Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Robotic control with deep-learned structured policies.

Dissertation thesis

Ing. Teymur Azayev

Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics
Supervisor: Doc. Ing. Karel Zimmermann, Ph.D.

Prague, November 2022

Thesis Supervisor:
Doc. Ing. Karel Zimmermann, Ph.D.
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nam. 13, Praha 2
Czech Republic

Declaration

I hereby declare I have written this doctoral thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, November 2022

.....
Ing. Teymur Azayev

Abstract

Data-driven methods for robotic control have been steadily gaining popularity over the past decades, showing high-performing results for complex, high-dimensional robot morphologies. Such methods usually entail learning a parametric function approximator (policy/control law) that maps a set of sensory inputs to action outputs. The most popular class of such function approximators is the neural network Multi-layer perceptron (MLP), consisting of several linear layers separated by a non-linearity. In conjunction with powerful trial and error algorithms such as Reinforcement learning, we are able to learn control policies that can maximize a given reward (cost) function in almost any domain. While most research is focused on learning algorithms and improving sample efficiency and sim-to-real adaptation, there is less attention to the actual function approximator that represents the control law. In this thesis, we show that for some robotic morphologies, using a monolithic MLP or Recurrent Neural network (RNN) can lead to issues during the learning phase and well as an overall poor result, especially for structured tasks such as locomotion. In our first published work, we show this in experiments on learning adaptive locomotion behaviors for legged hexapod robots and show that it can be effective to partition the control problem into several discrete tasks, learning an optimal policy for each one and then learning to switch between them. In subsequent work, we enforce well-explainable structural elements into the overall architectural design while preserving the end-to-end training. This is done by starting with an initial hand-designed algorithm and successively replacing various heuristic decision points with neural network modules that can then be trained using black-box optimization methods. In our third published work, we show yet another way in which we can structure the control policy as a hybrid of hand-designed knowledge and learnable elements, giving a sample-efficient and more interpretable architecture that can be used to learn autonomous flipper control for articulated tracked robots. We verified our experiments in this work on a real platform in conjunction with a full navigation stack, as well as deployed part of our algorithm in the DARPA Subt urban competition with good results. Finally, we describe various methods of simulation to real robot policy transfer and discuss how the various methods relate to a theoretical Bayesian approach in an increasing learning complexity hierarchy of agents.

Keywords: structured policies, neural networks, reinforcement learning, robotics

Abstrakt

Metody řízené daty pro robotické řízení si v posledním desetiletí neustále získávají popularitu a ukazují slibné výsledky pro složité, vysoce rozměrné morfologie robotů. Takové metody obvykle zahrnují učení aproximátoru parametrických funkcí (řídicí pravidlo), který mapuje sadu senzorických vstupů na akční výstupy. Nejoblíbenější typ takových aproximátorů je neuronová síť Multi layer perceptron (MLP), skládající se z několika lineárních vrstev oddělených nelinearitou. Ve spojení s výkonnými algoritmy na principu pokus-omyl, jako je posilované učení, jsme schopni se naučit řídicí pravidla, která můžou maximalizovat dané funkce odměny (nákladů) téměř v jakékoli doméně. Zatímco většina výzkumů je zaměřena na učení algoritmů, zlepšování efektivity vzorků a adaptace mezi simulátorem a realitou, je věnována menší pozornost na skutečný aproximátor funkcí, který představuje řídicí pravidlo. V této práci ukazujeme, že pro některé robotické morfologie, řízení pomocí monolitické MLP nebo rekurentní neuronové sítě (RNN) může vést k problémům během fáze učení a také k celkově špatným výsledkům, zejména pro strukturované úkoly, jako je lokomoce. V naší první publikované práci ukazujeme na experimentech učení adaptivního lokomočního chování pro šestinohé roboty, že může být efektivní rozdělit problém řízení do několika samostatných úkolů a naučit se optimální řídící pravidlo pro každý z nich a pak se naučit mezi nimi přepínat. V následné práci prosazujeme dobře vysvětlitelné konstrukční prvky do celkového návrhu architektury při zachování end-to-end učení. To se provádí tak, že se začne s počátečním ručně navrženým algoritmem a postupně se nahradí různé heuristické rozhodovací body moduly neuronové sítě, které lze následně trénovat pomocí metod black-box optimalizace. V našem třetím publikovaném díle ukazujeme ještě jinou cestu, kterou můžeme strukturovat kontrolní politikou, jako hybrid ručně navržených znalostí a naučitelných prvků, což poskytuje vzorově efektivní a lépe interpretovatelnou architekturu, kterou lze použít k učení autonomního ovládání flipperu pro kloubové pásové roboty. V tomto článku jsme ověřili naše experimenty na skutečné platformě ve spojení s komplétním navigačním systémem, a také nasadili část našého algoritmu v soutěži DARPA Subt s dobrými výsledky. Nakonec popisujeme různé metody simulace na přenos pravidla řízení robotů mezi simulatorem a reálným robotem a diskutujeme, jak se různé metody vztahují k bayesovskému přístupu s rostoucí hierarchii složitosti učených agentů.

Klíčová slova: Strukturované ridicí pravidla, neuronové sítě, posilované učení, robotika

Překlad názvu Řízení robotu pomocí strukturovaného hlubokého učení.

Acknowledgements

I am thankful to my Alma Mater, the Czech Technical University, and especially the department of Cybernetics, led by Prof. Tomas Svoboda, for providing me with the facilities that allowed me to pursue my doctoral studies. I am also grateful for the Czech Grant Organization (GACR) [1], the Research Center for Informatics (RCI) [2] and the SGS Conference Grant [3] for sponsoring my research. Finally, I am thankful for my supervisor, Doc. Karel Zimmermann, for his advice and feedback that helped me to finish my studies, and my parents for their support.

[1] 20-29531S
[2] reg. No.: CZ.02.1.01/0.0./0.0./16_019/0000765
[3] SGS22/111/OHK3/2T/13

List of Tables

List of Figures

Contents

Part I

Introduction and preliminaries

0.1 Introduction

Robots are becoming increasingly prevalent in a wide range of domains such as manufacturing [1], inspection [2], delivery [3], human assistance [4], surgery [5] and search and rescue [6], [7]. In some places, such as factories, the role of robots is irreplaceable. Although these machines are very complex and take considerable effort to engineer, in many cases, the software control of the robot is still the weakest link, requiring substantially more research and development. This is because, outside a predefined and known environment, robots have to use high dimensional inputs from various modalities such as video and LIDAR, as well as estimate various unobservable variables in order to set up a control problem that can be solved to infer robot movement. Robotic control tasks are traditionally approached using first principle methods. That is, using physical and mathematical knowledge of the system in order to derive an efficient control law for that system. This works very well for rigid robotic platforms where the interaction with the surrounding environment can be modeled reasonably well. A suitable example is a robotic hand that moves heavy loads from one conveyor belt to the next. The robot model is known, and the positions of the object are well-defined, which gives a problem that can be solved using a mixture of analytical and numerical techniques. This approach, however, can become problematic when the system dynamics are difficult or exhibit a high dimension. One such example is the locomotion problem of a 6-legged robot, whose interactions with the terrain are difficult to model and are often unknown. This results in a high dimensional control problem that is computationally demanding and needs to be solved using heuristic approaches. Another such example is fluid dynamical systems that are notorious for requiring hours of simulation for only a handful of frames [8].

Data-driven methods have enabled users to learn impressive robotic control algorithms [9] [10] for various robotic platforms with little or no expert knowledge required. The data-driven approach is to leverage robot interaction with the environment to extract a suitable control law, in contrast to deriving it from first principles. The advantage is that we don't require domain and control theory expertise, but on the other hand, it means that we need to be able to simulate rich, high-fidelity interactions of the robotic platform in the environment that it's going to be deployed on. This often includes rendering modalities such as cameras and LIDAR, which can add to the rendering cost. Such environment interactions, also called rollouts, in conjunction with a user-defined reward function, can be used with various learning approaches to extract powerful control laws on difficult-to-control and or high-dimensional systems. The most powerful approaches use trial and error methods such as Reinforcement learning [11] and Random Search [12], [13]. This can be simply explained as trying various actions and strategies and using the reward function to improve the policy that generates the actions. In this regard, it is similar to how humans learn to perform various tasks. There are various ongoing directions of

research around this data-driven paradigm. One of them is improving the actual learning algorithms themselves, improving sample efficiency and performance. Another direction is using previously learned information to bootstrap a newly instantiated agent from available information rather than learning *tabula rasa*. This area is called transfer learning. Another interesting direction, called Meta learning, uses an outer, more general learning loop which improves the inner learning loop, leading to more efficient training.
One drawback of data-driven models is that they usually leverage powerful black box functions, which is one of the reasons that these methods require a large number of trials and sometimes generalize poorly. A general black box function approximator assumes very little about the task at hand. In fact, in most practical tasks that involve structured data, appropriate Neural Network architectures have to be used to obtain good results. For example, in computer vision, 2D and 3D Convolutional Neural networks (CNN) currently achieve state-of-the-art results [14], [15]. In audio and other sequence recognition, mostly Recurrent Neural Networks (RNN) [16] are used, as well as 1D CNNs [17], with newer architectures such as Transformers showing significant improvements in Natural Language Processing [18].

In our work, we focus on using structured function approximators for robotics control that have better inductive priors and can lead to better training dynamics, better efficiency and generalization, and higher performance. This is in an effort to merge first principles and heuristics with data-driven approaches to leverage the power of computation. This is not a new idea, as there have been several efforts that attempt to use suitable priors for specific robotic morphologies, such as the use of graphs for legged robots [19]. Other works, such as [20], attempt to break down neural networks into linear and non-linear parts that are individually dedicated to various parts of the state space. Nevertheless, we feel that this area is understudied and can bring many improvements to data-driven control, as most methods usually focus on the more efficient and robust training algorithms [21], adaptation [22] and few shot learning techniques [23].

0.2 Aims and contributions of the doctoral thesis

The main goal of this thesis is to progress the state-of-the-art in data-driven robotic control for complex robotic morphologies and to demonstrate feasible methods that can be applied and used on real platforms. Although there are many components that go into making this goal possible, our main focus is on the structure of the learnable control function (policy). Our aim is to bridge hand-designed knowledge of a given platform, and black-box approaches into learnable hybrid functions. We show that such functions are more suitable than monolithic black box function approximators such as the commonly used Multi-Layer Perceptron (MLP).

We describe the data-driven robotic control problem and the various state-of-the-art approaches that have enabled high-performing results in research and industry. We also give an overview of the well-known problem of distribution mismatch, also known as *covariate shift* [24], when deploying a simulator-learned control policy on the real platform. In this section, we also take a broader look at general Reinforcement Learning in an unknown environment, how it relates to the sim-to-real transfer problem, and where it stands in comparison to proposed theoretical Bayesian approaches.

One of the platforms that we study is the six-legged hexapod morphology. This is a high-dimensional control problem whose solutions are structured, low-entropy gait sequences that are adapted for various terrains. In our publication titled: "Blind Hexapod Locomotion in Complex Terrain with Gait Adaptation Using Deep Reinforcement Learning and Classification", published in the "Journal of Intelligent & Robotic Systems", we showed that it is possible to learn optimal locomotion using Reinforcement Learning in simulation on a wide range of challenging terrains. Our contribution was to show that it's very difficult to learn a single monolithic policy that can perform well on a wide range of terrains, but we can learn *expert* policies that perform very well on individual terrains. We then learned a recurrent neural network policy that classifies terrains from state-action history sequences and picks the appropriate optimal policy network for locomotion. We showed that we are able to seamlessly switch between difficult terrains using our hierarchical structure, something that was not possible to learn end to end using a single policy using an MLP or a Recurrent Neural Network (RNN) [25].

In further work on the hexapod platform, we showed that we could improve hand-crafted cyclic locomotion generators by substituting various gait decision points with learnable Neural Network modules. By using Random Search methods such as CMA-ES [26], we are able to optimize gait phase decision, leg swing height, and body direction, leading to superior results compared to the baseline variant and compared to monolithic black box approaches. This work, named "Improving procedural hexapod locomotion generation with neural network decision modules", was presented as a conference paper in the "Modelling & Simulation for autonomous systems conference (MESAS2022)". The point of this paper was to demonstrate that we don't necessarily need to start with a clean slate when doing data-driven control. Instead, we can start with a working approach that was derived from first principles or use heuristics and replace various sub-functions or discrete decision points with learnable Neural Network modules and use Random Search to improve the overall performance.

Another platform that we worked with was the articulated tracked robot morphology that has steerable tracks and four individually controllable flippers. We studied the problem of obstacle negotiation and proposed an autonomous flipper controller that used exteroceptive data

to imitate a small set of demonstrations by the robot operator. This work, named ”Autonomous State-Based Flipper Control for Articulated Tracked Robots in Urban Environments,” was accepted and published in the ”Robotics and Automation Letters (RA-L)” journal and the ”IEEE IROS2022” conference, where it was presented. Similar to our previous work on the hexapods, the goal was to demonstrate that there are aspects of the control that can easily be hand-coded and other aspects that are best learned using suitable Neural Network modules. After analyzing the obstacle negotiation task, we formulated the control problem as a sequence of discrete phases, each of which is humanly interpretable, and implemented suitable hand-crafted flipper templates and low-level behaviors for each phase. The decisions to transition between the discrete phases were learned using Imitation Learning [27]. As a suitable neural network structure, we proposed a novel soft-differentiable state machine architecture that keeps a complete probability distribution of all phases at each time step. This allows it to essentially be in all phases at the same time, allowing the propagation of temporal gradients throughout the decision process. We showed that our approach is sample efficient, learning from only several minutes of data and outperforming the previous state-of-the-art, as well as other baseline architectures. Our approach also has excellent zero-shot performance on the real platform. A preliminary part of the work was used to control our articulated tracked platform in difficult urban environments in the DARPA Subterranean challenge, published in the ”Field robotics” journal under the title ”System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA Subterranean Challenge”.

Besides our academic publications, we also describe some of our additional contributions during the participation in the DARPA Subterranean challenge in the Urban and Final rounds. The appendix includes several technical and algorithmic contributions, mainly to Clearpath Husky A200 and the Boston Dynamic Spot platform.

0.2.1 Summary of academic publications

Articles in Peer-reviewed Journals

- T. Azayev and K. Zimmerman, “Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification”, *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3, pp. 659–671, 2020
- T. Azayev and K. Zimmermann, “Autonomous state-based flipper control for articulated tracked robots in urban environments”, *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–8, 2022

- T. Roucek, M. Pecka, P. Cízek, *et al.*, "System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA subterranean challenge", *CoRR*, vol. abs/2110.05911, 2021. arXiv: 2110.05911. [Online]. Available: https://arxiv.org/abs/2110.05911

Articles in conference proceedings

- Teymur Azayev, Jiri Hronovsky and Karel Zimmermann, "Improving procedural hexapod locomotion generation with neural network decision modules" . Presented at the Modelling & Simulation for autonomous systems conference (MESAS2022).

0.3 Learning robot control

0.3.1 Introduction

The task of robot control can be formulated by mapping a set of observations to actions at every discrete time step, given a distribution of starting states. The most logical approach to this task is to start from first principles and engineer an analytic or heuristic approach that solves the task optimally or with satisfactory performance. Unfortunately, most real systems are complex and non-linear, often with high dimensional input and output spaces which can make it difficult to come up with a good solution. Some platforms such as soft robots [31] or more exotic morphologies such as tensegrity robots [32] are good examples where models can become extremely complex and might not be feasible to work with.
An alternative approach is to define the control law (policy) using a general learnable function that maps observations to actions. A suitable performance evaluation procedure and algorithm can then be used to learn the parameters of the policy. Specific approaches are discussed below.

0.3.2 Control problem formalization

We consider the control law as an agent which interacts with an environment E in discrete time steps. We can describe the complete state of the agent, along with the environment, using a state vector $s \in S$, starting at an initial distribution ρs. The state s implies a Markov property which says that $P(s_t|s_{t-1},...,s_0) = P(s_t|s_{t-1})$. The environment state emits observations o_t, which are informational subsets of states s_t. We define the agent as a function $\pi : o_t \rightarrow a_t$, which maps observations $o_t \in O$ to actions $a_t \in A$. The environment is stepped in discrete time steps after receiving an action from the agent and can be described by transition function $T : s_t \times a_t \rightarrow s_{t+1}$. We also define a reward function $R : s_t \times a_t \rightarrow r_t \in \mathbb{R}$ and a discount factor $\gamma \in [0,1]$. The above form a markov decision process (MDP) denoted by the tuple

$$(S, A, T, R, \gamma)$$

In the case where the observations received by the agent are o_t instead of s_t, we say that the environment is partially observable and, therefore, a Partially Observable Markov Decision Process (POMDP). The general objective in the MDP is to obtain agent parameters θ, which maximize the reward gathered by the agent within a finite horizon h, shown in equations 1.

$$J = \sum_{t=0}^{t=h} R(s_t, \pi_\theta(o_t)) \tag{1}$$

$$\theta^* = \arg\max_{\pi_\theta} J \tag{2}$$

0.3.3 Imitation learning

Imitation learning is a powerful learning technique that allows us to learn an agent policy π from given data obtained by an expert policy π^e, which can be a human demonstrator or another algorithm. The dataset $D = \{\tau^m, ..., \tau^m\}$ consists of state-action trajectories $\tau^i = \{(o_0^i, a_0^i), ..., (o_h^i, a_h^i)\}$ that we consider being suboptimal and noisy in some cases. The simplest way how to obtain such a policy π from dataset D is to formulate the problem as supervised learning, as shown in equation 3.

$$\pi_{\theta^*} = \arg\max_{\theta} \sum_{\tau \in D} \sum_{(o_t, a_t) \in \tau} L(\pi(o_t), a_t) \tag{3}$$

This is called behavioral cloning in literature. It works well for some systems but suffers a well-documented issue problem called *covariate shift* [24], which can be briefly explained as follows. The trajectories in dataset D are collected from a demonstration policy π^e, giving trajectories samples given by $\tau \sim P_{\pi^e}$. During inference, however, the trajectories are instead sampled from the learned policy π_θ, giving trajectory samples $\tau \sim P_{\pi_\theta}$. The mismatch in the trajectory distributions is referred to as the covariate or distribution shift. Intuitively, one can think of the issue as the learned policy making small mistakes which make it end up in states that it has not seen before, resulting in larger errors that compound and lead to failure. This issue is more prominent in inherently unstable control systems. We describe a system in our work on articulated tracked robots [33] where this method works well. Another limitation of this approach is that the performance is limited by the quality of the demonstrations.

A potential improvement to the above imitation learning can be to use offline Reinforcement learning [34] in which we can learn a more robust and potentially higher performing policy than we can achieve with behavioral cloning. The idea is that if we are given a large amount of data (possibly noisy), then we can specify an additional reward function $R(s,a)$ and learn a value

function or an action-value (Q-function) on the given data. The Q-function could be used to learn a policy, similarly to Deterministic Policy Gradients (DDPG)[35], without the need for further interactions with the environment.
More sophisticated methods which attempt to learn a policy from demonstrations include Inverse Reinforcement Learning (IRL) [36] where the task is to learn a reward function that best explains the given demonstrations and then learns a policy from the given reward function. This is a difficult task, as there are multiple reward functions for which a given set of demonstrations is optimal. Another disadvantage of IRL is that we require an inner reinforcement learning loop which makes the process computationally demanding.

0.3.4 Trial and error learning

A powerful way to learn agent policies can be achieved by trial and error algorithms. This can be summarized by the agent exploring random actions around its current policy and updating its own parameters based on how well those actions were evaluated. This approach is very general because it only requires the ability to query a reward function and makes no assumption on the differentiability of the reward function with respect to the parameters of the agent. Trial and error approaches can be divided into two main categories: Random Search (RS) and Reinforcement learning (RL). Random search encompasses all types of Evolutionary Algorithms (EA), but more commonly refers to techniques named Evolutionary Strategies (ES) that are often used as black box optimizers. We show using a general heuristic random search algorithm in figure 1 that the difference between these two approaches is minor and can be summarized in the following few points:

- RS uses a distribution of candidate parameters, whereas in RL usually, we usually keep a single agent at a given iteration.
- RS samples random candidate solutions, whereas RL evaluates the same candidate multiple times during a single iteration.
- RS performs evaluation usually deterministically, whereas RL depends on random rollouts for exploration.

The key takeaway is that RS explores in parameter space, whereas RL explores in action space. This can be significant depending on the system in question. They each have their advantages and disadvantages, but both are equivalent *complete* in the sense of learning power.

Amongst the most powerful RS algorithms is an Evolutionary Strategies algorithm named CMA-ES [26], as well as Augmented Random Search (ARS) [37]. The authors showed that ARS could train complex policies and is competitive with RL methods.

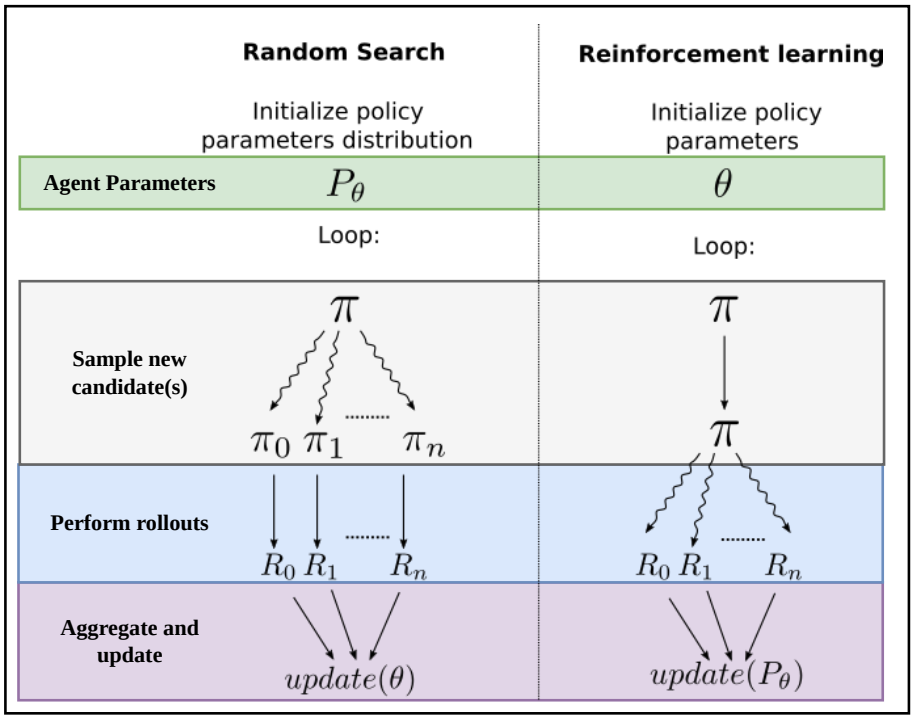

Figure 1: General Heuristic random search approach. The left shows general Evolutionary Strategies, and the right is Reinforcement learning. The wiggly lines denote a stochastic choice.

Learning complexity Trial and error methods usually have a high sample complexity, meaning that a large number of interactions are required with the environment to learn a good policy. The least efficient are Random Search (RS) methods. On the other hand, they are very simple to implement and can be *embarrassingly* (easily) parallelized, leading to low wall clock times when compute is available. On-policy RL algorithms, based on the policy gradient theorem [11], such as [38] and [39] are more efficient, requiring an order of magnitude fewer interactions than RS algorithms and can be parallelized up to a point. Off-policy RL algorithms [21], and [40] make the most efficient use of data and require an order of magnitude fewer data than On-policy RL algorithms. They are, however, sometimes more difficult to train. Using state-of-the-art Offline RL algorithms, it is even possible to train some real robotic tasks on the real platform using several hours of experience.

However efficient the learning algorithm, Trial and error methods still heavily rely on fast simulation. Fortunately, we have access to increasing compute power that comes with newer CPU architectures, as well as the use of accelerators such as Graphics processing units (GPU) and Tensor Processing Units (TPU) [41]. There are several dynamic simulation frameworks that are built to use such accelerators [42], [43], [44] and offer several orders of magnitude speeding up training.

Generalization, interpretability and control guarantees When designing a control law for a system, we almost never work with the actual system but with a model. This is true for first

principles approaches, as well as for data-driven approaches. For the latter, the model of the real system usually consists of interactions with a simulator. Both approaches suffer from the fact that the model and simulator differ from the corresponding real system, sometimes insignificantly, and sometimes by a large amount that can cause control performance degradation or failure. For various safety-critical systems, control theory is still the preferred approach because there are often provided theoretical guarantees that the control will work under a certain set of assumptions. These assumptions are usually made on approximations and linearization of the system, which can, in many cases, differ from the real system. It is, however, better to have some guarantee about an approximation of the system rather than nothing. When using data-driven control using function approximators, we generally don't have any theoretical guarantees due to the non-linear complexity of the function approximators. However, one thing that we can get is an empirical performance guarantee under various randomizations of the environment. This means simulating the system from various starting states, with all possible disturbances and all possible system parameter variations both during training and testing. In some cases, this can perhaps be more credible than theoretical guarantees on the proxy model of the system since it provides significantly more flexibility. In complex systems consisting of large perception, estimation, and control pipelines, such as self-driving cars, it is only possible to evaluate the system empirically. Empirical guarantees under simulation randomization can be seen as probabilistic Monte Carlo estimations of guarantees for a system of any complexity and non-linearity. The advantage of this approach is that it is easily parallelizable and thus scales with additional computing and given time. As mentioned in the above paragraph, some modern simulators, such as the NVIDIA Isaac [44], use the advantage of accelerated hardware and can even run a hundred or thousand instances in parallel, even on consumer GPUs.

0.3.5 Neural network function approximators

A data-driven control policy (agent) π_θ consists of a function that maps observations o_t to actions parametrized by learnable parameter vector θ. It has to be sufficiently expressive for the task at hand and has to be efficiently learnable within a given learning framework. For example, linear functions or low-order polynomials are not powerful enough to express complex policies required for a dynamic gait of a legged robot. Similarly, some function classes, such as decision rules in conjunctive normal form (CNF) and decision trees, are much more difficult to train because of the lack of differentiability. In most cases in learning control for difficult systems, the policy function π represents some form of Neural Network or a combination thereof. The most basic Neural Network is a multi-layer perceptron (MLP), which is a succession of linear layers separated by a non-linearity to create a resultant non-linear parametric function. The use of such classes of function approximators refers to Deep learning control. What makes Deep Neural Networks an attractive function class to use is that they automatically extraction

non-linear features on input data and are relatively easy to optimize.

Besides the raw learning power of the function approximator, the performance will also be heavily tied to the degree of observability of the system when receiving observations o_t. In most cases, some partial unobservability exists. This can manifest itself, for example, in the delay between control and observations, in unmodelled higher-order dynamics in robot actuators, or incomplete states due to a lack of sensors. In many cases, this can be ignored, and an MLP policy function is sufficient. In some cases, partial observability is significant, and we need to add memory to alleviate the effect. As an example, one can imagine playing a first-person shooter game from images. Without memory, the agent does not know what happened in the previous frames and therefore doesn't know how to continue. Similarly, if we remove velocity information from a simple inverse pendulum system, then it's unclear what the optimal action should be. To solve such a problem, we require a Neural Network architecture that either efficiently processes entire sections of time series or processes observations individually while carrying over a learned memory vector that is used as input in the next time step. Such a memory vector represents relevant information about the history of the past observation-action pairs. The following list describes the various architectures that are often used when we require an agent that has memory, starting from more traditional to more state-of-the-art and experimental approaches.

- **MLP with memory:** One solution is to concatenate a history of observations o_t, o_{t-1}, o_{t-h} for a chosen horizon h and feed them as input to the MLP. This can work well if the dimension of the observations is not too large, but it ignores the correlated temporal structure in the observations and will therefore not be as sample-efficient and is prone to overfitting to temporal noise.

