Arrangements of the Inputs and Outputs in the Multi-Robot Continuous Control Problem

Sida Liu
University of Vermont

Arrangements of the Inputs and Outputs in the Multi-Robot Continuous Control Problem

A Thesis Presented

by

Sida Liu

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Computer Science

May, 2021

Abstract

The Multi-Robot Continuous Control (MRCC) problem in Deep Reinforcement Learning requires a single neural controller (agent) to learn to control the behavior of multiple robot bodies. When learning to control a single robot body, sensors and motors are arbitrarily connected to the input and output layers of the neural controller, respectively, and this arrangement does not affect the learnability of target robot behaviors. If and how such arrangement can affect learnability in MRCC—when dealing with multiple robots with different body plans—is as of yet unknown. In this thesis, I demonstrate the following: (1) A neural controller can control a small number of robot bodies with an arbitrary arrangement of sensors to control inputs, and control outputs to motors for locomotion, which explains why arrangements can be ignored in this case. But such arbitrary arrangements do not work well when the number of robot bodies increases. (2) For a given set of robot bodies, some arrangements can make the MRCC problem easier. In certain cases, the variation in MRCC facilitation provided by different arrangements is pronounced. This fact holds both in bodies with parametric differences (e.g. short and long legs) and bodies with topological differences (e.g. differing numbers of legs). Arrangement thus provides a heretofore unknown optimization opportunity in MRCC: searching for arrangements that increasingly facilitate learning of a single policy for different robots.

To Anne, Zimo, and the future.

Acknowledgements

Throughout the development of this thesis I have received a great deal of support from my advisor, Professor Josh Bongard. We discussed about every aspect of the research and the writing. We have spent many weeks together discussing and debating. I really enjoyed it and have learned a lot from it.

I would like to thank my colleague, Sam Kriegman Ph.D. for sharing the knowledge and for many interesting discussions.

I would like to thank two friends, Lapo Frati and Keith Epstein, for giving suggestions for the research and the writing.

I also would like to thank all the kind and interesting people I have met in Vermont.

Last but not least, computations were performed, in part, on the Vermont Advanced Computing Core.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem

In the *Multi-Robot Continuous Control* (MRCC) problem, the goal is to control multiple different robots to do the same task with one shared controller (Figure 1.1).

Consider two robots with different body shapes, for example. Suppose one has four legs (a quadruped), and the other has only two legs (a biped), and the goal is to use one controller to make either one walk forward.

First, sensory information must be provided to the controller. Assume there is one sensor on each leg, and at each time step, there is one unit of information coming from each leg. A unit of information is defined to be a set of values from one sensor. For example, one unit of information of a leg can be the angle of joint on that leg and the first derivative of the angle. So the quadruped has four units of information, but the biped only has two. And, assume the controller can take four as input. One might decide to input the four units of information from the quadruped in order, but in what order should the two units of information be supplied from the biped?

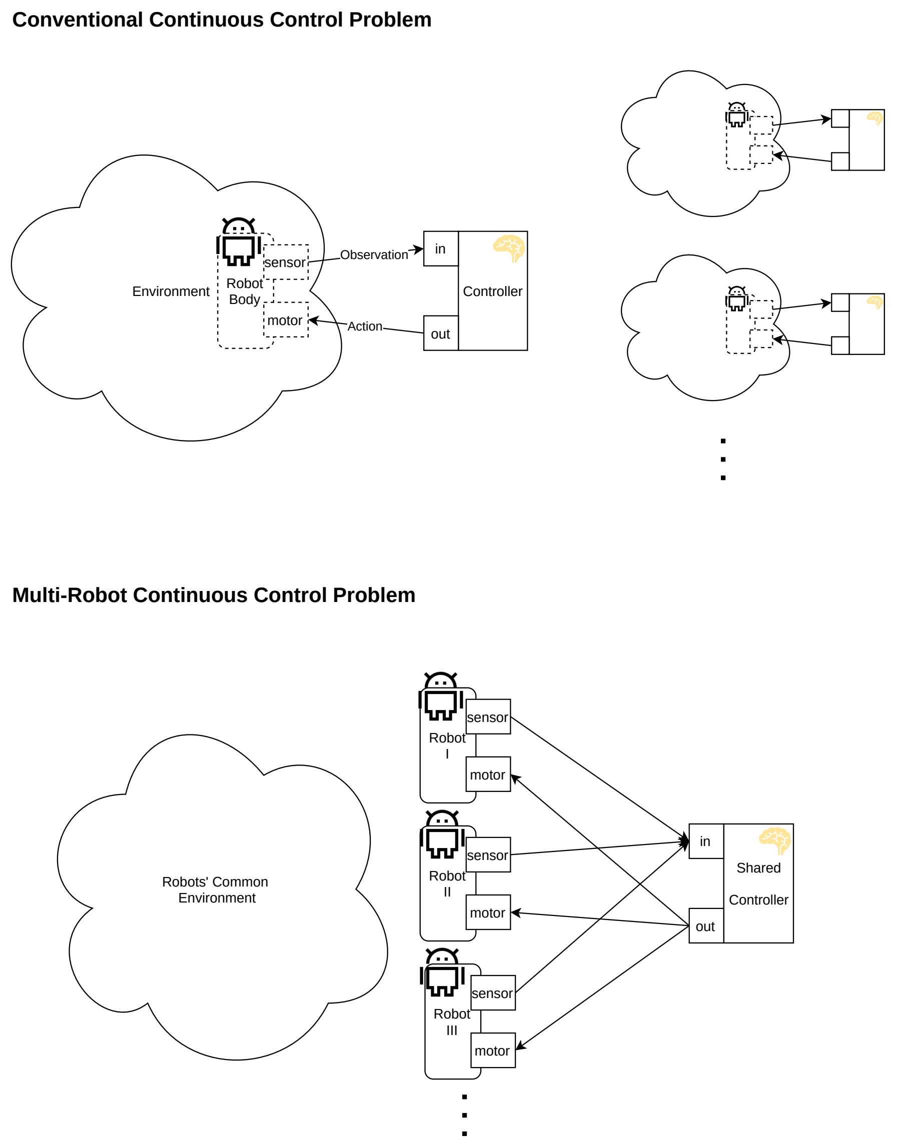


Figure 1.1: In contrast to the conventional Continuous Control Problem, in the Multi-Robot Continuous Control Problem, multiple different robots are controled by one shared controller.

Figure 1.2 shows one possible way to arrange the inputs. Here, $a3$ from the quadruped corresponds to $b1$ from the biped, $a4$ corresponds to $b2$, and $a1$ and $a2$ from the quadruped have no corresponding inputs in the biped. By aligning input in this way, the controller will treat the corresponding inputs (e.g., $a3$ and $b1$) the same as if they have identical meaning. The term *arrangement* will be used to mean such one particular solution, i.e., one particular way of arranging the orders of inputs from a set of robot bodies.

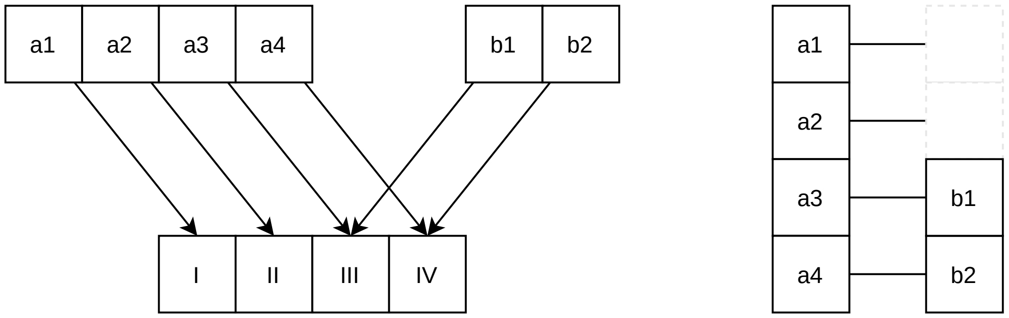


Figure 1.2: One possible way to arrange the inputs. a1, a2, a3, a4 are the sensory information from the quadruped. b1, b2 are the sensory information from the biped. Here, $a3$ corresponds to $b1$, $a4$ corresponds to $b2$, and $a1$ and $a2$ from the quadruped correspond to nothing.

Of course, Figure 1.2 provides only one possible arrangement. There are many other possible arrangements. For example, one can also decide to make the correspondence between $a1$ and $b1$, $a2$ and $b2$, and leave $a3$ and $a4$ unmatched.

The same decision also needs to be made on the output side. Assume the controller outputs four units of information, and one need to decide which two motors (or actuators in general) from two robots receive the same unit of output. To simplify the problem, assume that the output to the motors is arranged in the same way that the input from the sensors is arranged, i.e., if a correspondence between $a3$ and $b1$ is made on the input side, the same correspondence is also made on the output side.

Formally, let $\mathcal{O}_{r,t}$ be the overall sensor observation of robot r at time t; let $\mathcal{C}^{\text{in}}_{r,t}$ be the overall observation of robot r that is inputted to the controller at time t. It is assumed that the arrangement does not change over time, so for simplicity, the subscript t will be omitted henceforth. Without loss of generality, it is also assumed that each robot has multiple joints, and each joint can produce one unit of sensor observation and can take in one unit of motor action, and a unit of sensor observation and motor action can be null. Thus, $\mathcal{O}_r$ is a vector, and it contains multiple units of sensor observation: $\mathcal{O}_r = [\mathcal{O}^{(1)}_r, \mathcal{O}^{(2)}_r, \cdots]$. Here, $\mathcal{O}^{(i)}_r$ is the i-th unit of sensor observation from i-th joint. Each unit of sensor observation can itself be a vector that contains multiple real numbers: $\mathcal{O}^{(1)}_r = [\mathcal{O}^{(1,1)}_r, \mathcal{O}^{(1,2)}_r, \cdots]$. $\mathcal{O}^{(i,j)}_r \in \mathbb{R}$ is the j-th number in i-th unit of sensor observation from the i-th joint. As an example, $\mathcal{O}^{(1,1)}_r$ denotes the position of the first joint and $\mathcal{O}^{(1,2)}_r$ denotes the velocity of the first joint. Let J_r to be the cardinality $|\mathcal{O}_r|$, so J_r is also the number of joints of the r-th robot.

Let $\mathfrak{A}$ be the arrangement for a list of robots. $\mathfrak{A}$ contains one permutation matrix for each robot: $\mathfrak{A} = [\mathfrak{A}_1, \mathfrak{A}_2, \cdots]$, where $\mathfrak{A}_r$ is the permutation matrix for the r-th robot. $\mathfrak{A}_r$ has J_r columns and J_r rows. Thus:

$$\mathcal{C}^{\text{in}}_r = \mathfrak{A}_r \mathcal{O}_r \tag{1.1}$$

Equation (1.1) represents the order of units (joints) in $\mathcal{O}_r$ is changed by the permutation matrix $\mathfrak{A}_r$ into $\mathcal{C}^{\text{in}}_r$, this process is the arrangement of observations for the r-th robot, and thus, $\mathfrak{A}$ is the arrangement of a list of robots.

To express a permutation matrix in a more readable way, let z be a vector of length J_r, so that $z_i = i$. Let $\mathfrak{a}_r$ be the shorthand of $\mathfrak{A}_r$, the arrangement for robot

r, so that $\mathfrak{a}_r = \mathfrak{A}_r z$. For example, if $\mathfrak{A}_r = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$, then $z = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ and $\mathfrak{a}_r = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$. To make it more readable, the vector can be written as $\mathfrak{a}_r = [2, 3, 1]$.

Now consider the arrangement of outputs. The controller outputs actions to the motors. So, similar to $\mathcal{C}_r^{\text{in}}$ and $\mathcal{O}_r$, let $\mathcal{C}_r^{\text{out}}$ be the overall action that is outputted from the controller; let $\mathcal{A}_r$ be the overall action that robot r applies to its motors. Similarly, $\mathcal{A}_r^{(i)}$ is the i-th unit (joint) of action, and $\mathcal{A}_r^{(i,j)}$ is the j-th number in the i-th unit (joint) of action. In this thesis, one unit of action only contains one real number, which is the torque for the motor, but in more complex cases, one unit can contain several real numbers just like the observation. Recall it is assumed that each joint can produce one unit of sensor observation and can take in one unit of motor action, so $|\mathcal{A}_r| = |\mathcal{O}_r|$. Note the total number of real numbers contained in the observation and action are in general be different, but they have the same number of units (joints). It is assumed that, for each robot r, the output side uses the same arrangement $\mathfrak{A}_r$ as the input side, because the sensor observation and the motor action are both in the order, which is the order of joints. That is:

$$\mathcal{C}_r^{\text{out}} = \mathfrak{A}_r \mathcal{A}_r \tag{1.2}$$

This is similar to Equation (1.1). Usually, the controller will produce $\mathcal{C}_r^{\text{out}}$ first, then the robot applies $\mathcal{A}_r$ to the motors, so Equation (1.2) can also be written as:

$$\mathcal{A}_r = \mathfrak{A}_r^{-1} \mathcal{C}_r^{\text{out}} \tag{1.3}$$

$$\mathcal{A}_r = \mathfrak{A}_r^{T} \mathcal{C}_r^{\text{out}} \tag{1.4}$$

Here, the inverse of the permutation matrix $\mathfrak{A}_r$ is its transpose.

Let $\mathfrak{a}_r' = \mathfrak{A}_r^T z$ Take the previous example, when $\mathfrak{a}_r = [2, 3, 1]$, $\mathfrak{a}_r' = [3, 1, 2]$.

Our question of this thesis is, in MRCC, given a list of robots with different arrangement $\mathfrak{A}$'s, will the resulting control problems be different? Will some arrangements result in easier control problems? Will other arrangements result in harder control problems?

1.2 Background

1.2.1 Robotics

The ideas of building a robot can be traced back to the very early days, long before the word "robot" had been invented. Among those early ideas, one very interesting example is from a book written in the 4th century C.E. in ancient China called *Liezi* [1] (Page 110 - 111). The story was about a craftsman named Yen-shin. He made a humanoid robot and brought it to the king. The humanoid sang when its cheek was pushed, it danced when its hand was clasped, and it did innumerable other tricks. The king believed that it was a man, but when the craftsman cut open the performer, there were only leather, wood, glue and lacquer. When they were put together, the humanoid could perform again. It is hard to know whether this story actually took place, but the author of that book clearly documented the idea of an entertainment robot. And this robot, if realized, would be quite advanced even in today's standard.

