \mathbf{y})\right].\end{aligned} \tag{2.5}$$

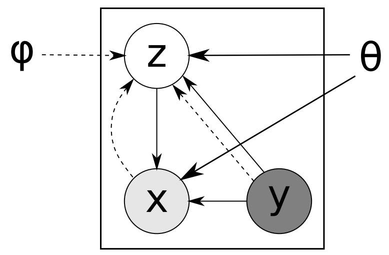


Figure 2.5: Schematic representation of the directed graphical model for a CVAE. Solid lines denote the generative distribution $p_\theta(\mathbf{x}, \mathbf{z}|\mathbf{y})$, and dotted lines denote the variational approximation, $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$

CVAEs allow for more flexibility in modelling the generative process as compared to VAEs. The same latent variable, **z**, can now result in different generative distributions on the observed variable, **x**, depending on **y**. Similarly, we can have different approximate posterior distributions on **z** given **x**, depending on the value of **y**. Traditional VAEs can be seen as a special case of CVAEs where the variables **y** are absent or assume a constant value for all data. In the rest of this thesis, our methodology is described for CVAEs but can be used directly with VAEs by simply dropping the variable **y**. We provide experimental results for models using VAEs as well as CVAEs in Ch. 4.

There have been a number of recent works extending this framework to sequential data, and learning state space models [5, 14, 20]. Here, we extend the CVAE to a sequential generative process represented by a hidden Markov model (HMM) as shown in Fig. 2.6. The prior in this case is a Markovian prior, $p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)$ and the observed variable is sampled from a conditional distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{y}_t)$ at every time step. We choose an approximate posterior, $q_\phi(\mathbf{z}_t|\mathbf{x}_t, \mathbf{y}_t)$, and derive the following lower bound on the conditional log likelihood of the observed data following a similar procedure to [5]:

$$\log p_\theta(\mathbf{x}_{1:T}^{(i)}|\mathbf{y}_{1:T}^{(i)}) \geq \sum_{t=1}^{T} \Big[-D_{KL}\left(q_\phi(\mathbf{z}_t|\mathbf{x}_t^{(i)}, \mathbf{y}_t^{(i)}) \Big\| \, p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t^{(i)}) \right) + \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t^{(i)}, \mathbf{y}_t^{(i)})} \left[\log p_\theta(\mathbf{x}_t^{(i)}|\mathbf{z}_t, \mathbf{y}_t^{(i)}) \right] \Big] . \quad (2.6)$$

There has also been some recent research on using variational autoencoders to learn discriminable skills for robots using context variables [1, 10]. These works aim to learn a diverse set of behaviors without an extrinsic reward function. The learned policy represents a discrete set of distinguishable behaviors, each corresponding to

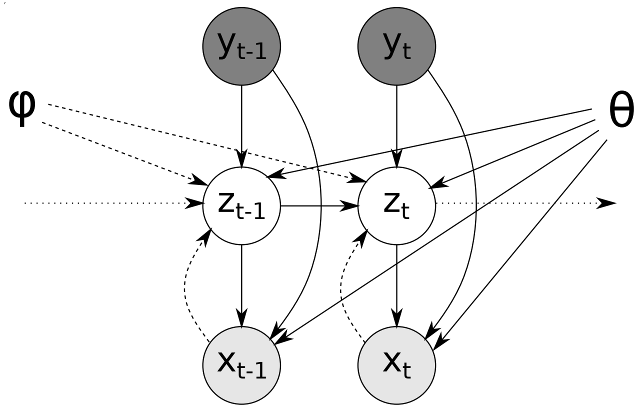


Figure 2.6: Schematic representation of the HMM directed graphical model. Solid lines denote the generative distribution $p_\theta(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}|\mathbf{y}_{1:T})$, which factorizes as $\prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)p_\theta(\mathbf{x}_t|\mathbf{z}_t, \mathbf{y}_t)$

a region in the latent space. Our approach differs from these approaches in that we learn a continuously parameterized family of behaviors that can be modulated by a small set of parameters. Our proposed architecture would be compatible with the methodology presented in these works, and would enable the diverse set of skills learned to be individually modulated.

Chapter 3

LEARNING FAMILIES OF BEHAVIORS

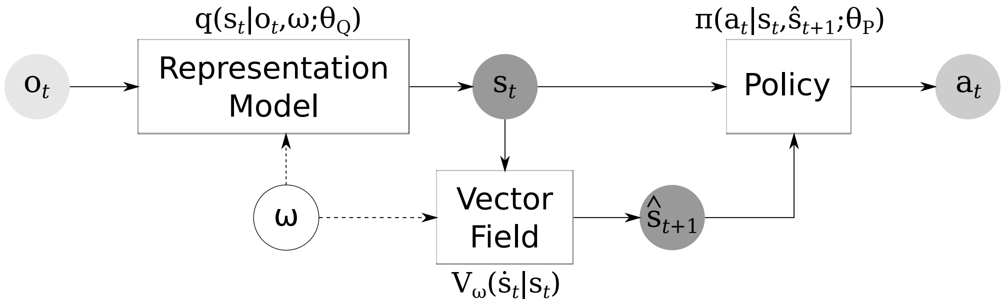
In this chapter, we present our approach for learning families of behaviors for legged locomotion using deep reinforcement learning. Since we would like our policy to represent a family of behaviors modulated by a small set of parameters, P, we define our policy as $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t, P)$. We presently restrict our attention to the case where P only consists of a single parameter, ω, which will be shown to correspond to the frequency of the gait executed by the robot. In the following sections, we outline the deep neural network architectures along with the training procedure that helps us achieve end-to-end learning of a policy representing a family of behaviors for legged locomotion.