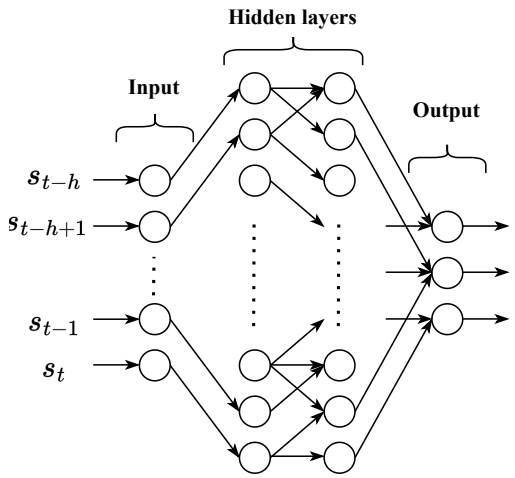

Figure 2: Multi Layer Perceptron on univariate temporal data with windowsize of h

- **Temporal convolutions** Temporal convolution networks (TCN) [17] are similar to 2D convolution networks used in various computer vision tasks, with the difference that they

operate on a single-dimensional sequence with multiple channels. This is a direct improvement to the above-discussed temporal MLP architecture in that we have multiple simple convolutional kernels that operate on the entire sequence (convolution operation); therefore, *reusing* the weights in the kernels. The outputs are fed into yet more convolutional kernals, which result in processing at various time scales and abstractions. Temporal convolutions are sometimes favored over other architectures, such as Recurrent Neural Networks (RNN) due to their simplicity and because they don't exhibit vanishing gradient issues that are common with RNNs. One common improvement in sequence processing with convolutional networks is temporal causality, where we make sure that subsequent layers only attend to inputs that have occurred in the past to prevent unwanted *lookahead* of the network. Figure 3 shows such a modification visually.

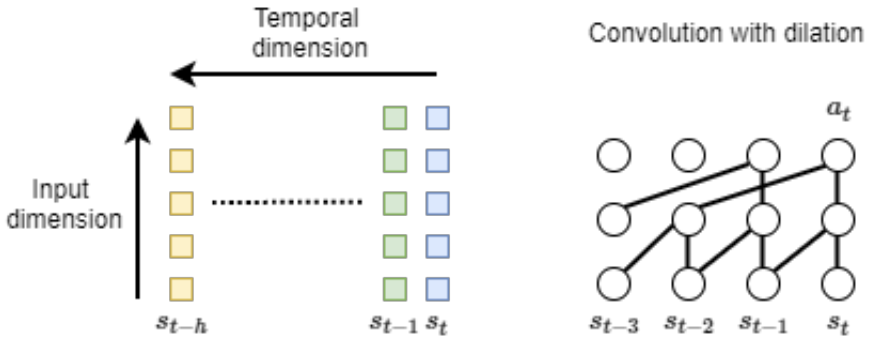

Figure 3: Temporal convolutions.

- **Recurrent neural networks (RNN)** [45]: When processing time-series data sequentially, it is necessary to consider the current observation in the context of the previous time steps up to a certain horizon. As shown in the above MLP architecture, one way of doing this is to use part of the history as input. The disadvantage of this is that this leads to an increasing amount of inputs with the history size. A more effective way to achieve a similar goal is to summarize the history at every timestep using a memory vector. RNNs do this by using the previous memory vector, along with the current observation, to predict an output, along with the next memory vector. This entire process is learned differentiably through the sequence. RNNs have been used for a wide variety of sequential tasks and have shown state-of-the-art performance in audio, speech, image sequence, and other problems. There are, however, issues connected with holding a memory vector over long time spans, such as memory fade and the gradient vanishing problem. There have been several architectures, such as the GRU [46] and LSTM [25], that propose advanced memory-gating mechanisms that alleviate the mentioned issues and allow the learning of sequences over large time spans. This is due to their ability to work over very long sequences. RNNs can be very difficult to learn in trial-and-error approaches such as Reinforcement learning due to the added complexity of deciding what to remember in the

next time step. A more detailed discussion on this matter can be found toward the end of this section.

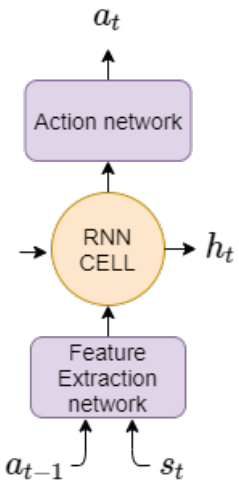

Figure 4: Recurrent Cell.

- **Attention mechanisms** A relatively new neural network architecture, illustrated in 5, initially introduced in [47] has lately grown very popular for certain sequential tasks such as natural language processing (NLP). The architecture calculates pointwise similarities (attention) between the inputs at various timesteps and then calculates the relevance of a specific input in the given temporal context. This results in the ability to accept very long sequences and calculate similarities between the inputs in a hierarchical fashion. This mechanism is good for sequences that do not necessarily have a smooth temporal correlation, such as natural language. One disadvantage of this mechanism is the quadratic computational complexity. In time series tasks, attention mechanisms encoder/decoder architectures, called Transformers [18] are sometimes used in conjunction with other methods such as LSTM and Temporal Convolutions.

0.4 Sim-to-real transfer

0.4.1 Introduction

In the introduction, we described a data-driven control paradigm using which we can obtain a control policy π_θ for a physical system by optimizing parameters π using trial and error-based methods such as Reinforcement Learning (RL). Due to the large number of random trials required, the training is usually done in simulation on a model that resembles the physical control system that we want to deploy our policy on. Unfortunately, there is often a discrepancy between the simulation and the real system. Efforts

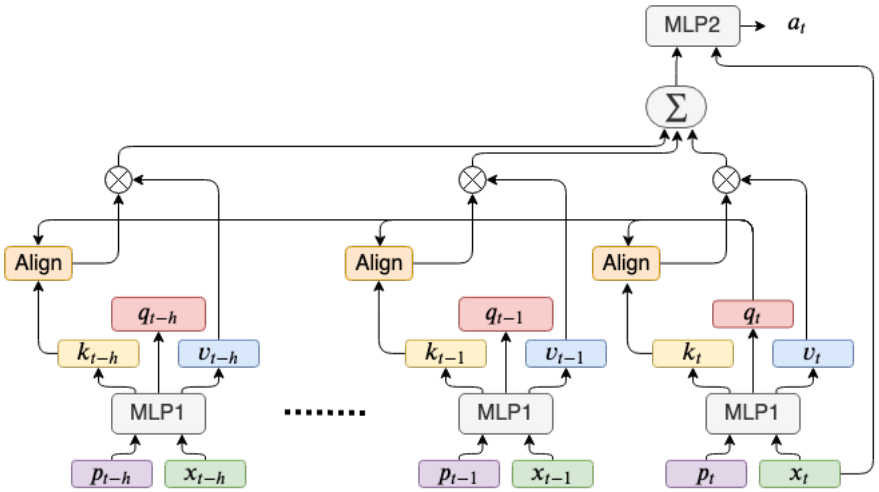

Figure 5: Attention mechanism.

that aim to close this gap are referred to as sim-to-real transfer methods. In this section, we will go over state-of-the-art sim-to-real methods, their advantages and disadvantages, and potential applications.

0.4.2 Related works

There have been a plethora of proposed techniques that can be used to bridge the Sim-to-real gap and enable simulator-trained policies to work better on a real platform. We consider grouping the most used techniques into several main categories.

System Identification The lowest hanging fruit when transferring a policy learned in simulation to the real platform is to match the system in simulation as much as possible to the real platform. This can be done manually but also more effectively using a data-driven approach. If data can be gathered from the real platform beforehand, then we can use the performed actions to drive the agent in simulation and then compare the state trajectories with the ground truth data and optimize the parameters so that the model trajectories are as close as possible to the ground truth data. Efficient black-box approaches such as evolutionary strategies [13], or Bayesian approaches [48] can be used to optimize simulation parameters. There are also parallel differentiable simulators [43] that are already available that can perform numeric simulations of dynamical systems in a differentiable way, allowing us to use methods such as gradient descent to optimize the parameters.

Transfer learning In some cases, it is possible to reuse knowledge gained from one task to perform another task. This can simply be done by training a neural network on task T_1 and using the learned weights as an initialization for learning task T_2 with a lower

learning rate. In this case, we would refer to it as *fine tuning*. In some cases, such as computer vision, it is possible to use entire pre-trained feature extractors on large datasets and train a smaller NN on top of it. This can improve sample efficiency and generalization and is a technique that is very often used in machine learning. In the context of sim-to-real transfer, a policy could be trained in simulation and then finetuned with a significantly smaller set of real rollouts. Besides training and transferring feature extractors, it is also possible to train and transfer parts of policies if structured in a hierarchical manner, such as in Feudal Learning [49] or the Options framework [50]. There are specialized methods of training that address multi-task Reinforcement learning explicitly, such as in [51]. Training on multiple tasks can enable the transfer of skills that are common to those tasks.

Domain randomization and adaptation Domain randomization can be described as training in simulation on randomized instances of our environment. The rationale is that if the agent sees randomized environments during training time, there is a much larger chance that the real environment is an instance of the distribution of trained randomized environments, giving a policy that is more robust than one trained on a single instance. Domain randomization is usually divided into perception randomization and dynamics randomization.

- **Perceptual** High-dimensional sensor rendering is improving with every generation of simulators. State-of-the-art engines allow for almost real-time raytraced camera renders that are close to photorealistic. Simulation environments such as Gazebo and Nvidia Isaac [44] provide high-quality renders of LIDAR and other range sensors. Nevertheless, it often helps to randomize various rendering parameters of the scene to obtain robust feature detectors. For images, this is straightforward as we can randomize scene textures and common camera parameters while training an agent policy, as demonstrated in the task of learning dexterous manipulation behaviors in simulation [52]. It is also possible to adapt image inputs from the simulator using data from the real platform (or the other way around) with techniques such as Generative Adversarial Neural networks (GANs) [53]. The idea is that we can use a learned modifier Neural network that changes the input image to look like a canonical simulation image [54] and therefore decrease the domain shift between simulation and real images.
- **Dynamics** Domain randomization can also be done on system dynamics, such as in [55]. The work of [56] uses real data to steer the randomization distribution of the simulation. Dynamics randomization is often used with state estimation methods,

described in the next section

Latent state estimation The task of sim-to-real transfer can be treated as a Partially Observable Markov Decision Process where the variability in the randomized instances of the simulator and the real platform can be represented by a latent vector that encodes this variability. If we can encode this representation in a vector and give it as input to the agent during training and test times, then the agent should have an idea of which *type* of environment it is in and therefore be able to perform optimally. During training, we have the ground truth parameter vector that we use to generate the randomized instances of the environment, but during test time, we do not. One of the most popular ways of solving this revolves around the idea of using ground truth labels to learn an estimator of the latent vector from a series of agent observations and actions. This estimator can then be deployed on the real platform, and the estimation can then be used as an input to the agent [57]. This is conceptually similar to state estimation techniques that are used in control theory, except that, in this case, the estimated states are latent vectors that parameterize the random environments. It is also possible to use data-driven inverse dynamics to calculate corrections during deployment for the trained simulation policy [58].

Meta RL The sim-to-real gap can also be closed by using techniques that allow the agent to learn fast. This is done by formulating the *inner* training loop as a training example and using an *outer* optimization loop, which optimizes for the performance of the agent that is a result of the inner training loop. One commonly used example of this in machine learning is hyperparameter optimization. In the case of Reinforcement learning, we can use an outer optimization loop that can teach an agent to learn to adapt to the changes in the environment from a small number of rollouts (few-shot learning) or from a single interaction episode (one-shot learning). There are several aspects of the agent's policy that can be *metalearned*.

- **Hyperparameters** As mentioned above, the simplest form of meta-learning is through tuning hyperparameters of the learning algorithm, which leads to faster learning and higher performance. Such parameters can be the number of layers or units in the neural network policy, the learning rate and momentum of the optimizer, and other variables pertaining to the environment, such as episode length, etc.
- **Optimizer** Function approximators such as Deep Neural Networks have complex gradient descent mechanics due to the very high dimensional parameter space, and finding the optimal update rule is not trivial. The most commonly used techniques

use some sort of adaptive momentum or second-order information. There have been several works, such as [59], showing that learning an update rule for such an optimizer that is a function of the current updates and features from the history can increase the performance of the optimizer, decreasing learning times in several tasks.

- **Weight initialization** Neural networks are usually learned from scratch with random weights. It can be beneficial to learn a suitable starting point for a range of tasks that will enable faster learning from consequent gradient updates. A well-known example is the MAML algorithm [60] and its follow-up works, which learn a good initial parameter initialization that can be adapted from several episodes of experience.
- **Architecture** Using standard neural networks can be unsuitable for some tasks, especially when high dimensional inputs are involved. It can make sense to learn part of the structure in an outer loop that suits the task more. Several works, such as [61], have shown such an approach to convolutional neural networks (CNN) for image-related tasks. Earlier works on neuro-evolution, such as NEAT [62], use a similar approach to optimize the architecture along with the weights simultaneously.
- **Exploration** The problem of exploration is one of the most fundamental aspects of trial-and-error methods such as Reinforcement learning. The stochastic aspect of the policy is what allows the gathering of new experiences to optimize a reward function. This means that exploring relevant parts of the state space can make a big difference in the quality of the experience that is obtained from the rollouts. We could meta-learn an exploration function as a separate agent, which would lead to faster learning of the Reinforcement Learning agent. Ideally, we would like to learn an agent that learns to explore as an instrumental task while optimizing the target task. Such an agent would have to be learned end to end using a recurrent policy (with memory). Such work is explored in [63].
- **Adaptation** Another approach that is similar to the state estimation described above is to learn adaptation between various random instances of simulation or tasks. In other words, an agent could learn to infer the necessary parameters, such as environment dynamics, from the history of state action pairs. One such work uses temporal architectures such as convolutions and attention mechanisms to meta-learn robust policies [64].

[64].

0.4.3 General Reinforcement Learning

Sequential decision with instrumental goals We usually consider sequential decision tasks within the framework of Markov Decision Process (MDP) in which we can train an agent π to perform optimally for a specific reward function. In the general case, however, the environment (MDP) has variables that are not directly observable. These unobserved states can be due to unavailable sensory information. It is also possible to formulate varying physical parameters or outside disturbances as unobserved parameters. In some cases, the agent might see a different environment (MDP) with every new episode (meaning starting from $t = 0$). All these scenarios lead to solving a partially observable MDP (POMDP). One such typical case is the sim-to-real scenarios described in this chapter, where the agent is trained on an environment μ_a, but is deployed in μ_b, for example. Such decision problems are more difficult than standard reinforcement learning problems due to having to also deal with the instrumental goal of inferring missing information about the environment. Not only does the agent need to balance exploration vs. exploitation during training, but now the optimal strategy consists of exploring and exploiting during inference in order to gather the necessary information, which can enable the agent to make optimal actions with respect to the cumulative reward.

Reinforcement Learning formalism. Consider a sequential task that consists of a series of actions and observations $(o_0, a_0)...(o_t, a_t)$ that result from an interaction of agent π with known environment μ. We will use the notation $oa_{<t}$ to denote observation-action history ranging from the start to a timestep t. The action of agent π maps all possible environment histories to actions and can be expressed as:

$$\pi : O \times A^* \to A \tag{4}$$

The environment is conditioned on the history and current action of the agent and can be expressed as.

$$\pi : O \times A^* \times A \to A \tag{5}$$

The goal of the agent is to maximize expected cumulative discounted rewards given a history of past observation-action sequences.

$$R_t = E_{\pi,\mu}\left[\sum_{h=t}^{\infty} \gamma^{h-t} \cdot r_h\right] \tag{6}$$

In Reinforcement Learning, we usually assume a Markov property on the environment, namely that $\mu oa_{<t} = \mu s_t$, where we denote $s \in S$ to be a variable which defines the complete state of the environment. This above-described formalism that we described is a slight modification of the one used throughout most of this thesis. However as mentioned in the introduction of this section, we will consider the more general case where the Markov property does not hold. We define a cumulative episodic knowledge q_t for each timestep and a knowledge update function $\mathbb{I}$ that updates q_t based on the previous observations.

$$q_{t+1} = \mathbb{I}(q_t, o_t, a_t) \tag{7}$$

We can then write the action that optimizes for such an intermediate goal as:

$$a^* = \arg\max_a \mathbb{I}(q_t, o_t, a_t) \ \max \sum_{h=t}^{\infty} \gamma^{t-h} R(\pi(o_h, q_{h-1})) \tag{8}$$

Equation 0.4.3 shows that the optimal action $a*$ is given by the reward of the action, conditioned on the information gain from the previous action.

Non-Markovian Reinforcement Learning It is useful to consider the framework of Non-Markovian Reinforcement Learning as an instance of sim-to-real transfer. We consider only a small part of the environment to be unknown, parametrized by vector q. This can be a set of physical parameters of a system or a latent vector that encodes qualitative properties of a high-dimensional modality such as a camera image. We can then use a Deep learning model as the agent function that is temporally persistent across timesteps, such as an RNN [25]. If trained on the target goal of maximizing rewards, such an agent will be forced to learn to maximize the instrumental goal of inferring information about the vector q so that it is able to make better decisions about the target goal down the line. Several sim-to-real approaches described at the beginning of this chapter use a similar approach. In some approaches, the instrumental goal of maximizing the knowledge of q is performed separately, which makes the training easier but reduces performance. The following is a summary of agents ranging from a static agent to an agent that performs full sequential Bayesian decisions.

- **Statically trained agent** We consider this agent as a baseline for making decisions in an MDP that is not fully observable. Such an agent will have worse than optimal performance due to the lack of information to infer the current state. A specific

example of this is learning an agent in simulation and then deploying it on a real model with no changes.

– **Randomized agent** Such an agent is trained on a set of various randomized instances of a specific environment in simulation. This results in an agent that performs well on average in these environments. It is, however, likely that in specific environments, the agent performance can be significantly worse or completely fail in comparison to the optimal agent. If the distribution of environments is wide enough, it can also happen that some difficult instances will destabilize the training process for the rest.

– **Separated latent vector estimation** In this case, we have a separate agent that estimates the latent parameters of the environment that make it observable and feeds it to the acting agent. This concept is quite standard in optimal control and is called an observer. In deep learning, we have also seen similar works, such as in [57]. The idea is to train a neural network that estimates system parameters from ground truth labels simultaneously with the agent. It is one of the simpler and more powerful sim-to-real approaches that one could implement.

– **Active latent vector estimation as an instrumental goal** In this case, the agent treats the issue of partial observability as an instrumental goal and optimally balances the exploration/exploitation during both training and inference in order to perform the actions that will simultaneously lead to high rewards and provide information about the environment which will lead to better actions downstream. In a sense, this agent can learn to perform optimal decisions, perhaps to a Bayesian extent, based on a prior environment distribution and model. Using deep learning, such an agent could be implemented using an RNN and trained using RL end-to-end by optimizing the target reward function, such as in [63]. The idea is that the instrumental goal is to be inferred as part of the task.

– **AIXI agent** The problem of sequential decision in an unknown environment $\mu \in M$, similar to the one posed in equations 0.4.3 and 0.4.3 have been proposed in Marcus Hutter's AIXI agent formulation [65]. AIXI is a general Reinforcement Learning agent that optimizes actions by considering all possible future outcomes, weighted by the probability of being in each environment $\mu \in M$. At the same time, during each timestep, the agent updates its belief that it is currently in a given environment. The AIXI formulation uses a universal prior $2^{-\mathbf{L}(\mu)}$ to weight the individual environments μ where μ is the length of the program used to compute μ. Using Occam's Razor, it prefers shorter programs. The decision rule, described in [66] can be written as follows:

$$a_k = \arg\max_{a_k} \sum_{o_k r_k} \cdots \max_{a_m} \sum_{o_m r_m} [r_k + \cdots + r_m] \sum_{q:U(\mu,a_1..a_m)=o_1 r_1..o_m r_m} 2^{-l(\mu)} \quad (9)$$

U is a universal Turing machine used to compute the environment. This approach is Bayesian-optimal as it considers a prior over all possible environments, which is updated at every step and selects actions based on current beliefs. Such a model, however, is incomputable, and although several computable approximations exist, it's not something that can be used on a real problem.

Part II

Hexapod locomotion

Chapter 1

Hexapod locomotion by expert policy multiplexing

In this part we study the possibilities of adaptive hexapod locomotion through switching between control policies that have high performance on various complex types of terrain. This is a two-level neural network architecture that infers various terrain types from temporal sequences of interoceptive data. We assume that the target complex terrain can be modeled by N discrete terrain distributions that capture individual difficulties of the target terrain. Expert policies (physical locomotion controllers) modeled by Artificial Neural Networks are trained independently in these individual scenarios using Deep Reinforcement Learning. These policies are then autonomously multiplexed during inference using a Recurrent Neural Network terrain classifier conditioned on the state history, giving an adaptive gait appropriate for the current terrain. Our work demonstrated several tests to assess policy robustness by changing various parameters, such as contact, friction and actuator properties. We also show experiments of how such a control policy can be controlled for practical locomotion by use of geometrical target goal-based positional control. We also show that we can select various gait criteria during deployment, giving us a complete solution for blind Hexapod locomotion in a practical setting. The Hexapod platform and all our experiments are modeled in the MuJoCo [67] physics simulator. This work is largely taken from our publication titled "Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification", cited as [28], published in the Journal of Intelligent & Robotic Systems. We modified some parts of the paper and added a few additional sections and figures that helped to expand on some of the explained concepts. An additional contribution that we have made in this thesis is a pipeline for procedural synthesis of heightmap terrains that are statistically similar to ones provided by a series of photographs by the user. This pipeline makes use of structure from motion, as well as patch-based synthesis, described in section 1.6.4

1.1 Introduction

The majority of control tasks involving ground-based mobile robots require some capability of traversing the terrain in which the robot is being deployed. Wheel-based robots are simple to control but their capabilities are limited by the size of their wheels relative to the terrain. Other alternatives, such as tracked systems [68], legged flipper systems [69] and tensegrity structures [32] each have their advantage in various rough terrains.

Robotic morphologies that are tailored to a specific terrain/purpose can outperform other designs in that domain. It can, however, be expensive and time-consuming to develop a custom platform for each task. A legged robotic platform offers universality in its design by being adaptable to many types of terrain. Legged robots provide a maximal amount of flexibility and do not require a continuous path to navigate their environment. This enables the traversal of the most complicated type of terrains, which feature sudden changes in height, holes, bumps, slants, etc. These robots are unfortunately also the most difficult to control, as each leg usually consists of several independent joints leading to a high dimensional continuous control problem. As an example, a Hexapod robot has six legs, each consisting of 3 links having their own joint, which leads to 18 degrees of freedom.

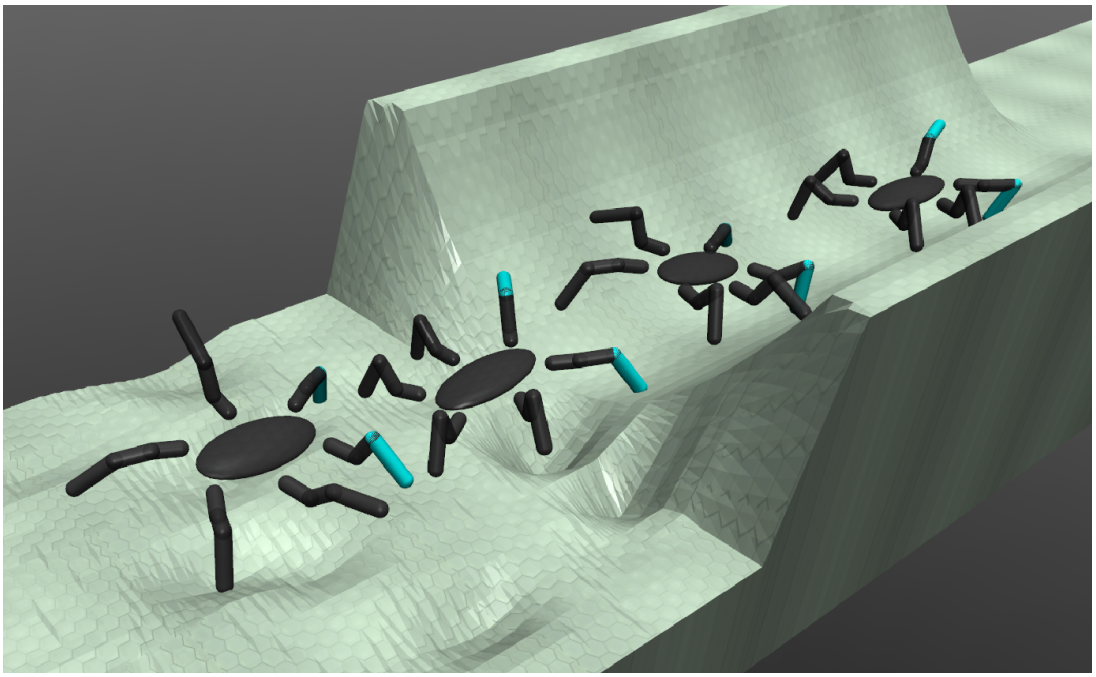

Figure 1.1: Hexapod locomotion in simulation. The front legs are distinguished by the blue color. Supplementary video material available at `https://youtu.be/OXAJ0jmdCZ0`, Github: `https://github.com/silverjoda/nexabots`

Locomotion on uneven terrain requires the ability to perceive the environment. This can be done using exteroceptive sensors such as LIDAR [70] or depth camera [71]. With such an approach, a local elevation map is usually reconstructed, and planning or other optimization

techniques are used to calculate foot trajectories and placement positions. These are then fed into low-level tracking controllers and executed by the robot actuators. This approach can fail if there is any inaccuracy with the local perception map, such as a puddle which causes issues for depth sensors.

A more minimalistic approach is to use only interoceptive sensing, that is by sensing contact between the leg and the ground in addition to onboard IMU and joint angle data. This approach is not as powerful as one that uses rich sensory data about the environment since there is less information available about the surrounding terrain. It does, however, have the obvious advantage of being simple and not requiring complex, heavy, and expensive sensors that have several known failure modes. The lack of high dimensional sensors also means a much lower computational load on the system, allowing the use of cheaper and lighter hardware, such as an embedded microcontroller for both sensing and locomotion. Another major advantage is that such a policy learned in simulation is less prone to overfitting and should be able to generalize to the real platform without issue.

In our work, we use the latter approach. This means that planning is out of the question due to a lack of knowledge of the surrounding environment. To perform successful locomotion, the robot has to feel the terrain with its feet, similar to how a human would navigate blindfolded between rooms with outstretched hands. We use a Deep Learning approach to help us learn a gait for the Hexapod platform, which adapts to the environment in real time, using binary contact sensors at the feet. Our work is focused on data-driven methods for locomotion because they are powerful and scalable. The user only has to be able to model sufficiently rich terrain in simulation. There is little to no domain knowledge required on how the robot should move. The user can specify certain criteria of the gait in terms of a reward function, adding as little or as much domain knowledge as he or she wants which in part contributes to the flexibility of this approach.

Our paper elaborates and contributes to the following topics:

- Insight into the blind locomotion problem for a hexapod platform.
- New approach to robust adaptable locomotion for legged robots by training on rich terrain using a two-level locomotion architecture which autonomously multiplexes expert control sub-policies that are appropriate for the each type of terrain.

1.2 The hexapod platform

The most common legged robot configurations are bipedal, quadrupedal, and hexapodal variants. Bipedal robots require the fewest actuators. They are, however, the least stable and need

a fast control loop to keep balance. Even with the right balance, they are sensitive to slippage and can fall over in unpredictable terrain. Some examples feature [72]. Quadruped platforms are probably the most popular and are used by several groups to research locomotion and other tasks [73] [74] [72]. Hexapod configurations are statically stable, meaning that they can stay upright with little or no active control. This advantage can be attributed to the redundant number of available legs. This feature also offers the highest potential terrain traversability out of the platforms mentioned above, as there are more points of contact with the ground. Stability and traversability clearly increase with the number of legs on the robot, but so does the complexity of construction and control. The hexapod configuration is a good compromise between complexity and utility.