Many ideas of robots had appeared with many different names before the word "robot" was eventually invented in 1920. The word was introduced by a science-fiction play called R.U.R (Rossumovi Univerzální Roboti) written by a Czech writer Karel Čapek [2]. In the story, the word "robot" was created to describe the artificial people, who are made of synthetic organic matter such as artificial flesh and blood and can think for themselves. They worked for humans at first, but later they rebelled and extincted the human race. These robots, if realized, would be far beyond today's technology. But the author could express his idea of robots in terms of a play, and the audience could visually observe movements and reactions of the robots.

In 1954, a device called a *Programmed Article Transfer* was designed [3], and later in 1961, the Unimate robot was built based on it. The Unimate robot was considered to be the first industrial robot [4]. The main part of the Unimate was a robot arm which had multiple rigid parts and multiple joints. It was practical since it could move the arm and grab other things, but compared to those early-day imaginations, the functionality was very limited.

As shown in the three examples above, the ideas of the robots usually occur before the realization of the physical robots. Like Valentino Braitenberg wrote in his book *Vehicles* in 1986, in which he introduced his idea of how to make vehicles so that they appear to have certain feelings:

> Our vehicles may move in water by jet propulsion. Or you may prefer to imagine them moving somewhere between galaxies, . . . It does not matter. Get used to a way of thinking in which the hardware of the realization of an idea is much less important than the idea itself.

If an idea can capture the essence of one aspect of robotics, then it is important.

With the help of modern digital computers, researchers can make their ideas even more precise and realistic. In 1994, Karl Sims [5] showed the world his evolved virtual creatures in simulated three-dimensional physical worlds. The simulation utilized its computational power to compute the dynamics of the worlds, so that the virtual robots could be evaluated in those environments. Compared to only using human imagination to evaluate the ideas of robots, this method can show us which ideas are more physically plausible. Today, the computer simulations and virtual creatures are even more important [6], and most researchers utilize them to help design novel robots for the future.

The ideas of robots were presented in different forms. A spectrum can be constructed to represent the development of the ideas of robots, as shown in Figure 1.3.

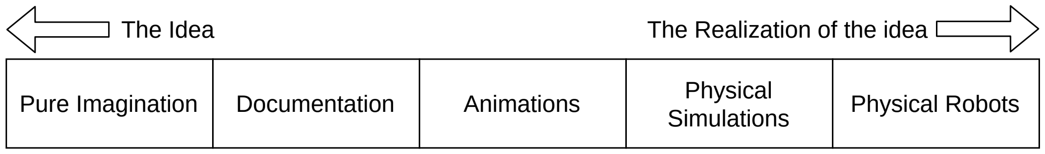


Figure 1.3: The spectrum from ideas of robots to the real physical robots.

Now let us take a close look at Karl Sims' virtual creatures. Each creature consisted two parts: the brain and the body. The brain was the control system. The body followed the control of the brain, provided sensory information to the brain, and followed the laws of the physical simulation. This formulation became a common pattern in robotics.

For convenience, later researchers tend to focus on one of those two parts, so the field of robotics can be viewed to consist of two main areas.

One of the main area focuses on the body. Sims [5] modeled the robot using rigid cuboids and different types of joints. This abstraction was widely accepted. But people are still trying to find better other ways to construct the body. For example, Lipson [7] showed that the robot can be made of bars and ball joints. Hiller [8] showed that by using voxels to model the material used by the robot bodies rather than cuboids and joints, it is possible to simulate and design soft robots. Cheney [9] showed that by specifying the structure of the body, it is possible to construct robots that locomote even with almost trivial controllers. Kriegman [10] showed that the robot body can be not just soft, but can made of living cells. Beside the material, researchers are also interested in the higher level structures. For example, Bongard [11] showed that changing the body plan from anguilliform to legged form can make the locomotion behavior more robust.

The other area focuses on the controller. There are many ways of creating the controller. One of the most recent methods, which is closely related to this thesis, is the use of learned controllers, which will be discussed in the next two sections. For example, Timothy [12] applied algorithms derived in other domains to control the robots. Schulman [13, 14] applied Reinforcement Learning that demonstrated significant improved results in controlling robots in the simulation.

It is worth noting that in addition to those examples, there are also works that took both the body and the controller into consideration, and thus are in the intersection of the two main areas. For example, Bongard [15] showed that by modeling its body in the simulation running inside the robot, the controller can make better decisions. Kwiatkowski [16] showed the self-modeling can also be done by utilizing not a simulation but a neural network. Cheney [17] showed that the body and the controller can be optimized together using the Evolutionary Algorithm. Ha [18] also showed that the body and the controller can be optimized together, but using Reinforcement Learning.

And this thesis is also one of the works in the intersection.

1.2.2 Reinforcement Learning

Classic Reinforcement Learning (RL) [19] is one of the ways to design learned controllers. RL is learning what to do to maximize a numerical reward signal, so that the human designers do not need to specify the detailed actions for the robots. The

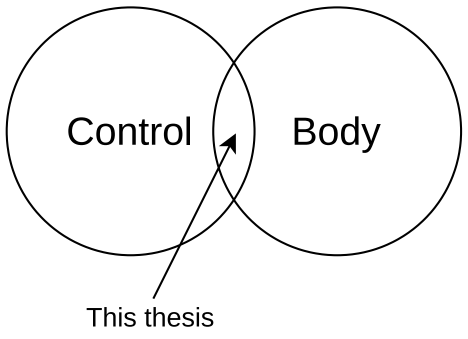


Figure 1.4: The illustration of two main areas of robotics and the position of this thesis.

designer only need to give the goals (reflected by the reward signals), and the robot will learn how to act on its own by interacting with its environment and considering reward or punishment it received.

In the fundamental RL setting (Figure 1.5), the controller is called the *Agent*, the all the rest other than the *Agent* is called the *Environment*. The *Environment* can be either physical or non-physical. When RL is used to control a robot, the body of the robot is included in the environment.

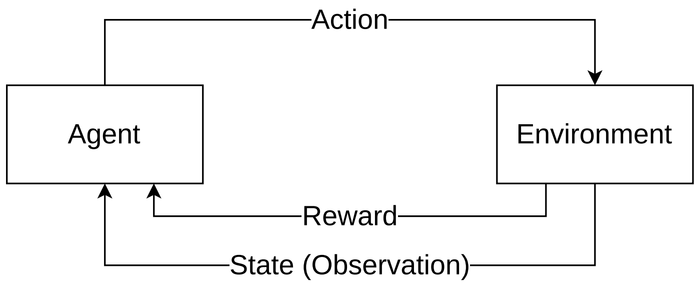


Figure 1.5: RL: The Agent interacts with the Environment in a Markov decision process.

The time is discretized into multiple time steps. At each time step t, the *Agent* will receive a *State* S_t (or *Observation*) from the *Environment*, it will also output an *Action* A_t that can influence the *Environment*. The *Environment* will provide a *Reward* R_t and the next *State* S_{t+1} based on the current *State* S_t and the *Action* A_t

of the *Agent*. This is called the Markov Decision Process (MDP), which means the *Environment* is memoryless and the next *State* is only depend on the current *State* and the *Action* of the *Agent*.

In the MDP setting, the *State* contains all the information of the *Environment* that can determine the future. It is not realistic in many applications. Luckily, for most current RL algorithms, the MDP assumption can be relaxed [19]. In the relaxed MDP setting, only part of the *State* is observed by the *Agent*, so the *Agent* receives a *Observation* instead of a *State*.

Note that, instead of simply replace the *State* with the *Observation*, more rigorous methods exist [19]. For example, the Bayesian approach Partially Observable MDP (POMDP) takes probability into account, or the Predictive State Representations (PSRs) method uses future predictions rather than current observations. But these methods are beyond the scope of this thesis.

One of the most important components of an RL algorithm is the *Value function*. A *Value function* specifies "what is good in the long run" [19]. An RL algorithm is trying to estimate the *Value function* according to the *Rewards* it has received over time.

Another important component is the *Policy*. A *Policy* produces *Action*s based on its *State*s (or *Observation*s).

The relation between the *Value function* and the *Policy* is that, if there is a good estimation of the *Value function*, it is easy to obtain a good *Policy* that can get more *Reward*; but at the same time, the *Value function* is depend on the current *Policy* because the future depend on the *Action*s it produces. So the estimation of the *Value function* and the *Policy* need to be improved together. Different RL algorithms are different ways to improve these two, including the one that was used in this thesis.

The RL theory was well developed in 1990s [20], but the power of RL was strongly experienced in recent years. This is because the development of another branch *Deep Learning*.

1.2.3 Deep Learning

Deep Learning (DL), as a field, is studying how to learn abstract representations from data using multiple layers of non-linear *Artificial Neural Networks* (ANNs).

Inspired by the structure of human brains, ANNs were proposed as a computational model [21]. An ANN is typically organized into multiple layers. Figure 1.6 shows a simple multi-layer ANN which contains the *Input Layer*, the *Hidden Layers*, and the *Output Layer*. The input layer and the output layer are exposed to the user, which is the most relevant to this thesis.

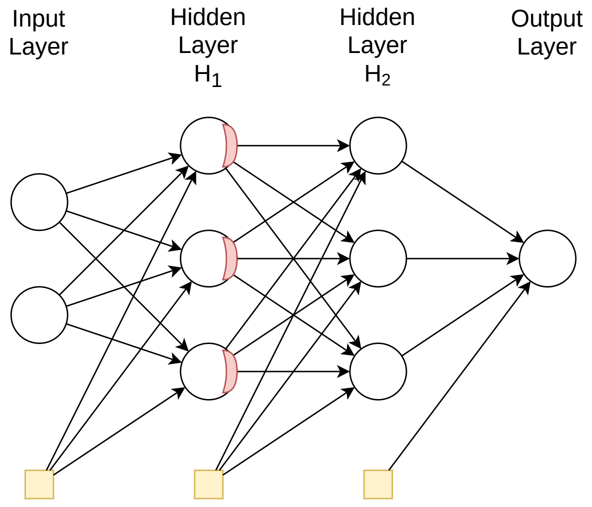


Figure 1.6: A simple multi-layer ANN.

In each layer, there are circles and a yellow square. A circle is a unit that can store a real number, it is usually called a *Neuron* in the ANN. Once the value is placed in a neuron in a hidden layer and if the next layer is also a hidden layer, it will be passed into a non-linear function called the *Activation* function (indicate by the red crescent). Any differentiable, non-linear function can be an activation function. The activation functions make the ANN much more expressive while keeping the whole network still differentiable. Then the result of the activation functions will be passed to multiple arrows simultaneously. A yellow square contains a real-number called a *Bias* of that layer. The bias is also passed to multiple arrows. One arrow represents multiplication of the input by a *Weight* (also a real number) of that arrow and the resulting value flows to the neuron pointed by the arrow. The results from multiple arrows to the neuron then are summed into one value and placed in that neuron. The whole process keeps carrying out until the neurons in the output layer have their values placed. This is called *Feedforward*. If there is a model with fixed parameters (weights and biases), feedforward can be used to get the output of an ANN. But in order to get the right parameters, another algorithm called *Back-propagation* is needed.

The *Back-propagation* algorithm [22] is the keystone of DL. After feedforward is done for a particular input, an output can be obtained from the model. Before the model is trained, the output of the model is not the desired output. A *Loss function* is needed to specify how bad the output is, and the result of the loss function can be minimized by adjusting the parameters so that next time, hopefully, the model can produce a better output. Because the whole network is differentiable, once the value of the loss function is computed, the gradient of all parameters can be computed. Using a process called *Stochastic Gradient Descent* (SGD), the parameters are adjusted a small amount towards the desired value whenever samples of data are given. Gradually, the parameters of the whole network are adjusted so that the output is more desirable.

In the case of *Supervised Learning*, the data contains the desired output, so the loss function can be defined as the difference between the desired output and the output of the network. Although the whole process was proposed back in the 1980s, due to its compute-intensive nature, the DL showed its great potential only in recent years [23], thanks to the development of the hardware and increasing computational power.

Many higher-level architectures were proposed to increase the capability of DL. *Convolutional Neural Network* (CNN) [24] used shared kernels to process data across different dimensions of input. It is well-known for its success in image-processing applications. *Recurrent Neural Network* (RNN) [22] can reuse earlier values in the neurons so it relaxed the i.i.d. assumption and gave the network the ability to memorize. The gating mechanism was introduced into RNN to stabilize the system. Architectures such as *Long short-term memory* (LSTM) [25] and *Gated Recurrent Unit* (GRU) [26]. *Attention* [27] applied the gating mechanism beyond RNN to the whole DL field. *Autoencoder* [28] brought the ability to compress the representation even when there is no signal other than the input itself. *Generative Adversarial Nets* (GAN) [29] introduced the idea of multiple different neural networks with conflicting loss functions can be trained together, so that each neural network can keep improving.

Those tools and many others make DL very powerful and general. Once a loss function is defined, all the tools can work together to make a good model that can minimize the loss function. This brings us back to RL again.

1.2.4 Deep Reinforcement Learning

Recall RL was discussed in Section 1.2.2. RL estimates the value function according to the rewards it has received over time. Once the estimation of the value function was obtained, one can apply DL to create models that are consistent with the estimated value function by specifying the loss function based on the value function. And once those DL models are trained (the parameters of these models are adjusted so that they are consistent with the estimated value function), the agent can use one of those models to make decisions based on its observations. This approach is called *Deep Reinforcement Learning* (DRL).

In 2013, Mnih et al. [30] connected an RL algorithm to a DL network so that the agent can play seven Atari games with raw input image data. This DRL approach outperformed all previous approaches on these games, and the performance of the agent even surpassed a human expert on three of them.

Game is an important testbed for DRL, algorithms such as the AlphaGo [31] and later the MuZero [32] achieved superhuman performance in Go, chess, and shogi.

Beyond games, researchers started to tackle the continuous control problem including the robotic control problem in simulation, Trust Region Policy Optimization (TRPO) [13] and Stochastic Value Gradient (SVG) [33] showed that a DRL agent can learned to control robots in simulation to locomote by trial-and-error. Algorithms such as Proximal Policy Optimization (PPO) [14], Soft Actor-Critic (SAC) [34], Deep Deterministic Policy Gradient (DDPG) [12], and Twin Delayed DDPG (TD3) [35] were developed based on the experiments in simulation.

Researchers utilized those algorithms along with other DL methods, developed learned controllers in simulation first, and then applied them to physical robots. The real-world applications include locomotion [36], robot arm grasping [37], solving Rubik's cube [38], etc.

1.2.5 Previous Work on MRCC

Using DRL, researchers approached one problem after another. One of those is the MRCC problem: how one controller could be used to control multiple robot bodies. Like any other problem, this problem can be approached in different ways.