3.1 Model Architecture

We create an architecture consisting of three components: a representation model, an encoded vector field and a policy (Fig. 3.1). The representation model transforms the observation $\mathbf{o}_t \in \mathcal{O}$, at time t, to an encoded state $\mathbf{s}_t \in \mathcal{S}$, given the parameter ω. A vector field $V_\omega : \mathcal{S} \to T_s\mathcal{S}$, parameterized by the same parameter ω, maps the encoded state $\mathbf{s}_t$ to an encoded velocity $\dot{\mathbf{s}}_t \in T_s\mathcal{S}$. Finally, a policy network maps the current encoded state $\mathbf{s}_t$, and the next encoded state predicted by the vector field, $\hat{\mathbf{s}}_{t+1}$, to a probability distribution over possible actions $\mathbf{a}_t$. The notations for these components are:

$$\begin{aligned}
&\text{Representation Model:} && q(\mathbf{s}_t|\mathbf{o}_t, \omega; \theta_Q) \\
&\text{Encoded Vector Field:} && V_\omega(\dot{\mathbf{s}}_t|\mathbf{s}_t) \qquad (3.1)\\
&\text{Policy:} && \pi(\mathbf{a}_t|\mathbf{s}_t, \omega) = \pi(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{s}}_{t+1}; \theta_P)
\end{aligned}$$

The vector field is predefined and can be modulated by the parameter ω. The representation model and the policy will be trained to jointly maximize a task-dependent cumulative reward function and the likelihood of the next encoded state $\mathbf{s}_{t+1}$ being the same as the predicted next encoded state, $\hat{\mathbf{s}}_{t+1}$. This will enable the model to learn behaviors modulated by ω that can achieve the task objective with their state distribution being consistent with the prescribed vector field.



$q(s_t|o_t,\omega;\theta_Q)$
$\pi(a_t|s_t,\hat{s}_{t+1};\theta_P)$
o_t
Representation Model
s_t
Policy
a_t
ω
Vector Field
$\hat{s}_{t+1}$
$V_\omega(\dot{s}_t|s_t)$

Figure 3.1: Model architecture. The representation model maps the current observation, $\mathbf{o}_t$, to the state, $\mathbf{s}_t$, in the encoded state space. The vector field takes in the gait frequency ω as input and maps $\mathbf{s}_t$ to a predicted state $\hat{\mathbf{s}}_{t+1}$. The current and predicted states are then passed through the policy to output a distribution over actions $\mathbf{a}_t$.

Representation Model

The representation model is used to map the observation space of the robot $\mathcal{O}$ to the encoded state space $\mathcal{S}$. As described in Sec. 2.3, we use a sequential variant of the CVAE as described in Sec. 2.3 as the representation model. We assume that the evolution of the state distribution under the policy can be modelled as a hidden Markov model as shown in Fig. 3.2. We use a Markovian prior derived from the vector field, as described in the following sections. The posterior distribution is approximated by a probabilistic encoder $q(\mathbf{s}_t|\mathbf{o}_t,\omega;\theta_Q)$, represented as a deep neural network with parameters θ_Q, that maps $\mathbf{o}_t$ to a probability distribution over $\mathbf{s}_t$. We are free to choose the form of the approximate posterior over the encoded states, $q(\mathbf{s}_t|\mathbf{o}_t,\omega)$, and we assume it to be a multivariate Gaussian with diagonal covariance,

$$q(\mathbf{s}_t|\mathbf{o}_t,\omega;\theta_Q) = \mathcal{N}(\mathbf{s}_t;\mu_{\mathbf{o}_t,\omega},\sigma^2_{\mathbf{o}_t,\omega}\mathbf{I}), \tag{3.2}$$

where the mean and standard deviation, $\mu_{\mathbf{o}_t,\omega}$ and $\sigma_{\mathbf{o}_t,\omega}$, are outputs of the encoding deep neural network. We choose the dimensionality of the encoded state space, $\mathcal{S}$, to be the same as that of the observation space, $\mathcal{O}$, to minimize loss of information since we are operating in fully observable environments.

Vector Field

One of the core ideas behind our proposed model is to learn a policy whose state distribution evolves according to a vector field that contains a stable limit cycle. We would like to then modify the behavior of the robot by modulating parameters of this vector field. In order to achieve this, we first specify a desired vector field

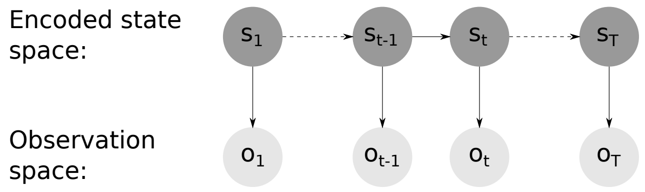


Figure 3.2: The state distribution of a trajectory obtained by executing the policy in the environment is modelled as a Hidden Markov Model.

containing a stable limit cycle, and use this to define a prior on the evolution of the encoded state for the HMM shown in Fig. 3.2. The policy is learned so as to perform actions that result in states dictated by the vector field. The details of how this is achieved are explained in the following sections.

Since the representation model and the policy are parameterized by a deep neural network, we expect our architecture to be able to learn arbitrary mappings between the encoded state space and the robot's actuator space. We assume that this flexibility in function approximation offered by deep neural networks will allow us to learn effective behaviors for any choice of a vector field that contains a stable limit cycle. We use the vector field specified in Eq. 2.1 that is comprised of a weakly nonlinear oscillator with a stable circular limit cycle. We rewrite it here for clarity of presentation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{\omega}{r} \begin{bmatrix} -y \\ x \end{bmatrix} + \frac{\alpha}{r} \left(r_0^2 - r^2 \right) \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.3}$$

The parameter, ω represents the angular frequency of the oscillator's motion around the limit cycle. The idea is to learn a policy whose encoded state distribution evolves according to this vector field. The limit cycle would then represent the nominal gait behavior exhibited by the robot, and modulating the parameters of the vector field will allow us to vary the nominal gait behavior. Therefore, ω corresponds to the angular frequency of this nominal gait behavior, and is henceforth referred to as the gait frequency parameter.