The hexapod platform consists of six legs, each having three links called the Coxa, Femur, and Tibia, respectively. The Coxa is connected to the Thorax (body) with a joint, as are the links between themselves, resulting in a total of 18 motorized joints. The joints are usually actuated using readily available or purpose-built servos. One of the design choices for such a platform is whether to use position or direct torque control. Positional servos are more readily available and have their own inner feedback control loop, which takes care of the low-level control. Joint actuators also require current joint angle information and preferably the instantaneous current or torque that is being applied to it. In-built current limiting and overheating protection is preferred in such a platform as stall conditions are likely to happen, especially during experimentation or in the case that the control policy fails and behaves undesirably. Regular geared servos exhibit backlash, but this is not an issue in such a platform as we do not require precise control and placement of the legs. The platform used in our work is modeled to match the dimensions of the hexapod platform by Trossen Robotics [75] in the MuJoCo simulator [67].

1.3 Related work

There has been a significant amount of work which addresses the task of locomotion for hexapods robots. Some of the more popular methods for legged robot locomotion involve the use of Central Pattern Generators (CPGs) [76], [77]. This is a bio-inspired method that attempts to mimic certain aspects of the sensory-motor nervous system of animals to achieve a successful gait. The advantage of this method is that it induces a strong prior on the action space of the agent, meaning significantly fewer parameters and a more natural gait. The disadvantage of this method is the domain knowledge required to craft the oscillator network and interconnections. It is also unclear how to integrate high-dimensional sensory feedback into these networks. CPG methods for legged robot locomotion usually include the use of inverse kinematics to control the servos and a method of footstrike detection using an IMU, servo position feedback [78] or a button sensor at the tip of the leg [79]. Most of these approaches focus on the bio-inspirational aspect

of locomotion and on gait stability analysis. Our approach, however, directly optimizes for locomotion performance. To our knowledge, there is no CPG based approach that is robust and can handle a variety of challenging terrains.

A contrasting approach is to use neural networks together with data-driven methods to learn locomotion. One such method is Deep Reinforcement Learning (DRL) that supports higher dimension state and action spaces using neural networks as function approximators. One of the simplest and highest performing DRL algorithms is the Proximal policy optimization (PPO) [38] algorithm. This algorithm efficiently estimates the policy gradient [11] on a batch of states and performs multiple updates while making sure that the updated policy does not deviate too much from the current one. We use the PPO algorithm in all our experiments.

Learning locomotion on various terrains can be thought of as learning several skills and choosing the right skill at the right time, which is essentially what our approach does. There is a related work by [80] where the authors teach the agent locomotion end-to-end on flat terrain using a mixture of sub-policies. However, the quality of results demonstrated using this algorithm are unsatisfactory, even for much simpler tasks. Furthermore, this algorithm requires an actor-critic architecture that is unstable for training locomotion tasks for many-legged robots, especially in a recurrent setting, which would be required in our case.

One of the drawbacks of using a standard feed-forward neural network architecture for locomotion is that it is ignorant of the morphology of the robot. It is preferable to use a structure which imposes a prior on the problem, similarly to convolutional neural networks are used in computer vision. There has been some work on learning locomotion using structured neural network policies such as [19], where the authors propose a policy class for legged robots. Although the authors demonstrate good performance on repetitively-linked morphologies such as centipedes, all the experiments were done on flat terrain.

While dealing with tasks that require memory, recurrent neural networks (RNN) are an appropriate choice of model. We use Long short-term memory (LSTM) networks [25] that are models that have achieved very high performance on various time-series tasks [81], [82], [83]. When using the policy gradient algorithm on RNNs, it is modified to update on batches of episodes instead of batches of states.

1.3.1 Conclusion

Legged robot locomotion has been thoroughly studied but most results fall into the following categories: a)Bio-inspired methods and gait analysis, b)optimization methods that require complex sensors and pipelines and c)experimental data driven methods which demonstrate improvements in neural network architectures or training algorithms. To our knowledge, there is no method which approaches the problem by modeling various complex terrains and training a locomotion policy with minimal sensory input that could be used with high-level control, such

as in our work.

1.4 The blind locomotion problem

Locomotion for legged robots is a high-dimensional combinatorial problem. One way how to solve it is to constrain the action to a periodic movement called a gait that is suitable to a given terrain. A suitable gait is one that allows it to traverse the terrain in a manner that is not damaging to the physical platform and while respecting specific criteria such as energy efficiency or smoothness. The simplest gait example is on flat terrain. An efficient gait would consist of symmetric 3-4 legged (meaning that these legs are in contact with the ground) movement without moving lifting its legs unnecessarily high, minimizing energy consumption and wear and tear on the system. If we, however, consider a gait with tiles of various heights, then such a gait would cause the legs to collide with the various height tiles. What is instead necessary, is a stable gait where the legs are always lifted very high up, with slightly larger strides. The challenge is then to recognize which type of gait is required in a given situation, and to adapt to it autonomously. This could again be imagined walking blindfolded room. If told in advance that the next several meters are flat ground, then we can simply walk in a normal fashion. If, however, we are told that somewhere along the floor there will be a small step then our gait will be careful, slower and maybe even asymmetric so that we are more balanced in the case that a leg gets hooked at the step. At this point, it is necessary to clarify what an adaptive gait means. In literature on gait analysis, an adaptive gait usually means a gait that works on terrain other than a flat plane. In other words, it is able to sense the kinks in the terrain and slightly adjusts the gait so that it works. We will refer to this as micro-adaptation. In our work, by *adaptive*, we mean a locomotion policy that can change the entire gait pattern given the situation, as in the example given above on blindfolded walking in a room. We refer to this as macro-adaptation. Our locomotion policy does both.

1.4.1 Overview

We assume that the target environment (where we plan to deploy our agent) is somewhat structured and can often be geometrically categorized. One example is almost any man-made environment that consists of hard ground, slippery wet ground, stairs, tunnels, pipes, ground obstacles, etc. We work with the assumption that if the agent is capable of navigating these discrete terrain distributions, then it can navigate the target terrain distribution. Note that we refer to the terrains as distributions because we assume that an instance of a specific terrain is sampled from some distribution that generates that type of terrain.

At this point, it would be beneficial to clarify as to why the concept of a gait on distinct

terrain distributions is required and why we don't learn a locomotion policy which simply solves the entire complex problem. As mentioned previously, treating this problem combinatorially comes with an exponential compute cost. A gait is essentially a prior or a limitation of the state space that allows us to solve such a difficult problem. What we can hope to do is to modify or perturb the gait slightly to adapt to the unevenness of the terrain. This is why we believe that it is better to have distinct gaits that are suited to their specific terrain types and to work from there instead of having a universal gait that has to perform complex adaptations to all the various terrain types that we are planning to deploy on.

1.4.2 Problem definition

We consider an agent that interacts with the environment in discrete timesteps. At each timestep, the state of the agent and the environment can be jointly described by a vector $s \in S$ called the state vector. By definition, this vector should contain all the necessary information required to choose an optimal action for the current time step. In most cases, the environment is only partially observable, and therefore we work with incomplete observation vectors $o_t \in O$. The agent performs an action $a_t \in A$, after which the environment advances by a single time step before returning the new observation vector $o_t \in O$. Each state-action transition is evaluated by reward function $r : S \times A \to R$. At the beginning of each episode, the agent finds itself in a random initial state $s_0 \sim \rho(s)$. The above element, along with an environment state-transition distribution $T(s', s)$ and reward discount factor γ are described by a Markov Decision Process (MDP). If the state s is not fully observable, then we refer to the process as a Partially Observable MDP, or POMDP.

The agent is denoted by a parameterized function $\pi_\theta : S \to A$ also called a policy function. This is essentially a mapping from state s_t to action a_t at every time step. The parameter θ denotes the weights of the neural network which we are optimizing.

We assume that most of the terrains can be modeled by and composed of several distinct terrain distributions $t = 1 \ldots N$ such as stairs, slopes, pipes, slippery patches, etc. The task is to train the policy, which performs locomotion successfully on all these types of terrain. Desirable properties of quality gait, such as smoothness or speed, are determined by the state-action reward function r.

We define a trajectory $\tau = \{(s_1, a_1), \ldots (s_h, a_h)\}$ to be the sequence of consecutive states and actions with a finite horizon h. The sum of rewards collected along a trajectory τ is the cumulative reward

$$R_\tau = \sum_{(s,a)\in\tau} r(s,a)$$

Terrain type t and the policy π_θ uniquely determine trajectories probability distribution

$p(\tau|t,\pi_\theta)$. The goal is to find a policy that performs well on all types of terrain:

$$\pi_{\theta^*} = \arg\max_{\pi_\theta} \mathbb{E}_{\tau\sim p(\tau|t,\pi_\theta)}\{R_\tau\}$$

The fact that the agent does not know what type of terrain it is on makes the task partially observable that can be formally described by a POMDP. A purely reactive policy is insufficient to solve this task as it requires memory to know what type of terrain it is currently on. Therefore π_θ has to be recurrent, meaning that it keeps an internal representation of the current trajectory at a given time step. Due to the high-dimensional underlying state-action space and the variety of terrain, direct optimization of the policy parameters θ using a recurrent neural network is difficult. Experiments show that this approach does not learn distinct gaits, but rather a single gait which performs sub-optimally on the individual terrains. This can be thought of as a local optimum as far as a locomotion gait is concerned. Therefore we exploit the compositionality prior of a given terrain and propose a novel two-level structure of the policy as shown in Figure 1.2. Our proposed compound architecture essentially separates the policy into two components. The first part consists of the policies that perform well on the individual terrains that we call the expert policies. The second is a multiplexer policy that classifies the current terrain and selects the appropriate expert.

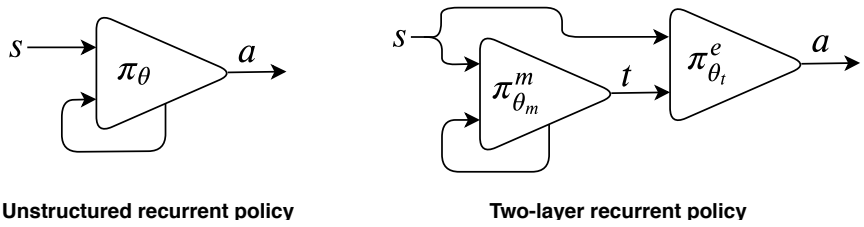

Figure 1.2: Policy structure: Left general recurrent policy π_θ, Right proposed two-level structure for terrain locomotion consisting of a recurrent multiplexer policy $\pi^m_{\theta_m}$ and a set of reactive expert policies $\pi^e_{\theta_t}$.

We assume that for a given terrain type t, the task is fully observable and there exists a near-optimal *expert* policy $\pi^e_{\theta_t}(s) : S \times T \to A$ which is purely reactive. In our experiments, we found that recurrent policies slightly outperform their reactive counterparts because even the discrete terrains exhibit a certain degree of partial observability due to blindness. Nevertheless, reactive policies are used for simplicity's sake and for proof of concept. Consequently, the proposed structure of the policy is a concatenation $\pi_\theta = \pi^e_{\theta_t} \circ \pi^m_{\theta_m}$ of a *multiplexer* recurrent policy $\pi^m_{\theta_m}$, which switches between terrain-expert policies $\pi^e_{\theta_t}$.

1.5 Training pipeline

The proposed learning scheme is summarized in the following Algorithm:

1. Obtain terrain-expert policies $\pi^e_{\theta_t}$ for all terrain types $t \in T$ by learning parameter vectors θ_t using reinforcement learning for each terrain type t independently.

$$\theta^*_t = \arg\max_{\theta} \mathbb{E}_{\tau \sim p(\tau|t,\pi^e_{\theta_t})}\{R_\tau\} \quad \forall_{t \in T}$$

2. Use the learned expert policies to gather rollouts on compound terrains. Learn a recurrent multiplexer policy to classify current terrain from the history of observations on the gathered dataset by minimizing classification cross-entropy between the predictions and terrain labels.

$$\theta^*_m = \arg\min_{\theta} \mathbb{E}_{\tau \sim p(\tau|t,\pi_\theta)}\{\sum_{t \in \tau} L(\pi^m_\theta(o_t, ..o_0), l_t)\}$$

   where l_t is the current terrain label at time t. Terrain types can change multiple times throughout an episode.

3. Optionally iterate.

In the following sections, we will take a look at the entire pipeline in detail on how to train a blind locomotion policy that adapts the gait to several terrains.

1.5.1 Environment creation

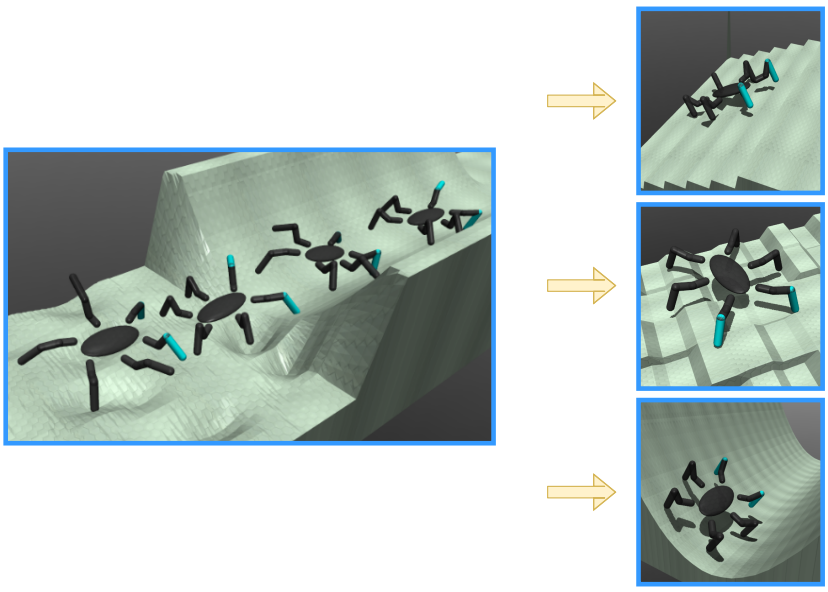

Figure 1.3: A scenario of a complex structured terrain which is decomposed into several distinct terrains.

The first step is to create an environment in which we want to train the agent. The environment consists of the physical world (terrain), as well as how the agent interacts with it and how

it is rewarded. These three factors shape the agent locomotion policy. As mentioned previously, we are assuming that the robot will be deployed in some complex terrain that can be modeled as N distinct terrain distributions. This means that we will have to create N distinct environments.

The physical terrain consists of a heightmap which can be modeled manually in some cases and procedurally otherwise. In our work we generate a new terrain instance every few episodes so that the agent is able to interact with various different difficulties and doesn't overfit to a single instance, resulting in a robust policy. It is also important that the terrain is appropriately difficult. Otherwise, no interesting or complex behavior will emerge as a result. If it's too difficult, then the agent will be unable to solve it. Depending on the terrain, there are several approaches that we use to tune the difficulty. For specific cases such as steps and stairs, it is appropriate to use realistic dimensions that a robot is expected to encounter in the field. For other regular patterns such as tiles and slanted surfaces, the dimensions are evaluated experimentally by incrementally increasing the difficulty and observing the limits of the robot. Terrains with abnormal geometries such as narrow pipes that require specific gaits are more or less done similarly as the regular terrains, with the robot limits in mind. Outdoor surfaces modeled by noise fields such as Perlin noise [84] are tuned by hand by generating many examples and modifying the distribution and scaling parameters of the algorithm so that interesting examples are generated every time. Simple checks, such as the absolute difference between the minimal and maximal height, can be used to determine whether the specific sample is too easy and can be discarded. Occasional inappropriate samples can be considered as noise and do not affect the learning process significantly.

Compound terrains: In our experiments, we use compound terrains to demonstrate gait adaptation. These consist of samples from various terrain distributions which are joined together to form a single instance. The tricky part is to make sure that the stitching is done seamlessly, meaning that there are no step-differences in height between terrains. A straightforward way is to pass a height averaging sliding window over the joining region. One downside is an introduced *valley* artifact that arises in this region due to the averaging. Some sections, such as stairs, can randomly end at a certain height meaning that subsequent sections have to be height-adjusted appropriately.

Agent interaction: The interaction is defined by the input and output spaces of the agent. The input has to be informative enough for the agent to perform the task. If the policy is reactive (memoryless), then the translational torso velocity and joint angular velocities have to be added as well. Otherwise, the task will be ill-defined. Formally, the observation of the agent at time t consists of $o_t = \{j_t^1, .., j_t^n, \dot{j}_t^1, .., \dot{j}_t^n, c_t^1, .., c_t^m, q_t^w, q_t^x, q_t^y, q_t^z\}$ where j_t^i is the i_{th} joint angle, $\dot{j}_t^n$ is the i_{th} joint velocity, c_t^k is a binary value representing the k_{th} contact of leg k with the ground and

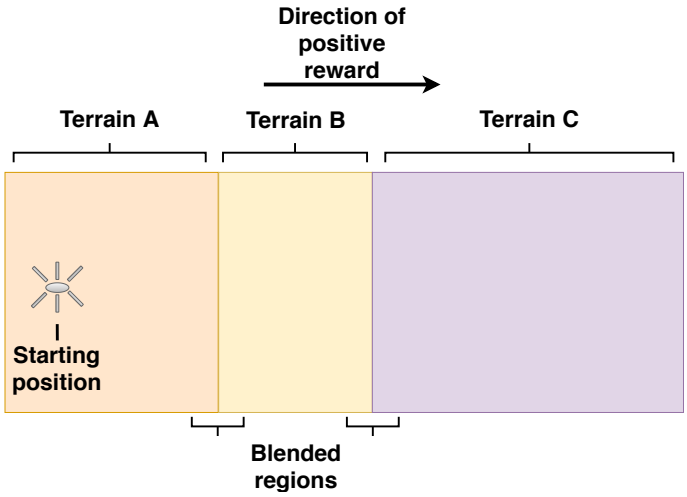

Figure 1.4: Top-down illustration of compound terrain consisting of 3 joined instances

q_t^j are individual parts of the quaternion of rotation of the torso. It is important to set the range of the joints reasonably. The larger the joint space, the more difficult it will be to learn a gait, and the more likely it will lead to a gait that is asymmetric or neglecting one or more joints.

1.5.2 Training expert policies

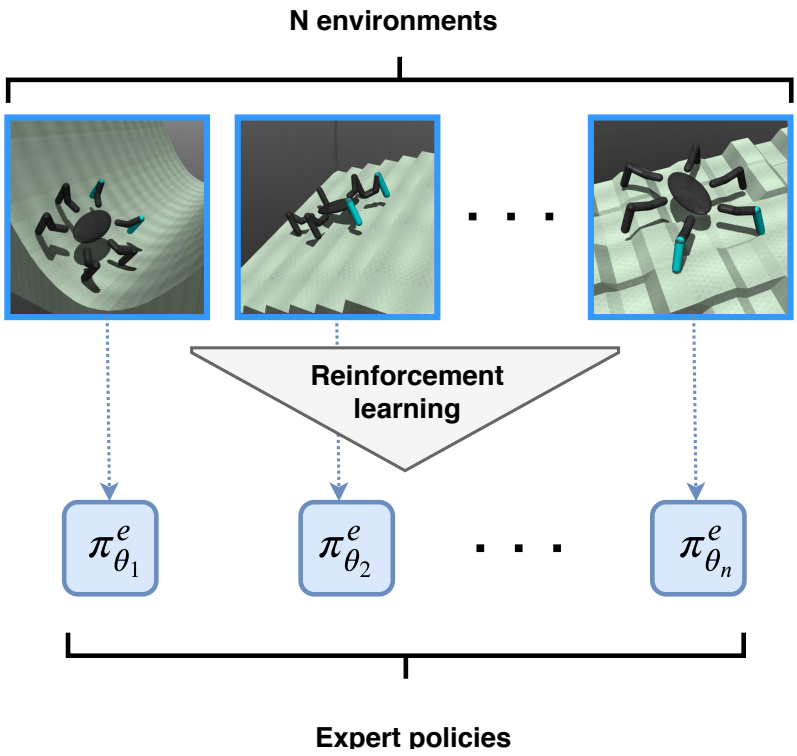

Figure 1.5: Use RL to train expert policy for each environment.

We denote expert policies trained on the i_{th} environment as π_i^e. Policies are modeled as a multilayer feed-forward neural network, also called a multilayer perceptron (MLP). The net-

works typically have two hidden layers, with 64-96 units in each layer. The MLP is parameterized by a weight vector θ. The MLP policies are essentially a learnable mapping from observation to action that solves a particular Markov Decision Process (MDP) in our case. Even though we divided the complex target terrain into simpler terrains, it does not mean that these individual terrains are fully observable, also though we assume them to be. This means that reactive memoryless policies learned will always be sub-optimal. The reason why we use reactive policies as experts and not recurrent ones is that they are much easier and straightforward to work with when using the multiplexer policy.

The policies can be trained by ascending the following unbiased policy gradient, similar to what is derived in [11]. The term $A(a_t, o_t)$ can be any estimate of the advantage, or *goodness* of action a_t.

$$\nabla_\theta J(\theta) = \sum_{o_t, a_t \in \tau} \nabla_\theta log \pi_\theta(a_t|o_t) \cdot A(a_t, o_t) \tag{1.1}$$

A single iteration of the algorithm can be summarized in a few simple steps:

1. Gather a batch of trajectory rollouts τ using the current stochastic policy π_θ
2. In each trajectory, for every action taken, calculate the advantage A. We use an unbiased Monte Carlo estimate of the return discounted by γ

$$A(a_t, o_t) = \sum_{t=k}^{h} \gamma^{t-k} r(a_t, o_t) \tag{1.2}$$

3. Calculate policy gradient $\nabla_\theta J(\theta)$
4. Update policy π_θ by ascending the policy gradient

We use a slightly modified update rule from the PPO algorithm for more efficient updates. Inputs and outputs to the policy are normalized to the interval $[-1, 1]$ where applicable. This enables faster training and better performance. Each expert on a single CPU thread can take anywhere from 4-8 hours to train, depending on the terrain difficulty and desired performance. The environment is generated randomly in each episode from the environment distribution, and an episode of 200 steps is performed. Batches of 20-40 episodes are used to obtain the policy gradient to update the network. In our experiments, we set the same reward function for all environments so that we can compare them, but they can be tailored to each environment separately. The agent is rewarded in the following way:

$$r_t = \lambda_1 \cdot r_v \lambda_2 \cdot r_\theta - \lambda_3 \cdot \phi_t^2 - \lambda_4 \cdot \psi_t^2 - \lambda_5 \cdot \dot{z}_t^2 - \lambda_6 \cdot \tau_t^2 \tag{1.3}$$

where

$$r_v = \frac{1}{abs(\dot{x}_t - v_{tar}) + 1} - \frac{1}{v_{tar} + 1} \tag{1.4}$$

$$r_c = |\theta_{t-1}| - |\theta_t| \tag{1.5}$$

The total reward r_t consists of rewards r_v, r_c and various penalties. The velocity reward r_v is the reward that motivates the agent to move forward and is tuned so that the agent receives a reward for a positive velocity up to a certain point, peaking at the target velocity. It is important to set the maximal velocity reward to a specific target. If we simply optimized for maximal reached distance, then we would get a very fast gait and erratic jumping behavior, which is not something we desire as it can make training in simulation more difficult and could destroy the real platform. The term r_θ rewards the agent for correcting heading errors and works significantly better than penalizing heading deviations. The issue is that if slippage occurs or a robot leg gets caught on an obstacle, the robot can suddenly find itself facing the wrong way. If we penalized heading deviations, then the agent would start to accrue large amounts of penalty for something it didn't do. This seems like a trivial issue, but for a legged robot, changing direction is a task in and of itself. This is why, at the beginning of each episode, the robot is placed with a small random yaw rotation of roughly 1 radian. Formulating the r_c, as shown above, leads to a policy that is motivated to correct the heading first and then walk towards the goal direction. Lastly, various penalties can be added which modify the gait. Some of the simplest ones that we added are torso angle and acceleration penalizations in the form of L_2 loss. Angles ϕ, ψ, and velocity $\dot{z}$ represent the current pitch, roll, and z-axis movement of the hexapod torso. These are being included to produce a gait which keeps the torso level and smooth. It can be useful if we have a camera or a scanning LIDAR sensor on board. We can also optimize for energy consumption. This penalty is given in the term τ and denotes the sum of actuator torques at time t.

1.5.3 Training a terrain classifier

At each time-step, the RNN receives the joint angles and binary leg contact information from the agent and predicts a softmax probability distribution over the terrain classes. The maximum over the softmax is then used as the label to choose the expert policy. We use an LSTM network with 3 stacked memory layers. The supervised learning task is solved by descending the following gradient in an iterative fashion:

$$\nabla_\theta \frac{1}{|D|} \sum_{(h_t, y_t) \in D} L(\pi_{obs}(h_t), l_t) \tag{1.6}$$

Training is done on batches of episodes. It is essentially a sequence to sequence task where

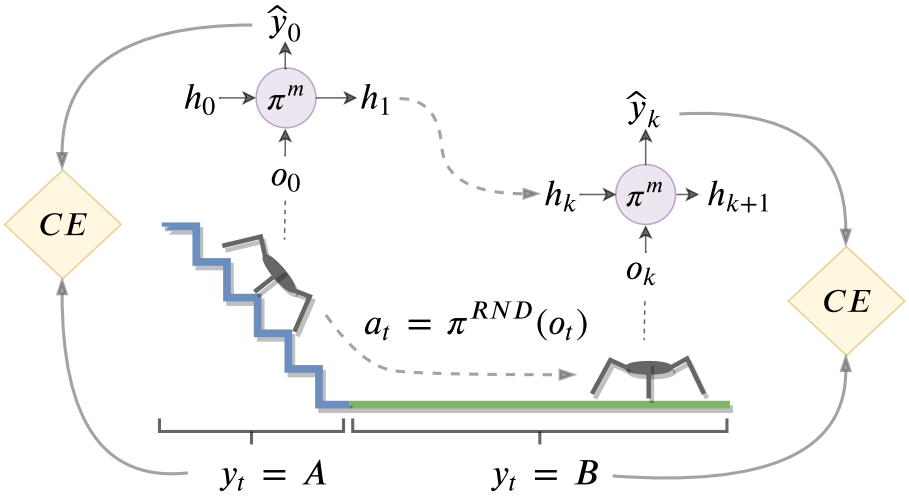

Figure 1.6: Training of a recurrent terrain type classifier that is then used as a multiplexer. Observations o_t are seen at every time-step by the classifier, and a label $\hat{y}$ is predicted. The predictions are then compared against ground truth labels using a cross entropy loss, marked as *CE* in the illustration

we map current observation histories h_t to environment labels that the agent is currently on. The training dataset is gathered by generating random combinations of terrain instances with various lengths and joining them into a single terrain. The transition points are stored along with the instances so that the correct terrain labels can be generated during training time. The hexapod then navigates the combined terrain using randomly picked expert policies at random times, and the observations and terrain labels are recorded at each time step. The policies have to be randomized and cannot be chosen according to the current terrain; otherwise, the classifier will fit the policy and not to the terrain. Specifically, the episode is started with a randomly chosen expert policy, and at every step, there is a probability p of choosing another expert policy. Another way would be to iteratively learn the classifier and sample the policy choices from it that would, in theory, lead to a trajectory distribution which would be the same in training and test time. However, we found that the training procedure is unstable.

The terrain transitions are probably the most troublesome part of the whole process. The terrain labels are updated a heuristic amount of steps after the agent crosses onto new terrain. This makes sense because the agent cannot immediately know that it is on a different terrain without first having interacted with it.

1.6 Experiments

In this section, we will demonstrate several different experiments that show the strengths and weaknesses of our approach. These will consist of an evaluation of the expert policies, and

our proposed two-level approach with a trained multiplexer selecting the appropriate policy. In addition, we provide experiments to show how changes in robot and environment parameters affect performance to get an understanding of the robustness of the policies. Further experiments show how we can affect gait parameters such as walking height and speed using auxiliary inputs to the network. We conclude the experiment section by demonstrating that our policy can be controlled in a waypoint fashion that can be used in conjunction with a planner or human operator.