Multi-Task Reinforcement Learning

Following the line of Multi-Task Learning (MTL) [39] in DL, researchers formulated the MRCC problem as the Multi-Task Reinforcement Learning (MTRL) problem. However, MTL methods were usually evaluated on tasks that are from a very narrow distribution. For example, Landolfi, 2019 [40] evaluated algorithms on different tasks with the only differences in desired locomotion speed/direction.

Henderson [41] provides a set of tasks that can be used for MTRL and Meta-RL, which includes several parametric modifications to the robot bodies. They modified one body part at a time, the resulting size is either "Big" (times 1.25) or "Small" (times 0.75). For example, a valid task is HopperBigFoot, indicating that the body part called "foot" is enlarged based on the body plan called "Hopper". Each reported benchmark groups contain several different bodies: some groups such as the Hopper group contain 8 bodies, other groups like the Humanoid group contain 14 bodies. However, the number of robot bodies in a group can significantly affect the results.

In Devin et al. [42], there were different robot arms–two were 3 Degree of Freedom (DoF) and the third was 4 DoF. Although there were only three different body plans, one of them was topologically different. The author used a shared network module for the same task, but different network modules for different bodies, so that different bodies can perform the same task.

Noticeably, Yang, 2020 [43] integrated multiple modularized expert networks with the gating mechanism to control a physical robot perform different tasks like trotting, turning, and fall recovering. However, no morphological modifications are involved.

Meta-Reinforcement Learning

Closely related to MTRL, Meta-Reinforcement Learning (Meta-RL) aims to solve previous unseen tasks in a few-shot learning manner.

Finn et al. [44], Rothfuss et al. [45], Fernando et al. [46], and Rakelly et al. [47] evaluated their algorithm on tasks that differ in locomotion direction, velocity, target position, or simulation parameters. No morphological difference was introduced.

Meta-World [48] provides a set of tasks that can be used for MTRL and Meta-RL. They evaluated six state-of-the-art meta-RL and MTRL algorithms at that time (2019), and the results showed that non-parametric tasks are much harder than parametric tasks, and all algorithms struggled to learn the non-parametric tasks in their tasks set. Recent researches started using this environment [49] In addition, their parametric and non-parametric tasks only vary in the task itself. For example, the parametric task variations include reaching puck at different locations, and the non-parametric task variations include both reaching puck and opening window. But the robot body (a robot arm) is always the same.

Nagabandi et al. [50] introduced topologically different body plan. In their experiments, a robot could lose a limb during operation. For each experiment, only two possible bodies were involved: one was before losing the limb and the other was after losing the limb. In this case, they disabled the sensor and the motor. Since the other part of the body did not change, they did not consider to re-arrange the observation and the action space.

Sim2Real/Sim2Sim Transfer

Sim2Real/Sim2Sim transfer is to handle the problem that, after people have made (or trained) an agent in the computer simulation, how could the controller be used directly in a physical robot or a different simulation. In the RL setting, this is closely related to MTRL and Meta-RL, because the task in the physical robot or the other simulation can be viewed as a new task. Usually, this transfer only needs to deal with parametrical differences because it is easy to make a new environment with a similar topological structure.

Tobin et al. [37] applied *Domain Randomization* when they transfer learned policy to real robots. The policy was trained in many simulated environments with different object colors, background colors, camera positions, etc., so that when the policy was deployed in the real robot, the policy can handle the new environment as if it is just another random environment. Here the input to the agent is the camera images, so random camera position changed the content of the input. CNN was used to enhance the ability to handle image inputs. However, there is only one camera, so the input only has one unit of information (i.e. $|\mathcal{O}_r| = 1$ as defined in Section 1.1). And the robots in different environments are topologically identical.

Following the idea of domain randomization, Peng et al. [51] utilized RNN to enhance the agent's ability to capture the dynamics of different environments, so the performance of the policy on the real robots gets better.

Also following this idea, OpenAI et al. [38] transfers the agent that can manipulate Rubik's cube in simulation to the real robot hand. They train agents that can handle robot bodies with different parametric differences, e.g. the size of each finger. The purpose is to train agents to handle the whole distribution of the environments, and hopefully the reality can be treated approximately as one of the environment from the distribution.

Zhang et al. [52] showed the possibility to learn the correspondence between two domains using DL techniques prior to RL training. Once the correspondence can be represented as several neural networks, the RL agent trained in one domain can utilize them to directly get the corresponding actions in the other domain.

Shared Modular Policy

Very recently, Huang et al. [53] challenged the MRCC problem at a higher level. Not only the topology of the robots are different, but also the number of different robots involved in training is significantly larger than in previous work.

They proposed a method called *Shared Modular Policy* (SMP). This method assumed each body part of the robot has one joint, one sensory input, and one motor. This assumption is identical to ours (defined in Section 1.1). The architecture of SMP contains a collection of identical modules. The core component of the whole network is the unique module. The core module has one unit of input and one unit of output, which is exactly the number assumed in each body part. Whenever the policy needs to handle a body, it needs to be rewired so that each body part corresponds to one module. The connections between these identical modules are determined by the connections between different body parts. In other words, the architecture is arranged to have the same topological structure as the body.

Apart from those approaches that solely focused on the parametrical difference, but here, the parametrical differences were ignored by this algorithm. The algorithm suggests that any topologically identical bodies would be treated as if they are the same body.

1.3 Learnability, the Measurement

In the works mentioned in the previous section, the performances of the RL algorithms were measured to determine which algorithm learns faster. The performance of an agent is the total reward it can accumulate during a test episode.

In this thesis, The focus is on how different arrangements affect this performance during and after training. For example, suppose there is a set of robots, and two different arrangements are applied resulting in two different tasks. One agent with a given RL algorithm and hyperparameters is trained from scratch for each task. The *Learnability* of a task describes how easy the task is for an RL agent to learn.

In addition, two aspects of the learnability will be reported: (1) the *Learning Curve* and (2) the *Final Distribution*.

The *Learning Curve* shows the improvement during training. In the example above, during training, the agent is tested, and the performance is obtained at certain time steps. The learning curve of the task is the plot of performance over time.

The *Final Distribution* shows the variations in the final performance. At the end of the training, the agent will be tested and a final performance will be recorded. Due to the stochasticity of the RL learning process, only reporting common summary statistics is considered to be not enough [54]. Thus the estimated distribution of the final performance will also be reported, in the form of the probability density plot. Because with only the mean and the variance, one might assuming the distribution to be a Normal distribution, but the distribution is not always unimodal Normal, it could be bimodal.

1.4 Overview of Three Experiments

With the definition of *Arrangement* in Section 1.1 and the definition of *Learnability* in Section 1.3, the main research question of this thesis is:

Given a set of different robot bodies and an RL algorithm, will different arrangements result in different learnabilities?

To investigate the research question, three experiments will be conducted.

The first experiment which explores the *Parametrically Different Bodies* (PDB) problem will be presented in Chapter 2.

In the first experiment, four sets of bodies are procedurally constructed. Each set contains multiple parametrically different bodies. However, the topology structures of the bodies in each set are identical. The PDB problem is investigated first because most of the previous related works are dealing with this case. In the PDB problem, the topology structure is identical, thus the most intuitive arrangement is to align each body part across different robots. Two groups are compared, one is a set of robots with an aligned arrangement and the other group is the same set of robots with randomized arrangements. It is observed that the number of bodies in a set is important.

The second experiment which explores the *Topologically Different Bodies* (TDB) problem will be presented in Chapter 3.

In the second experiment, one set of four topologically different bodies are used. These four different bodies are from four popular locomotion tasks. Because there is no obvious correspondence among those bodies, Two arbitrary arrangements are compared. It is observed that the difference between two groups is statistically significant.

The third experiment which explores the *Topologically Different Bodies with Obvious Correspondence* (TDBOC) problem will be presented in Chapter 4.

In the third experiment, one set of eight topologically different bodies were constructed based on a common body plan. Two limbs are added to the prototypical body plan in different ways to obtain those topologically different bodies. Because there is an obvious correspondence, there is one possible optimal arrangement *M0*: two newly added limbs are treated arbitrarily, but other body parts follows the order from the prototypical body. Then other arrangements are created by randomly permuting M0 multiple times, and the learnability of these arrangements are compared. It is observed that there are gradients around the M0, which indicates that it might be possible to follow the gradients and find the optimal M0 in other cases where the M0 is unknown. It is also observed that the differences in learnability exist between two arrangements of the same permutation distance.

Chapter 2

Parametrically Different Bodies

2.1 Overview

Several solutions mentioned in the previous chapter only consider the case of parametrically different bodies [55,56], because this is the simplest case of different bodies. In this case, the robot bodies are topologically identical. Each robot has the same number of body parts, and they are connected in the same way.

In this chapter, four body types will be introduced, and four sets of experiments will be done: one experiment for each body type. For example, in one set of experiments, multiple parametrically different *Walker2D*s are generated, each with slightly different length, thickness, etc, and one shared agent is trained to control those multiple bodies.

Two groups are compared: (1) the *Aligned* group and (2) the *Randomized* group. For parametrically different bodies, there is a natural choice to align the sensors and motors across bodies, which is according to their original topological positions. This arrangement is called *Aligned*. The *Aligned* group contains only one such arrangement.

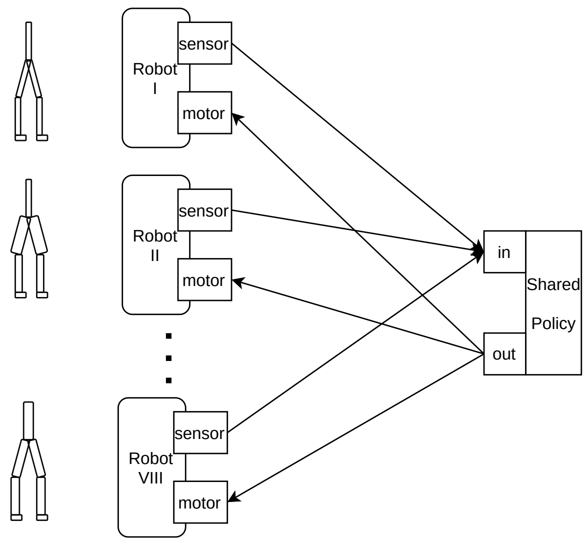


Figure 2.1: An illustration of the MRCC problem on Parametrically Different Bodies. Here are eight *Walker2D*s, and they are controlled by one shared policy. There will be experiments on different numbers and different body types.

On the contrary, random arrangements that have no particular meaning are generated each time before training, and they form the *Randomized* group.

The hypothesis here is whether the learnability is different between the *Aligned* group and the *Randomized* group. The difference in learnability between the *Aligned* group and the *Randomized* group is compared for each body type.

The main results are: (1) two groups do not differ much in learnability when there are only two bodies in each group, but when the number of bodies increases in each group, the difference becomes pronounce; (2) as expected, the learnability of the *Aligned* group is higher than the *Randomized* group.

2.2 Methods

2.2.1 Rigid-body Physics Simulation

In DRL robotics research, the proprietary software *MuJoCo* [57] is the most widely used rigid-body robot physics simulator. But recently the open-source alternative *PyBullet* [58] became more and more popular. In this thesis, *PyBullet* is used to conduct all the experiments.

The *PyBullet* environment is a three-dimensional virtual world, but by adding additional constraints to the robots, it can be viewed as a two-dimensional space. There can be a floor in the virtual world. The default flat floor is used in the experiments. One can also specify the direction and magnitude of gravity of the virtual world. For locomotion tasks, a low gravity world usually means an easier task. The default gravity ($g = 9.8\,\mathrm{m/s}^{-2}$) is used in the experiments.

A robot in *PyBullet* consists of a tree of body parts (a body part is sometimes referred to as a *link*). The root part is called the *Torso*. The torso has no sensor or motor. Start from the torso, a child part can attach to the parent part at an arbitrary position with a *Joint*. The joint is a sensor, and it is also a motor. At each time step, each joint will produce two real numbers: one is the position of the joint, the other is the velocity of the joint. Note that, the position of the joint does not mean the euclidean position of the joint in the virtual world, but it is the current angle value of the joint (only hinge joints are used). And the velocity of the joint means the change of the angle value.

Each body part has a geometric shape. The body parts can collide with the floor according to their shapes. But different parts do not collide with each other by default. This is because it is simple to construct body shapes without considering self-collision. It is possible to construct more complex body shapes with physically plausible joints so that self-collision can be enabled [59], but the default locomotion tasks without self-collision are used in the experiments.

2.2.2 Gym interface

OpenAI *Gym* [60] is a toolkit for DRL that contains benchmark problems. The interface is well-designed and easy to use. Due to its popularity, the OpenAI *Gym* interface became the de facto standard in DRL research, and its formulation is reshaping people's consensus on Reinforcement Learning.

Gym follows the Partially Observed MDP assumption mentioned in Section 1.2.2. So, it refers to the input to the agent as the *Observation*, not the *State*.

Usually, *Gym* has finite-length episodes (usually less than 1,000 time steps), which means the simulation will be reset after a finite amount of time steps. This is to avoid the learning to fail because the agent would enter some bad situation and never be able to recover to a situation that learning can continue. Also, there will be some random noises at the beginning of each episode to help the agent explore.

The key interface between the environment and the agent is the function *step*. The *step* function takes in an action and returns an observation, a reward, and additional information (such as does the current episode end). An action is an array of real numbers, and an observation is also an array of real numbers. These two concepts was discussed in detail in Section 1.1. Because *Gym* mainly serves DRL research, the designer assumed that the order in each action array and the order in each observation array are arbitrary but fixed over time. It is a reasonable assumption when the agent always interacts with an identical environment, because the orders of the input/output layer of a neural network as long as they are connected to a fully-connected hidden layer and the orders do not change over time. However, when an agent interacts with a different environment, this might become an issue. For example, an agent learned to control a wheel, and then it is used to control a hinge joint, although a wheel and a hinge joint both take one real number as their commands, the meanings of the real number are different in two commands. In MRCC, there are more than one environment, so this assumption might not hold.

2.2.3 Locomotion Tasks

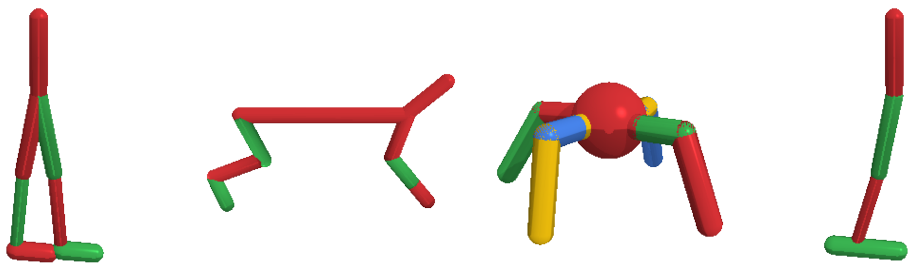


Figure 2.2: Robot body plans in four popular locomotion tasks: (1) Walker2D, (2) HalfCheetah, (3) Ant, and (4) Hopper.