The vector field is defined over a two-dimensional manifold and the encoded state space will naturally have many more dimensions than two. At this point, we chose to group the component dimensions of the encoded state space into pairs, where each pair is then propagated along the vector field for one time step to form $\hat{\mathbf{s}}_{t+1}$. We could have chosen to define a higher dimensional vector field but chose to use groups of two-dimensional ones to keep our network architecture (and our understanding of it) as simple as possible.

Policy

As previously stated, we would like the learned policy to be modulated by the gait frequency parameter. The policy model must therefore be conditioned on the current state as well as the gait frequency parameter, ω. To achieve this, we use the vector field which depends on ω to predict the state at the next time step $\hat{\mathbf{s}}_{t+1}$. The current encoded state, $\mathbf{s}_t$, estimated by the representation model, along with the predicted next state, $\hat{\mathbf{s}}_{t+1}$, are input to the policy network which is represented as a deep neural network, $\pi(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{s}}_{t+1}; \theta_P)$, with parameters θ_P. The policy network outputs a probability distribution over the action space, $\mathcal{A}$, given by,

$$\pi(\mathbf{a}_t|\mathbf{s}_t, \omega) = \mathcal{N}(\mathbf{a}_t; \mu_{\mathbf{s}_t,\hat{\mathbf{s}}_{t+1}}, \sigma^2_{\mathbf{s}_t,\hat{\mathbf{s}}_{t+1}}) \tag{3.4}$$

where the mean and standard deviation, $\mu_{\mathbf{s}_t,\hat{\mathbf{s}}_{t+1}}$ and $\sigma_{\mathbf{s}_t,\hat{\mathbf{s}}_{t+1}}$, are outputs of a deep neural network.

To summarize the operation of the architecture, the representation model serves as a feedback model and maps the robot's observation to the encoded state space. The vector field uses this encoded state to predict the encoded state at the next time step, and the policy predicts an action that would result in this predicted state. We would like to train the architecture so that modulating the gait frequency parameter, ω, will result in a change in the observed frequency of the robot's gait. In the following sections, we describe the methodology that helps us achieve this while simultaneously learning to solve the task objective.

3.2 Reward Function

Depending on the particular task objective, the agent receives an extrinsic reward from the environment at every time step, t, which we denote by r^e_t. In our case, this reward consists of a positive component corresponding to the forward progress made by the robot, and a small negative component penalizing high actuator torques. The extrinsic reward provides the agent with a reinforcing signal that enables it to learn walking behaviors. In order to ensure that the policy state distribution follows the vector field and to encourage actions that result in states consistent with the vector field, we provide the agent with an intrinsic reward, r^i_t at every time step, that penalizes the agent for departure from the vector field. More precisely, the intrinsic reward is defined as

$$r_t^i = -\beta \left\| s_t - \left[s_{t-1} + \int_{t-1}^{t} V(s_\tau) \mathrm{d}\tau \right] \right\|^2, \tag{3.5}$$

where $\beta > 0$ is a hyperparameter. For ease of implementation, we use Euler's method to integrate for one time step, the encoded velocity dictated by the vector field. The total reward, r_t, at every time step is given by $r_t = r_t^i + r_t^e$.

3.3 Training the Policy Network

To enable the agent to solve the locomotive task at hand, the representation model and the policy network are jointly trained to maximize cumulative reward over time. Since we would like the policy to work for a range of gait frequencies, we maximize cumulative reward over a pre-specified distribution over gait frequencies, $\omega \sim p(\omega)$:

$$\max_{\theta_P, \theta_Q} \mathbb{E}_{\omega \sim p(\omega)} \mathbb{E}_{\pi_{\theta_P}, q_{\theta_Q}} \left[\sum_t r_t \right]. \tag{3.6}$$

While our architecture may be used with any policy gradient algorithm, in the experiements presented here, we use Proximal Policy Optimization [34], an on-policy actor-critic algorithm to optimize this objective. We use generalized advantage estimation [33] to estimate the advantage values required to compute the policy gradient.

Since we would like the encoded state distribution induced by the policy to follow the specified vector field, we use the vector field to specify a prior for the representation model. Let ψ be the parameters of the true distribution, $p_\psi(\mathbf{s}_{1:T}, \mathbf{o}_{1:T} | c)$. We assume a Markovian prior on the distribution of the encoded state, $\mathbf{s}_t$, at time t given by,

$$p_\psi(\mathbf{s}_t | \mathbf{s}_{t-1}, \omega) = \mathcal{N}\left(\mathbf{s}_t; \mathbf{s}_{t-1} + \int_{t-1}^{t} V_\omega(\mathbf{s}_\tau) \mathrm{d}\tau, \sigma_o^2 \mathbf{I} \right), \tag{3.7}$$

$$\hat{\mathbf{s}}_t = \mathbf{s}_{t-1} + \int_{t-1}^{t} V_\omega(\mathbf{s}_\tau) \mathrm{d}\tau$$

which is a diagonal Gaussian centered at the encoded state $\hat{\mathbf{s}}_t$ predicted by the vector field, given the encoded state at the previous time step, $\mathbf{s}_{t-1}$, and the gait frequency parameter, ω.

The representation model is trained by maximizing the Evidence Lower Bound (ELBO) over the policy state distribution. We would like the representation model to work for different values of the gait frequency prameter, ω. Therefore, we maximize the expectation of the variational bound for the HMM described in Sec. 3.1 under the pre-specified distribution $p(\omega)$. For the choice of approximate posterior, $q(\mathbf{s}_t|\mathbf{o}_t, \omega; \theta_Q)$, this maximization can be expressed as

$$\max_{\theta_Q,\psi} \mathbb{E}_{\omega\sim p(\omega)} \left[\sum_t \mathbb{E}_q \left[-\log \frac{q(\mathbf{s}_t|\mathbf{o}_t, \omega; \theta_Q)}{p_\psi(\mathbf{s}_t|\mathbf{s}_{t-1}, \omega)} + \log p_\psi(\mathbf{o}_t|\mathbf{s}_t, \omega) \right] \right] \tag{3.8}$$

Since the true distribution parameters, ψ, are not known, we represent the decoding distribution $p_\psi(\mathbf{o}_t|\mathbf{s}_t)$ by a deep neural network.