1.6.1 Experiment details

The platform used in experiments is modeled to match the dimensions of the Mark II hexapod platform by Trossen Robotics in the Mujoco simulator [67]. The model dimensions and masses are modeled by taking measurements from the real platform. Contact dynamics include tangential, torsional and rolling friction. Since we have a legged robot, we are mostly interested in tangential friction which we set to a suggested default. Material surfaces are not simulated as this would be excessively complex and likely not a contributing factor for hexapod locomotion. The robot is controlled by servos with internal feedback loops. The servos have an armature, dampening and a loop gain parameter. These are set in the simulation so that responses to step joint angle changes respond similar to the real platform. Simulation step size is set to the suggested $0.02s$ per step. The simulation gravity vector is set to $9.81ms^{-2}$. Parameter details can be found in the provided GitHub repository.

1.6.2 Evaluation of expert policies

We picked five terrain types with varying degrees of contrast and train reactive policies that perform well on those specific distributions. These policies are then cross-compared to get an idea of how a specific policy might perform on a terrain that is similar or very different from the one that it was trained on. The evaluation consist of episodes of 300 steps over 100 randomly generated instances for each terrain. This is done for each policy.
We use two metrics for comparison. The first is the mean achieved reward of the environment given by the terrain and reward function, as described in chapter 1.5.2. We can refer to this as the gait quality metric, as it includes penalization for unnecessary torso movement and other terms such as energy consumption. The second is the mean success rate of reaching a specific distance. Tables 1.1 and 1.2 shows the results. The average rewards are unit-normalized to the value achieved by the native policy. This is done because various terrains have various maximal achievable average rewards for a specific reward function.

The results give insight as to how policies perform on environments that are similar and different from the ones that they were trained on. We can see that most policies perform well on

Table 1.1: Normalized gait quality performance. Rows are terrain types. Columns are the expert policies evaluated on given terrain type.

	π_{Flat}	π_{Tiles}	π_{Slants}	π_{Stairs}	π_{Pipe}	π_{Perlin}
Flat	1.00	0.98	0.59	0.68	0.37	0.94
Tiles	0.32	1.00	0.47	0.45	0.06	0.71
Slants	0.23	1.61	1.00	0.96	0.16	1.95
Stairs	0.23	0.66	-0.04	1.00	0.07	0.52
Pipe	-0.43	-0.09	-0.33	0.16	1.00	0.05
Perlin	-0.21	0.88	0.38	0.15	-0.05	1.00

Table 1.2: Mean achieved distance in a fixed amount of timesteps. Rows are terrain types. Columns are the expert policies evaluated on given terrain type.

	π_{Flat}	π_{Tiles}	π_{Slants}	π_{Stairs}	π_{Pipe}	π_{Perlin}
Flat	1.49	1.43	1.31	0.97	0.97	1.42
Tiles	0.62	1.20	0.92	0.59	0.46	1.19
Slants	0.49	1.20	0.96	0.65	0.40	1.31
Stairs	0.30	1.08	0.63	0.73	0.56	1.00
Pipe	0.32	0.76	0.67	0.60	1.61	0.62
Perlin	0.47	1.08	0.89	0.59	0.29	1.17

the flat environment as it admits almost any gait. The difference is only in how efficient that gait is and its quality. Both tables show that the flat policy fails on almost all terrain except on the native one. This is expected as the hexapod doesn't lift its legs up high enough from the ground that causes collisions in terrains with sudden changes in height. We see that the *Tiles* and *Perlin* policies are somewhat interchangeable and perform similarly. This shows that a policy trained on sufficiently rich terrain performs well on a variety of features. The difficult part comes in terrains where a different gait is required, such as the *Pipe* and *Stairs* terrains. Here most policies perform poorly or fail completely. One such example can also be seen in the supplementary video and shows why we need to change the gait completely in certain cases. There is, however, an anomaly seen in the performance of the *Slants* policy that is outperformed in its own native environment by several other policies. This is most likely due to a poorly designed environment during training. In this case, particularly, the slants were physically too high for our hexapod morphology. Since the hexapod is unable to make progress, the reward feedback is uninformative, and the learning process suffers, leading to a poor result. This was fixed subsequently but we decided to leave the result as it was to demonstrate the failure mode. The achieved distance table shows more or less the same results.

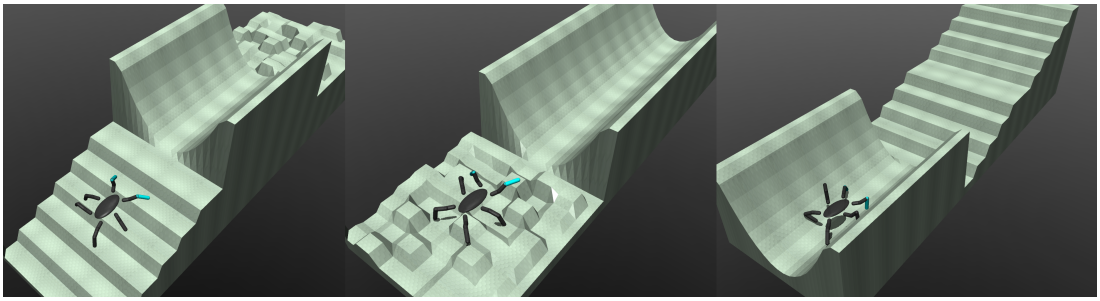

Figure 1.7: Examples of generated compound environments from 3 terrain types.

Table 1.3: Easy compound environment

	Oracle	Proposed two-level policy	End to end RNN
Normalized gait quality	1.0	0.79	0.31
Average distance reached	2.03	1.79	1.64

Comparison of Expert-Multiplexer architecture with plain RNN

In this subsection, we evaluate the performance of our proposed two-level architecture that autonomously multiplexes expert policies on a compound terrain. This is then compared with a ground truth multiplexer and an RNN based policy that was trained on the compound environment end-to-end. All policies are trained with the same reward function as described in the 1.5.2 section. We pick two sets of terrains to evaluate our experiments on. The first consists of the terrains *tiles, slants, flat*, which is a relatively simple combination as far as required gait variety is concerned. The second consists of the set *tiles, stairs, pipe* that is significantly more difficult and requires dedicated gaits to attain good performance. When creating a compound terrain instance, the terrains are sampled from the set with various lengths and with replacement so that any combination is possible. Figure 1.7 shows several examples.

Tables 1.3 and 1.4 show the results of all 3 methods on a test set of 100 sampled instances. We look at the gait quality metric, and average distance traveled within a given amount of steps. We can see that our autonomous multiplexing almost matches in performance the ground-truth policy selection in both easy and difficult sets. We can also see that the RNN policy performs worse in the easy environment and significantly worse in the challenging one.

We also show the attained performance of two multiplexed experts versus an end-to-end trained policy using an RNN. This can be seen in Figure 1.8. This, however, comes with a

Table 1.4: Challenging compound environment

	Oracle	Proposed two-level policy	End to end RNN
Normalized gait quality	1.0	0.80	0.12
Average distance reached	1.86	1.63	0.85

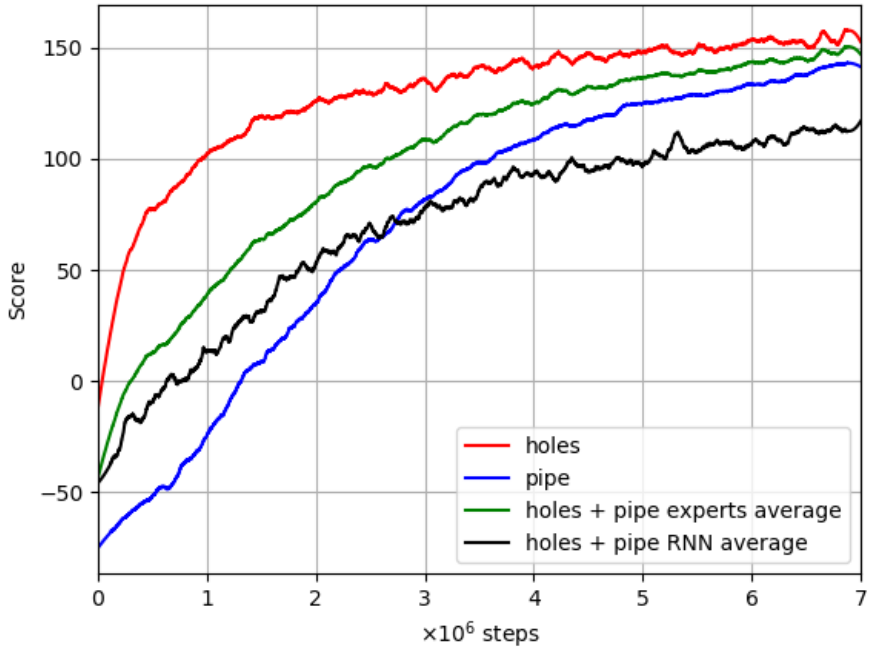

Figure 1.8: Average performance comparison of expert policies versus RNN policy trained end-to-end. Graphs show filtered mean learning curves of 30 training sessions in each scenario. We can see that on terrains on contrasting terrains, the end-to-end policy learns slower and does not attain the performance of individual experts on each terrain.

slight caveat as any classification inaccuracy at the environment transitions can lead to a loss of performance. This is, however, insignificant if we consider the long terrain and sparse transitions.

Generalization and robustness experiments

We perform several experiments in an attempt to quantify the robustness of a learned policy to various changes in the parameters of the environment. On flat terrain, even heavier perturbations don't affect the performance significantly. For a policy trained in an environment with randomly slanted surfaces, we noticed the following behavior:

- Contact friction: We test the whole range of values that give a stable simulation. High values have no effect, whereas low values cause slippage as expected. In extreme cases, this sometimes causes the policy heading to deviate. Small to medium changes in friction don't have a significant effect on the policy.

- Mass: The standard torso mass is defined as 5kg. Perturbations range from 0.1kg to 20kg. These don't have a significant effect on performance as the low-level joint controller is a

closed feedback loop.

- Link lengths: The tibia (the 3rd link of the leg) in our model is roughly 10cm long. If the learned policy is asymmetrical and more dependant on one leg than the rest, then random shortening of the tibia of that leg can cause performance degradation or failure. We also tested lengthening of all the tibia links by an equal amount and found that up to a point (3cm-4cm), the policy performs similarly to the default lengths. When the tibiae are too long, then the policy gets unstable and can overturn.

- Controller parameters: The main parameters of servo actuators and the armature and feedback gain value. The armature is a parameter which is roughly equivalent to the rotational inertia of the servo actuator. Setting this parameter several times lower than the trained value can cause the policy to be too hasty on rough terrain that leads to flailing and policy failure. Higher armature simply causes a sluggish gait. Similar behavior is observed with the gain feedback value of k_p. In addition, a k_p value that is too low is unable to drive the joints to their commanded angles if the weight of the robot is too large.

It is possible to include these perturbations into the environment during training so that the policy learns to perform well over a range of parameters. This is called domain randomization and is often used to learn robust functions in several computer vision and robotics tasks with varying degrees of success [85], [86].

Gait analysis and shaping

As mentioned previously, using a learning approach to obtaining a locomotion policy, the user has the option of adding optional optimization criteria that can shape the locomotion policy. The default locomotion criterium is merely optimizing the gait for a target velocity. The gait that emerges depends on the physical size and parameters of the robot, the joint angle ranges, and the terrain that it is being trained on. For a flat reward function on flat ground, the emergent gait is usually tripodal. If the joint angles range is not excessively large, then the emergent gait is roughly symmetric. A common problem with gaits obtained by uninformative objective functions like the one described above is that the gait is non-natural looking, asymmetric, or in general, can be severely defective. Defects usually manifest themselves as the agent neglecting one or several joints or limbs. These issues happen as a result of the agent getting stuck in a local minimum. A defective or bad gait is a sub-optimal solution. In our experience, this happens more often when trying to optimize a torque driven robot. One way how to solve this issue is

to shape the objective function. This can be in the form of penalizing the agent for not using all the legs uniformly or penalizing excessive joint angles or torques. An effective method of enforcing symmetry is by adding a state-action symmetry penalty such as in [87]. The authors of [87] also use curriculum learning by using varying levels of assistance during training time that guides the agents and prevents bad local minima.

Real time gait manipulation

We can use reward shaping to train gait variations in real-time by providing the desired parameters to the input and modifying the reward function appropriately. The agent learns to understand how to modify its gait conditioned on the input. This is no different than normal reward shaping except that the specific parameters that we want to manipulate are sampled randomly every episode during training and provided by the user during test time. This is also an instance of goal-based learning, where the criteria are given to the input as goals so that they can be manipulated during inference. As an example, we teach the agent to keep the torso at a height and velocity provided by the user.

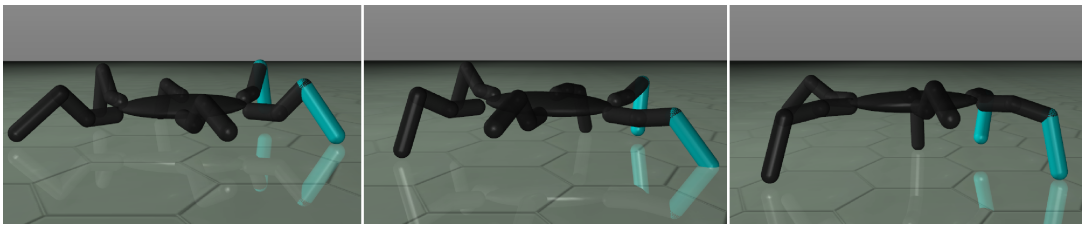

Figure 1.9: Hexapod gait at three different body-height levels.

The agent is trained to be able to walk at heights from 4cm to 11cm and at velocities from $0.2ms^{-1}$ to $0.8ms^{-1}$. These target inputs are normalized to the range $[-1,1]$ to improve the behavior of the network. We use input using a GUI fader to manipulate the torso height and velocity. This type of parameter manipulation can be useful when the user wants to have more control over the locomotion policy of the robot. Criteria such as energy consumption, smoothness, gait velocity, stability could be programmed and specified during test time using this method. The disadvantage is that the policy has to be retrained if the user wishes to add additional criteria.

High-level control

Most deep reinforcement learning results feature a locomotion policy that moves in a single direction to maximize a distance along a given axis. We propose a goal-based method of control where the agent receives as input relative x, y coordinates, and is trained to always walk towards the given coordinates, being rewarded for minimizing the distance between itself and the goal target.

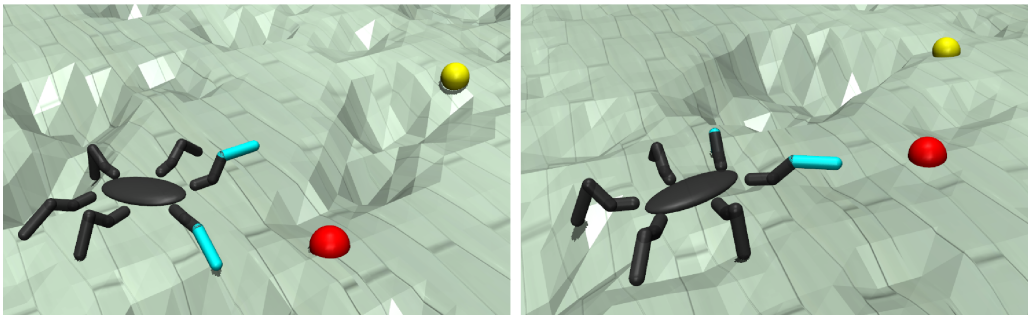

Figure 1.10: Double goal-based locomotion. Red is the immediate goal and yellow is the pending goal. The agent receives as the coordinates of the red goal relative to its own and the yellow goal relative to the red.

We use a double goal input where the agent is conditioned on the next two goals. The purpose of this is that the agent can learn to position itself for the next goal if necessary. As soon as the agent is within proximity of the first goal, therefore completing it, the next goal becomes the current one, and a new goal is generated or added. The above methods are compatible with real-time user control. It can be used in conjunction with a mouse-click input method. It can also be used by a high-level planner that uses information from a global map.

1.6.3 Solving POMDPs with Recurrent Neural Networks

If we treat the problem of decomposition of complex terrains into N discrete ones as a partially observable task with a latent variable τ denoting the current terrain that the agent is on, it should theoretically be possible to learn a recurrent master policy which can learn to infer this hidden variable and pick the gait appropriately. Indeed a well-constructed LSTM has sufficient representational power to perform this task. We can show this by learning the LSTM policy to imitate the expert policies on joint terrains directly. The imitation works, but as in most behavioral cloning tasks, the compounding domain shift is significant, and the policy often finds itself in a state where it does not know how to perform. This can be mitigated using a technique called DAGGER [88], but we refrained from going down this path to avoid adding additional complexity. Another reason is that expert policies trained using RL will always be more robust than an imitated policy.

The question remains, though, why a recurrent agent does not learn an adaptive gait if trained end-to-end on random joint environments. We speculate that the discovery of a gait already results in a strong local minimum, which is difficult to get out of to adapt to other required scenarios. A a result, the policy learns a single gait, which is a compromise between all the different environments that it was trained on instead of learning a distinct g it for each environment. To mitigate this issue, the policy would have to somehow explore at the latent gait level, which we do not as of yet know how to do.

There is another reason as to why it is advantageous to multiplex expert policies instead of

having a single master policy. If we discover that our agent performs poorly on a specific part of the target environment, we can model that environment and quickly train an expert policy on just that environment. The time it takes to train a single expert on an environment as well as to retrain the multiplexer policy using supervised learning is significantly less than retraining a master policy on all the different terrains.

1.6.4 Synthetic ways of generating real sampled terrain

Learning adaptive legged locomotion in simulation requires programming a rich enough set of terrains using various spatially correlated noise functions such the Perlin generator, described in section 1.5.1. We propose a method to synthesize realistic terrain which can be sampled from the actual target terrain that we want to learn locomotion on. This can be done by first capturing a substantial amount of photographs (50-200) of a sample of the target terrain by hand. A smartphone camera is sufficient. The photographs are then fed into a free software package such as VisualSFM [89] [90] which reconstructs the poses of the camera and produces a dense pointcloud. This can further be imported into a free package called Meshlab [91] which creates a mesh from this pointcloud which can then be rendered as a height map. Having a heightmap we would then like to be able to synthe- size random heightmaps similar to this sample. For this task we use EbSynth [92] which is an efficient implementation of patch-based synthesis which supports annotation of various segments of the example and output image. A heightmap in this regard behaves similarly to a texture. Figure 1.11 shows the pipeline to synthesize new samples from real terrain.

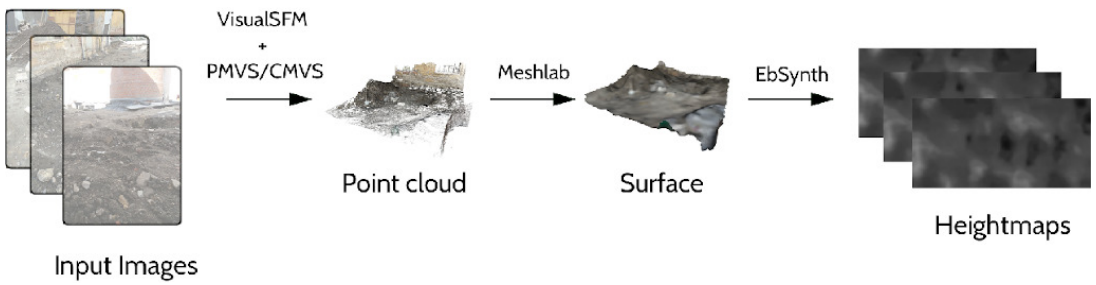

Figure 1.11: Terrain synthesis pipeline. The user provides photographs of a patch of terrain from various angles which are converted into a pointcloud using structure from motion. This is then turned into a heightmap from which similar heightmaps are synthesized using patch based synthesis.

The whole process takes roughly 1 hour to complete, including computation times for the reconstruction. A sample of the results of synthesized terrain for images taken near s sand-piled construction site can be seen in figure 1.12. One disadvantage in using this technique is that there is a trade of between small and large scale features. If detailed terrain features are required then the target patch should be small, in the order of 1m2 so that small scale details can

be captured. It is also theoretically possible to do a heighmap synthesis with RGB texture on top, but this would require modification of the patch based algorithm to work with an additional dimension. This is something that we left for future work.

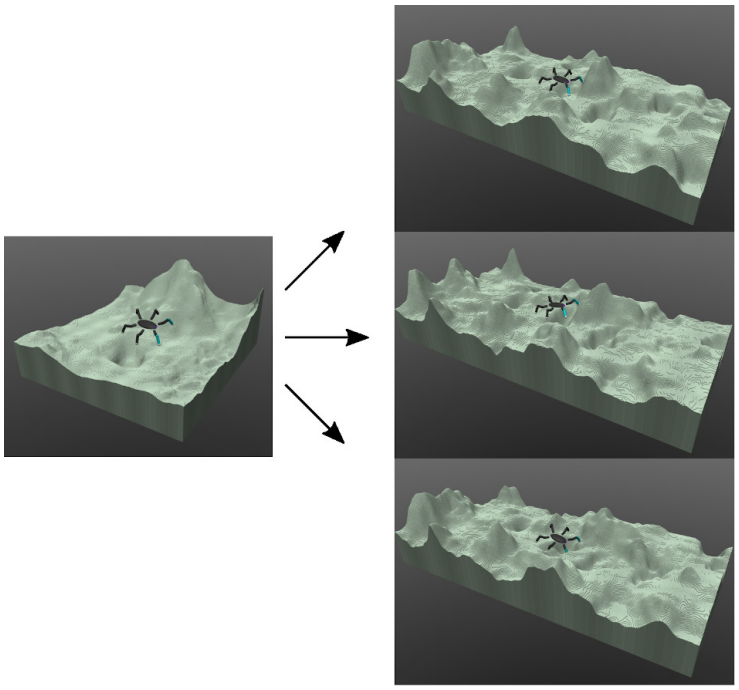

Figure 1.12: Examples of terrain synthesized using EbSynth from a reconstructed heightmap using VisualSFM.

1.7 Discussion

We presented a data-driven approach to hexapod locomotion that requires almost no domain knowledge and no hand-crafting of the locomotion policy. Our approach could be used in a setting where the target environment is structured and that we have an idea on what to expect. The whole concept of the N-terrain decomposition was made with the idea that the operator can analyze the terrain beforehand and pick out N difficult aspects that are then used to train locomotion. It's not necessary not capture every single feature of the target terrain, as we saw that policies that have been trained on sufficiently rich terrain perform well on a range of terrains, including ones that it has not seen during training. On the other hand, more difficult sections such as stairs and pipes require switching to a policy that was purpose-trained.

There are still several drawbacks in our approach, such as having to combine trained experts through a multiplexer policy. A better method would be one that automatically trains a recurrent policy that works on all terrains. As described in section 1.6.3, this problem is difficult and requires more exploration in suitable recurrent architectures and training algorithms. Another drawback is the necessity to manually program a terrain distribution generator from which ter-

rain samples can be generated. Future work might feature automatic terrain generation from gathered photos or videos using structure from motion and heightmap synthesis.

In our work, we strive to promote a practical application of data-driven locomotion for legged robots. One of the issues with transferring a learned policy from the simulator to a real platform is that the dynamics, sensing, and actuation can often differ. This is a well-known open problem in deep learning. In our case, sensory input is not an issue as we only use interoceptive data. Dynamics and actuation should not cause problems since the hexapod platform is statically stable, and the movement is more or less kinematic. Moreover, experiments show that our policies are robust to certain changes in parameters which attempt to model some of the differences between the simulated and real model.

Concerning learned locomotion policies, a visual analysis of the learned gaits shows that using a non-structured policy class such as an MLP exhibits asymmetries in the rhythmic movement, which could cause more wear to some servos more than others on a real platform. This expected as the MLP imposes no prior on the morphology of the robot. Graph-based policy classes such as in [19], [93] could help mitigate this issue. Unfortunately the authors only demonstrate their results on flat terrain which does not properly test the performance of the method. As mentioned throughout the paper, many of the difficulties of only emerge when the robot needs to turn and change gait. This is an open problem and we leave it to future work to experiment with various policy classes.

Given the current trend in all the various subfields of AI, it is clear that more general approaches that can leverage large sources of compute will dominate hand-crafted ones. We believe that the future of robotic locomotion is going to be almost entirely data-driven. Simulated training environments will likely be generated procedurally or in an adversarial fashion, and locomotion policies will be optimized on those environments using RL or some other heuristic search algorithm such as Natural Evolution Strategies (NES) [94]. This is arguably the most flexible and scalable approach, and as argued by [95], scalability is key for the future of any almost AI technique.

Chapter 2

Learned procedural hexapod locomotion

Most data driven control methods start from a black box function approximator such as a neural network and use reward shaping and auxilliary tasks [96] to guide the policy into performing as the designer expected. In this section we take the opposite approach. We study a case where we can improve an existing procedural hexapod locomotion algorithm using neural networks and black-box learning methods. Despite the high dimensionality of the overactuated hexapod platform, the locomotion task exhibits both morphological and temporal structure. Approaches such as procedural locomotion generation use inverse kinematics and leg scheduling to move the torso towards a target location. Such approaches are prominent in graphical animation. However, considering kinematics alone can often cause failure in dynamical environments due to slippages and other unforseen effects. We propose taking such an approach and replacing the leg scheduling and other heuristic decision points with neural network modules. We show that we can formulate the gait-phase decision for each leg as learnable state machines and use Evolutionary Strategies in simulation to optimize the free parameters. We also show that other useful locomotion traits, such as torso and leg height can be optimized as well. We propose several weight-sharing schemes between the legs of the hexapod that lead to fewer learnable parameters, a stronger inductive bias that leads to faster learning. The result improves on the baseline procedural approach on difficult terrains such as stairs. Our approach attempts to inject learnable experience from simulation into an algorithm with a high inductive prior, resulting in a hybrid control policy that has superior performance on a wide range of uneven terrains. The work in this section is mostly based on our publication titled ”Improving procedural hexapod locomotion generation with neural network decision modules”, that was presented at the Modelling & Simulation for autonomous systems conference (MESAS2022). We added several figures and additional descriptions that we did not see fit to include in the paper.

2.1 Introduction

Legged robots are one of the most versatile robotic platforms due to the large variety of difficult terrain that they are capable of traversing. This has motivated many areas of research for the construction and locomotion of such platforms. Various morphologies, such as two, four, and six-legged platforms have their own advantages [97] and drawbacks. The Hexapod platform has six legs with three degrees of freedom for each leg, totaling in 18 degrees of freedom. The most prominent advantage of such a configuration is static stability, meaning that locomotion can be generated kinematically, without significant consideration for the dynamics of the system. Such approaches are often used in graphics animations [98] and high-end game engines. On real robotic platforms, where slippage, obstructions, and other difficulties are present, we may find that various heuristics are required to tune the algorithm to perform robustly in a given environment. Such tuning can be difficult and time-consuming. Instead what can be done is to modify the handcrafted algorithm and replace various heuristic decision points with learnable functions such as neural networks. These can then be optimized in simulation on a wide variety of complex terrains until suitable behavior is achieved. This approach is in contrast to the classical end-to-end learning paradigm where we assume that we know nothing about the problem and train a large neural network to solve the task using Reinforcement Learning. Such methods are powerful and have shown impressive performance in various tasks, but require a large number of training steps and are more difficult to generalize to unknown settings, due to the lack of inductive prior of the underlying algorithm structure.

2.2 Related work

Various approaches have been developed over the past years which study hexapod locomotion. Biologically inspired solutions are popular and mostly revolve around using Central Pattern Generators (CPG) for generating a locomotion gait. Such generators can be tuned or learned [99] according to a given reward function. CPGs have also been implemented for legged robots in analog circuits [100]. Some works have also used plain oscillators with variable frequencies for each leg with interdependent signals to generate locomotion [101]. Other approaches include using leg planning [102] and proprioceptive sensing methods for gait adaptation such as in [103]. Powerful learning methods such as Reinforcement learning (RL) can be used to generate locomotion for hexapod robots. The work of [104] uses neural networks with focus on decentralized control. Inductive priors such as graph-like structures can also be applied to neural networks in conjunction with RL for hexapod robots [19]. The work of [28] uses a heirarchical structure of expert policies that provide adaptive locomotion in difficult terrains.