Controlling different robot bodies to walk on a flat floor is formulated as different locomotion tasks. One task is an RL environment that defines a world, a robot body, and a reward function. There are four popular locomotion tasks (Figure 2.2) in PyBullet that are originally constructed by MuJoCo. Due to their popularity, people reconstructed these four tasks in PyBullet. However, despite the similar appearance of the robots, the dynamics are quite different. So, the learnability of these tasks is different from their counterparts in MuJoCo, and the resulting reward and distance are not comparable with experiments done in MuJoCo. The tasks in PyBullet are considered harder problems.

The original observation of each task consists of three major parts: (1) *Torso*: the information about the torso, (2) *Joints*: the information from all joints, and (3) *Floor*: the information about whether certain body parts are in contact with the floor.

Table 2.1: Meaning of each number in the observation space of the locomotion tasks.

Parts	Number meanings
(1) Torso	1. Height relative to initial state. 2. Sine of the yaw angle. 3. Cosine of the yaw angle. 4. Velocity in the x direction. 5. Velocity in the y direction. 6. Velocity in the z direction. 7. Roll angle. 8. Pitch angle.
(2) Joints	9. Position of Joint 1. 10. Velocity of Joint 1. 11. Position of Joint 2. 12. Velocity of Joint 2. $\vdots$
(3) Floor	Whether certrain Parts are contacting with the floor. (Removed from experiments)

The meaning of the numbers in the observation space is listed in Table 2.1. The format of the *Torso* part of the observation is identical across different tasks, so it is left untouched. The focus is on the *Joints* part, which depends on the order of the joints. Each joint produces two numbers: the position and the velocity. As mentioned in Section 2.2.1, the meaning of the position is the angle of the joint, and the velocity is the first derivative of the angle of the joint. The arrangement of a set of robots is specifed as the orders of joints for all robots. The *Floor* part is also different across tasks, but for simplicity, this part is removed, so the agent will not have the input of whether any part is in contact with the floor.

The reward functions for these four locomotion tasks are also in a similar structure. The reward function consists of four parts: (1) Progress: the major part of the reward is the progress the robot made at the current time step, the progress is the speed moving in the desired direction (if only this part of the reward is considered, then maximizing the episodic reward is equivalent to maximizing the total distance it traveled in the desired direction); (2) Alive bones: to encourage the robot to stand and walk rather than crawl, the robot can receive a bonus for keeping the position of the torso in a good range; (3) Electricity cost: to avoid the robot "waste" electricity, it will receive a small punishment proportional to the torque produced by the motors; (4) Joint-at-limit cost: to discourage the robot keep the joints at their limits, it will receive a small punishment when every there is joint its limit.

It is well-known that learnability is highly sensitive to the design of the reward function, so the default reward functions are used.

There is an additional constraint for these robots except the *Ant*: the robot bodies were constrained in their sagittal planes, so the robots can only move forward or backward but can not move aside, and the virtual world can be viewed as a two-dimensional space.

Now, let us examine these four locomotion tasks individually:

Walker2D

The name of the task is *Walker2DPyBulletEnv-v0*, or *Walker2D* for short.

There are one torso and six other body parts (thus six joints). The default order of the six joints are: thigh, leg, foot, thigh-left, leg-left, foot-left.

At each time step, if the robot keeps the height and pitch angle of its torso in certain ranges, it receives an alive bonus.

HalfCheetah

The name of the task is *HalfCheetahPyBulletEnv-v0*, or *HalfCheetah* for short.

There are one torso and six other body parts (thus six joints). The default order of the six joints are: back-thigh, back-shin, back-foot, front-thigh, front-shin, front-foot.

At each time step, if the robot keeps the pitch angle of its torso in certain ranges, it receives an alive bonus.

Ant

The name of the task is *AntPyBulletEnv-v0*, or *Ant* for short.

There are one torso and eight other body parts (thus eight joints). The default order of the eight joints are: front-left-hip, front-left-ankle, front-right-hip, front-right-ankle, back-left-hip, back-left-ankle, back-right-hip, back-right-ankle.

At each time step, if the robot keeps its torso above the floor, it receives an alive bonus.

Hopper

The name of the task is *HopperPyBulletEnv-v0*, or *Hopper* for short.

There are one torso and three other body parts (thus three joints). The default order of the three joints are: thigh, leg, foot.

At each time step, if the robot keeps the height and pitch angle of its torso in certain ranges, it receives an alive bonus.

2.2.4 Procedurally Generated Bodies

In this experiment, one policy is used to control multiple parametrically different bodies. Before start, those different bodies need to be generated first. [1]

Four sets of bodies are generated based on four prototypical bodies from the locomotion tasks mentioned in Section 2.2.3. The newly generated bodies are parametrically different from the prototypical body, but the topology is identical.

As illustrated in Figure 2.3, two types of body parts are used in these bodies: (1) Capsule-shaped parts, (2) Spherical parts. There are two parameters for each capsule-shaped part: the length of the cylinder l and the radius of the hemisphere r. Them stands for the length and thickness of the part. There is one parameter for each spherical part: the radius r. The capsule-shaped body parts are widely used in the bodies, e.g., the foot of a *Walker2D*. The spherical body part is only used as the torso of *Ant*.

The third type of parameters that can be varied is the default angle of two connected body parts. The customized default angles only exist in the *HalfCheetah* body

[1]Link to the source code file on GitHub

plan. The default angles in other body plans are either 90° or 180°.

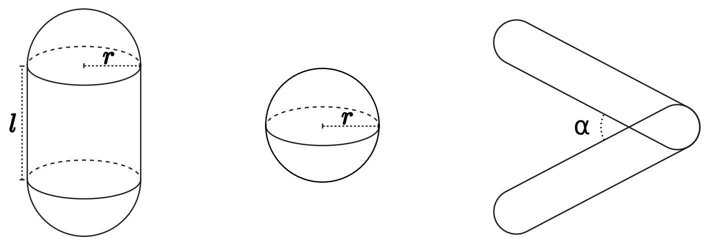


Figure 2.3: (1) A capsule-shaped body part with two parameters: l and r, (2) A spherical body part with one parameter: r, (3) An angle between two parts: α.

When generating new bodies, noises are added to all parameters. First, a random variable r that follows a Gaussian distribution with zero mean and standard deviation of $\frac{1}{3}$ is created, this standard deviation results that most of the time (99.7%), the random variable will be in the range of (-1,1). Then, all parameters are multiplied by 1.4^r. The number 1.4 is an arbitrary choice, and most of the time(99.7%), 1.4^r will be in the range of $(\frac{1}{1.4}, 1.4)$. Four sets of parametrically different bodies are obtained by this method, with 50 variations in each set.

However, there will be small changes that a generated body does not work, and experimenting on these broken bodies will be a waste of computational resource. To avoid this, and to make the experiments more efficient, only 16 variations from the generated 50 are selected based on the learnability of each variation while training a policy on it individually. The detailed training process will be discussed in Section 2.2.6, and the training results of this selection will be discussed in Section 2.3.1.

2.2.5 Specifying Arrangements

In total, 16 variations are obtained for each body type, and there are four body types.

Recall our goal of this experiment is to see whether the learnability is different between the *Aligned* group and the *Randomized* group. Since there is no topological difference within each body type, the default orders of newly generated bodies are *Aligned* (Figure 2.4). On the other hand, the *Randomized* group needs to be generated before each training process (Figure 2.5).

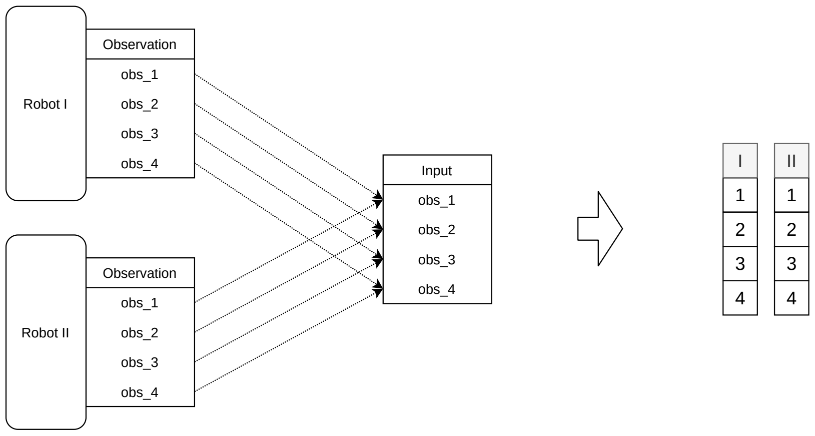


Figure 2.4: An illustration of the Aligned arrangement.

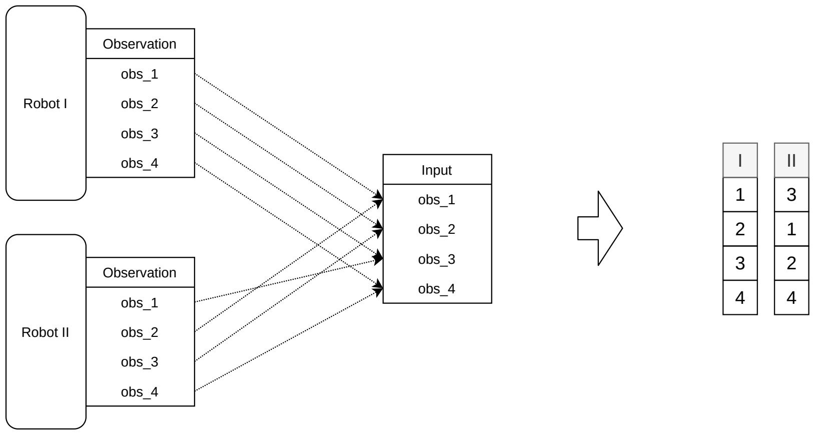


Figure 2.5: One example of the Randomized arrangements.

A wrapper was implemented to specify the orders for different arrangement[2]. A wrapper in the Gym framework is a general extension of existing environments. One

[2]Link to the source code file on GitHub

can modify the observation and action spaces of environments using a wrapper. Recall that different bodies exist as different Gym environments. By using a wrapper, before passing the observation to the policy, the orders of the observation can be specified, and before passing the action to the motors in simulation, the orders of the action can also be specified. By specifying orders for all Gym environments that are used for training, the arrangement used in the experiment is specified.

For example, as illustrated in Figure 2.5, the second unit of Robot II's observation is mapped to the first unit of input. This indicates that, on the output side, the first unit of output would be mapped to the second unit of Robot II's action.

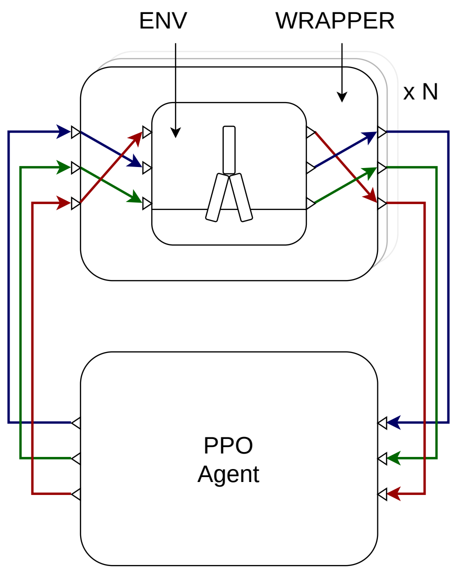


Figure 2.6: An illustration the wrapper and training schema.

Figure 2.6 shows an illustration of the wrapper. The PPO agent will be introduced in the next section.

2.2.6 Training

Our implementation of the joint training is based on the Actor-Critic style *Proximal Policy Optimization* (PPO) [14] algorithm from *Stable Baselines3* [61]. Stable Baselines3 is a set of high-quality implementations of popular DRL algorithms in PyTorch [62].

PPO algorithm is used because its *on-policy* nature. *On-policy* methods update the policy that is currently used for decision making, while *off-policy* methods update the policy that is different from the one that is used to generate the data. In Stable Baselines3, the training process of the online-policy algorithm can be easily parallelized, so the total training wall-time is much shorter than off-policy ones. In addition, the parallelism also allow me to easily stack different environments (i.e., different robot bodies) and train them simultaneously. Although on-policy algorithms are in general less data efficient, since a fast physics simulation is used and the structure of the robots are relatively simple, choosing PPO can significantly reduce the wall-time for the experiments.

The PPO algorithm assumes an discrete, discounted, finite state MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $R : \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor.

The initial state s_0 follows the distribution of the initial state: $s_0 \sim \rho_0(s_0)$. The next state s_{t+1} follows the transition probability distribution: $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$.

Let π be a stochastic policy, so that $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The action a_t follows the distribution produced by the stochastic policy: $a_t \sim \pi(a_t|s_t)$.

The $state - action - value$ function under policy π is Q_π, so that:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots}\left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+l})\right]$$

The $state - value$ function under policy π is V_π, so that:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \cdots}\left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+l})\right]$$

Note $V_\pi(s_t)$ is averaging all possible a_t according to $\pi(a_t|s_t)$, while $Q_\pi(s_t, a_t)$ only consider the action a_t.

Then, the *advantage* function under policy π is A_π, so that:

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

Let π_θ be the policy that is parametrized by vector θ, and and let $\hat{A}_t$ be the estimation of A_π at time t.

The main objective function proposed by the PPO algorithm is:

$$L(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \operatorname{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ be the probability ratio.

According to the results in the PPO paper [14], the hyperparameter *Clipping* ϵ is changed back to 0.2 instead of using a linearly decreasing clipping from 0.4 to 0.0 as implemented in Stable Baselines3. The reason Stable Baselines3 uses a linear decreasing clipping is to encourage exploration by allowing more aggressive parameter updates at the beginning of training and allow less and less updates towards the end of training. A constant clipping ϵ makes the algorithm simpler and thus will not introduce additional artifacts to our experiments.

Generalized Advantage Estimation [63] is used to facilitate learning.

State Dependent Exploration [64] is used to significantly reduce the training time.

All trainings were done using the same PPO algorithm.

All trainings were done on the servers of Vermont Advanced Computing Core (VACC). Each training process was treated as one Slurm task, so multiple tasks can be executed parallelly as long as there are free nodes in the cluster. One typical training process of 2 million time steps will take 0.5 - 3 hours.