The overall optimization problem can now be written as a combination of Eqs. 3.6 and 3.8,

$$\max_{\theta_P,\theta_Q,\psi} \mathbb{E}_{\omega\sim p(\omega)} \mathbb{E}_{\pi_{\theta_P}, q_{\theta_Q}} \sum_t \left[r_t + \lambda \left[-\log \frac{q(\mathbf{s}_t|\mathbf{o}_t, \omega; \theta_Q)}{p_\psi(\mathbf{s}_t|\mathbf{s}_{t-1}, \omega)} + \log p_\psi(\mathbf{o}_t|\mathbf{s}_t, \omega) \right] \right], \tag{3.9}$$

where $\lambda > 0$ is a scalar used to weigh the importance of the variational loss against the importance of the policy gradient loss.

Based on Eq. 3.9, it is evident that we are jointly optimizing the expected cumulative sum of rewards and the expected variational lower bound for the representation model along trajectories resulting from the execution of the policy. The rewards are a combination of extrinsic and intrinsic rewards. Maximizing extrinsic rewards results in good performance on the task objective, while maximizing intrinsic rewards results in actions that are consistent with the predictions of the vector field. Further, maximizing the variational lower bound for the representation model, along trajectories resulting from execution of the policy forces the corresponding encoding state distributions to follow the vector field.

Chapter 4

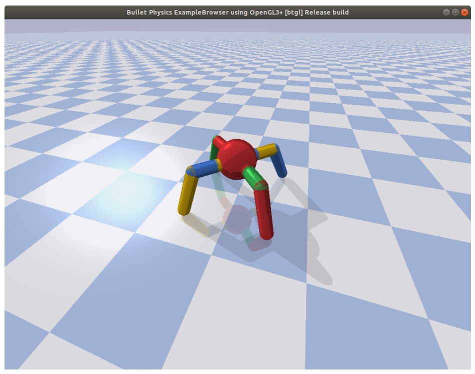
EXPERIMENTS AND RESULTS

In this chapter, we explore the capabilities of the presented framework through experiments in simulation. We start off with the details of the experiments – the simulation environments used to test our algorithms, hyperparameters used in the experiments, and the metric used for evaluation. This is followed by two sets of experiments – learning a family of behaviors whose gait frequency can be modulated by the gait frequency parameter, ω, and learning individual behaviors whose gait frequency can be pre-specified before training.

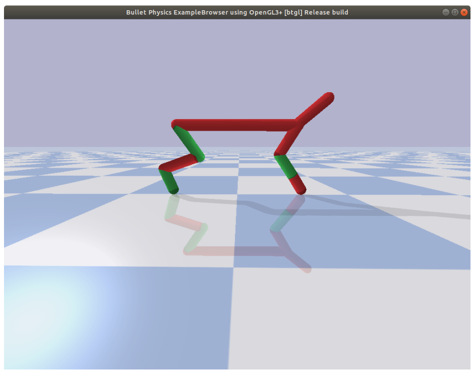
4.1 Experimental details

Environments

We demonstrate the efficacy of the presented algorithm on two continuous control environments from the Pybullet Gym Environment Suite [7], namely the `AntBulletEnv-v0`, and the `HalfCheetahBulletEnv-v0` shown in Fig. 4.1. The observation from the environment consists of the height of the agent's center of mass (CoM) from the ground, velocity of the CoM, the orientation of the agent, joint angles and velocities, and binary indicators for the feet being in contact with the ground. The agent's action space is the space of feasible motor torques for each of its joints.


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

(a) `AntBulletEnv-v0`


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

(b) `HalfCheetahBulletEnv-v0`

Figure 4.1: The environments used to test the proposed algorithm

Training Details

We use Proximal Policy Optimization, PPO [34], to compute the policy gradient for Eq. 3.6. The hyperparameters used have been listed in Tab. 4.1. Hyperparameters specific to particular experiments have been listed in the following sections. The encoder and decoder networks used in the representation model, as well as the policy network, consist of two fully connected layers with 256 units each and tanh activation for every non-output layer. The value network consists of two tanh-activated hidden layers with 64 units each. We also use frameworks for vectorized environments and normalization of rewards and observations from [23, 9]. The parameters of the vector field described in Eq. 2.1 are set to be $\alpha = \omega/25$ and $r_0 = 2.4$. The dimensionality of the encoded state space is chosen to be equal to the dimensionality of the observation space of the robot.

Number of workers	8
Number of minibatches	32
Number of optimization epochs	10
Learning rate	5×10^{-5}
PPO clip parameter	0.2
λ for GAE	0.95
Discount Factor, γ	0.99
Optimizer	Adam

Table 4.1: Hyperparameters used in the experiments

In the experiments described in Sec. 4.2, we consider two variants of the representation model, one with a CVAE and another with a VAE, as described in Sec. 2.3. We refer to these as the CVAE and VAE variants respectively. In the CVAE variant, both the encoder and the decoder must be conditioned on the gait frequency parameter, ω. We use a tanh-activated hidden layer of size two times the dimensionality of the robot's observation space, to process the gait frequency parameter, ω. The output of this layer is then appended to the observation, $\mathbf{o}_t$ or the encoded state, $\mathbf{s}_t$, to be input to the encoder or the decoder respectively.