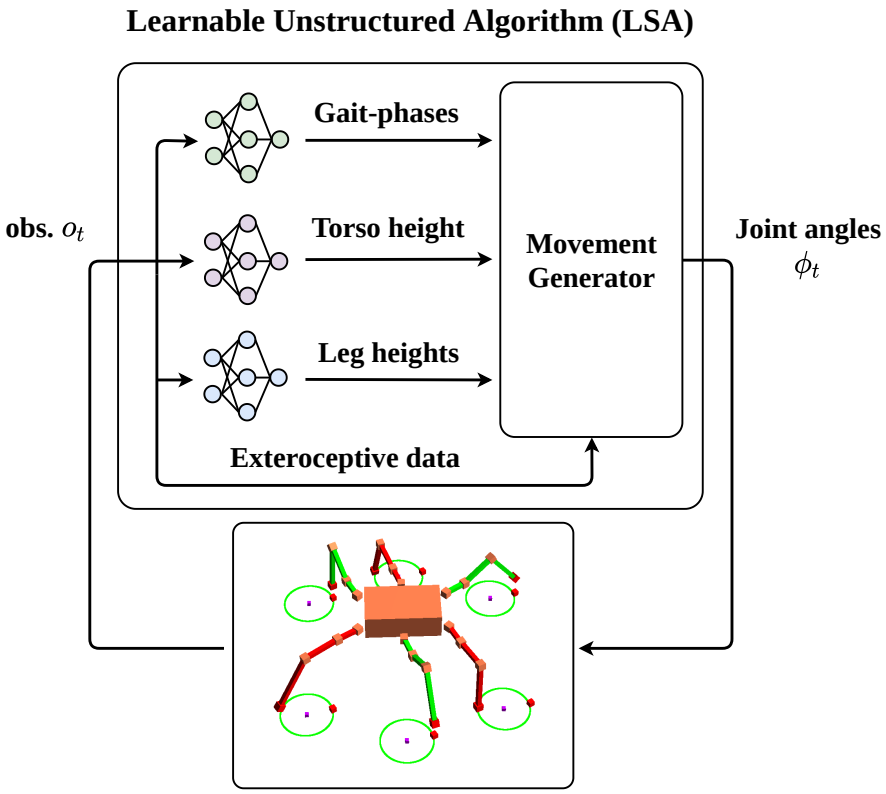

Figure 2.1: Learnable unstructured Algorithm (LSA) illustration

Alternative, non-learning approaches include using kinematic models and procedural generation for locomotion, such as the work of [98]. These algorithms are strongly structured, perform well, but experience issues when deploying on a physical engine. Some works have attempted to model hexapod dynamics [105] to assist with this. Nevertheless, it is necessary in such methods to relax some of the heuristic parts of the algorithms and replace them with neural networks so that we can learn more complex behavior through trial and error. Black box algorithms such as Heuristic Random Search algorithms such CMA-ES [106] and Augmented Random Search (ARS) [12] can be used to optimize any set of free parameters in a given algorithm by using interactions with an environment that are evaluated by a reward function. Some works such as [107] have leveraged similar algorithms to improve openloop sinusiodal locomotion generation for hexapod robots. Other works such as [108] use similar search algorithms to improve programmed algorithms by optimizing for locomotion within a constrained gait space. Our approach is similar to [108], but we propose to use HRS algorithms to improve a kinematic approach similar to [98] by replacing various modules using neural networks and using blackbox optimization to find the parameters which improve the initial algorithm on difficult terrains such as stairs and high tiles.

Our contributions can be summarized in the following points

- A hybrid locomotion algorithm that consists of hand designed kinematic logic as well as learnable decision points.

- Training procedure and performance evaluation of the baseline procedural locomotion method against our improved version with learnable state-machine leg scheduling, and torso and leg height regression.
- A brief analysis of the training difficulties of unstructured vs structured algorithms.
- An environment of several difficult terrains in the Pybullet simulation for locomotion testing.

2.3 Locomotion generation

In this section we will describe the hexapod platform in brief, particularly perception and actuation. We also breakdown the components of our procedural locomotion algorithm, and describe how we attempt to improve the performance by replacing various components with neural network modules.

2.3.1 Hexapod environment overview

The platform that we use is a model of the MKII Phantom X hexapod in the Pybullet simulator. This hexapod has six legs with three joints in each leg, shown in 2.2.

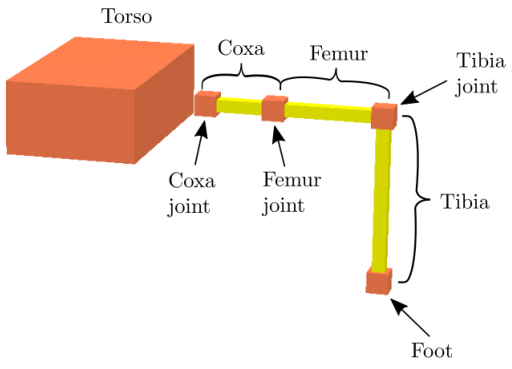

Figure 2.2: Hexapod leg joints description.

The actuators are modeled as servos that accept target joint angles and implement control using a PID regulator. Proprioceptive observations include current joint angles and velocities, as well as torso attitude. To obtain an algorithm that can be realistically used on a real platform, we use sparse point cloud observations that could be obtained from a depth camera or lidar. From the sparse point cloud we compute a height map by dividing the point cloud regions into a regular grid, shown in figure 2.3. Empty voxel heights are calculated by interpolation from neighboring voxels.

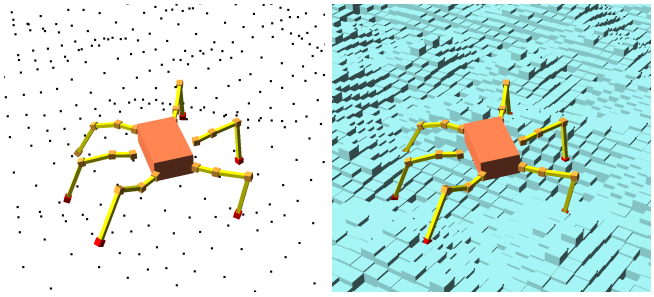

Figure 2.3: Point cloud heightmap approximation.

We also assume that we will be running an accurate localization algorithm such as ICP so in simulation we use ground truth position data for our experiments.

2.3.2 Algorithm structure

The locomotion algorithm consists of a movement generation module which accepts a set of pre-computed gait parameters, set of observations and target control inputs. In this section we will describe all the above components.

Observations and control inputs

The observations and target control inputs include the following:

- Current state of the hexapod $s_t = (x_t, q_t, j_t)$ which consists of position x_t, orientation q_t and joint angles j_t
- Control inputs θ_t^{tar}, and v_t^{tar} defines the target direction angle and the xy velocity respectively. These are provided by the user or by a waypoint follower.
- Point cloud heightmap pc_t provides exteroceptive data for the algorithm, shown in figure 2.3.

Movement generation module

A single step of the movement generation module is described in algorithm 1.

Algorithm 1 Movement generator step

Estimate surface normal
Update torso pitch and roll based on the surface normal
if no legs are at the operating boundary **then**
    Use IKT to move torso position and yaw based on control input
Update torso height based on input control
Calculate torso transformation matrix
for each leg **do**
    Update leg given precomputed gait parameters and observations (see 2.3.2)
write joint angles to servos

The leg update function, mentioned in algorithm 1 consists of calculating the target foot position using direct kinematics, and then deciding the movement depending on whether the leg is in *stance* or *swing* phase. If in *stance* phase, then the leg follows the height of the corresponding position of the terrain. If in *swing* phase, then the leg height and position is updated according to the progress of the swing. Phase decisions are described in 2.

Target foot position calculation

Foot placement depends on the control input. The following examples can be seen in Figure 2.4.

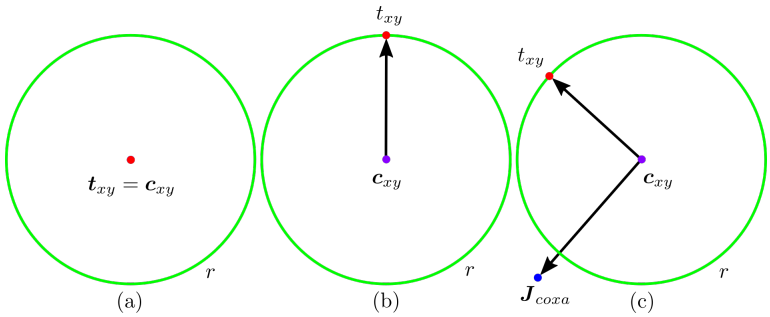

Figure 2.4: Visualization of foot placement defined by Equations 2.1 and 2.3.

When the input control is static, the foot target t_{xy} is held in the middle c_{xy} of the boundary. Otherwise, the step-down target t_{xy} is placed within range $\boldsymbol{r}$ in the direction $\boldsymbol{d}$ of translational or rotational control, shown in equation 2.1.

$$\boldsymbol{t}_{xy} = \boldsymbol{c}_{xy} + r\frac{\boldsymbol{d}}{\|\boldsymbol{d}\|} \tag{2.1}$$

Turning control is shown in equation 2.3

$$\boldsymbol{w} = (\boldsymbol{J}_{coxa} - \boldsymbol{c}) \times \boldsymbol{z}_{world} \tag{2.2}$$

$$\boldsymbol{t}_{xy} = \boldsymbol{c}_{xy} + rd\frac{\boldsymbol{w}_{xy}}{\|\boldsymbol{w}_{xy}\|} \tag{2.3}$$

The leg z coordinate is then sampled from the heightmap location corresponding to the target foot position

2.3.3 Precomputed gait parameters

Precomputed gait parameters: We consider the following important three gait parameters that have a significant impact of the locomotion: a) Gait-phases: These variables decide whether a leg will transition into one of the possible $\{stance, swing\}$ phases. b) Torso height: Torso height can be important for locomotion and it is not always straightforward to decide how high it should be based on underlying terrain. c) Leg lifting height: Leg lifting can have an impact on the quality of the gait, stability as well as collision with the neighboring terrain. In the following two subsections we describe how these gait parameters are computed manually and how we propose to replace them using neural network modules.

Hand designed gait parameters

At a given time, an individual leg can be in the *stance* or *swing* phase. When in the *stance* phase then the leg remains in this phase until it is nearing defined kinematic reach boundary of that given leg, where it automatically goes into the *swing* phase, described in algorithm 2. We use a stable tripod gait which activates groups of three legs at once, shown in figure 2.5

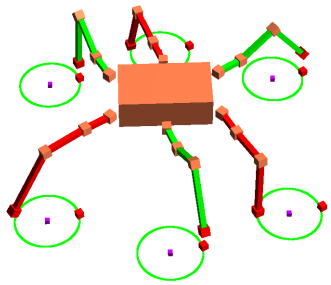

Figure 2.5: Tripod gait. The leg groups can be distinguished by the red and green color.

Algorithm 2 Gait-phase transition

```
if Group 1 is in stance phase and some leg from Group 2 is stuck then
    Put Group 2 in swing phase
if Group 2 is in stance phase and some leg from Group 1 is stuck then
    Put Group 1 in swing phase

for leg in All legs do
    if leg distance from target foot position ¡ some threshold then
        Put leg in stance phase
```

The movement generator calculates the torso height as follows:

$$h_{t_z} = \frac{1}{6}\sum_{j=1}^{6} t_{j_z} + h_d + h_i \tag{2.4}$$

Where h_{t_z} is the torso height in world coordinates, $t_{1_z}, t_{2_z}, \ldots, t_{6_z}$ are z-coordinates of the target positions, h_d is the default height value, and h_i is the torso height control parameter.
Leg height when lifting is calculated as follows:

$$h_l = h_t + h_d + h_i \tag{2.5}$$

Where h_l is the leg height in the world coordinates, h_t is the terrain height sampled from heightmap, h_d is the default lifting height, and h_i is the leg lifting height control parameter.

Neural network gait parameters

Gait Phase decisions We implemented a neural network for each of the 6 legs that can decide to switch between the *stance* and *swing* phases given interoceptive and exteroceptive data. The observations o_t^i for leg i and timestep t consist of current leg phases ph_t, leg tip ground contacts $cp_t \in \mathbb{R}^6$ of all legs, leg joint angles $j_t^i \in \mathbb{R}^3$, distance to the target position $d_t^i \in \mathbb{R}^3$, surface normal $sn_t^i \in \mathbb{R}^3$ and leg boundary conditions $lb_t \in \mathbb{R}^6$ of all legs. The whole observation is shown in equation 2.6.

$$leg_t^i = \{ph_t, cp_t, lb_t, j_t^i, d_t^i, sn_t^i\} \in \mathbb{R}^{25} \tag{2.6}$$

Each leg gets detailed information such as joint angles only of itself, but global information such as leg contact points of all other legs as well. This way the observation can stay relatively low dimensional but still be informative. The neural networks can collectively decide which legs have slipped, which are stuck and can adapt to unforseen behaviors.

The neural network is a Multi Layer Perceptron (MLP) with ReLu activation functions and a hyperbolic tangent (tanh) function at the output. Each network $pi^i >_{gait_{\theta_i}}$ maps input leg_t^i to an output $a_{t+1}^i \in \mathbb{R}$. The phase was decided according to equation 2.7

$$\text{leg phase} = \begin{cases} \text{swing-phase}, & a_1 > a_2 \\ \text{stance-phase}, & a_1 \leq a_2 \end{cases} \tag{2.7}$$

Figure 2.6 shows the structure of such a network.

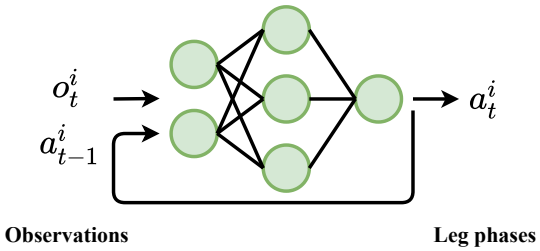

Figure 2.6: Generic Neural Network structure

Torso and leg height For predicting the Torso and leg heights we use a similar neural network architecture as the gait phases, but with slightly different inputs. For the Torso height prediction the observation consists of gait phases ph_t, leg boundary conditions lb_t, leg contacts cp_t, surface normal sn_t^i, torso orientation $torso_t^{ang}$ and the torso height $torso^{hm}$ sampled from the heightmap. This totals to a dimension of 25.
For the leg height prediction of leg i we only a small 8 dimensional vector consisting of the surface normal sn_t^i, leg joint angles j_t^i, leg ground contact cp_t^i and current leg height $l_h^i eight$.

Weight sharing patterns for gait phases Given that we will be making decisions for each leg of the hexapod robot, it makes sense to consider the structure of the morphology so as better to distribute the weights of the neural network and to avoid redundancy. We propose three options of weight sharing. The first is to allow each leg neural network module to have unique weights. This leads to a large amount of parameters and is more difficult to train. The second scheme is to use bilateral weight sharing to exploit the bilateral symmetry present in the robot. This would mean that we only have 3 sets of weights, and the rest will be mirrored. The last scheme is to use the same weights for each leg of the hexapod, resulting in only one set of weights.

We experimentally found that the third proposed scheme to share a single set of weights across all 6 legs works best. Some locomotion results can even be seen after a single epoch of training. The only difficulty with the bilateral and full weight sharing scheme is to correctly route the inputs of the other legs to each other input.

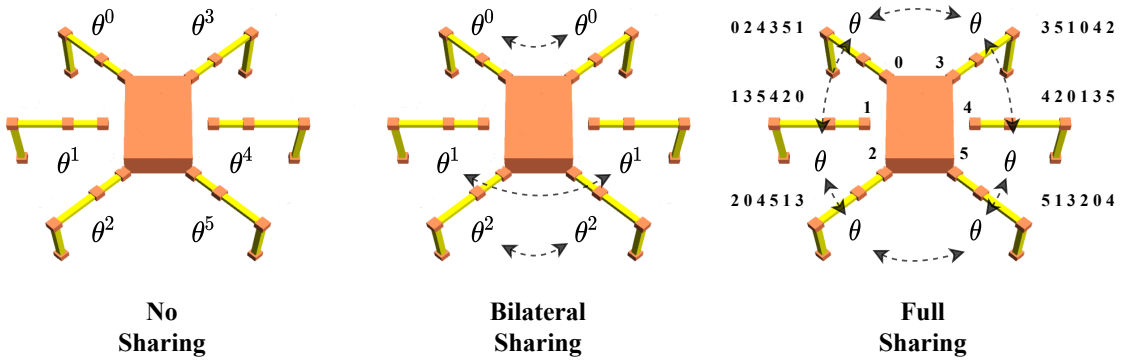

Figure 2.7: Neural network weight sharing schemes.

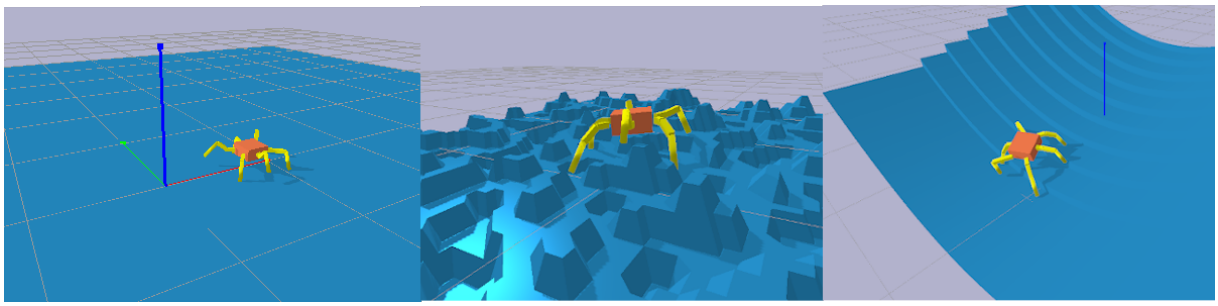

Figure 2.8: Experimental terrains: Flat, rocks, stairs

2.4 Experiments

Training and testing is done in the PyBullet simulator. The environment consists of the physics engine, hexapod robot, and the reward function. Episodes start with the robot spawned at a given position. The main goal is to achieve successful locomotion in the direction of the positive x axis, along with other criteria described below. We test three different terrains with varying difficulties and challenges, shown in figure 2.8. The rocky terrain is generated by discretizing the height of a 2 dimensional Perlin noise function. [1]

Training is done using the CMA-ES algorithm by sampling a batch of candidate solutions from the current distribution, evaluating them for a single episode each and then then updating the candidate solutions according to the rewards. We consider three different criteria for evaluation: a) Distance travelled, which is calculated by summing the instantaneous $\dot{x}$ velocities, b) Locomotion smoothness per distance, calculated by summing torso translational and angular accelerations a_t and ε_t, normalized by the distance travelled, shown in equation 2.8 c) Power spent, by summing the product of instantaneous joint torques and velocities, shown in equation 2.9.

$$r_s = -\|\boldsymbol{a}_i + c\boldsymbol{\epsilon}_i\|^2 \tag{2.8}$$

[1] Video: `https://vimeo.com/744114592`

$$r_p = \sum_{k=1}^{18} |\tau_k| \cdot j_k \tag{2.9}$$

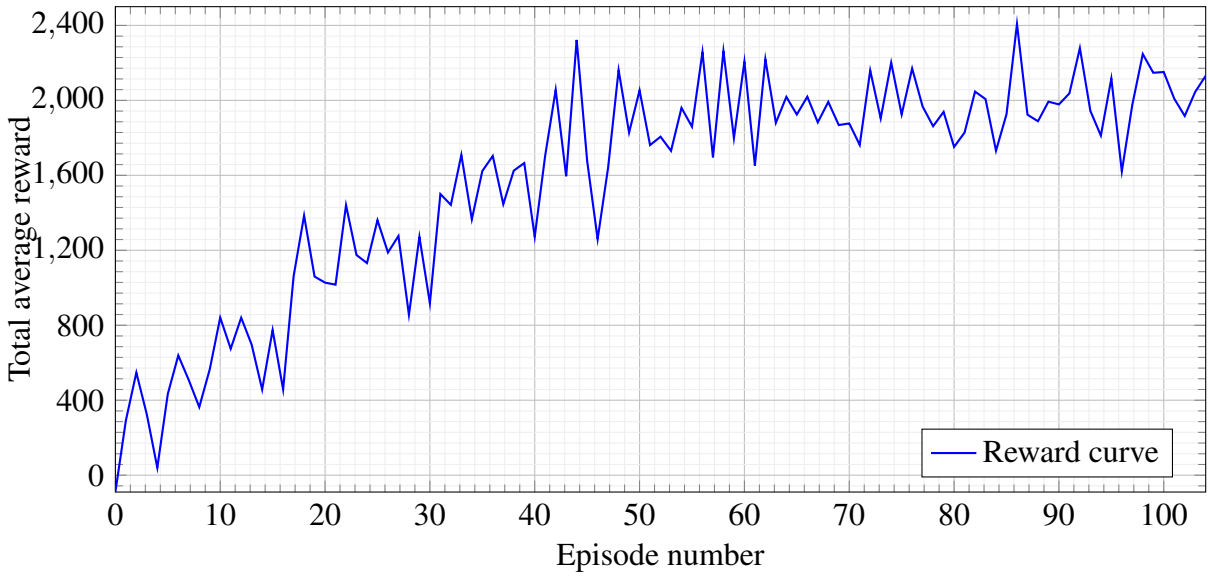

Figure 2.9: A training curve of an optimization procedure of our learnable structured algorithm approach

We trained and evaluated our baseline structured algorithm (SA) approach, against the improved proposed learnable structure algorithm (LSA) and also an end-to-end (E2E) neural network. The approaches were trained individually on all three terrains. Results are shown in tables 2.3. D is the distance reached, S is the smoothness per distance, P is the power consumption per distance.

We can also see from figure 2.10 that our proposed learned variant (LSA) is superior to the baseline algorithm (SA) with increasing terrain difficulty in distance travelled and power consumption. This suggests that the baseline algorithm is suboptimal and admits improvement.

Reward landscape One interesting thing to note when training a highly structured algorithm versus a neural network is that the reward landscapes are vastly different. A small change to the weight of the E2E neural network has a smooth and roughly proportionate effect on the reward. The structured algorithm, on the other hand, is much more sensitive, especially when optimizing something like the gait phase scheduling. We attempt to show the various landscapes by projecting parameter-reward pairs to a 2 dimensional plot using Principle Component Analysis (PCA). Figure 2.11 shows the structured LSA algorithm on the left, and the unstructured neural network on the right. The color intensity denotes reward. We can see that in the unstructured algorithm there is quite a distinct path in 2D which leads the weight vector from the random initialization to the optimal solution. This is something that can make training algorithms with

Policy	D	S_d	P_d
SA	28.58	**-7.40**	-94.85
LSA	32.58	-9.19	-86.67
E2E	**135.21**	-339.17	**-61.88**

Table 2.1: Performance on flat terrain

Policy	D	S_d	P_d
SA	16.31	**-202.74**	-262.66
LSA	**23.03**	-210.42	**-190.03**
E2E	15.31	-1906.96	-643.16

Table 2.2: Performance on rocky terrain

Policy	D	S_d	P_d
SA	6.97	**-251.06**	-603.37
LSA	8.24	-481.10	**-588.32**
E2E	**9.37**	-2374.00	-818.58

Table 2.3: Performance on steps terrain

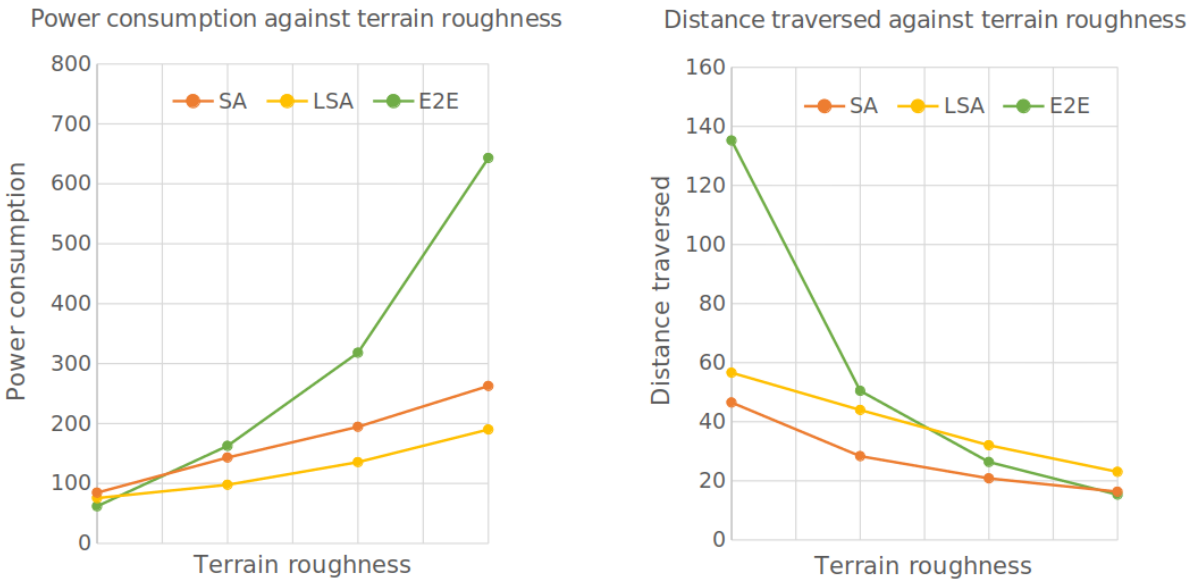

Figure 2.10: Trends comparing the various policies

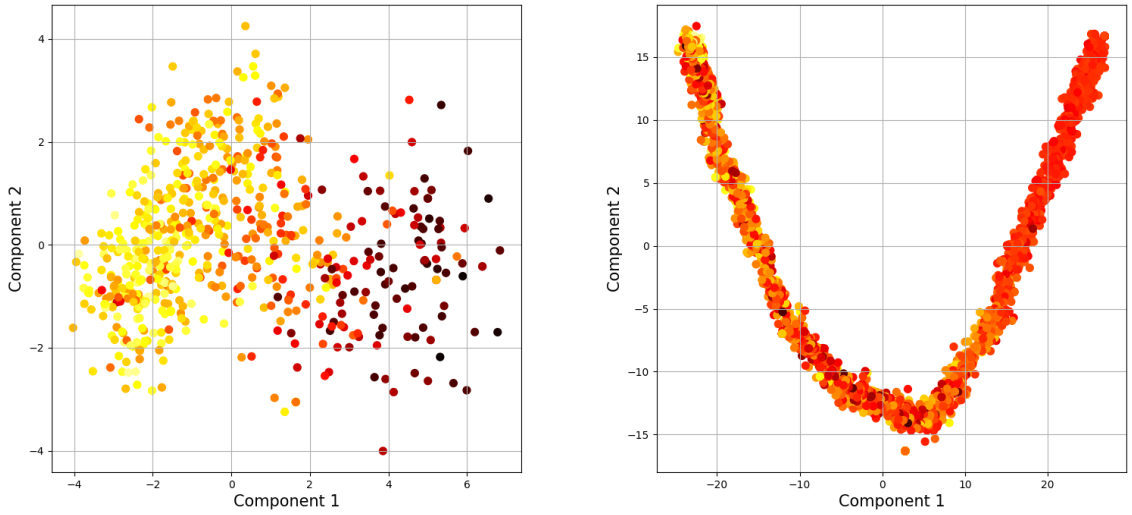

Figure 2.11: Parameter vectors of the LSA (left) and E2E (right) projected to 2D with PCA. The figure suggests that the E2E reward landscape is significantly more smooth and informative than the LSA.

high inductive prior quite difficult using techniques such as random search.