2.3 Results

2.3.1 Selection of Bodies

The bodies generated in Section 2.2.4 differ in learnability. After those bodies were generated, the learnability of each is measured.

Bodies are generated based on four locomotion tasks. There are 50 bodies for each body type. Each body was trained individually three times (this is the standard DRL approach: one agent learns to control one body). Figure 2.7 shows the estimated probability density of final episodic reward distribution.

This is a plot of the episodic reward, not the distance traveled. The reward function contains more than just distance traveled. Details of the reward function were discussed in Section 2.2.3.

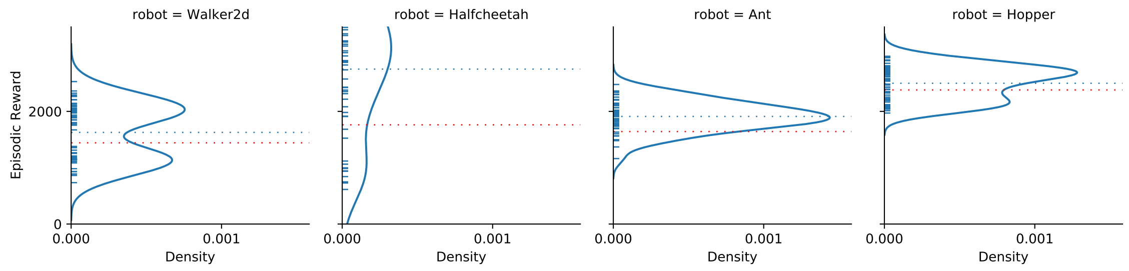


Figure 2.7: Learnability of all randomly generated bodies. The episodic reward was measured at the end of 2M-step training. The red dashed line is the original bodies from PyBullet. The blue dashed line is the average value of generated bodies.

It is observed that the distribution for *Walker2D* has two peaks, one is around 1,000, the other is around 2,000. The lower peak can be understood as that the robot learned how to keep balance and does not fall during the test episode, and the higher peak can be understood as the robot walks forward.

It is also observed that the distribution for *HalfCheetah* has a high variance. And the generated bodies are much better than the original body.

The distributions for *Ant* and *Hopper* have smaller variance.

In order to reduce the impact of the worst bodies in *Walker2D* and *HalfCheetah*, for each body type, 16 bodies are selected from the generated 50 bodies. Figure 2.8 shows the probability density of selected bodies. It is observed that less *Walker2D* idling and less *HalfCheetah* with low learnability.

Here, one blue dashed line stands for the average value of the 16 selected bodies. And these blue dashed lines will be used as a baseline in the following results.

All selected variations are visualized in Figure 2.9.

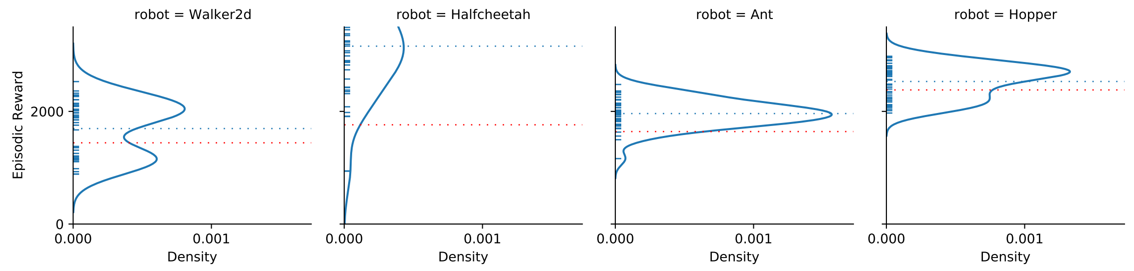


Figure 2.8: To have less idling in Walker2D and less HalfCheetah with low learnability, 16 bodies were selected from the generated 50 bodies for each body type.

2.3.2 Training On A Set Of Bodies

In this section, the learnability of a set of bodies will be examed.

The first choice to make is to decide how many bodies a controller should handle in one experiment. A natural choice is to ask a controller to learn to control two bodies with either *Aligned* or *Randomized* arrangement, repeat the experiment multiple times, and compare the difference between those two arrangements. However, this is a bad choice.

Figure 2.10 shows the learning curves of different experiments.

From top to bottom, the experiments vary in the size of the set. The results are measured with a set size of 2, 4, 8, and 16. (A set of 16 robot bodies means the agent learns to control 16 different robot bodies in one training process.)

From left to right, the experiments vary in body type. The results are measured in four body types: Walker2D, HalfCheetah, Ant, and Hopper. (One agent only learns to control robot bodies of one type in one training process.)

The x-axis is the time step in the simulation. There are in total 2 million time steps in each experiment. According to Section 2.2.2, one episode is less than 1,000 time steps, which indicates that the agent learned to walk after many episodes.

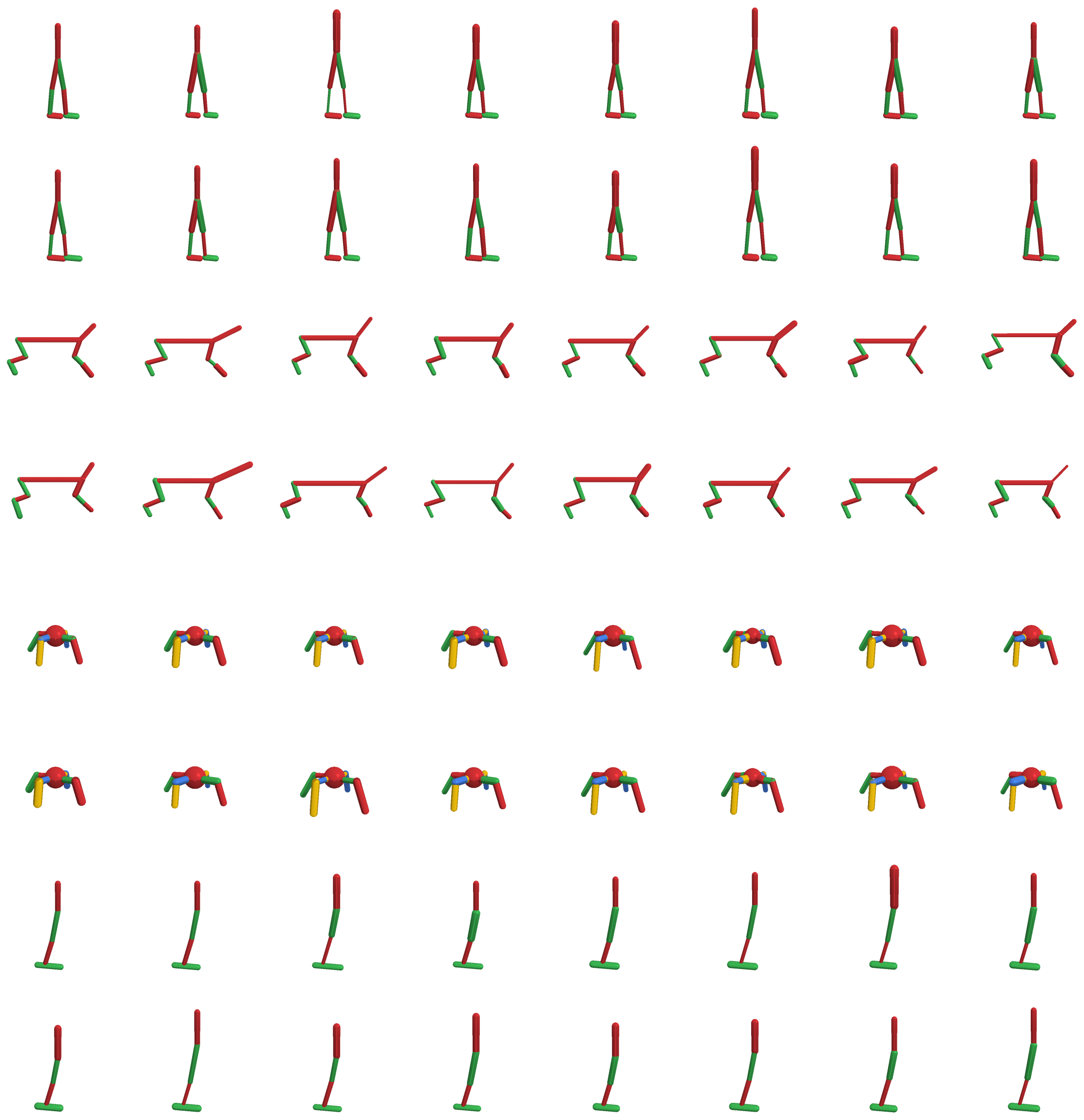


Figure 2.9: All selected parametrical variations: 16 Walker2Ds, 16 HalfCheetahs, 16 Ants, and 16 Hoppers.

The y-axis is the episodic reward. During training, the agent is periodically asked to control all the bodies in independent test episodes. The total reward the agent accumulated is recorded as the episodic reward. When time is close to time step 0, the episodic reward is close to 0, which indicates the agent performs badly before learning (equivalent to a random policy). And the agent performs better after 2M time steps of training.

There are two curves in each experiment. The blue one is the *Aligned* group, in which all the observation (input) and the action (output) are in the correct order. The orange one is the *Randomized* group, in which all the observation and the action orders are randomized for each run. The shaded areas are the 95% confidence interval.

The blue dashed lines are from the previous section, which are the baselines of training on one body (the standard DRL approach). As expected, training on a set of bodies is harder than training on one body in general. (Two curves are lower than the dashed baselines.)

Figure 2.11 shows the same data from the experiments using the estimated probability density. The layout, axes, colors, and the baselines are the same with Figure 2.10. In addition to the previous figure, more details of the final performance after training are shown.

According to these two figures, the difference between the *Aligned* group and the *Randomized* group is pronounce, which can support our hypothesis–different arrangements result in different learnability.

In addition to this finding, it is also observed that the difference is not that obvious in the case that there are only two bodies in the set. As mentioned earlier, if one only look at the result of an agent controlling two Walker2Ds or two HalfCheetahs, the difference between the *Aligned* group and the *Randomized* group is not obvious. This was surprising to me at first. It is a *Randomized* group, in which an agent is controlling two robots, but the orders of the observation and action are randomized in each run. One real number in the action could mean two different commands for

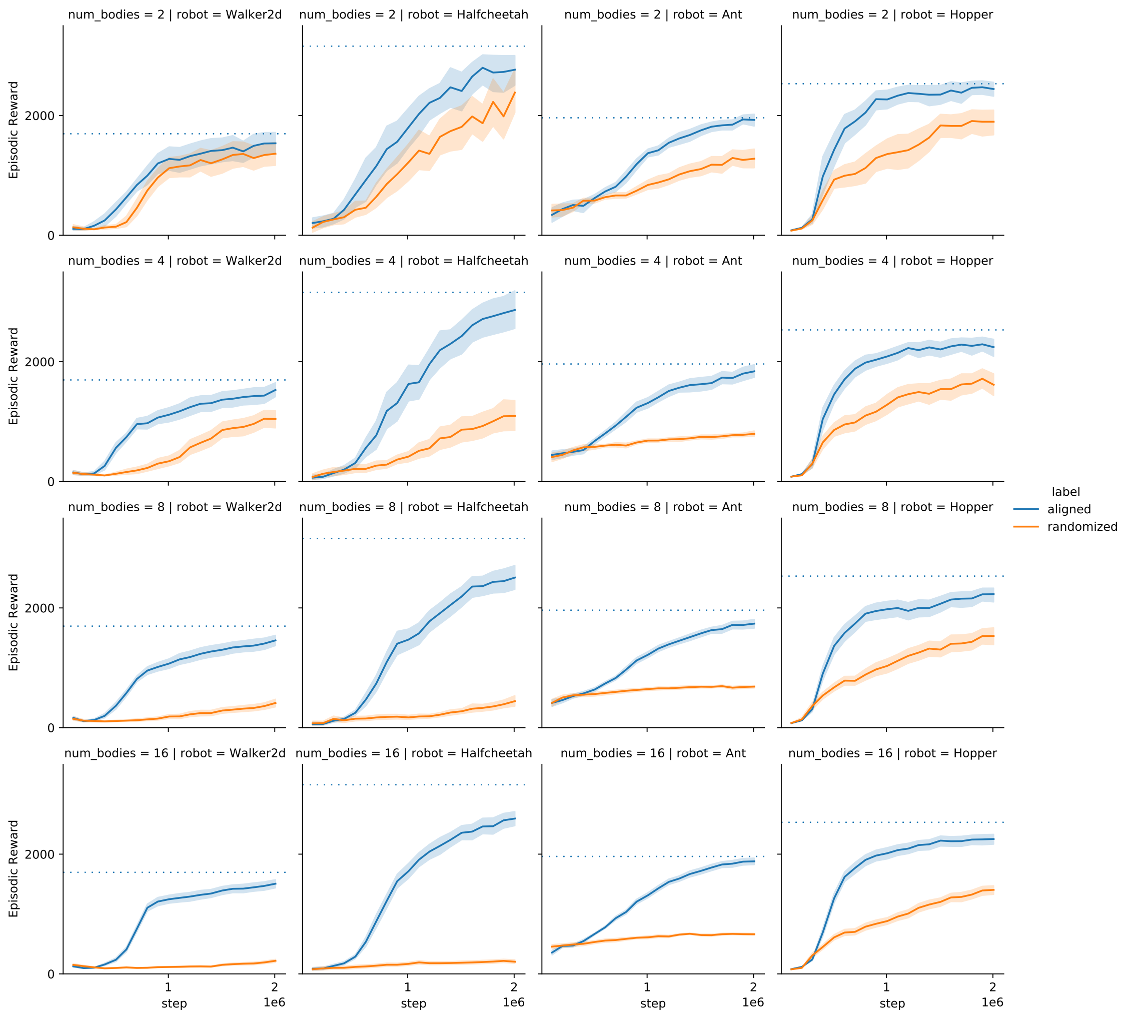


Figure 2.10: One controller learns to control a set of bodies. Sizes of the set are 2, 4, 8, and 16 (from top to bottom). Different body types are investigated in the experiments: Walker2D, HalfCheetah, Ant, and Hopper (from left to right). The blue dashed line is the average value of training on individual bodies.

two different motors. However, the agent has learned to control these two bodies. My interpretation of this phenomenon is that this is a feature of the deep neural network. The network inside the agent is performing both classification and control. The classification can tell the difference between two bodies based on the observation, and the action outputted by the agent is conditioned on the knowledge of the current