Evaluation Metric

As discussed in Sec. 3.1, our model attempts to learn a policy whose encoded state distribution evolves according to the vector field described by Eq. 3.3. The limit cycle would thus be expected to represent the nominal gait behavior learned by the policy, and the gait frequency parameter, ω, would be expected to correspond to the angular frequency of this nominal gait behavior. We use this correspondence to evaluate the effectiveness of the presented algorithm. We define the observed angular frequency of the gait executed by the robot using the learned policy as 2π times the average number of steps taken by the robot per second per limb. The discrepancy between this observed angular frequency and the gait frequency parameter ω is used as the evaluation metric.

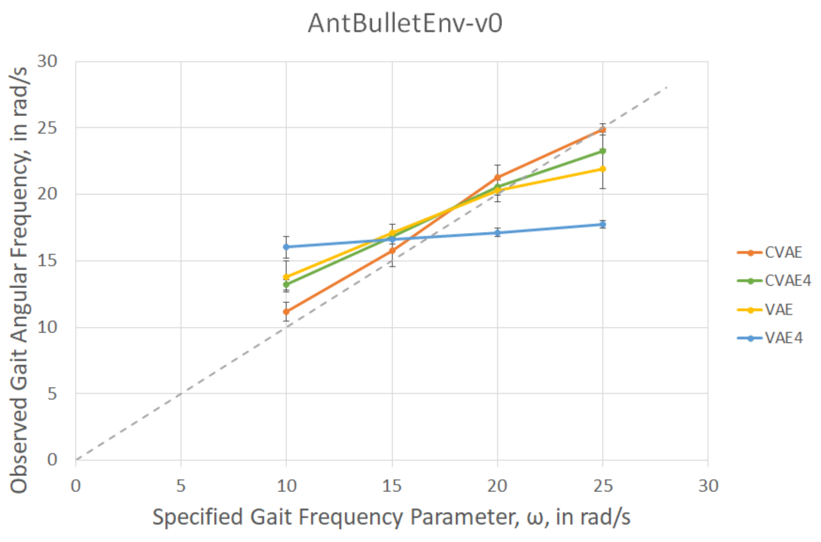
4.2 Learning policies for a range of gait frequencies

In this section, we demonstrate the legged robotic agent's ability to learn a family of behaviors that are modulated by the gait frequency parameter, ω. In order for the learned policy to be able to operate at different frequencies, we must specify a probability distribution over ω and minimize the expected value of the loss as described in Sec. 3.3. We restrict the value of ω to lie between 10 rad/s and 25 rad/s. We consider two discrete uniform distributions over this range: one defined only at the upper and lower limits of the desired frequency range, and another defined at four evenly spaced values of ω over the specified range, i.e. 10 rads/s, 15 rad/s, 20 rad/s, 25 rad/s. We refer to these as the 2-value and 4-value variants respectively. We refer to the experiment by the choice of representation model, i.e. CVAE or VAE, followed by the number of possible ω values. The hyperparameters for these experiments are listed in Tab. 4.2.

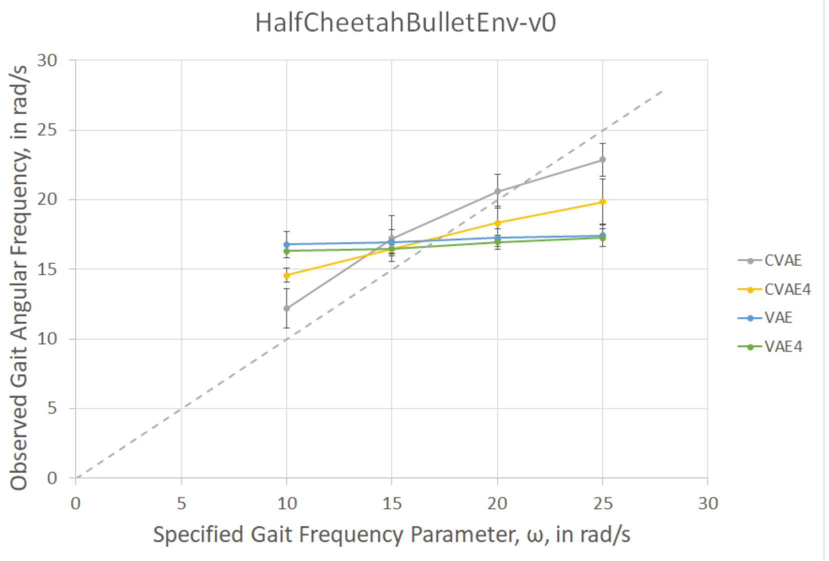
	`AntBulletEnv`		`HalfCheetahBulletEnv`	
	CVAE	VAE	CVAE	VAE
Number of timesteps	3e6	3.5e6	3e6	3.5e6
Intrinsic reward coef., β (Eq. 3.5)	10.0	7.5	2.5	2.5
Variational loss coef., λ (Eq. 3.9)	0.007	0.007	0.004	0.007

Table 4.2: Hyperparameters for experiments described in Sec.4.2

To test the effectiveness of the proposed method, we execute the learned policy for a fixed length of time with different values of the gait frequency parameter, ω. Fig. 4.2 shows the observed gait angular frequencies for each of these values. We see that the CVAE variants learn behaviors whose observe gait frequency is close to the gait frequency parameter, ω. Note that the framework is capable of interpolating behaviors for values of the gait frequency parameter that it hasn't been trained for. The observed gait angular frequency is seen to more closely match the specified gait frequency parameter, ω, in the 2-value variants. This behavior could be attributed to the additional constraints added to the policy by requiring the learned behavior family to match the gait frequency parameter for two additional values of ω. Experiments were also performed for $p(\omega)$ being a continuous uniform



(a) AntBulletEnv-v0



(b) HalfCheetahBulletEnv-v0

Figure 4.2: Observed gait angular frequencies for different values of the gait frequency parameter, ω, for `AntBulletEnv-v0` and `HalfCheetahBulletEnv-v0`.

distribution, but the learned policy in this case was observed to be a single behavior, nearly agnostic to the specified ω.