2.5 Discussion and Conclusion

We proposed several experiments to compare the performance of our procedural locomotion algorithm (SA) and the variant where we replace the gait scheduling and several other control parameters with neural network modules. From the table 2.3 we can see that the algorithm with the largest traversed distance and lowest power is the end-to-end neural network (E2E). This is, however misleading as we can see from video results that the hexapod cheats by making large jumps and strides, gaining a large distance. It shows that it is more difficult to train such a task without a strong prior and more work has to go into the reward function to prevent such artifacts. Tables 2.3 show that our proposed neural network modules alow the structured algorithm (LSA) to traverse significantly more terrain than the static variant (SA). This is also confirmed in graphs shown in figure 2.10. Figure 2.10 also shows that although the distance traversed by the E2E algorithm is large for smooth terrain, it is overtaken by our LSA variant in more difficult terrains. By visually inspecting the resultant locomotion policies in the supplementary video we can see that the unstructured variant is very shaky and

We also note that both approaches, structured and unstructured, that were presented have their own difficulties during training. The unstructured approach (E2E) requires more involved reward engineering due to the lack of inductive prior in the policy structure. The structured variant, however, can be more difficult to train due to the noisier reward landscape. This can be

seen in the two dimensional structure of candidate parameter vectors of the CMA-ES algorithm, obtained by PCA projection, shown in 2.11.

In conclusion, we have shown that we can improve already programmed locomotion algorithms by replacing various decision points by neural network modules. We think that this approach is useful not only for legged robots, but for other platforms where we can make use of a mix of problem domain knowledge and learned experience.

Part III

Articulated tracked robot control

Chapter 3

Hybrid control policies

In this section we look at tracked robot control. Specifically, we study the autonomous control of independent robot flippers on two different platforms with similar morphologies. The motivation of this work is to enable semi-autonomous of full autonomous deployment of such a platform in complex urban and outdoor terrains by providing with an algorithm that correctly sets the individual flippers in suitable position. We demonstrate a hybrid approach to autonomous flipper control, focusing on a fusion of hard-coded and learned knowledge. The result is a sample-efficient and modifiable control structure that can be used in conjunction with a mapping/navigation stack. The backbone of the control policy is formulated as a state machine whose states define various flipper action templates and local control behaviors. The state machine transitions are also used as an interface that facilitate the gathering of demonstrations using a D-PAD gamepad to train the transitions of the state machine. One of our contributions is a soft-differentiable state machine neural network that mitigates the shortcomings of its naively implemented counterpart and improves over a multi-layer perceptron baseline in the task of state-transition classification. We show that by training on several minutes of user-gathered demonstrations in simulation, our results show a considerable increase in performance over a previous competing approach in several essential criteria such as traversal smoothness and physical shock. We demonstrate zero-shot domain transfer from simulated training to a wide range of obstacles on a similar real robotic platform. We successfully deployed an earlier version of our work in the Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge to alleviate the operator of manual flipper control. In the urban circuit our approach was able to autonomously traverse stairs and other obstacles, that enabled the robot to get to hard to reach place, improving map coverage and attaining a higher overall score. The work on this section is taken from our publication titled ”Autonomous state-based flipper control for articulated tracked robots in urban environments”, published in the IEEE Robotics and Automation Letters Journal, and presented at the IROS 2022 conference. We also include additional mathematical descriptions to our proposed soft-differentiable state machine, as well as illustrations of various

parts of the system that could not make it to the publication due to page constraints.

3.1 Introduction and overview

Robotic control is a challenging problem usually approached by applying first principles or engineering a heuristic solution based on an underlying knowledge of the problem. This often results in intractable solutions or limited performance due to highly non-linear dynamics or high-dimensional observations. Lately, many works have shown impressive robotic control performance in various difficult tasks [109], [110], [111] using trial and error-based approaches such as Reinforcement Learning or Random Search [37]. This approach typically assumes little or nothing of the underlying task and therefore allows the use of flexible parametric control functions such as neural networks [112]. Such control functions are robot-agnostic and can lead to a less than desirable behavior, requiring tedious reward engineering to alleviate the issue. The weak inductive prior in such flexible functions manifests itself in high sample complexity. This requires very fast simulations and typically tens of millions of training steps to get decent results. Finally, the result of such a monolithic algorithm is a non-interpretable black box. We aim to bridge the extremes of hard-coded control with learning-based methods and explore an approach that attempts to unify and make use of the best of both worlds. We hypothesize that by analyzing how a human operator performs a robotic task manually, we can structure our approach appropriately. The idea is to partition the control into a module that is effectively learnable from demonstrations and another module that is simple enough to be heuristically hard-coded. The combination of these modules results in a control approach that preserves modularity and allows a degree of interpretability. We demonstrate this approach on the task of flipper position control of an articulated tracked robot for purposes of obstacle traversal.

From extensive interaction with the robot, we observe that for most structured obstacles, it is sufficient to divide the obstacle negotiation into several discrete phases: *Neutral (N)*, *Ascending-front (N)*, *Ascending-rear (AR)*, *Stairs-up (SU)*, *Descending-front (DF)*, *Descending-rear (AR)*, *Stairs-down (SD*. The state names correspond to the phase of obstacle negotiation at which the robot is currently at. Figure 3.1 shows these phases (states), and their corresponding transitions, which together form a state machine. Transitions are learned from demonstrations by a human operator using a novel soft-differentiable state machine architecture. Each state additionally defines local flipper controllers, which allow us to hard-code desired behaviors into the system. At each timestep, the state machine classifier decides the next state transition. The selected local flipper controller then overlays flipper template positions, as well as roll stabilization and escape maneuvers to generate the final flipper target angles. For clarity, we will refer to our proposed control system as the hybrid flipper controller (HFC).

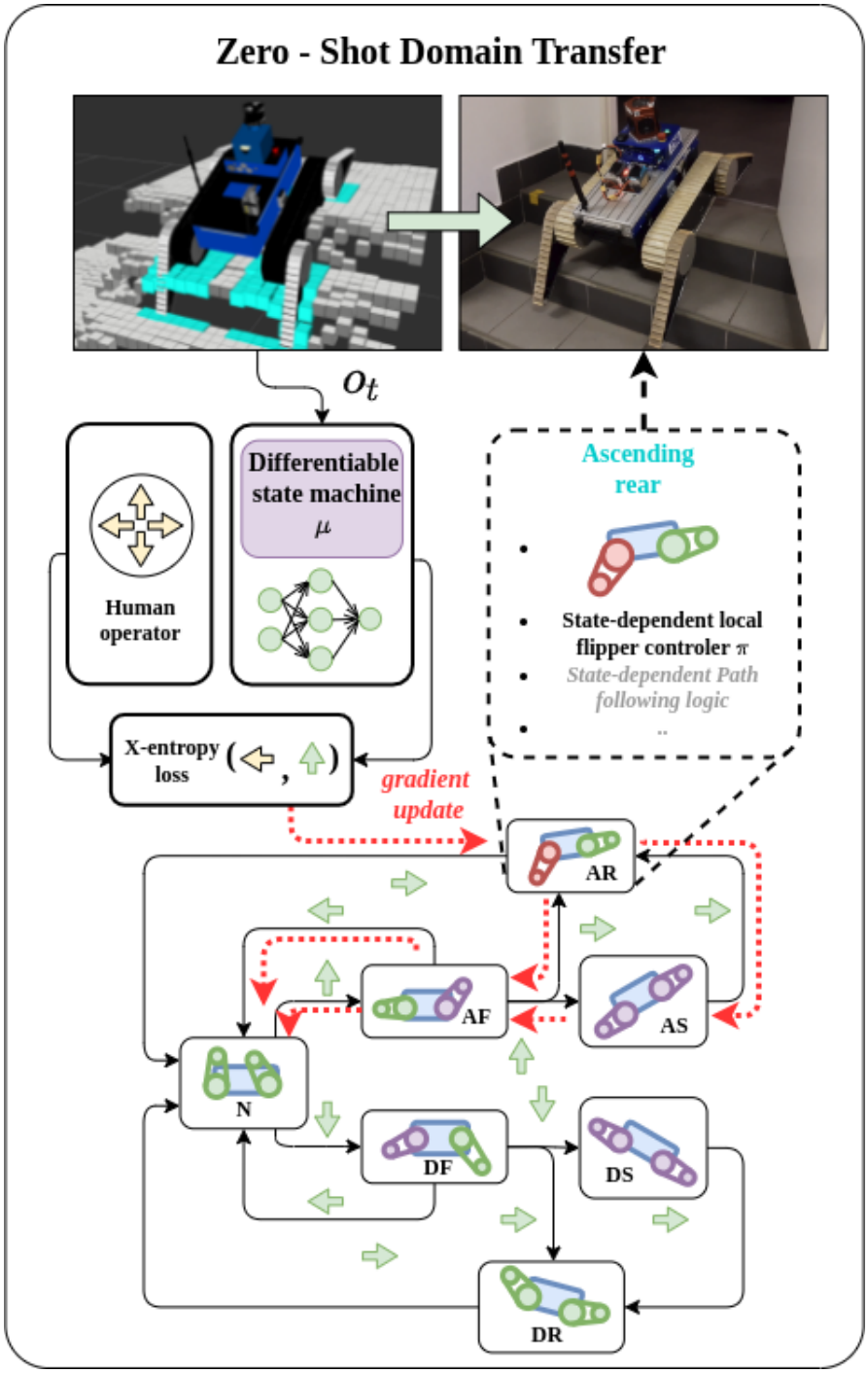

Figure 3.1: An illustration of our hybrid approach. The red dotted lines show the gradient flow from the loss function. The defined flipper states and their transitions are shown using black arrows. Self-transitions are omitted for clarity. In all states, the robot is shown as right-facing. The flipper colors correspond to predefined flipper torques with green being the lightest and red the heaviest. State acronyms are expanded in Section 3.4.

Our work is summarized by the following contributions:

- A state-based control architecture for articulated tracked robots combining learnable and hard-coded control elements which achieves zero-shot domain transfer performance on a variety of difficult urban obstacles.
- Proposal and comparison of several state-of-the-art structures for learning state-transition functions, including our proposed soft-differentiable state machine.
- Intuitive state-based human control interface for gathering quality locomotion data for learning a state-transition policy for autonomous control.
- A training and test obstacle course environment in the OSRF ignition gazebo simulator with accurate tracked robot plugins [113] for comparing and testing the performance of proposed control algorithms.

3.2 Related work

Articulated tracked robot control is a complex perception and control problem. The choice of approach can depend on the target environment and the available sensory configuration. The authors of [114] demonstrate that simple geometric approaches with a limited sensory payload can be used to estimate suitable target flipper angles which keep the robot stable. The work of [115] uses Reinforcement Learning (RL) to optimize a neural network policy but focuses specifically on stairs traversal by using externally calculated edge features. The work by [116] explores safety in RL and demonstrates safe learning on a single low palette, whereas our work is focused on a variety of difficult obstacles. A similar work uses a safe RL algorithm for staircase negotiation for assistive robots [117]. Computationally demanding approaches such as planning [118] have been used in traversing certain obstacles. Planning requires a fast and flexible simulator that allows placing the robot in various positions. It also requires privileged information, which we do not have because our observation features are built from an elevation map that is discovered on the fly. The authors in [119] teach a linear classifier to predict a suitable flipper template by learning a q-function from random actions at various poses, annotated by noisy human rewards.

We build on the idea of using discrete flipper templates from [119], extending and improving the work in several ways: Firstly, we constrain the problem to a sequential decision task by explicitly modeling phase transitions using a state machine. We propose a learnable

soft-differentiable neural state machine architecture that solves several issues of a naively implemented neural state machine, described in Section 3.4. Secondly, we expand the notion of flipper templates to states which define other attributes and hard-coded behaviors, allowing us to inject domain knowledge into the system and train the classifier under the distribution of the entire system. Thirdly, we use a cleaner imitation learning approach using high-quality demonstrations made possible by our state machine structure. We show extensive experiments on a complex variety of urban obstacles both in simulation and a corresponding real platform and show that our hybrid approach significantly outperforms the previous work. Imitation learning has seen use in several robotic tasks, such as autonomously following a forest trail from visual input [120], or car steering imitation [121]. This learning approach makes sense and can be preferred over Reinforcement Learning when we can easily gather high-quality demonstrations, such as in our case.

3.3 Formulation of the control and optimization problem.

For consistency with existing robotic learning literature, we will use a subset of a Markov Decision Problem with an additional latent state to formally describe the control problem.

- A set of states $s \in S$ of a robot in which each state uniquely describes the configuration of the robot and environment.
- Observations $o \in O$ which denote what the control agent sees at each time step. Observations consist of robot body roll and pitch information, and exteroceptive feature vectors $f_t^i \in R^4$ for each flipper index $i \in 1,2,3,4$. These vectors represent the height-map terrain region pc_t^i in defined bounding box regions around the front and rear flippers.

$$o_t = \{f_t^1, f_t^2, f_t^3, f_t^4, r_t, p_t\} \tag{3.1}$$

- Set of latent states $q_t \in Q = \{1,..,n\}$ of the state machine. Vector $p_t \in P \subset R^{|Q|}$ represents the probability of being in state i at time t. State transition function $p_{t+1} = \mu_\phi(o_t, q_t)$ is parameterized by learnable parameter vector ϕ.
- Actions $a \in A$ that a control policy $\pi_q = \pi(o_t, \text{argmax}\, p_t) : o_t \rightarrow a_t$ takes at each time step. These actions represent the target flipper positions of the robot. We use a total of $|Q|$ action policies π^q, each defining flipper behavior in latent state q. In our case, π consists of hard-coded behaviors which have no learnable parameters.
- World transition function $s_{t+1} = T(s_t, a_t)$ which advances the environment one time step after taking a specific action a_t. The robot and navigation stack is part of the environment.

The complete behavior of the flipper controller is defined by the combination of a learnable state-transition function μ_ϕ and hard-coded knowledge represented by policy π, as summarized in algorithm 3:

Algorithm 3 Flipper control inference

$p_1 = [1, 0 \dots 0]$ % *Initialization*
for t=1:T **do**
  $o_t = \text{get\_measurement}()$
  $q_t = \text{argmax}\, p_t$ % *Current latent state*
  $a_t = \pi_{q_t}(o_t)$ % *Control with hand-crafted policy*
  $p_{t+1} = \mu_{\phi_{q_t}}(o_t)$ % *Following latent state prob. distr.*

We learn the latent state-transition policy μ_ϕ from a set of demonstrations $D = (o_0, q_1), ..., (o_{m-1}, q_m)$ where q_{t+1} are latent states chosen by an expert human operator for observation o_t, described in more detailed in Section 3.6. The learning problem can be formulated as a sequential supervised classification of the following latent states with a cross-entropy loss l.

$$\phi^* = \underset{\phi}{\text{argmin}} \sum_{(o,q)\in D} l(\mu_\phi(o), q) \tag{3.2}$$

An alternative to learning from demonstrations would be to use trial and error methods such as Reinforcement Learning. Despite their recent successes of such methods, we decided to forgo them due to the following reasons: A simulator running sensor rendering and the entire navigation stack achieves a real-time factor of less than 1x on a modern 6-core CPU, which is roughly two orders of magnitude less than desirable. Secondly, it is almost infeasible to reliably reset an entire navigation stack on a system for many trajectory rollouts.

3.4 Proposed hybrid flipper control (HFC) architecture

In this section we describe in detail several learnable neural network architectures for state-transition classifier μ_ϕ as well as the components of the hard-coded flipper policy π. We also describe the data gathering procedure using the proposed state machine.

3.4.1 State-transition classifier μ_ϕ neural networks.

Deciding the next state can be represented by a function conditioned on a current observation o_t and in the case of state machines, additionally on latent state q_t. We consider the following three architectures:

Multi layer perceptron (MLP): The simplest structure is a vanilla neural network with two hidden layers which reactively map current observations to next state decisions. The input is the observation vector o_t and the output is a probability distribution p_{t+1} over all states Q.

$$p_{t+1} = \mu_\phi(o_t) \tag{3.3}$$

Training is done on randomly sampled mini-batches from the dataset.

Crisp neural network State Machine (SM) consists of $|Q|$ transition functions represented by $|Q|$ neural networks $\mu_{\phi_1}, \dots \mu_{\phi_{|Q|}}$ with the same architecture, but different parameters ϕ_i. To predict the next state, we use the $\mu_{\phi_{q_t}}$ which corresponds to the previously predicted state q_t, as shown in the following equation.

$$q_{t+1} = \text{argmax } \mu_{\phi_{q_t}}(o_t) \tag{3.4}$$

$$p_{t+1} = onehot\ (q_{t+1}) \tag{3.5}$$

In contrast to the MLP, we train the state machine on sequences because we require the current latent state q_t to result from the previous prediction rather than the previous ground truth state. This allows the state machine to train under its own state distribution and prevents degenerate solutions caused by highly temporally-correlated labels. Given the sequential nature of the training, it often happens that at a given time step, the current classification subset does not contain the correct label, resulting in an undefined loss. This issue sometimes causes instabilities in training and results in poor performance.

Soft-differentiable state machine (SDSM): To help alleviate the above issues with the crisp state machine, we propose a differentiable, fuzzy variant of the state machine. Similarly to the alternative described above, we define neural networks $\mu_{\phi_1}, \dots \mu_{\phi_{|Q|}}$ for each of state $q \in Q$. The key difference here is that at each time step, we keep a complete probability distribution over all states p_t. The probability output of the current state is calculated by taking the weighted probability of all transition functions $\mu_{\phi_{q_i}}$ as follows:

$$p_{t+1} = \sum_i p_t^i \cdot \mu_{\phi_{q_t}}(o_t) \tag{3.6}$$

This is equivalent to taking the expectation of all possible state trajectories simultaneously, giving a well-defined loss at each time step and improved performance. This is made a bit more clearly when describing the computation as matrix multiplication in equation 3.4.1, where we denote transition functions from state i to state j as $f(o_t)^{i,j}$.

$$
\begin{aligned}
p_{t+1} &= \begin{bmatrix} p_{t+1}^0 \\ \vdots \\ p_{t+1}^{m-1} \end{bmatrix} \\
&= \begin{bmatrix} p_t^0 \cdot f(o_t)^{(0,0)} & + & \cdots & + & p_t^{m-1} \cdot f(o_t)^{(m-1,0)} \\ & & \vdots & & \\ p_t^0 \cdot f(o_t)^{(0,m-1)} & + & \cdots & + & p_t^{m-1} \cdot f(o_t)^{(m-1,m-1)} \end{bmatrix} \\
&= \underbrace{\begin{bmatrix} f(o_t)^{(0,0)} & + & \cdots & + & f(o_t)^{(m-1,0)} \\ & & \vdots & & \\ f(o_t)^{(0,m-1)} & + & \cdots & + & f(o_t)^{(m-1,m-1)} \end{bmatrix}}_{T(o_t)} \cdot \begin{bmatrix} p_t^0 \\ \vdots \\ p_t^{m-1} \end{bmatrix} \\
&= T(o_t) \cdot p_t
\end{aligned}
$$

The SDSM also has the added benefit of making our architecture temporally differentiable, as illustrated in figure 3.2. An auto diff package such as Pytorch [122] allows us to keep the predicted state as Pytorch tensor objects which builds a computational graph in the background which we can then backpropagate through. In a sense, our architecture is similar to a recurrent neural network (RNN) but with a meaningful hidden state. During inference, we propagate the probability distribution the same way as during training and take the *argmax* at each timestep as the inferred state for policy π.

3.4.2 Local flipper control policy (LFC) π:

A given state q defines a local control policy π^q, which provides flipper target positions $[a^1, a^2, a^3, a^4]$. Action a^i is an overlay of three separate actions.

- **Flipper template position a^i_{temp}:** Action templates are illustrated in figure 3.1 and are chosen to be suitable for each phase of the traversal process for the tracked robot. These templates also define target torque levels, shown by color in figure 3.1.

- **Body roll stabilization:** We implement a proportional-derivative (PD) regulator that attempts to stabilize the roll of the body by acting on the appropriate (left/right) flipper group. Actions belonging to the left and right flipper groups are defined as $[a^1, a^3]$ and $[a^2, a^4]$ respectively. Stabilization actions are then calculated as

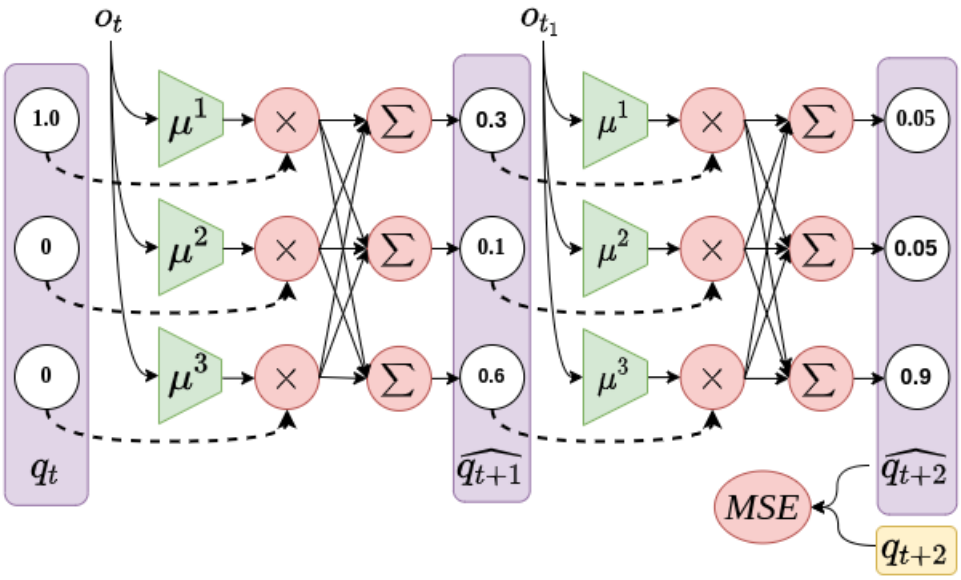

Figure 3.2: Computation graph of our proposed soft-differentiable state machine with reduced to 3 states for simplicity, computed through 2 time steps. The dashed lines show the differentiable links which are carried over to the next state.

$$\begin{aligned} a_{stab}^{1,3} &= -(\phi \cdot k_p - \dot{\phi} k_d) \\ a_{stab}^{2,4} &= (\phi \cdot k_p - \dot{\phi} k_d) \end{aligned} \tag{3.7}$$

- **Escape maneuvers:** We define a feature $st \in [0,1]$ which we denote as *stagnation*, denoting how much the robot is currently stuck. This feature is computed by comparing the estimated linear velocity and angular velocity of the robot against the target differential steering velocity given by the path follower (or operator). The individual flipper escape maneuver (em) actions a_{em}^i are state-dependent and are a linear function of the *stagnation*. For example, for the *ascending-rear* state, the rear action group modifier is calculated as

$$\begin{aligned} a_{em}^{1,2} &= -st \cdot 0.3 \\ a_{em}^{3,4} &= st \cdot 0.5 \end{aligned} \tag{3.8}$$

  The resultant action lowers the rear flippers in attempt to contact the ground more and lifts the forward flippers in case there is a conflicting obstacle during ascent, as shown in figure 3.3. It is impractical to make the template action steeper to fix this issue. The reason is that when traversing lower obstacles, the rear of the robot would rise significantly off the ground, resulting in a higher motor current consumption and a higher potential covariate shift at inference.

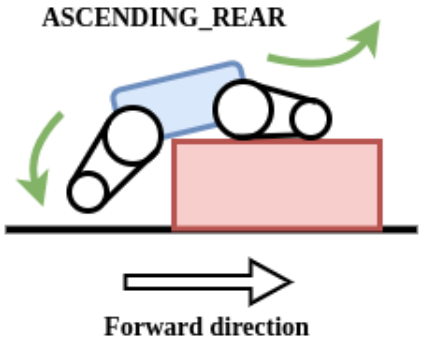

Figure 3.3: An escape maneuver in the *ascending rear* state, executed by moving the flippers in the directions shown by the green arrows. This maneuver allows for the stuck robot to regain flipper contact with the ground.

The total action a^i is calculated as a sum of the three above actions:

$$a^i = a^i_{temp} + a^i_{stab} + a^i_{em} \tag{3.9}$$

3.5 Data gathering

The key learning mechanism of our approach revolves around imitating demonstrations gathered by a human operator on a variety of obstacles using a standard gamepad and third-person visual input. Transitioning between the states on our proposed state machine structure is done by discrete button presses on the D-PAD of the gamepad. The arrows corresponding to the transitions are shown in figure 3.1. Despite the intimidating number of arrows on the diagram, the control is quite simple and intuitive. Figure 3.4 shows a typical obstacle traversal sequence. We have found that this approach leads to fewer mistakes and a lower cognitive load than other interfaces that we have tried.

To gather the training dataset, we use our training obstacle course in the OSRF gazebo simulator. The robot is spawned in the initial position of the obstacle course, and the navigation stack is initialized. The operator then drives the robot manually along a predefined path around the entire training obstacle course, similar to what is shown in figure 3.8. We vary object traversal velocities to get a wide variety of dynamics on the obstacles. We found that the velocity with which the robot traverses the obstacle has a significant impact on whether it is susceptible to getting stuck due to the low clearance of the robot.

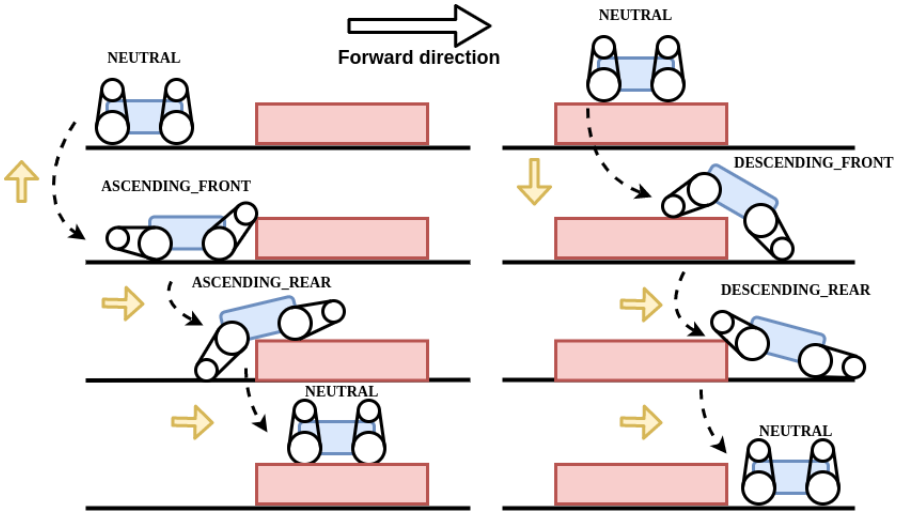

Figure 3.4: A typical traversal sequence of a positive obstacle is shown in red. The yellow arrows correspond to the respective directions on the gamepad DPAD that the operator has to press to transition between the given sequences.

3.6 Navigation stack and exteroceptive features

Platform and navigation stack

The platform is a 50 kg articulated tracked robot equipped with four independent, continuously rotating flippers, shown in figure 3.5 producing a maximum torque of roughly $120\,\mathrm{N}\cdot\mathrm{m}$.