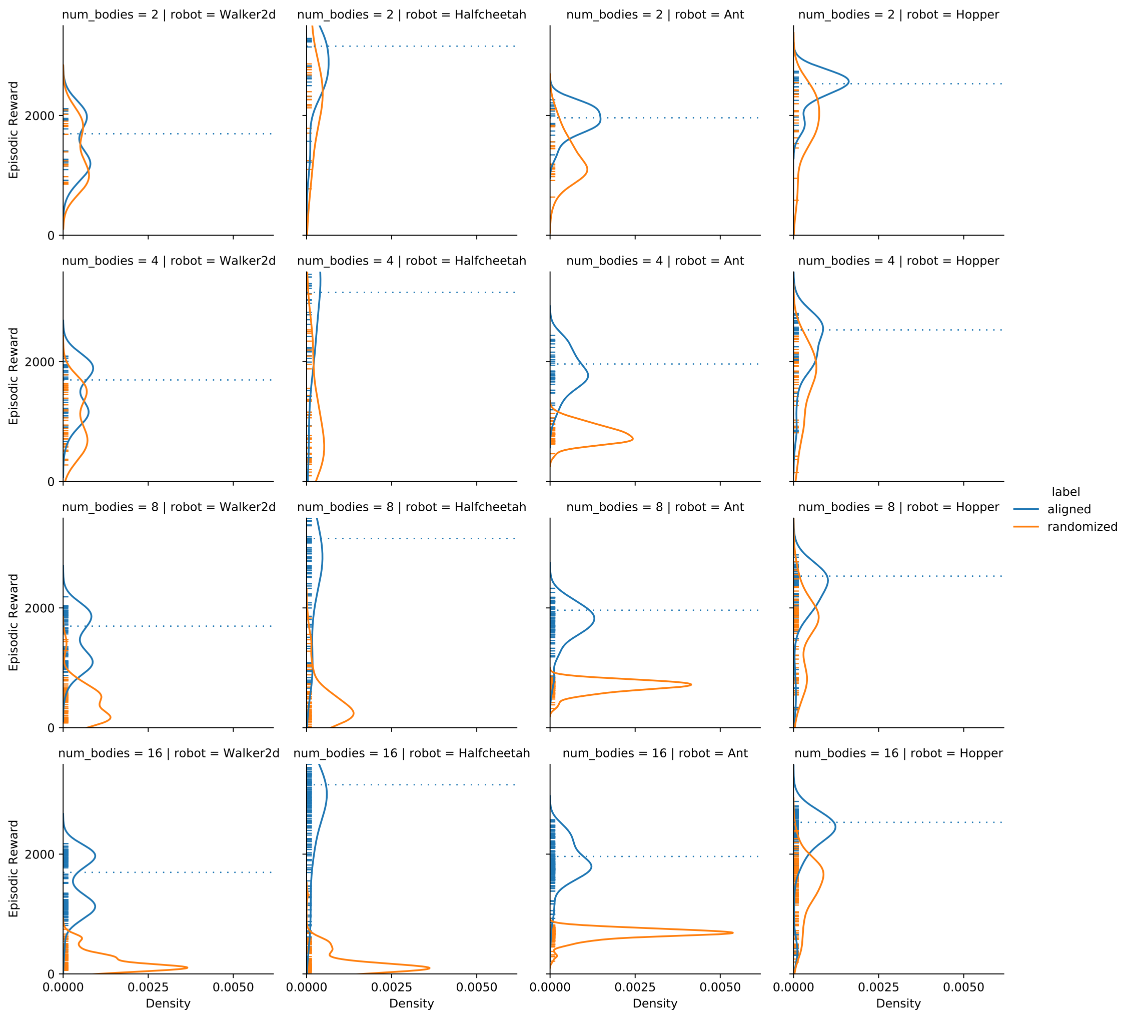


Figure 2.11: The probability density of the final distribution of the same data with Figure 2.10.

body it is controlling.

Researchers in the field sometimes ignore this effect, and might compare results of RL agents that controlling a different number of bodies. Thus, it is important to address how many robots in a set matter in the MRCC problem.

The *Aligned* group (blue) shows that learning to control a set of (less or equal to 16) bodies with parametrical differences is not a very hard problem. But, if the arrangements were *Randomized*, the difficulty increases rapidly.

Chapter 3

Topologically Different Bodies

3.1 Overview

As shown in the previous chapter, controlling parametrically different robot bodies is not particularly hard, as long as the aligned arrangements are used. In this chapter, the problem of one agent controlling a set of topologically different bodies (TDB) will be examinated. The TDB problem is considered to be a harder problem than the PDB problem.

The agent needs to learn to control a set of four robot bodies from the original PyBullet locomotion tasks. In Figure 3.1, there is no obvious correspondence between different robots. So, the optimal arrangement is unknown.

In order to see whether different arrangements result in different learnability, Two arrangements are chosen. Then the learnabilities of the set of four bodies with these two arrangements are measured. The difference between these two groups is statistically significant.

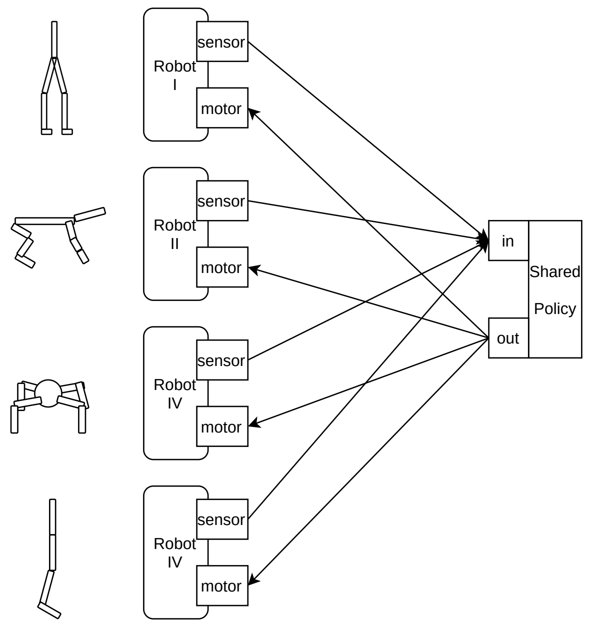


Figure 3.1: An illustration of the MRCC problem on Topologically Different Bodies.

3.2 Methods

In this chapter, the same simulation and training methods are used as the previous chapter.

Details of the four robot bodies were discussed in Section 2.2.3.

In order to see whether different arrangements result in different learnability, two arrangements need to be compared.

The Random Search methods are used to obtain a good arrangement in early attempts. In the random search, multiple different arrangements were generated. Then training was performed using each arrangement once, and the best arrangement was reported. However, it is observed that most of the arrangements constructed in the search process have a high variance in learnability, and the difference in mean is small, which means the resulting best arrangement might very likely due to random

Table 3.1: Two arbitrary arrangements

Arrangement	Waler2D	HalfCheetah	Ant	Hopper
#1	4,3,2,5,0,6,1,7	4,5,6,2,3,1,7,0	0,5,1,3,2,7,6,4	4,0,2,1,7,5,3,6
#2	6,2,3,0,4,5,7,1	2,1,3,5,6,7,4,0	7,4,0,3,1,2,6,5	6,4,2,5,7,3,0,1

Eight joints are ordered from 0 to 7, to be consistent with the source-code file.

chance. This caused the random search process to unable to find solutions with very different performances. If the search process only run training on the arrangement once, most of the variance is due to random chance. To estimate which arrangement is better, one need to run training on the arrangement many times. Thus the time used for random search is intractable. Instead, two arbitrary arrangements are chosen to compare. (Arrangement #1 was the best arrangement indicated by the random search, however, as explained, the random search is not reliable in this case, so it is referred to as an "arbitrary" choice.)

These two arrangements are showed in Table 3.1. There are at most eight units (joints) in each robot body. The i-th number j means the i-th unit of sensory data should be put into j-th unit for the agent to read. If a robot bodies does not have so many joints, null joints will be padding into the end of the sensory data.

The two arrangements can be visualized in Figure 3.2. The first row is arrangement #1, and the second row is the arrangement #2. The black parts are the torsos, which have no sensor and motor and the order do not change. Colors represent the correspondence across different bodies. For example, as illustrated in Figure 3.3, in arrangement #1, the red thigh of the Walker2D corresponds to the red shin of the HalfCheetah, and it corresponds to the red thigh of the Ant, and nothing of the Hopper (a null body part) are red. The color means that the agent treats the sensory

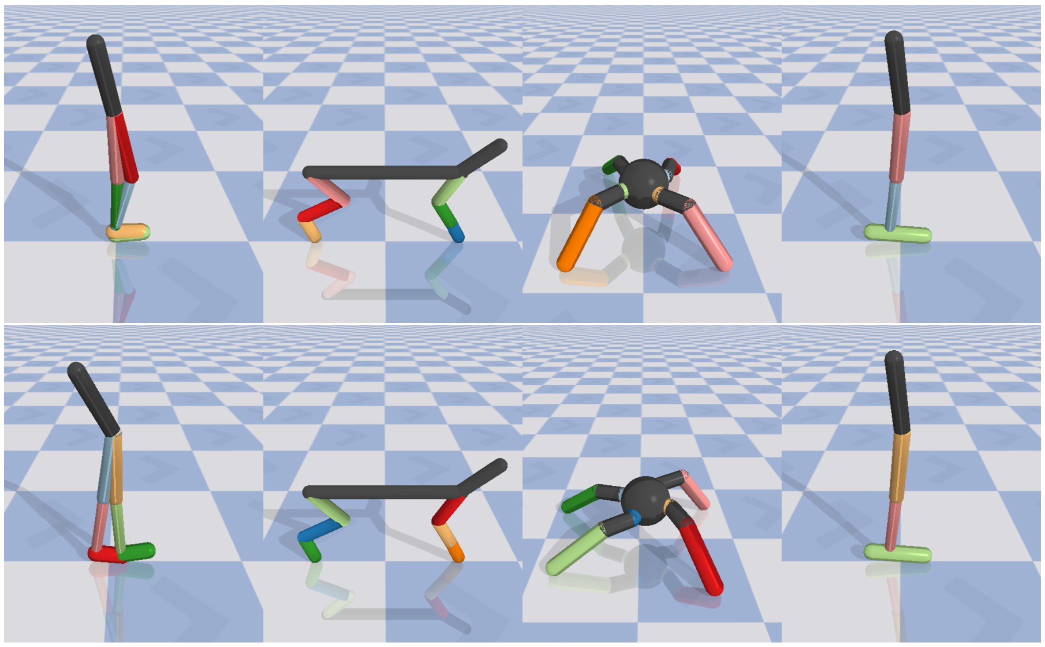


Figure 3.2: Two sets of four bodies. Each row shows an arbitrary arrangement.

data produced by the joints of those body parts the same, and the number the agent produced outputs to those parts as commands.

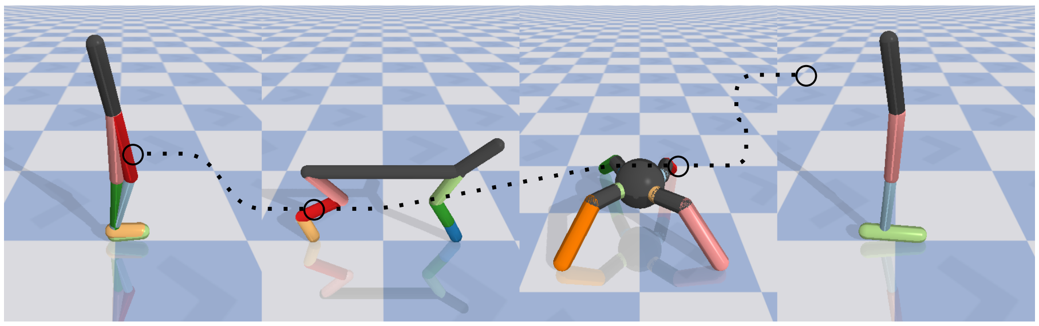


Figure 3.3: For example, the red body parts on all four robots in arrangement #1.

So the two treatment groups to compare are the arrangement #1 and the arrangement #2. Due to the high variance and the small difference in mean, Agents are trained with each arrangement 100 times using different random seeds.

3.3 Results

The resulting learning curves and probability densities are showed in Figure 3.4. Please refer to Figure 2.10 and 2.11 for the detailed explanations for these two types of plots.

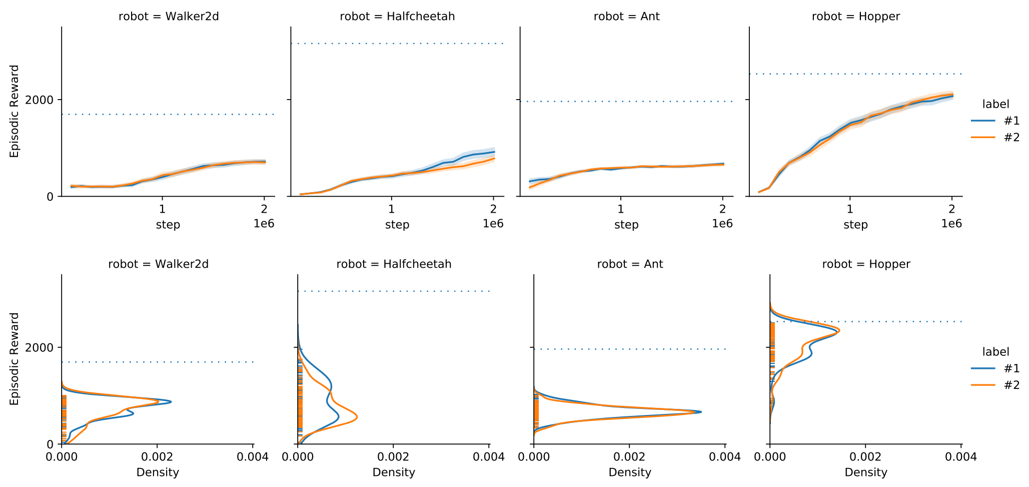


Figure 3.4: Learning curves and probability densities.

Here, the blue lines are the treatment group that uses arrangement #1, and the orange lines are the treatment group that uses arrangement #2.

Tthe performance of both groups are very similar.

However, it is observed that a small difference exists in the HalfCheetah. This means when a trained agent was tested on four bodies using different arrangements, they only perform slight differently in the HalfCheetah.

The question is whether different arrangements result in different learnability in HalfCheetah. The *t-test* is used to analyze such a small difference. The two-sided p-value is 0.01375. [1]

[1]Link to the source code file on GitHub

So this experiment still supports the hypothesis: different arrangements result in different learnability.