It is also worth noting that the CVAE variants fare better at achieving the desired gait frequencies than their VAE counterparts for the `AntBulletEnv-v0` environment, as is evidenced by Fig. 4.2. In the case of the `HalfCheetahBulletEnv-v0` environment, the VAE variants are seen to learn a single policy that does not depend on the specified gait frequency parameter, ω. This difference between the

CVAE and VAE variants can be attributed to the higher level of flexibility afforded to the representation model by conditioning on ω. This conditioning allows the same observation to map to a different encoded state depending on the value of ω. This enables the state distribution of the learned policy to vary with ω. The VAE variant on the other hand, forces the agent to visit the same set of states at different frequencies, and is therefore only capable of learning behavior families that visit the exact same set of states when operating at different frequencies.

In order to further validate the working of the proposed method, we visualize the evolution of the encoded state through the execution of the policy. As described in Sec. 3.1, the vector field is used to specify a prior on the encoded state space of the agent, by splitting the encoded state into pairs of two, and evaluating the vector field on each pair of components. Fig. 4.3 shows the evolution of each of these pairs through the execution of the policy for 500 time steps.

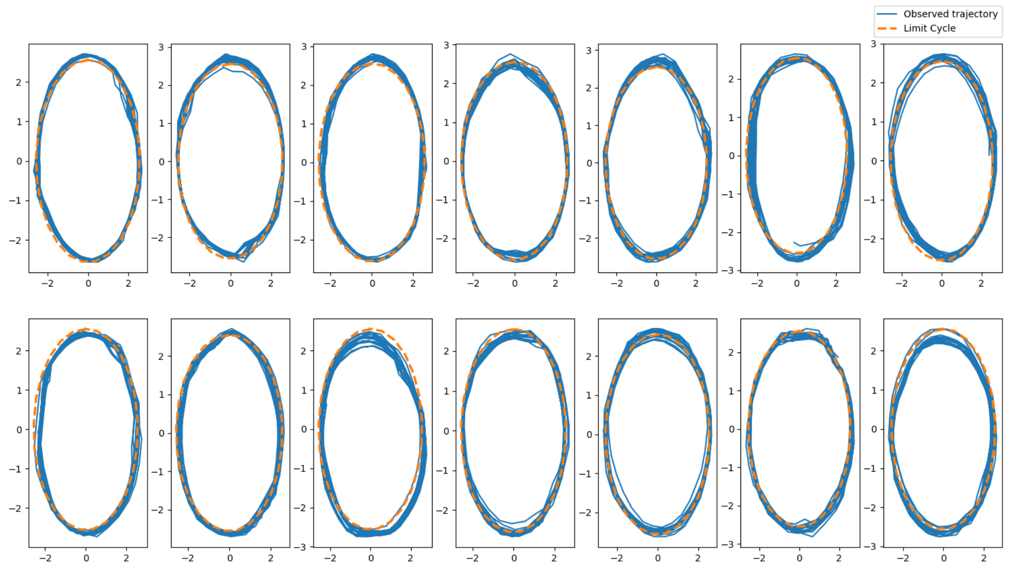


Figure 4.3: An example of an encoded state space trajectory through the execution of a learned policy. The dotted lines represent the limit cycles corresponding to Eq. 3.3, while the blue lines correspond to the evolution of the encoded state components along the trajectory when the policy is executed. It can be seen that the encoded state components remain close to the limit cycle and seem follow the vector field.

It can be seen that the trajectories in the encoded state space stay close to the limit cycle, and seem to be obeying the vector field dictated by the prior on the state space. This serves as validation of the fact that our method indeed works.

4.3 Learning policies with a specific gait frequency

We can also use this framework to learn a single behavior with a pre-specified gait frequency. This corresponds to the special case where $p(\omega)$ is a dirac delta function. This means that the architecture is trained for a specific value of ω so as to have a specific gait frequency. Fig. 4.4 shows the observed angular frequency of the gait for different values of the specified gait frequency parameter, ω. Since the value of ω is constant in this case, the CVAE and VAE variants of the representation model are identical.

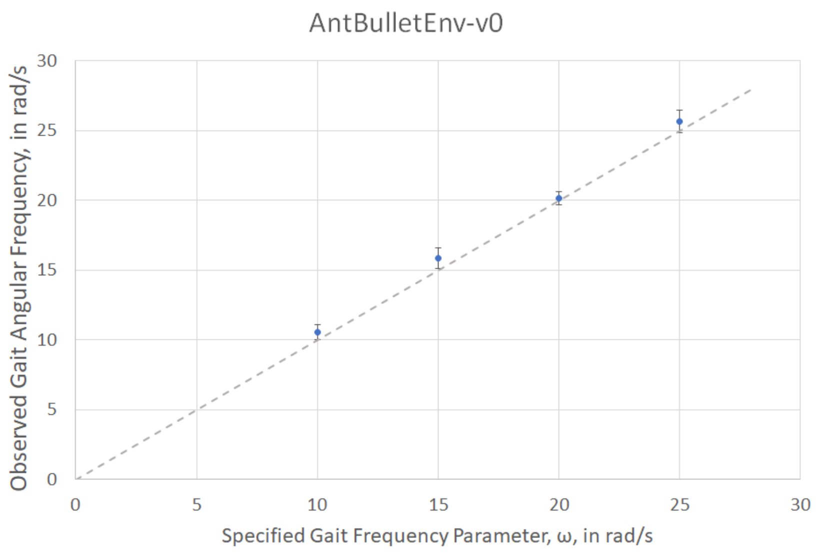
4.4 Comparison with joint-space CPGs

As discussed in Sec. 2.1, conventional CPG controllers generate trajectories in the robot's joint space which must then be tracked by a lower level controller. This performance is extremely sensitive to the parameters of the tracking controller. We demonstrate this sensitivity in the following experiment.