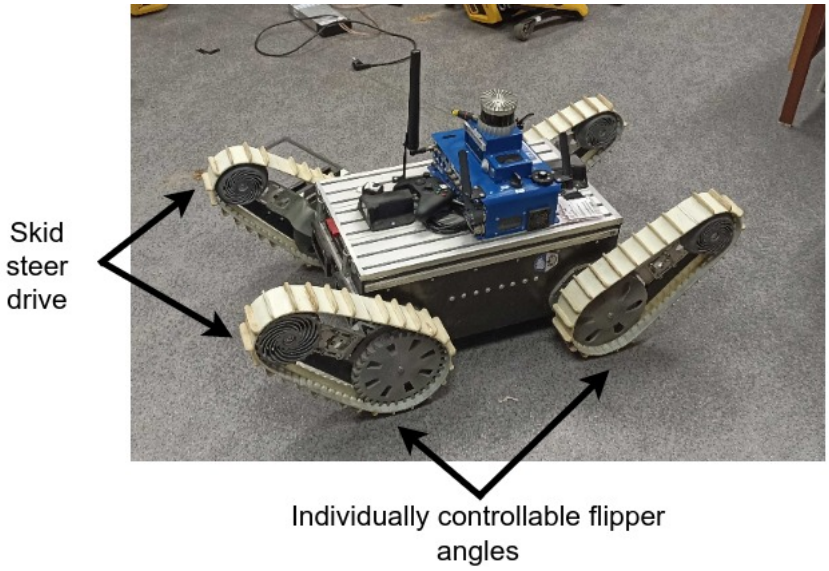

Figure 3.5: MARV (Mobile Autonomous Rescue Vehicle): Articulated tracked platform that we use in our experiments.

The flippers are equipped with a ribbed polyamide belt driven by each of the four main wheels implementing a skid-steer control design. The primary sensing device is an Ouster OS-128 lidar that has a 90° horizontal field of view consisting of 128 scans and a circular resolution

of 512 scans. Although this provides adequate visibility of the terrain around the robot, there is still a radius of roughly 60cm where the data is obstructed by the robot body, shown in figure 3.6. This means that reactive methods on LIDAR data can't be used directly, and a mapping method is required to put together the scans and preprocess it to give enough information about the terrain.

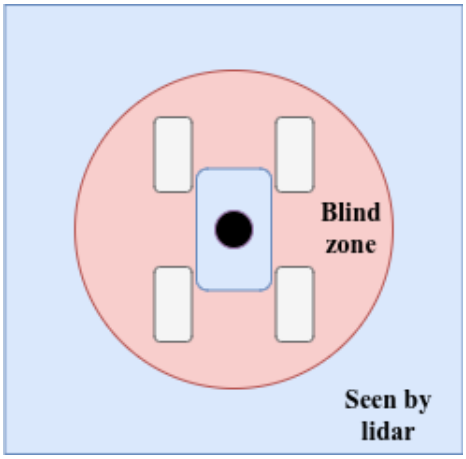

Figure 3.6: Illustration of the blind zone of the LIDAR sensor on the MARV platform.

We use an ICP-based mapping algorithm based on [123] to obtain laser scan transformations for localization. These transformations are then used by a traversability estimation [124] algorithm, which computes a dense 2.5D point cloud map. The robot navigates toward target waypoints generated either by a high-level command from the operator or by an autonomous planner in conjunction with a mission plan or exploration agent.

Exteroceptive features

Each flipper has a zero-pitch bounding box region which defines point cloud regions pc_i, as shown in figure 3.7. The feature vector for each flipper is a combination of the median height of all present points, the median of the highest 10% of points, which we denote as $pc_i^{h-10\%}$ and the lowest 10% as $pc_i^{l-10\%}$ pc_i as well as the number of points present in the bounding box divided by the maximum possible amount, giving a feature in the range of $[0,1]$. This generalizes well to noisy lidar inputs on the physical platform. The complete feature vector is shown in equation 3.10. We initially included an additional feature vector that represented some aspects of curvature of the underlying terrain for each flipper but later removed it as it made the neural network more sensitive to noise. In contrast to legged robots, an articulated tracked robot does not require detailed knowledge of the terrain below it for locomotion purposes.

$$f_i = \{med(pc_i), med(pc_i^{h-10\%}), med(pc_i^{l-10\%}), \frac{|pc_i|}{max|pc_i|}\} \tag{3.10}$$

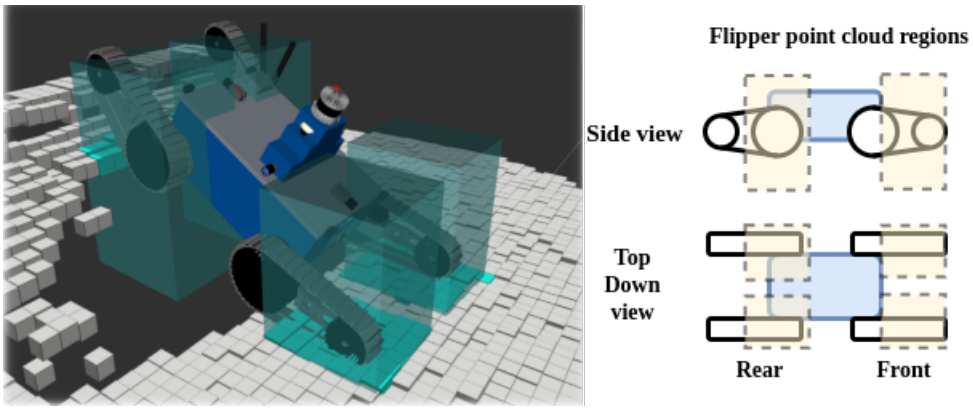

Figure 3.7: Bounding boxes showing the flipper point cloud regions. The traversability map often has large regions of missing data.

3.7 Experiments and results

3.7.1 Obstacle course traversal in simulation

We evaluate the performance of our entire system and compare our approach to a best-effort reimplementation of similar previous work [119]. The reimplementation uses our proposed flipper templates and quality demonstrations but keeps the original features, template prediction function, and a similar learning process to what is described in the paper. We add several of our improvements to [119] to compare the results and form an ablation. We, therefore, refer to [119] as the baseline state of the art (Sota) in our results table. We also compare several neural network architectures in state-transition classification, including our soft-differential state machine (SDSM). Demonstrations are available in the supplementary video. Our evaluation criteria consist of the following:

- Classification accuracy: This shows us how much a given architecture is able to fit the training data. The loss is a percentage of correctly classified data, calculated as follows:

$$classif.\,acc. = \frac{1}{|D|} \sum_{o_t, q_t \in D} \delta[q_t = \hat{q}_t] \tag{3.11}$$

- Traversal Failures: If the robot is stuck, requiring human intervention, then it counts as a failure.

- State changes: Besides traversal success, we care about the smoothness and safety of traversal. We can quantify this by counting the number of state changes that a policy makes during the traversal. More state changes mean more excessive flipper movement

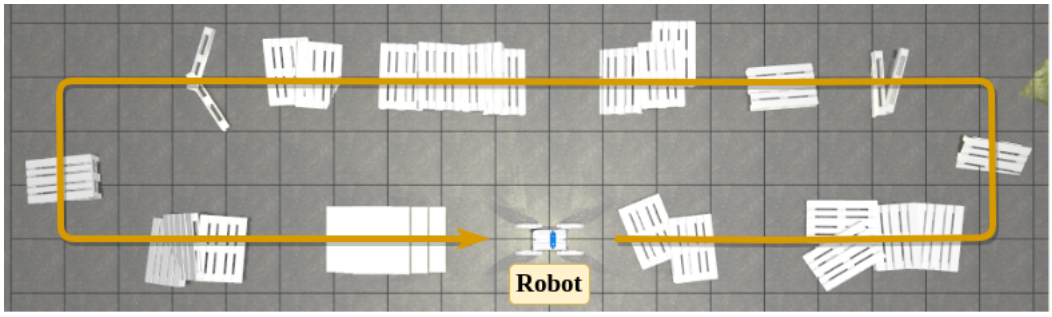

Figure 3.8: Top-down view of test obstacle course in simulation. The white pallets are stacked at various angles to form complex obstacles. The path is marked in yellow, and the grid size is 1 meter.

that makes the robot jerky and can lead to dangerous poses. Also, consistent state classification can be leveraged in path following logic for safe obstacle traversal. For this reason, we consider state consistency to be very important for real-world applications.

A single run of the experiment is performed as follows: The robot and navigation stack are initialized. The path following algorithm drives the robot at constant velocity along a predefined path around the obstacle course. If the robot gets stuck at a specific obstacle, the operator intervenes by manual assistance and the run continues.

The test obstacle course, shown in figure 3.8, consists of 12 heavily modified versions of the training obstacle course. The obstacles are compound and exhibit extreme height changes and asymmetry, and are designed to push the limits of what can be done with this approach. Some of the compound obstacles have several failure points, so we identify 16 total possible failure points per run, giving a total of 32 for two runs at different velocities, which we average over three iterations. All networks are trained on the same amount of epochs on the training course and evaluated on the test course.

Table 3.1: Results of the baseline comparison [119] and our approach on the test course in simulation averaged over two different test velocities.

Methods	Classif. acc.	Failures	State changes
[119] Orig (Sota)	0.52	24/32	-
[119] + MLP (Sota)	0.83	15/32	394
[119] + MLP + LFC (Sota)	0.83	9/32	401
HFC + MLP (Ours)	0.86	7/32	346
HFC + SM (Ours)	0.74	17/32	-
HFC + SDSM (Ours)	**0.9**	**3/32**	**230**

Effect of body roll balancing One of the low-level defined functionalities is active proportional-derivative regulation of flippers for body roll minimization. We use our learned SDSM state-transition policy and evaluate the effect of body roll stabilization on the minimum, and maximum roll experienced during the run of the test course and get the following result.

Table 3.2: Effects of body roll stabilization on the test circuit

	Roll min	Roll max
w \ stabilization (Ours) [rad]	**-0.33**	**0.14**
w \o stabilization [119] [rad]	-0.41	0.19

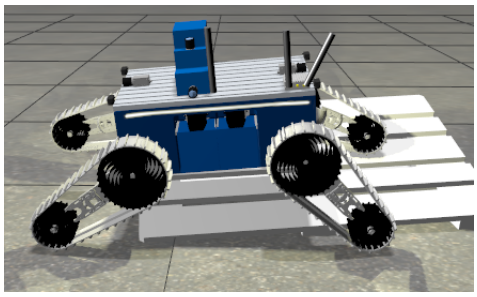

Figure 3.9: Active roll stabilization on a highly slanted obstacle. Left flippers are pushed down and vice versa.

We can see in Table 3.2 that the effect is not too large, but significant. Body roll limitation can be crucial in avoiding dangerous body roll situations where the robot might flip over. Another reason to keep a minimal body roll is better LIDAR and camera coverage in an autonomous mission.

3.7.2 Real platform experiments

Due to the current unavailability of the physical platform on which we have performed our tests in simulation, we demonstrate real-world experiments on a similar available platform. It mainly differs in having additional main tracks in the middle of the robot and higher body clearance. This means that the robot is significantly less likely to get stuck on an obstacle. Due to the non-deformability of the middle track, it requires more careful traversal to mitigate the fast swinging motion of the body, which can damage the motors. The test is performed on five different obstacles of various difficulty, including unseen, complex, and asymmetrical objects. We define two additional metrics shown in equation 3.12, which we call shock and swing, in addition to the number of state changes experienced by the transition policy as a proxy for locomotion smoothness.

$$
\begin{aligned}
shock &= \frac{maximum(|\ddot{x}| + |\ddot{y}| + |\ddot{z}| - 9.81, 0)^2}{N} \\
swing &= maximum(|\dot{\theta}| + |\dot{\phi}| + |\dot{\psi}| - 0.1, 0)^2
\end{aligned}
\tag{3.12}
$$

Table 3.3: Real platform zero-shot locomotion smoothness results averaged over three runs on five obstacles from easiest to most difficult.

Obst.	Methods	Shock	Swing	State ch.	Fail.
1	[119] + MLP	7.7	13.8	20	0/3
	HFC + MLP (Ours)	5.6	13.1	**8**	0/3
	HFC + SDSM (Ours)	**5.5**	**6.1**	9	0/3
2	[119] + MLP	2.1	3.6	12	1/3
	HFC + MLP (Ours)	**1.9**	**1.1**	10	0/3
	HFC + SDSM (Ours)	**1.9**	1.5	**7**	0/3
3	[119] + MLP	-	-	-	2/2
	HFC + MLP (Ours)	**5.1**	8.1	14	0/3
	HFC + SDSM (Ours)	6.2	**5.4**	**7**	0/3
4	[119] + MLP	2.8	**2.9**	11	0/3
	HFC + MLP (Ours)	2.3	5.0	12	0/3
	HFC + SDSM (Ours)	**1.8**	3.3	**9**	0/3
5	[119] + MLP	6.6	7.0	22	0/3
	HFC + MLP (Ours)	2.2	7.1	12	0/3
	HFC + SDSM (Ours)	**1.5**	**3.5**	**8**	0/3
Total score	[119] + MLP	24.3	35.4	79	3
	HFC + MLP (Ours)	17.1	34.4	56	0
	HFC + SDSM (Ours)	**16.9**	**19.8**	**40**	0

The original work from [119] which used a linear function approximator, as well as our crisp state machine (SM), was excluded from tests due to poor performance. Table 3.3 shows that the baseline generalizes poorly to the real platform and fails on several obstacles resulting

in robot tip-over, as can be seen in the supplementary video. Our approach leads to generally smoother and safer performance. Using our SDSM transition function leads to the fewest state changes, a negligible difference in overall translational shock or jerk experienced, but a significant improvement in swing, which can be dangerous for the robot and can result in an unstable traversal.

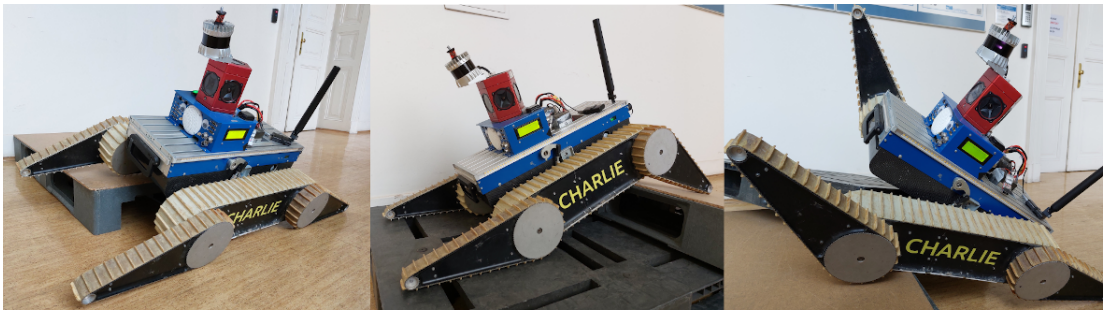

Figure 3.10: Several of the tested obstacles on real platform.

3.7.3 DARPA subterranean challenge deployment

This work came out of a necessity for reliable and smooth autonomous flipper locomotion for robotic missions in an unknown environment, such as the Urban circuit of the DARPA Subt challenge. We successfully deployed [125] a subset of the presented work in preference over previous work [119], due to its jerky and unreliable performance. We used our proposed state machine transition architecture but with if/else transition functions which were manually tuned on a very limited range of obstacles and mostly relied on depth data from a forward-facing depth camera. Our main robots were sent with confidence up and down steep staircases and traversed other obstacles in their path autonomously in order to increase map coverage and discover artifacts that lead to additional points.

3.8 Discussion and conclusion

We evaluated the practical use of our approach by demonstrating its performance within a complete navigation stack, as it would be deployed in a real world autonomous mission. In Section 3.7 we showed that our HFC architecture can reliably traverse a large variety of obstacles in simulation and real platform from only several minutes of demonstrations. Specifically, Table 3.1 shows that our HFC significantly outperforms the previous approach, even when augmented with elements from our HFC. The same table also shows that our proposed SDSM significantly outperforms the MLP decision function in the task of state classification. This is shown by having significantly fewer failures and state changes which leads to overall smoother locomotion, which is critical in autonomous exploration in unknown environments. As expected, the naively

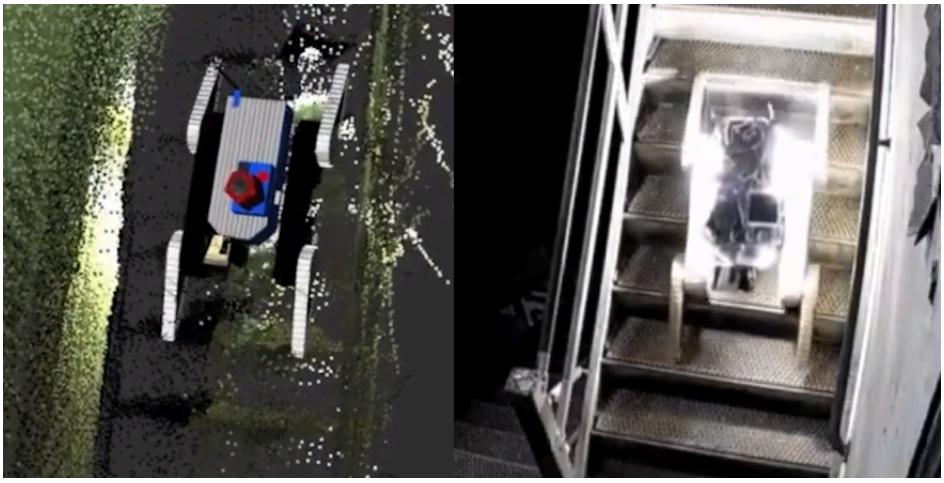

Figure 3.11: Autonomous flipper control in the DARPA subterranean challenge Urban circuit. The robot was remotely given waypoint goals on the map. The path follower algorithm controls the robot velocity, and our flipper controller sets the flippers in such a position that enables a smooth transition.

implemented SM does not perform well for reasons mentioned in Section 3.4. The results in 3.1 and 3.9 also show the effectiveness of our local flipper controller (LFC) in terms of robot roll limiting and reduction of traversal failures. We have found that allowing the temporal gradient propagation of the SDSM improves the training classification accuracy of the demonstrations from *0.87* to *0.92*, showing that it is able to leverage the temporal gradient to adjust the prior state distribution such that it provides a better decision outcome. We have also shown that our approach has excellent zero-shot generalization on a similar real platform, as shown in Table 3.3, again significantly outperforming the previous approach.

Our approach decouples flipper control with robot steering, with the latter being done by a path-following algorithm. One advantage of this choice is preserving modularity and compatibility with various path following and planning algorithms which is essential when working with a large team on a complex project. More importantly, tracked articulated robots have very poor steering characteristics when traversing an obstacle and can lead to excessive robot wear or overturning. For this reason, it is safer and less damaging to the robot to traverse an obstacle in a straight path and steer when the process is completed. Our method already reliably detects the phases of traversal, so a simple heuristic would be to wait until the robot is in the *neutral* state before allowing angular track velocities. However, we decided to leave this for future work, where we will examine the tight integration of the HFC with a planning/exploration algorithm.

Another potential issue that comes up when learning from demonstrations is the well-documented covariate shift [24]. It describes a loss in performance of the control system at test time due to a mismatch of the training and test trajectory distributions induced by the demon-

strator and control algorithm, respectively. After extensive experimentation and validation, we do not observe a significant difference between training and test performance and therefore cannot justify implementing the remedies proposed in [24]. We speculate that some of the reasons why we can circumvent this issue are the kinematic and inherently stable nature of the platform and a robust choice of exteroceptive features described in Section 3.6. In addition, the learned state transition classifier in our approach comprises only part of the control system.

One of the weaknesses of our approach is that the control decisions are only as good as the operator. A potential future work would be to jointly learn parameters in the state transition and our defined local policies using backpropagation. Our method fully supports this thanks to our soft-differentiable state machine, but would require training using Reinforcement Learning. Concerning the variability of obstacles that we are able to traverse, currently negative obstacles such as holes are an issue. These would require additional exteroceptive features and training examples. Another possible disadvantage of exteroceptive-based approaches such as ours is irregular and deformable terrain such as grass.

In conclusion, we demonstrated a complete flipper control approach for articulated tracked robots which can be learned in minutes from demonstrations and deployed in a real-life autonomous navigation scenario.

Part IV

Conclusion

3.9 Conclusion

Data-driven methods for practical applications in robotic control are still in their early stages, but there are already several cases where it rivals or has replaced more traditional optimization/planning-based control methods. We discussed some of the shortcomings of such methods, such as the high training sample complexity of trial and error optimization algorithms. We also described various sim-to-real transfer approaches that allow a trained policy to be used on a real robot. Our main contribution, however, focused on issues and the effect of various control law structures for a given robot morphology. We saw that, particularly for high-dimensional-legged robots such as hexapods, it might be more suitable to start with a heuristic approach that captures the prior of the action space (for example, cyclic motion) and replace various decision points with learnable modules. We also showed that it is difficult to train monolithic neural networks to perform adaptive locomotion using Reinforcement Learning and showed that structuring a hierarchical policy that learns to adapt locomotion gaits can be more suitable. General, low prior functions such as MLP Neural Networks also require significant reward engineering to get a practically usable result. One example of this is hexapod locomotion, where an MLP with a good performance might be walking asymmetrically, using only five of the six legs, which is unacceptable for real usage. Besides raw control performance, another metric of interest is interpretability. This is one of the negative points of using Deep Neural Networks such as MLP for control. In our work on articulated tracked robots, we took a look at how we can engineer a hybrid control function that consists of interpretable states that have learnable transition functions but hand-designed low-level controllers. We demonstrated significant improvements over previous state-of-the-art approaches and also showed that a correctly engineered feature and control space can give zero-shot generalization to the real platform. This concept fuses data-driven control with the possibility of inserting domain knowledge of system behavior. We believe that bridging first principle approaches and data-driven methods is the way to move forward because it will bring the best of both worlds. In conclusion, the fast advance of simulation environments, along with increasing computational power and specialized hardware accelerators, means that data-driven control will likely be increasingly prevalent in many robotic and control system domains. We believe that engineering and applying more suitable control policy structures is an important step in this endeavor and will lead to more mainstream use of data-driven control for robotic platforms.

3.10 Future work

There is an important computational aspect of data-driven control that is rarely addressed. Every function that maps a set of inputs to an output performs a sequence of computations on

that input. Neural Networks, for example, perform computation as a successive forward pass between layers. This works well for pattern recognition due to the compositional structure of the input space. However, we know that some tasks are inherently combinatorial and therefore require iterative computation while referencing an inner memory state. It would be interesting to experiment with policy structures that perform some form of iterative computation on a pre-computed set of input features and apply these to difficult robotic tasks. This would perhaps be more suitable for a task that has combinatorial subtasks, such as planning. An example of this would be complex foot placement for a hexapod robot or short-term navigational planning for mobile robots. There have been works that have explored learnable iterative architectures, but it has not been explored for robotics yet.

Appendix A

Additional contributions

A.0.1 Successfully supervised student Bachelor theses:

- Adaptive control using Neural Networks, Author: Švrčina Jan
- Learning Dynamic System Control on a Data Driven Model, Author: Aleksandr Barinov
- Learning a Structured Locomotion Algorithm for Hexapod Robots, Author: Jiří Hronovský

A.0.2 World robotic competition participations during PhD studies:

- Darpa Subterranean challenge Urban Round, Washington, USA (2020)
- Darpa Subterranean challenge Final Round, Kentucky, USA (2021)

Appendix B

Darpa Subt Challenge Final Round 2021

Besides research work on data-driven control methods for legged robot locomotion and articulated robots, we have also participated in the DARPA Subterranean challenge Urban and final rounds. A systems paper [125] was published by our team on the Urban round but not on the final round. As in the Urban round, a lot of work went into designing the payload computer systems and sensors. This also comprised the software integration necessary to run the system, which was mostly in the Robot Operating System (ROS), running on the Ubuntu 18 operating system. On top of the system ROS plumbing, we built a navigation stack that consists of an odometry estimator that fuses dead reckoning with a magnetometer for absolute orientation as a precursor to a LIDAR SLAM algorithm which provided the robot precise localization. The LIDAR point clouds were preprocessed and stitched into a traversability map consisting of a uniformly dense voxel grid, with each voxel informing the cost of the terrain for the given robot. Explicit or exploration goals could then be planned using an A^* planner and followed with a waypoint tracker. There were several minor modifications required for the spot robot that we will describe below.

Spot payload The Boston Dynamics Spot robot is equipped with an onboard computer that processes onboard depth cameras and sensors and computes low-level locomotion. It is a closed system that can be communicated through an API using TCP/IP. For custom autonomy, we had to design and mount an external payload that consisted of several computers and sensors. The main autonomy logic was run on an Intel NUC i9 computer. Camera object detection was done on an Nvidia Jetson Xavier, due to the requirement of CUDA GPU acceleration. Additional sensors consisted of an X-SENS IMU, 5 BASLER RGB cameras and a 360 degree 128 row OUSTER OS-1 LIDAR. The computers and sensors were organized and placed inside a cage made of aluminum extrusions and mounted with the help of 3D-printed parts. We contributed to some aspects of the design and construction of the payload. One notable part was the LIDAR holder, which was designed in Fusion 360 and printed using PETG plastic on a Prusa MK III

3D printer.

Figure B.1: Boston Dynamics Spot robot autonomy payload.

Spot assistant One part of the Spot robot that we contributed to is the high-level control logic software. The platform is more complex than a skid-steer wheeled robot and has various power modes and sitting/standing modes that we had to correctly configure before taking off. Sensor issues can cause system faults and errors that have to be managed. After falling, the platform has to also go over a sequence of error-clearing and self-righting procedures. Navigation using the Spot platform was the most complex aspect and included tasks such as detecting stairs and ramps in the path that required orienting spot to face the ascent/descent in the right direction and setting the appropriate gait through the API. Using additional traversability information from the LIDAR voxel map, we also switched to more careful gaits when necessary to prevent locomotion instability and falling.

Spot driver augmentation We added several additional monitoring and control topics, such as setting the terrain gait type, friction, and leg height parameters to the Clearpath ROS driver for the SPOT platform, available on our fork of the driver on github `https://github.com/silverjoda/spot_ros` . Additionally we made a ros interface for the inner terrain representation of the Spot platform, available on our github at `https://github.com/silverjoda/spot_ros_expansion`.

Virtual Bumper As described in the introduction of this section, the navigation of our robots are done by following waypoints that were generated using a planning algorithm in a LIDAR map. One potential issue with this approach is that errors in the traversability map and transport and computational delays can cause situations where the robot can collide with the environment or another robot, potentially causing damage. We found that a reasonable way to get around this

issue is to use an additional fast, low-level loop that uses raw LIDAR data in the robot frame to detect objects and prevent a collision. This is done by defining bounding boxes around the robot and using thresholding to detect if there is a foreign object in a specific bounding box. An illustration of this is shown in a screenshot from the RVIZ visualization tool in figure B.2.

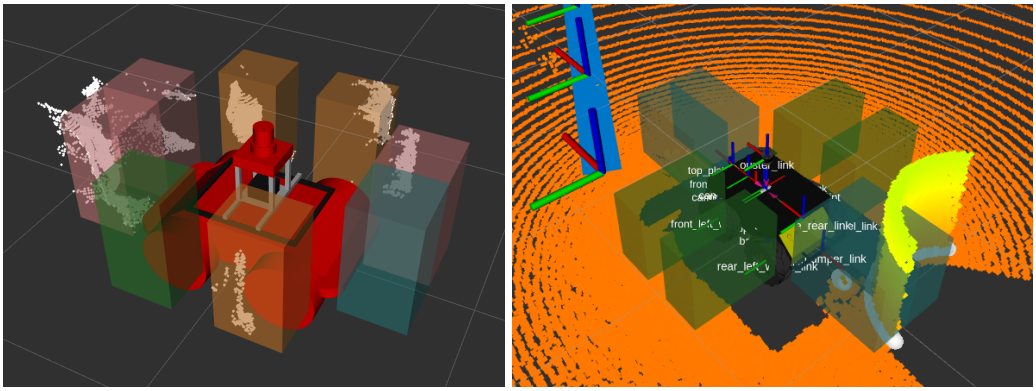

Figure B.2: Virtual bumper for the Clearpath Husky robot. The various colors of the boxes indicate the level of activation by the point cloud, shown in white.

Instead of stopping the robot when detecting that a collision is imminent, we instead attempt to steer the robot away reactively so that it can keep going. This is done by modifying the forward and angular target velocities taken from the waypoint follower. Algorithm 0 summarizes the logic that we use. It can be inserted into the navigation stack and enables the robot to seemlessly avoid obstacles and walls and even navigate around them to a certain extent. We call this our Virtual Bumper (VB). We use the VB on most of our skid steer platforms, and we deployed them in the DARPA challenge.