The main obstacle in this chapter is to find a good arrangement that can show the difference in an obvious way. However, the arrangement optimization is a hard problem (discussed in Section 5.2). In the next chapter, a special case will be constructed, in which an optimal arrangement is known.

Chapter 4

Topologically Different Bodies with Obvious Correspondence

4.1 Overview

In the previous chapter, the general TDB case is investigated. However, there exists no arrangement optimization method that can produce the optimal arrangement, so two arbitrary arrangements are tested.

In this chapter, a special case is constructed, so that one optimal arrangement is known. As illustrated in Figure 4.1, two small limbs were added to two arbitrary existing limbs of a Walker2D to obtain a variant. An RL agent need to learn to control eight such Walker2D variants. This problem can be called the *Topologically Different Bodies with Obvious Correspondence* (TDBOC). Because the resulting robot bodies are all similar to Walker2D, and the optimal arrangement might be the one that aligns the body parts with apparent same functional meanings.

The optimal arrangement is called the *M0*. Once *M0* is known, a slightly different

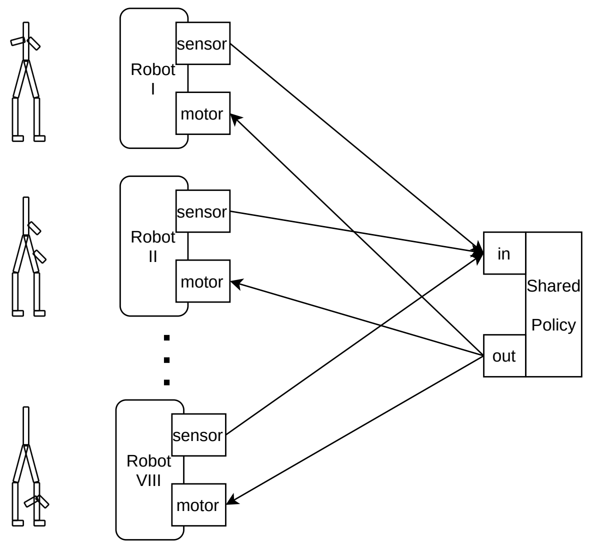


Figure 4.1: An illustration of the MRCC problem on Topologically Different Bodies with Correspondence. Here are eight topologically modified *Walker2D*s.

arrangement can be obtained by randomly permute *M0*. All arrangements that can be obtained by permuting *M0* twice are called *M2*. and it is called *M4* if the permutation is done four times, and so on.

Then the *M0* and those generated arrangements (*M2*, *M4*, etc.) are used in the experiments. It is observed that the less it is permuted, the better the learnability will be. And the learnabilities of the arrangements are different within the same amount of permutation.

4.2 Methods

In this chapter, the same simulation and training methods are used from the previous chapter. The only differences are (1) the robot bodies and (2) the way all the arrangements are obtained.

4.2.1 Manually Generated Bodies

The original Walker2D from PyBullet is used as a prototype. For each newly generated robot body, two small limbs were added sequentially. The resulting eight robot bodies are visualized in Figure 4.2.

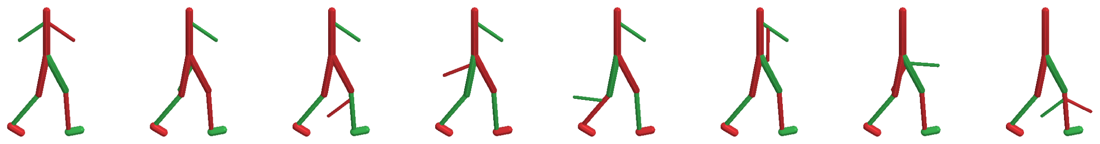


Figure 4.2: Eight manually constructed bodies.

One agent needs to learn to control a set of eight robot bodies simultaneously in one task.

4.2.2 Permutation Distance

After a set of robot bodies has been generated, the arrangement for those bodies needs to be specified.

There is a default arrangement. However, the orders of body parts in those robot bodies were disrupted because those small limbs were added to the body part list. So, there can be a better arrangement than the default one.

As shown in the visualization, there is a correspondence across robot bodies. For example, as illustrated in Figure 4.3, the left feet of all robots have the same functional meaning. And by examining the functional meanings for all parts, one optimal arrangement is constructed, and it is called the *M0*. Table 4.1 shows the *M0* used in the experiment. Note that the *M0* is not the unique optimal arrangement,

Table 4.1: M0: a possible optimal arrangement

Robot ID	Order of joints
I	0,1,2,3,4,5,6,7
II	1,2,0,3,4,5,6,7
III	1,2,3,0,4,5,6,7
IV	1,2,3,4,5,0,6,7
V	1,2,3,4,5,6,0,7
VI	1,0,2,3,4,5,6,7
VII	2,0,1,3,4,5,6,7
VII	2,3,0,1,4,5,6,7

because there is currently no way to prove whether there exists another arrangement that is equivalent or better than this arrangement.

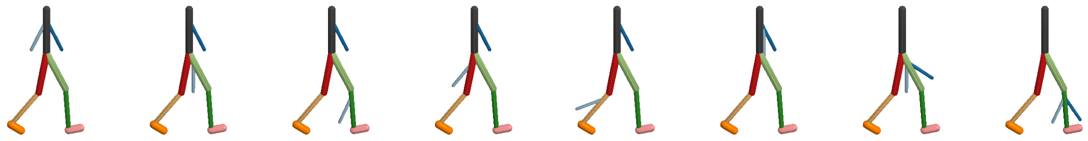


Figure 4.3: Robots are colored based on the M0 arrangement.

Once the *M0* is obtained, other arrangements are constructed by permuting *M0* randomly. The *Permutation Distance* is defined to be the times of permutation needed to obtained a particular arrangement. And by definition, the permutation distance of *M0* is 0.

The procedure of one permutation is defined to be (1) randomly select one robot, (2) randomly select two positions, (3) and swap the number in those two positions. [1]

This procedure is equivalent to a random mutation in the Evolutionary Algorithm.

And one *M2* arrangement can be obtained by randomly permute from *M0* twice,

[1]Link to the source code file on GitHub

one *M4* arrangement can be obtained by randomly permute from *M0* four times, one *M8* arrangement can be obtained by randomly permute from *M0* eight times, one *M16* arrangement can be obtained by randomly permute from *M0* 16 times, and one *M32* arrangement can be obtained by randomly permute from *M0* 32 times.

In total, 21 *M2* arrangements, 21 *M4* arrangements, 21 *M8* arrangements, 21 *M16* arrangements, and 21 *M32* arrangements are generated. [2]

4.3 Results

4.3.1 Gradient Around M0

First, agents are trained to control a set of robot bodies using the M0 arrangement. Figure 4.4 shows the results for M0.

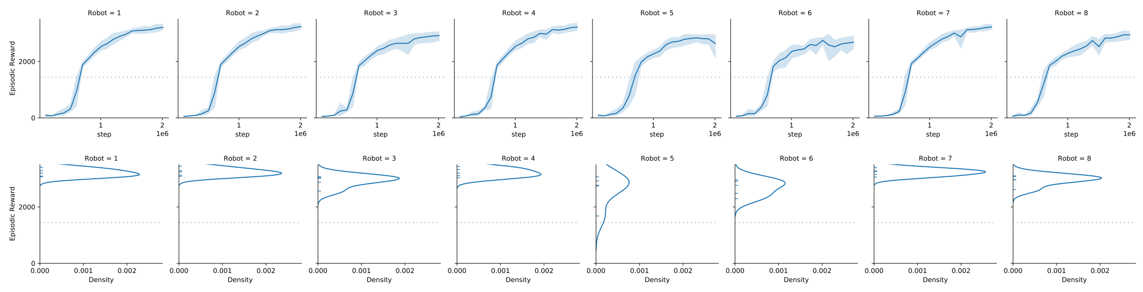


Figure 4.4: Learning curves and probability densities for M0.

In total there are 5 runs with different random seeds. From left to right, there are eight robot bodies. The blue dashed lines are the baselines from the original Walker2D in Section 2.3.1.

To my surprise, the learnability is much higher than the original Walker2D, which

[2]The number 21 is an arbitrary choice.

means the agents trained on this set of eight robots using M0 perform much better on a set of robot bodies than the agents trained on the original Walker2D. My guess to this is the added arms provide additional means to balance the body during training.

After the learnability of the robots with M0 is measured, the learnabilities of the robots with other arrangements (M2, M4, M8, M16, and M32) are also measured. Figure 4.5 shows the results for those arrangements.

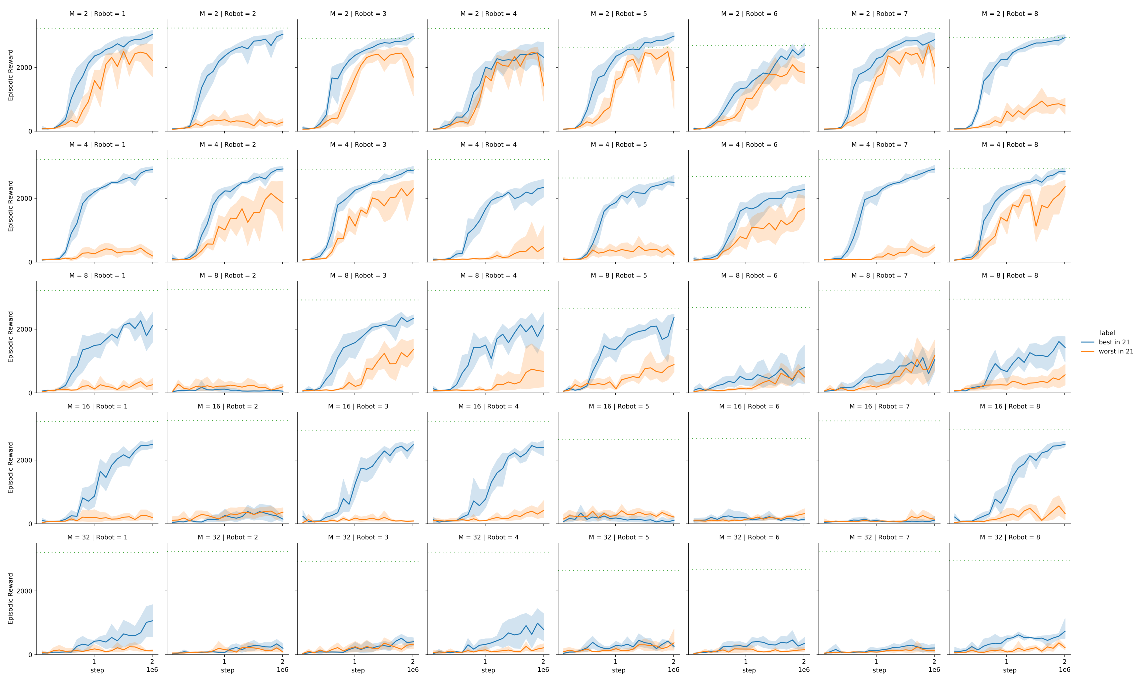


Figure 4.5: Learning curves and probability densities for M2, M4, M8, M16, and M32. The green dashed line is the mean values while using M0.

From left to right, there are eight robot bodies.

From top to bottom, the permutation distance increases: the first row is M2, and the second row is M4, and so on.

The green dashed lines stand for the average final episodic reward while using M0.

The blue lines are the arrangements that result in the best performance in each permutation distance. They are the best in the 21 arrangements. The orange lines are the worst in the 21 arrangements.

The agents perform worse while using arrangements with larger permutation distances to *M0*. In other words, on average, M2 is worse than M0, and M4 is worse than M2, and so on. This indicates that there are gradients around M0, and in theory, an optimization program could find M0 or any other optimal arrangement from an arbitrary arrangement. The possibility of optimization and the difficulties later in Section 5.2.

It is observed, in the learning curve for M2 and Robot 5, the performance of the best M2 arrangement is better than M0. This indicates that there might exist other arrangements that are better than the current M0. It reminds us M0 was constructed by me looking at the visualization. In a more complicated case, the visualization cannot reveal the correspondence, and optimization algorithms are needed to find a better arrangement.

4.3.2 Difference in same distance

In the previous section, the learnabilities of arrangements with different permutation distances were compared. One might ask, will there be a difference in the arrangements even when they are in the same permutation distance? For example, is an M2 arrangement differ from another M2 arrangement?

As suggested in Figure 4.5, there might be a difference, so the best and the worst arrangements are selected in each permutation distance, and agents are trained independently using these arrangements with 10 different random seeds, and the comparison of the learnabilities is shown in Figure 4.6.

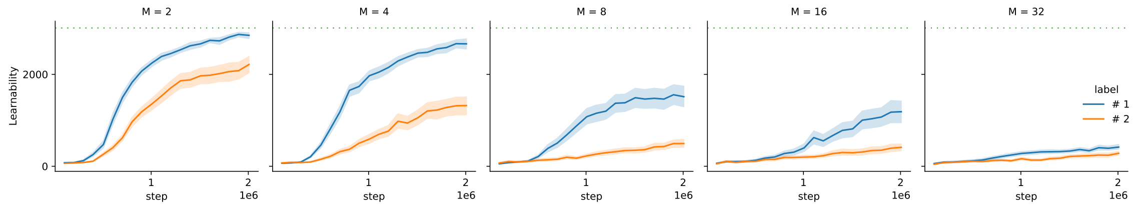


Figure 4.6: Independently compare the best arrangement (#1) and the worst arrangement (#2) in a certain permutation distance.

Here the layout is slightly different, from left to right, they are M2, M4, M8, M16, and M32.

The green dashed lines are the performance on M0, and they are also averaged across eight robots.

The blue lines labeled #1 are the best arrangements, and the orange lines labeled #2 are the worst arrangements.

An obvious difference is observed between two M2 arrangements, and between two M4 arrangements, and so on.

These 10 additional runs for each M are done independently, so these results show even in the same permutation distance, the difference in learnabilities exists.

Chapter 5

Discussion

5.1 Compare to the State-of-the-art

In all three experiments, some arrangements can make the MRCC problem easier than others.

But one would ask, as shown in Table 5.1, since Huang et al. 2020 [53] has provided a possible solution for the MRCC problem, why should one consider the arrangement problem in the first place?

To address this question, let us take the TDBOC experiment in Chapter 4 as an example to show why the optimal arrangement is important.

Recall that the problem is to learn to control eight robot bodies with different topology, but the problem is designed in a way that the optimal arrangement M0 is known.

First, as Huang et al. 2020 suggests, the baseline uses an arbitrary arrangement, and the agent using standard RL would fail to learn to control all eight different bodies.