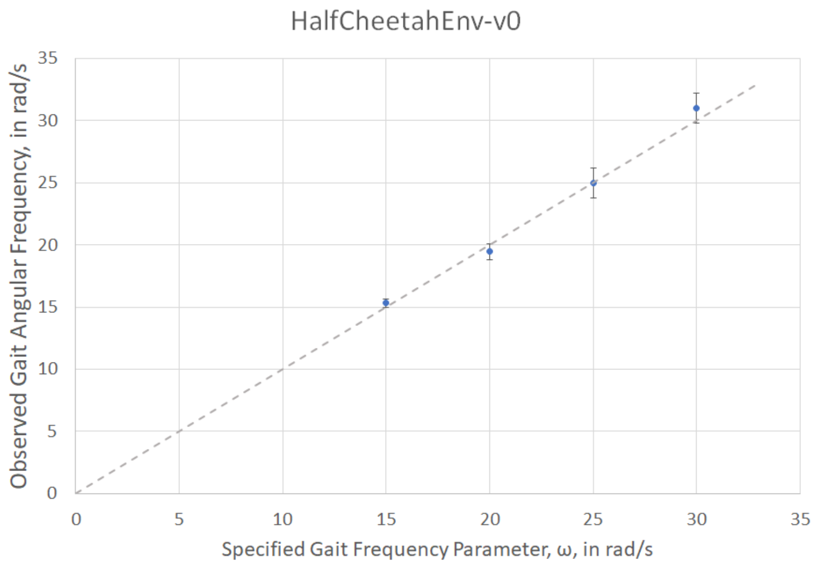
We implement a joint space CPG controller on the `AntBulletEnv-v0` environment and optimize the CPG parameters to maximize the expected cumulative extrinsic reward over a given distribution over ω, similar to Eq. 3.6. The extrinsic reward used in our experiments contains a penalty term for the joint torque magnitude. Since the CPG uses a PID controller to track joint positions, it uses significantly higher torques than policies learned using the proposed architecture, and fails to walk if the torque penalty is added. Therefore, we ignore the joint torque penalty when optimizing the CPG parameters, and only use the forward progress component in all our comparisons in this section.

We use the CPG formulation presented in [30]. The parameters of the vector field are the same as the ones described in Eq. 3.3, in addition to coupling terms that dictate the synchrony of the limbs and therefore the gait pattern. We use the trot gait which was found to have the best performance for our choice of robot and reward function.

For the trajectory tracking controller, we use Pybullet's [7] POSITION_CONTROL strategy to track the trajectory dictated by the CPG. This strategy implements a PID controller and offers position control by optimizing `position_gain` * (`desired_position - current_position`) at a higher frequency than that of the CPG controller. We fix the value of the `position_gain` parameter, and use an evolutionary strategy, CMA-ES [15], to optimize the expected cumulative sum of rewards over a distribution of angular frequencies, ω. We consider a discrete



(a) AntBulletEnv-v0



(b) HalfCheetahBulletEnv-v0

Figure 4.4: Observed gait angular frequencies for specified values of gait frequency, ω, for `AntBulletEnv-v0` and `HalfCheetahBulletEnv-v0`.

uniform distribution over four values of angular frequency – 10 rad/s, 15 rad/s, 20 rad/s, 25 rad/s. Fig. 4.5 shows a plot of the relative cumulative reward as the `position_gain` parameter, that scales the PID gains, is varied relative to the initial value, for the different values of ω.

It can be seen that the performance varies drastically as the `position_gain` parameter is varied. The effectiveness of the joint space CPG controller is heavily

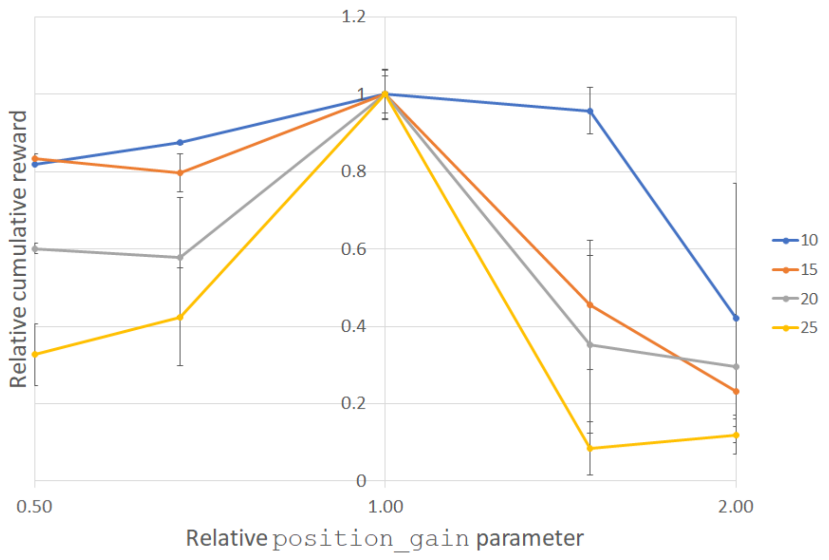


Figure 4.5: Relative cumulative rewards using the optimized joint space CPG as the value of `position_gain` is varied, for different values of ω.

dependent on the low level trajectory tracking controller being used. Our method avoids this problem by operating directly in the actuator space of the robot and yielding torque commands for the actuators. This means that the low level controller is integrated into the learned policy in our proposed model. Further, this allows more power-efficient control of the robot.

Chapter 5

CONCLUSIONS AND FUTURE WORK

In this thesis, we presented an algorithm that enables learning a policy of a family of implicit CPG-like behaviors for legged locomotion. The learned policy can be modulated by the desired gait frequency. We demonstrated the ability to learn such a controller from scratch, thus eliminating the need for expert knowledge that is crucial in designing effective CPG controllers. We also showed how the technique could be adapted to specify a priori a desired frequency for the policy being learned.