Algorithm 4 Part of logic for calculating velocity from binary bounding box activations

Initialize Target linear and angular velocities v_x, v_z and boolean bounding box activations $B = f_l, f_r, sf_l, sf_r, sr_l, sr_r$
procedure CALC COMMAND VELOCITY (v_x, v_z, B)
    if $v_x > 0$ and $f_l = True$ **then** ▷ Front partially blocked
        $v_x^{out} \leftarrow 0$
        $v_z^{out} \leftarrow -max(target_lin, abs(target_ang))$
    if $v_x > 0$ and $f_r = True$ **then** ▷ Front partially blocked
        $v_x^{out} \leftarrow 0$
        $v_z^{out} \leftarrow max(target_lin, abs(target_ang))$
    if $sf_l = True$ **then** ▷ Sides partially blocked
        $v_z^{out} \leftarrow min(v_z^{out}, 0)$
    if $sf_r = True$ **then** ▷ Sides partially blocked
        $v_z^{out} \leftarrow max(v_z^{out}, 0)$

Appendix C

Author Publications response

- T. Azayev and K. Zimmerman, "Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification", *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3, pp. 659–671, 2020
  - S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments", *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021
  - W. Ouyang, H. Chi, J. Pang, *et al.*, "Adaptive locomotion control of a hexapod robot via bio-inspired learning", *Frontiers in Neurorobotics*, vol. 15, p. 627 157, 2021
  - P. Manoonpong, L. Patanè, X. Xiong, *et al.*, "Insect-inspired robots: Bridging biological and artificial systems", *Sensors*, vol. 21, no. 22, p. 7609, 2021
  - G. Zhang, Y. Du, Y. Zhang, *et al.*, "A tactile sensing foot for single robot leg stabilization", in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 14 076–14 082
  - M. Schilling, J. Paskarbeit, H. Ritter, *et al.*, "From adaptive locomotion to predictive action selection–cognitive control for a six-legged walker", *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 666–682, 2021
  - P.-H. Kuo, S.-T. Lin, J. Hu, *et al.*, "Multi-sensor context-aware based chatbot model: An application of humanoid companion robot", *Sensors*, vol. 21, no. 15, p. 5132, 2021
  - F. Zhou and J. Vanschoren, "Open-ended learning strategies for learning complex locomotion skills", *arXiv preprint arXiv:2206.06796*, 2022
  - M. Thor and P. Manoonpong, "Versatile modular neural locomotion control with fast learning", *Nature Machine Intelligence*, vol. 4, no. 2, pp. 169–179, 2022
  - J. Homchanthanakul and P. Manoonpong, "Continuous online adaptation of bioinspired adaptive neuroendocrine control for autonomous walking robots", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 1833–1845, 2021
  - M. Schilling, A. Melnik, F. W. Ohl, *et al.*, "Decentralized control and local information for robust and adaptive decentralized deep reinforcement learning", *Neural Networks*, vol. 144, pp. 699–725, 2021
  - S. Yang, "Design and typical gait realization of a wheeled-foot integrated hexapod robot", in *2021 IEEE 4th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, IEEE, 2021, pp. 715–718

  - H. Hu and Y. Liu, "Blind adaptive gait planning on non-stationary environments via continual reinforcement learning", in *2021 IEEE International Conference on Unmanned Systems (ICUS)*, IEEE, 2021, pp. 280–284
  - P. Moazzeni Bikani *et al.*, "Robust proximal policy optimization for reinforcement learning", 2022
  - M. Li, Z. Wang, D. Zhang, *et al.*, "Accurate perception and representation of rough terrain for a hexapod robot by analysing foot locomotion", *Measurement*, vol. 193, p. 110 904, 2022
  - F. Zhou, "A study of an open-ended strategy for learning complex locomotion skills",
  - W Amri, L Hermes, and M Schilling, "Hierarchical decentralized deep reinforcement learning architecture for a simulated four-legged agent", *arXiv preprint arXiv:2210.08003*, 2022

- T. Roucek, M. Pecka, P. Cízek, *et al.*, "System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA subterranean challenge", *CoRR*, vol. abs/2110.05911, 2021. arXiv: `2110.05911`. [Online]. Available: `https://arxiv.org/abs/2110.05911`
  - A. Koval, S. Karlsson, S. S. Mansouri, *et al.*, "Dataset collection from a subt environment", *Robotics and Autonomous Systems*, vol. 155, p. 104 168, 2022
  - M. Kaufmann, R. Trybula, R. Stonebraker, *et al.*, "Copiloting autonomous multi-robot missions: A game-inspired supervisory control interface", *arXiv preprint arXiv:2204.0664* 2022
  - T. Yang, Y. Li, C. Zhao, *et al.*, "3d tof lidar in mobile robotics: A review", *arXiv preprint arXiv:2202.11025*, 2022
  - B. Zhou, H. Xu, and S. Shen, "Racer: Rapid collaborative exploration with a decentralized multi-uav system", *arXiv preprint arXiv:2209.08533*, 2022
  - V. Šalanskỳ, "Robot learning and perception in sensory deprived environment",

Bibliography

[1] Z. Liu, Q. Liu, W. Xu, L. Wang, and Z. Zhou, "Robot learning towards smart robotic manufacturing: A review", *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102 360, 2022, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2022.102360. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584522000485.

[2] D. Lattanzi and G. R. Miller, "Review of robotic infrastructure inspection systems", *Journal of Infrastructure Systems*, vol. 23, p. 04 017 004, 2017.

[3] S. Srinivas, S. Ramachandiran, and S. Rajendran, "Autonomous robot-driven deliveries: A review of recent developments and future directions", *Transportation Research Part E: Logistics and Transportation Review*, 2022.

[4] A. Mohebbi, "Human-robot interaction in rehabilitation and assistance: A review", 2020.

[5] B. S. Peters, P. R. Armijo, C. M. Krause, S. A. Choudhury, and D. Oleynikov, "Review of emerging surgical robotic technology", *Surgical Endoscopy*, vol. 32, pp. 1636–1655, 2018.

[6] Y. Tian, K. Liu, K. Ok, *et al.*, "Search and rescue under the forest canopy using multiple uavs", *The International Journal of Robotics Research*, vol. 39, pp. 1201 –1221, 2020.

[7] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments", *IEEE Robotics and Automation Letters*, vol. 4, pp. 610–617, 2019.

[8] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, "Machine learning–accelerated computational fluid dynamics", *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, e2101784118, 2021. DOI: 10.1073/pnas.2101784118. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2101784118. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

[9] N. M. O. Heess, T. Dhruva, S. Sriram, *et al.*, "Emergence of locomotion behaviours in rich environments", *ArXiv*, vol. abs/1707.02286, 2017.

[10] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain", *Science Robotics*, vol. 5, 2020.

[11] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds., MIT Press, 2000, pp. 1057–1063. [Online]. Available: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf.

[12] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning", in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/7634ea65a4e6d9041cfd3f7de18Paper.pdf.

[13] N. Hansen, "The cma evolution strategy: A tutorial", 2016, cite arxiv:1604.00772Comment: ArXiv e-prints, arXiv:1604.xxxxx. [Online]. Available: http://arxiv.org/abs/1604.00772.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[15] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition", *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 945–953, 2015.

[16] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2222–2232, 2017.

[17] A. van den Oord, S. Dieleman, H. Zen, *et al.*, "Wavenet: A generative model for raw audio", *ArXiv*, vol. abs/1609.03499, 2016.

[18] A. Vaswani, N. M. Shazeer, N. Parmar, *et al.*, "Attention is all you need", *ArXiv*, vol. abs/1706.03762, 2017.

[19] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks", in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=S1sqHMZCb.

[20] M. Srouji, J. Zhang, and R. Salakhutdinov, "Structured control nets for deep reinforcement learning", in *International Conference on Machine Learning*, PMLR, 2018, pp. 4742–4751.

[21] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, "Soft actor-critic algorithms and applications", *ArXiv*, vol. abs/1812.05905, 2018.

[22] A. Nagabandi, I. Clavera, S. Liu, *et al.*, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning", *arXiv: Learning*, 2019.

[23] Y. Wang, Q. Yao, J. T.-Y. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning", *arXiv: Learning*, 2019.

[24] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction", in *In AISTATS*, 2011.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[26] N. Hansen, "The cma evolution strategy: A tutorial", *arXiv preprint arXiv:1604.00772*, 2016.

[27] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods", *ACM Comput. Surv.*, vol. 50, no. 2, 2017, ISSN: 0360-0300. DOI: 10.1145/3054912. [Online]. Available: https://doi.org/10.1145/3054912.

[28] T. Azayev and K. Zimmerman, "Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification", *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3, pp. 659–671, 2020.

[29] T. Azayev and K. Zimmermann, "Autonomous state-based flipper control for articulated tracked robots in urban environments", *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–8, 2022.

[30] T. Roucek, M. Pecka, P. Cízek, *et al.*, "System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA subterranean challenge", *CoRR*, vol. abs/2110.05911, 2021. arXiv: 2110.05911. [Online]. Available: https://arxiv.org/abs/2110.05911.

[31] B. Jumet, M. D. Bell, V. Sanchez, and D. J. Preston, "A data-driven review of soft robotics", *Advanced Intelligent Systems*, vol. 4, no. 4, p. 2 100 163, 2022. DOI: https://doi.org/10.1002/aisy.202100163. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202100163. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202100163.

[32] M. Vespignani, J. M. Friesen, V. SunSpiral, and J. Bruce, "Design of superball v2, a compliant tensegrity robot for absorbing large impacts", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2865–2871. DOI: 10.1109/IROS.2018.8594374.

[33] T. Azayev and K. Zimmermann, "Autonomous state-based flipper control for articulated tracked robots in urban environments", *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7794–7801, 2022. DOI: 10.1109/LRA.2022.3185762.

[34] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems", *ArXiv*, vol. abs/2005.01643, 2020.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning", *CoRR*, vol. abs/1509.02971, 2016.

[36] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning", *ArXiv*, vol. abs/1710.11248, 2018.

[37] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning", *arXiv preprint arXiv:1803.07055*, 2018.

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

[39] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning", in *ICML*, 2016.

[40] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods", *ArXiv*, vol. abs/1802.09477, 2018.

[41] N. P. Jouppi, C. Young, N. Patil, *et al.*, "In-datacenter performance analysis of a tensor processing unit", *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[42] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. [Online]. Available: https://github.com/google-research/tiny-differentiable-simulator.

[43] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, *Brax - a differentiable physics engine for large scale rigid body simulation*, version 0.0.15, 2021. [Online]. Available: http://github.com/google/brax.

[44] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning", in *CoRL*, 2018.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation", 1986.

[46] K. Cho, B. van Merrienboer, Çaglar Gülçehre, *et al.*, "Learning phrase representations using rnn encoder–decoder for statistical machine translation", in *EMNLP*, 2014.

[47] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *CoRR*, vol. abs/1409.0473, 2015.

[48] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[49] P. Dayan and G. E. Hinton, "Feudal reinforcement learning", in *NIPS*, 1992.

[50] M. Stolle and D. Precup, "Learning options in reinforcement learning", in *SARA*, 2002.

[51] H. Shi, J. Li, J. Mao, and K.-S. Hwang, "Lateral transfer learning for multiagent reinforcement learning.", *IEEE transactions on cybernetics*, vol. PP, 2021.

[52] O. Andrychowicz, B. Baker, M. Chociej, *et al.*, "Learning dexterous in-hand manipulation", *The International Journal of Robotics Research*, vol. 39, p. 027 836 491 988 744, Nov. 2019. DOI: 10.1177/0278364919887447.

[53] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "Rl-cyclegan: Reinforcement learning aware simulation-to-real", *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 154–11 163, 2020.

[54] S. James, P. Wohlhart, M. Kalakrishnan, *et al.*, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[55] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[56] Y. Chebotar, A. Handa, V. Makoviychuk, *et al.*, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience", *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.

[57] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid Motor Adaptation for Legged Robots", in *Proceedings of Robotics: Science and Systems*, Virtual, 2021. DOI: 10.15607/RSS.2021.XVII.011.

[58] P. F. Christiano, Z. Shah, I. Mordatch, *et al.*, "Transfer from simulation to real world through learning deep inverse dynamics model", *ArXiv*, vol. abs/1610.03518, 2016.

[59] M. Andrychowicz, M. Denil, S. Gomez, *et al.*, "Learning to learn by gradient descent by gradient descent", *Advances in neural information processing systems*, vol. 29, 2016.

[60] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks", in *International conference on machine learning*, PMLR, 2017, pp. 1126–1135.

[61] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search", *arXiv preprint arXiv:1711.00436*, 2017.

[62] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies", *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[63] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, *et al.*, "Learning to reinforcement learn", *arXiv preprint arXiv:1611.05763*, 2016.

[64] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner", in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1DmUzWAW.

[65] M. Hutter, "A theory of universal artificial intelligence based on algorithmic complexity", *CoRR*, vol. cs.AI/0004001, 2000. [Online]. Available: https://arxiv.org/abs/cs/0004001.

[66] ——, *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Berlin: Springer, 2005, 300 pages, ISBN: 3-540-22139-5. DOI: 10.1007/b138233. [Online]. Available: http://www.hutter1.net/ai/uaibook.htm.

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

[68] G. Kruijff, I. Kruijff-Korbayová, S. Keshavdas, *et al.*, "Designing, developing, and deploying systems to support human–robot teams in disaster response", *Advanced Robotics*, vol. 28, no. 23, pp. 1547–1570, 2014. DOI: 10.1080/01691864.2014.985335. eprint: https://doi.org/10.1080/01691864.2014.985335. [Online]. Available: https://doi.org/10.1080/01691864.2014.985335.

[69] U. Saranli, "Rhex: A simple and highly mobile hexapod robot", *The International Journal of Robotics Research*, vol. 20, pp. 616–631, Jul. 2001. DOI: 10.1177/02783640122067570.

[70] M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda, "Controlling robot morphology from incomplete measurements", *CoRR*, vol. abs/1612.02739, 2016. arXiv: 1612.02739. [Online]. Available: http://arxiv.org/abs/1612.02739.

[71] M. Bjelonic, N. Kottege, T. Homberger, P. Borges, P. Beckerle, and M. Chli, "Weaver: Hexapod robot for autonomous navigation on unstructured terrain", *Journal of Field Robotics*, vol. 35, pp. 1063–1079, Jun. 2018. DOI: 10.1002/rob.21795.

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

[73] M. Hutter, C. Gehring, A. Lauber, *et al.*, "Anymal - toward legged robots for harsh environments", *Advanced Robotics*, vol. 31, no. 17, pp. 918–931, 2017. DOI: 10.1080/01691864.2017.1378591. eprint: https://doi.org/10.1080/01691864.2017.1378591. [Online]. Available: https://doi.org/10.1080/01691864.2017.1378591.

[74] *Boston dynamics, spot*, https://www.bostondynamics.com/spot, Accessed: 16-10-2019.

[75] *Trossen robotics*, https://www.trossenrobotics.com/, Accessed: 22-05-2010.

[76] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review", *Neural networks : the official journal of the International Neural Network Society*, vol. 21 4, pp. 642–53, 2008.

[77] P. Manoonpong, U. Parlitz, and F. Wörgötter, "Neural control and adaptive neural forward models for insect-like, energy-efficient, and adaptable locomotion of walking machines", *Frontiers in neural circuits*, vol. 7, p. 12, Feb. 2013. DOI: 10.3389/fncir.2013.00012.

[78] P. Čížek and J. Faigl, "On locomotion control using position feedback only in traversing rough terrains with hexapod crawling robot", *IOP Conference Series: Materials Science and Engineering*, vol. 428, p. 012 065, Oct. 2018. DOI: 10.1088/1757-899X/428/1/012065.

[79] Y. Isvara, S. Rachmatullah, K. Mutijarsa, D. E. Prabakti, and W. Pragitatama, "Terrain adaptation gait algorithm in a hexapod walking robot", in *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, 2014, pp. 1735–1739. DOI: 10.1109/ICARCV.2014.7064578.

[80] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning", *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*, vol. 35, no. 4, 2016.

[81] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning", *CoRR*, vol. abs/1609.05521, 2016. arXiv: 1609.05521. [Online]. Available: http://arxiv.org/abs/1609.05521.

[82] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks", *CoRR*, vol. abs/1303.5778, 2013. arXiv: 1303.5778. [Online]. Available: http://arxiv.org/abs/1303.5778.

[83] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137.

[84] K. Perlin, "Improving noise", *ACM Trans. Graph.*, vol. 21, no. 3, pp. 681–682, Jul. 2002, ISSN: 0730-0301. DOI: 10.1145/566654.566636. [Online]. Available: http://doi.acm.org/10.1145/566654.566636.

[85] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world", *CoRR*, vol. abs/1703.06907, 2017. arXiv: 1703.06907. [Online]. Available: http://arxiv.org/abs/1703.06907.

[86] OpenAI, M. Andrychowicz, B. Baker, *et al.*, "Learning dexterous in-hand manipulation", *CoRR*, vol. abs/1808.00177, 2018. arXiv: 1808.00177. [Online]. Available: http://arxiv.org/abs/1808.00177.

[87] W. Yu, G. Turk, and C. K. Liu, "Learning symmetry and low-energy locomotion", *CoRR*, vol. abs/1801.08093, 2018. arXiv: 1801.08093. [Online]. Available: http://arxiv.org/abs/1801.08093.

[88] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction", *CoRR*, vol. abs/1011.0686, 2010. arXiv: 1011.0686. [Online]. Available: http://arxiv.org/abs/1011.0686.

[89] C. Wu, "Towards linear-time incremental structure from motion", in *Proceedings of the 2013 International Conference on 3D Vision*, ser. 3DV '13, Washington, DC, USA: IEEE Computer Society, 2013, pp. 127–134, ISBN: 978-0-7695-5067-1. DOI: 10.1109/3DV.2013.25. [Online]. Available: http://dx.doi.org/10.1109/3DV.2013.25.

[90] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment", in *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR*, IEEE, 2011, pp. 3057–3064.

[91] P. Cignoni, M. Corsini, and G. Ranzuglia, "Meshlab: An open-source 3d mesh processing system.", *ERCIM News*, vol. 2008, no. 73, 2008. [Online]. Available: http://dblp.uni-trier.de/db/journals/ercim/ercim2008.html/CignoniCR08.

[92] O. Jamriska, *Ebsynth: Fast example-based image synthesis and style transfer*, https://github.com/jamriska/ebsynth, 2018.

[93] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, *et al.*, "Graph networks as learnable physics engines for inference and control", *CoRR*, vol. abs/1806.01242, 2018. arXiv: 1806.01242. [Online]. Available: http://arxiv.org/abs/1806.01242.

[94] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies", *Journal of Machine Learning Research*, vol. 15, pp. 949–980, 2014. [Online]. Available: http://jmlr.org/papers/v15/wierstra14a.html.

[95] *Bitter lesson, rich sutton*, http://www.incompleteideas.net/IncIdeas/BitterLesson.html, Accessed: 2019-04-18.

[96] M. Jaderberg, V. Mnih, W. M. Czarnecki, *et al.*, "Reinforcement learning with unsupervised auxiliary tasks", *ArXiv*, vol. abs/1611.05397, 2017.

[97] D. J. Todd, *Walking machines: an introduction to legged robots*. Springer Science & Business Media, 2013.

[98] A. A. Karim, T. Gaudin, A. Meyer, A. Buendia, and S. Bouakaz, "Procedural locomotion of multilegged characters in dynamic environments", *Computer Animation and Virtual Worlds*, vol. 24, no. 1, pp. 3–15, 2013, ISSN: 15464261.

[99] A. S. Lele, Y. Fang, J. Ting, and A. Raychowdhury, "Online reward-based training of spiking central pattern generator for hexapod locomotion", in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 208–209.

[100] L. Minati, M. Frasca, N. Yoshimura, and Y. Koike, "Versatile locomotion control of a hexapod robot using a hierarchical network of nonlinear oscillator circuits", *IEEE Access*, vol. 6, pp. 8042–8065, 2018.

[101] R. Campos, V. Matos, and C. Santos, "Hexapod locomotion, A nonlinear dynamical systems approach", in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, Glendale, AZ, USA: IEEE, 2010, pp. 1546–1551, ISBN: 978-1-4244-5225-5.

[102] P. Cizek, D. Masri, and J. Faigl, "Foothold placement planning with a hexapod crawling robot", in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada: IEEE, 2017, pp. 4096–4101, ISBN: 978-1-5386-2682-5.

[103] M. Bjelonic, N. Kottege, and P. Beckerle, "Proprioceptive control of an over-actuated hexapod robot in unstructured terrain", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 2042–2049.

[104] M. Schilling, K. Konen, F. W. Ohl, and T. Korthals, "Decentralized deep reinforcement learning for a distributed and adaptive locomotion controller of a hexapod robot", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5335–5342.

[105] M. Zangrandi, S. Arrigoni, and F. Braghin, "Control of a hexapod robot considering terrain interaction", *CoRR*, vol. abs/2112.10206, 2021. arXiv: 2112.10206.

[106] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks", *CoRR*, vol. abs/1604.07269, 2016. arXiv: 1604.07269.

[107] J. Vice, G. Sukthankar, and P. K. Douglas, "Leveraging evolutionary algorithms for feasible hexapod locomotion across uneven terrain", *arXiv preprint arXiv:2203.15948*, 2022.

[108] D. Belter and P. Skrzypczyński, "A biologically inspired approach to feasible gait learning for a hexapod robot", 2010.

[109] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies", *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[110] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning", *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[111] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning", *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

[112] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[113] M. Pecka, K. Zimmermann, and T. Svoboda, "Fast simulation of vehicles with non-deformable tracks", in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 6414–6419.

[114] K. Nagatani, A. Yamasaki, K. Yoshida, T. Yoshida, and E. Koyanagi, "Semi-autonomous traversal on uneven terrain for a tracked vehicle using autonomous control of active flippers", Oct. 2008, pp. 2667 –2672. DOI: 10.1109/IROS.2008.4650643.

[115] A. Mitriakov, P. Papadakis, J. Kerdreux, and S. Garlatti, "Reinforcement learning based, staircase negotiation learning: Simulation and transfer to reality for articulated tracked robots", *IEEE Robotics Automation Magazine*, vol. 28, no. 4, pp. 10–20, 2021. DOI: 10.1109/MRA.2021.3114105.

[116] M. Pecka, V. Šalanský, K. Zimmermann, and T. Svoboda, "Autonomous flipper control with safety constraints", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2889–2894. DOI: 10.1109/IROS.2016.7759447.

[117] A. Mitriakov, P. Papadakis, S. M. Nguyen, and S. Garlatti, "Staircase traversal via reinforcement learning for active reconfiguration of assistive robots", Jul. 2020. DOI: 10.1109/FUZZ48607.2020.9177581.

[118] Y. Yuan, Q. Xu, and S. Schwertfeger, "Configuration-space flipper planning on 3d terrain", in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 318–325. DOI: 10.1109/SSRR50563.2020.9292598.

[119] K. Zimmermann, P. Zuzanek, M. Reinstein, and V. Hlavac, "Adaptive traversability of unknown complex terrain with obstacles for mobile robots", in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5177–5182. DOI: 10.1109/ICRA.2014.6907619.

[120] A. Giusti, J. Guzzi, D. C. Cireşan, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots", *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016. DOI: 10.1109/LRA.2015.2509024.

[121] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to end learning for self-driving cars", *arXiv preprint arXiv:1604.07316*, 2016.

[122] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[123] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP Variants on Real-World Data Sets", *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, Feb. 2013.

[124] J. Bayer and J. Faigl, "Speeded up elevation map for exploration of large-scale subterranean environments", in *Modelling and Simulation for Autonomous Systems*, Springer International Publishing, 2020, pp. 190–202. DOI: 10.1007/978-3-030-43890-6_15.

[125] T. Rouček, M. Pecka, P. Čížek, *et al.*, "System for multi-robotic exploration of underground environments ctu-cras-norlab in the darpa subterranean challenge", *arXiv preprint arXiv:2110.05911*, 2021.

[126] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments", *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[127] W. Ouyang, H. Chi, J. Pang, W. Liang, and Q. Ren, "Adaptive locomotion control of a hexapod robot via bio-inspired learning", *Frontiers in Neurorobotics*, vol. 15, p. 627 157, 2021.

[128] P. Manoonpong, L. Patanè, X. Xiong, *et al.*, "Insect-inspired robots: Bridging biological and artificial systems", *Sensors*, vol. 21, no. 22, p. 7609, 2021.

[129] G. Zhang, Y. Du, Y. Zhang, and M. Y. Wang, "A tactile sensing foot for single robot leg stabilization", in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 14 076–14 082.

[130] M. Schilling, J. Paskarbeit, H. Ritter, A. Schneider, and H. Cruse, "From adaptive locomotion to predictive action selection–cognitive control for a six-legged walker", *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 666–682, 2021.

[131] P.-H. Kuo, S.-T. Lin, J. Hu, and C.-J. Huang, "Multi-sensor context-aware based chatbot model: An application of humanoid companion robot", *Sensors*, vol. 21, no. 15, p. 5132, 2021.

[132] F. Zhou and J. Vanschoren, "Open-ended learning strategies for learning complex locomotion skills", *arXiv preprint arXiv:2206.06796*, 2022.

[133] M. Thor and P. Manoonpong, "Versatile modular neural locomotion control with fast learning", *Nature Machine Intelligence*, vol. 4, no. 2, pp. 169–179, 2022.

[134] J. Homchanthanakul and P. Manoonpong, "Continuous online adaptation of bioinspired adaptive neuroendocrine control for autonomous walking robots", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 1833–1845, 2021.

[135] M. Schilling, A. Melnik, F. W. Ohl, H. J. Ritter, and B. Hammer, "Decentralized control and local information for robust and adaptive decentralized deep reinforcement learning", *Neural Networks*, vol. 144, pp. 699–725, 2021.

[136] S. Yang, "Design and typical gait realization of a wheeled-foot integrated hexapod robot", in *2021 IEEE 4th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, IEEE, 2021, pp. 715–718.

[137] H. Hu and Y. Liu, "Blind adaptive gait planning on non-stationary environments via continual reinforcement learning", in *2021 IEEE International Conference on Unmanned Systems (ICUS)*, IEEE, 2021, pp. 280–284.

[138] P. Moazzeni Bikani *et al.*, "Robust proximal policy optimization for reinforcement learning", 2022.

[139] M. Li, Z. Wang, D. Zhang, X. Jiao, J. Wang, and M. Zhang, "Accurate perception and representation of rough terrain for a hexapod robot by analysing foot locomotion", *Measurement*, vol. 193, p. 110 904, 2022.

[140] F. Zhou, "A study of an open-ended strategy for learning complex locomotion skills",

[141] W Amri, L Hermes, and M Schilling, "Hierarchical decentralized deep reinforcement learning architecture for a simulated four-legged agent", *arXiv preprint arXiv:2210.08003*, 2022.

[142] A. Koval, S. Karlsson, S. S. Mansouri, *et al.*, "Dataset collection from a subt environment", *Robotics and Autonomous Systems*, vol. 155, p. 104 168, 2022.

[143] M. Kaufmann, R. Trybula, R. Stonebraker, *et al.*, "Copiloting autonomous multi-robot missions: A game-inspired supervisory control interface", *arXiv preprint arXiv:2204.06647*, 2022.

[144] T. Yang, Y. Li, C. Zhao, *et al.*, "3d tof lidar in mobile robotics: A review", *arXiv preprint arXiv:2202.11025*, 2022.

[145] B. Zhou, H. Xu, and S. Shen, "Racer: Rapid collaborative exploration with a decentralized multi-uav system", *arXiv preprint arXiv:2209.08533*, 2022.

[146] V. Šalanskỳ, "Robot learning and perception in sensory deprived environment",