Table 5.1: A comparison with previous MRCC work with topologically different bodies

Paper	Maximum number of different bodies	Typical training time
Devin et al. 2016 [42]	2	-
Nagbandi et al. 2019 [50]	2	-
Huang et al. 2020 [53]	**23**	1 week
TDBOC	8	**3 hours**

Then, the state-of-the-art approach introduced by Huang et al. 2020 is to use one shared network module and rewire the architecture each time before controlling a body. If the approach could eventually learn to control all eight bodies, a typical learning process would take a week.

However, if the known optimal arrangement M0 is used along with the standard RL, the agent can learn to control all eight different bodies in less than three hours. The optimal arrangement makes the learning problem much easier.

But how to obtain the optimal arrangement for a given set of robot bodies?

5.2 Search for Optimal Arrangements

It was shown that the arrangement matters, the next step would naturally be developing optimization methods that can find optimal arrangements.

The simplest search method is the *Random Search* (RS). One might want to calculate how many possible arrangements are there. Suppose there are r robots, and each robot has j joints. Assuming $\mathfrak{A}_1$ does not matter because of the symmetry of the neural network. For each of the robots 2 to r, there are $j!$ possible orders. In total, the number of possible arrangements n is:

$$n = (j!)^{r-1} \tag{5.1}$$

If $r = 16$, $j = 8$, then $n \approx 10^{93}$. It will not realistic for RS to find the optimal arrangement in such a vast search space.

Now consider the *Evolutionary Algorithm* (EA). Is it possible for an EA to find the direction towards the optimal arrangement? For example, is it possible to find an M3 arrangement from an given M4 arrangement? Suppose, again, there are r robots, and each robot has j joints. In total, the number of possible permutations n is:

$$n = r \cdot \frac{j!}{2!(j-2)!} \tag{5.2}$$

$$n = \frac{1}{2} rj(j-1) \tag{5.3}$$

If $r = 16$, $j = 8$, then $n = 448$. Among those possible permutations, at least one permutation can turn an M4 into an M3. This is much better than RS. However, in practice, it might still take a long time. The reason is that it needs a complete run of training (0.5 - 3 hours in our case) to get one evaluation for an arrangement. In addition, since this outcome is stochastic, and the variance of it is high, multiple runs are needed to obtain one accurate evaluation of an arrangement. Suppose three independent runs are performed for each evaluation, there will be $3 \times 448 = 1344$ runs for an arrangement to move one step towards M0. So EA might be possible for simple problems, but it will be very expensive to scale.

Now consider the *Gradient-based Methods* (GM). Since DRL used by the agent is a gradient-based method, it might be possible to compute the gradients for the elements in $\mathfrak{A}'_r$ at each time step. There will be much more signals than EA. But GM can only operate regular matrices with real number values, and it can not update a permutation matrix directly. In the next section, the idea of *Soft Arrangement* will be introduced, which can relax the assumption for GM.

5.3 Soft Arrangement

In Section 1.1, $\mathfrak{A}$ is defined to be the arrangement of a set of robot, and $\mathfrak{A} = [\mathfrak{A}_1, \mathfrak{A}_2, \cdots]$, where $\mathfrak{A}_r$ the permutation matrix for the r-th robot. each $\mathfrak{A}_r$ is defined to be a permutation matrix. The input to the agent can be obtained using Equation (1.1):

$$\mathcal{C}_r^{in} = \mathfrak{A}_r \mathcal{O}_r$$

However, a permutation matrix needs to be square and there is only one non-zero number 1 in each row and each column. But since it is a matrix multiplication, and there is no constraint on $\mathcal{C}_r^{in}$, the constraints on $\mathfrak{A}_r$ can be relaxed to make it a regular matrix.

For example, if $\mathfrak{A}'_r = \begin{pmatrix} 0.1 & 0.3 & 0.5 \\ -0.2 & 0.1 & 0.2 \\ 1.2 & 0.1 & 0.2 \end{pmatrix}$, it will still result in a valid $\mathcal{C}_r^{in}$.

An arrangement that contains such regular matrices $\mathfrak{A}'_r$s can be called the *Soft Arrangement*. There might be several potential benefits from this relaxation:

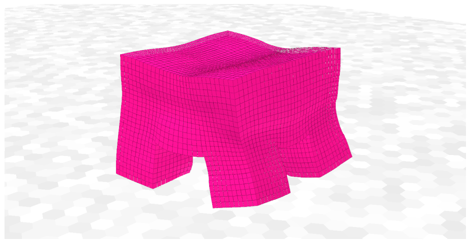
First, it might be able to solve the scale problem. For example, if there are two similar robots, the only difference between them is they interpret the motor commands differently. If one motor receives the real number 1.0, it might exert 1.0 N force, while the other might exert 10.0 N force. While the neural network could handle this difference, but it will be convenient to re-scale the commands through $\mathfrak{A}'_r$.

Second, it might be able to handle duplicate similar joints. For example, imagine there are two robots with many parallel limbs. One has 10 limbs, and the other has 20. When using a permutation matrix, one command can only be sent to one joint, no more. But if it is a regular matrix $\mathfrak{A}'_r$, one command can be duplicated to many joints.

Third, it can facilitate GM because the technique for optimizing a regular matrix was well developed.

5.4 Arrangements in Soft Robots

While a rigid-body robot usually have very limited body parts, a soft robot might need a large number of body parts to exhibit the softness of the body.


(a) A soft robot with 13,900 voxels

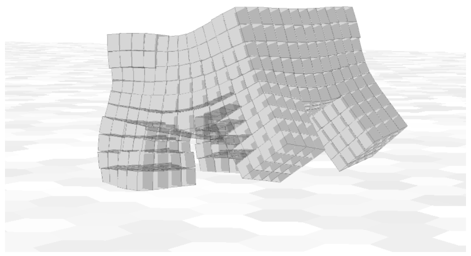

(b) A soft robot with 928 voxels

Figure 5.1: Two soft robot examples.

For example, Figure 5.1 shows two soft robots from the *Voxcraft* [65] simulation.

The pink soft robot on the left consists of 13,900 voxels, and thus 13,900 body parts, and 13,900 actuators. The grey soft robot on the right consists of 928 voxels, and thus 928 body parts, and 928 actuators. They are all quadrupeds, so there is a potential correspondence in morphology that can be exploited. However, the resolutions of these two bodies are different. When a DRL agent is used to control these two bodies, although it would be great to have an optimal arrangement that can map the legs of two robots to the same input unit of the agent, it is hard to find such an optimal arrangement.

To my best knowledge, there is no research discussing the MRCC problem in soft robotics. The problem of MRCC and optimize the arrangements will be much harder in soft robotics.

Chapter 6

Conclusion

This thesis investigated the arrangement of the inputs and outputs in the *Multi-Robot Continuous Control* (MRCC) problem in the domain of *Deep Reinforcement Learning* (DRL). The MRCC problem in DRL is a newly emerging field, and the arrangement becomes non-trivial when one learned agent is trying to control different robot bodies. In practice, no previous work has noticed the effect of the arrangement.

In this thesis, it is hypothesized that for a given set of different robot bodies, different arrangements will result in different learnability. In total three experiments are conducted to test this hypothesis in three different scenarios.

In the first experiment, Multiple parametrically different bodies are procedurally generated. The difference between the *Aligned* arrangement and the *Randomized* arrangements are compared. The results shows the control problem with the *Aligned* arrangement is easier (more learnable) than the one with the *Randomized* arrangements. It also shows that the number of robots in a set matter, and comparing MRCC problems with a different number of robots could be misleading.

In the second experiment, A set of four existing topologically different robot bodies are used. There is no obvious correspondence in the body parts across different robot bodies. Two different, arbitrary arrangements #1 and #2 are compared. The results shows that the control problem with the #1 arrangement is slightly easier than the one with the #2 arrangement. The difference is statistically significant.

In the third experiment, Multiple topologically different robot bodies with obvious correspondence are manually constructed. One optimal arrangement *M0* is known according to the correspondence. Other arrangements are obtained by permuting *M0*. The *Permutation Distance* to *M0* is defined to be the times of permutations needed to obtain an arrangement from *M0*. The results shows that, on average, smaller permutation distance will result in better final performance. The different arrangements with the same amount of permutation are also compared. The results shows that different arrangements with the same permutation distance still result in different learnability.

All three experiments provided positive evidence that supports my hypothesis: different arrangements result in different learnability.

In addition, the possibility and obstacles of searching for the optimal arrangement for an arbitrary set of robots are discussed. And how the MRCC problem would evolve in soft robotics.

Bibliography

[1] Angus C. Graham. *The Book of Lieh-Tzu.* Columbia University Press, 1960.

[2] K. Čapek and C. Novack. *R.U.R. (Rossum's Universal Robots).* Penguin classics. Penguin Books, 2004.

[3] G. Devol. Programmed article transfer, U.S. Patent US2988237A, 1954.

[4] A. Gasparetto and L. Scalera. A Brief History of Industrial Robotics in the 20th Century. *Advances in Historical Studies*, 08(01):24–35, 2019.

[5] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, pages 15–22, Not Known, 1994. ACM Press.

[6] Sam Kriegman. Why virtual creatures matter. *Nature Machine Intelligence*, 1(10):492–492, 2019.

[7] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974, 2000.

[8] Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012.

[9] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013.

[10] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859, 2020.

[11] Josh C Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239, 2011.

[12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019. arXiv: 1509.02971.

[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*, April 2017. arXiv: 1502.05477.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. arXiv: 1707.06347.

[15] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[16] Robert Kwiatkowski and Hod Lipson. Task-agnostic self-modeling machines. *Science Robotics*, 4(26):eaau9354, January 2019.

[17] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143):20170937, 2018.

[18] David Ha. Reinforcement Learning for Improving Agent Design. *Artificial Life*, 25(4):352–365, November 2019. arXiv: 1810.03779.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[20] Richard S. Sutton. Introduction: The Challenge of Reinforcement Learning. In Richard S. Sutton, editor, *Reinforcement Learning*, The Springer International Series in Engineering and Computer Science, pages 1–3. Springer US, Boston, MA, 1992.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 318–362. MIT Press, Cambridge, MA, USA, January 1986.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. Number: 6088 Publisher: Nature Publishing Group.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition, March 2008. Publication Title: http://dx.doi.org/10.1162/neco.1989.1.4.541 Publisher: MIT Press 238 Main St., Suite 500, Cambridge, MA 02142-1046 USA journals-info@mit.edu.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. Publisher: MIT Press.

[26] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, September 2014. arXiv: 1406.1078.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv: 1706.03762.

[28] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. _eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013. arXiv: 1312.5602.

[31] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. Number: 7587 Publisher: Nature Publishing Group.

[32] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020. Number: 7839 Publisher: Nature Publishing Group.

[33] Nicolas Heess, Greg Wayne, David Silver, Timothy Lillicrap, Yuval Tassa, and Tom Erez. Learning Continuous Control Policies by Stochastic Value Gradients. *arXiv:1510.09142 [cs]*, October 2015. arXiv: 1510.09142.

[34] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018. arXiv: 1801.01290.

[35] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. February 2018.

[36] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to Walk in the Real World with Minimal Human Effort. *arXiv:2002.08550 [cs]*, November 2020. arXiv: 2002.08550.

[37] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, Vancouver, BC, September 2017. IEEE.

[38] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. October 2019.

[39] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv:1706.05098 [cs, stat]*, June 2017. arXiv: 1706.05098.

[40] Nicholas C. Landolfi, Garrett Thomas, and Tengyu Ma. A Model-based Approach for Sample-efficient Multi-task Reinforcement Learning. *arXiv:1907.04964 [cs, stat]*, November 2019. arXiv: 1907.04964.

[41] Peter Henderson, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. Benchmark Environments for Multitask Learning in Continuous Domains. *arXiv:1708.04352 [cs]*, August 2017. arXiv: 1708.04352.

[42] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2169–2176, May 2017.

[43] Chuanyu Yang, Kai Yuan, Qiuguo Zhu, Wanming Yu, and Zhibin Li. Multi-expert learning of adaptive legged locomotion. *Science Robotics*, 5(49), December 2020. Publisher: Science Robotics Section: Research Article.

[44] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, July 2017. arXiv: 1703.03400.

[45] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. ProMP: Proximal Meta-Policy Search. *arXiv:1810.06784 [cs, stat]*, December 2018. arXiv: 1810.06784.

[46] Chrisantha Thomas Fernando, Jakub Sygnowski, Simon Osindero, Jane Wang, Tom Schaul, Denis Teplyashin, Pablo Sprechmann, Alexander Pritzel, and Andrei A. Rusu. Meta-Learning by the Baldwin Effect. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 109–110, July 2018. arXiv: 1806.07917.

[47] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. *arXiv:1903.08254 [cs, stat]*, March 2019. arXiv: 1903.08254.

[48] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. *arXiv:1910.10897 [cs, stat]*, October 2019. arXiv: 1910.10897 version: 1.

[49] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-Task Reinforcement Learning with Soft Modularization. *arXiv:2003.13661 [cs, stat]*, December 2020. arXiv: 2003.13661.

[50] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning. *arXiv:1803.11347 [cs, stat]*, February 2019. arXiv: 1803.11347.

[51] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018*

IEEE International Conference on Robotics and Automation (ICRA), pages 3803–3810, May 2018. arXiv: 1710.06537.

[52] Qiang Zhang, Tete Xiao, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Learning Cross-Domain Correspondence for Control with Dynamics Cycle-Consistency. *arXiv:2012.09811 [cs]*, December 2020. arXiv: 2012.09811.

[53] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control. *arXiv:2007.04976 [cs, stat]*, July 2020. arXiv: 2007.04976.

[54] Kaleigh Clary, Emma Tosch, John Foley, and David Jensen. Let's Play Again: Variability of Deep Reinforcement Learning Agents in Atari Environments. *arXiv:1904.06312 [cs, stat]*, April 2019. arXiv: 1904.06312.

[55] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. August 2018.

[56] Wenhao Yu, C. Karen Liu, and Greg Turk. Multi-task Learning with Gradient Guided Policy Specialization. *arXiv:1709.07979 [cs]*, March 2018. arXiv: 1709.07979.

[57] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[58] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[59] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[60] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016. arXiv: 1606.01540.

[61] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[63] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, October 2018. arXiv: 1506.02438.

[64] Antonin Raffin and Freek Stulp. Generalized State-Dependent Exploration for Deep Reinforcement Learning in Robotics. *arXiv:2005.05719 [cs, stat]*, May 2020. arXiv: 2005.05719 version: 1.

[65] Sida Liu, Sam Kriegman, David Matthews, and Josh Bongard. Voxcraft-sim, a highly parallelized physics engine that can simulate the voxel-based soft robots, May 2020.