A limitation of the current framework is that it only allows modulation of the learned policy using gait frequency. We would like to add capabilities to modulate the learned policy with other parameters to expand the set of behaviors that can be represented by a policy. A defining feature of CPG-based controllers is their modular nature and enforcing synchrony between different limbs of the robot through coupling between oscillators associated with each limb. An extension of this work would be learning modular policies that exploit the symmetric construction of most legged robots and allow modulation of coupling to realize a wide variety of gait behaviors.

Biologically, animals are seen to transition between gaits i.e. change footfall patterns, going from statically stable gaits at low speeds to dynamically stable ones at higher speeds. Gait transitions are generally related [32] to the absolute velocity of motion which may not always be positively correlated with the gait frequency of the motion. Moving forward, we would like to explore this correlation, and the ability of this approach to admit gait transitions.

The current framework relies on an environment reward function that incentivizes forward progress. Different formulations of the reward function to learn families of behavior that allow for turning, modulating step heights etc., are another avenue for future work. Additionally, we would also like to explore the application of this architecture to models like [10] and [1] that do not rely on a reward from the environment.

BIBLIOGRAPHY

[1] Joshua Achiam et al. “Variational option discovery algorithms”. In: *arXiv preprint arXiv:1807.10299* (2018).

[2] Shinya Aoi and Kazuo Tsuchiya. “Locomotion control of a biped robot using nonlinear oscillators”. In: *Autonomous robots* 19.3 (2005), pp. 219–232.

[3] Paolo Arena. “A mechatronic lamprey controlled by analog circuits”. In: *Proceedings of the 9th IEEE mediterannean conference on control and automation*. 2001.

[4] Peter W Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. In: *arXiv preprint arXiv:1806.01261* (2018).

[5] Junyoung Chung et al. “A recurrent latent variable model for sequential data”. In: *Advances in neural information processing systems*. 2015, pp. 2980–2988.

[6] Jörg Conradt and Paulina Varshavskaya. “Distributed central pattern generator control for a serpentine robot”. In: *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. 2003, pp. 338–341.

[7] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2019.

[8] Alessandro Crespi and Auke Jan Ijspeert. “Online optimization of swimming and crawling in an amphibious snake robot”. In: *IEEE Transactions on Robotics* 24.1 (2008), pp. 75–87.

[9] Prafulla Dhariwal et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[10] Benjamin Eysenbach et al. “Diversity is all you need: Learning skills without a reward function”. In: *arXiv preprint arXiv:1802.06070* (2018).

[11] Yasuhiro Fukuoka, Hiroshi Kimura, and Avis H Cohen. “Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts”. In: *The International Journal of Robotics Research* 22.3-4 (2003), pp. 187–202.

[12] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *arXiv preprint arXiv:1801.01290* (2018).

[13] Danijar Hafner et al. “Dream to control: Learning behaviors by latent imagination”. In: *arXiv preprint arXiv:1912.01603* (2019).

[14] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *International Conference on Machine Learning*. 2019, pp. 2555–2565.

[15] Nikolaus Hansen. “The CMA evolution strategy: A tutorial”. In: *arXiv preprint arXiv:1604.00772* (2016).

[16] Rein Houthooft et al. “Variational information maximizing exploration”. In: (2016).

[17] Auke Jan Ijspeert. “Central pattern generators for locomotion control in animals and robots: a review”. In: *Neural networks* 21.4 (2008), pp. 642–653.

[18] Auke Jan Ijspeert and Alessandro Crespi. “Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 262–268.

[19] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[20] Maximilian Karl et al. “Deep variational bayes filters: Unsupervised learning of state space models from raw data”. In: *arXiv preprint arXiv:1605.06432* (2016).

[21] Nathan D. Kent et al. “Inferring task-space central pattern generator parameters for closed-loop control of underactuated robots”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[22] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[23] Ilya Kostrikov. *PyTorch Implementations of Reinforcement Learning Algorithms*. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`. 2018.

[24] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[25] Shakir Mohamed and Danilo Jimenez Rezende. “Variational information maximisation for intrinsically motivated reinforcement learning”. In: *Advances in neural information processing systems*. 2015, pp. 2125–2133.

[26] Fabio Pardo et al. “Time limits in reinforcement learning”. In: *International Conference on Machine Learning*. 2018, pp. 4045–4054.

[27] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 16–17.

[28] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. “Stochastic backpropagation and approximate inference in deep generative models”. In: *arXiv preprint arXiv:1401.4082* (2014).

[29] Ludovic Righetti and Auke Jan Ijspeert. "Design methodologies for central pattern generators: an application to crawling humanoids". In: *Proceedings of robotics: Science and systems*. CONF. 2006, pp. 191–198.

[30] Ludovic Righetti and Auke Jan Ijspeert. "Pattern generators with sensory feedback for the control of quadruped locomotion". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 819–824.

[31] Guillaume Sartoretti et al. "Central pattern generator with inertial feedback for stable locomotion and climbing in unstructured terrain". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–5.

[32] Gregor Schöner, Wenying Y Jiang, and JA Scott Kelso. "A synergetic theory of quadrupedal gaits and gait transitions". In: *Journal of theoretical Biology* 142.3 (1990), pp. 359–391.

[33] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[34] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[35] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. 2015, pp. 1889–1897.

[36] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 3483–3491. URL: `http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf`.

[37] Mario Srouji, Jian Zhang, and Ruslan Salakhutdinov. "Structured control nets for deep reinforcement learning". In: *arXiv preprint arXiv:1802.08311* (2018).

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[39] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1995–2003.

[40] Manuel Watter et al. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in neural information processing systems*. 2015, pp. 2746–2754.