Journal of Intelligent & Robotic Systems (2020) 99:659–671
https://doi.org/10.1007/s10846-020-01162-8

Blind Hexapod Locomotion in Complex Terrain with Gait Adaptation Using Deep Reinforcement Learning and Classification

Teymur Azayev[1] · Karel Zimmerman[2]

Received: 23 May 2019 / Accepted: 28 January 2020 / Published online: 19 March 2020
© Springer Nature B.V. 2020

Abstract
We present a scalable two-level architecture for Hexapod locomotion through complex terrain without the use of exteroceptive sensors. Our approach assumes that the target complex terrain can be modeled by *N* discrete terrain distributions which 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. We perform several tests to assess policy robustness by changing various parameters, such as contact, friction and actuator properties. We also show experiments of goal-based positional control of such a system and a way of selecting several 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 [1] physics simulator. Demonstrations are available in the supplementary video.

Keywords Deep · reinforcement · learning · locomotion · hexapod · neural

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 [2], legged flipper systems [3] and tensegrity structures [4] each has 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 the 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 (Fig. 1).

Locomotion on uneven terrain requires the ability to perceive the environment. This can be done using exteroceptive sensors such as LIDAR [5] or depth camera [6]. 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 on-board IMU and joint angle data.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10846-020-01162-8) contains supplementary material, which is available to authorized users.

✉ Teymur Azayev
azayetey@fel.cvut.cz

Karel Zimmerman
zimmerk@fel.cvut.cz

[1] CVUT-FEL, E227, Karlovo nam. 13, Praha 2, Prague, Czechia
[2] CVUT-FEL, E226, Karlovo nam. 13, Praha 2, Prague, Czechia

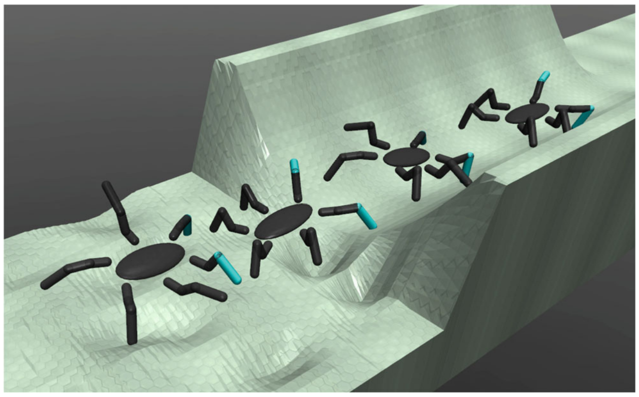

Fig. 1 Hexapod locomotion in simulation. Front legs are distinguished by the blue color. Supplementary video material available at https://youtu.be/OXAJ0jmdCZ0, Github: https://github.com/silverjoda/nexabots

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, then 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.

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 [7]. Quadruped platforms are probably the most popular and are used by several groups to research locomotion and other tasks [7–9]. 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 [10] in the MuJoCo simulator [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) [11, 12]. 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 [13] or a button sensor at the tip of the leg [14]. 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) [15] algorithm. This algorithm efficiently estimates the policy gradient [16] 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 [17] 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 [18], 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 [19] that are models that have achieved very high performance on various time-series tasks [20–22]. When using the policy gradient algorithm on RNNs, it is modified to update on batches of episodes instead of batches of states.

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.

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.

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 becausewe 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.

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 $s \in \mathcal{S}$ 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 $o_t \in \mathcal{O}$. The agent performs an action $a_t \in \mathcal{A}$, after which the environment advances by a single time step before returning the new observation vector $o_t \in \mathcal{O}$. Each state-action transition is evaluated by reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. 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 $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ 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 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.

We assume that for a given terrain type t, the task is fully observable and there exists a near-optimal *expert* policy $\pi^e_{\theta_t}(s) : \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{A}$ 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*

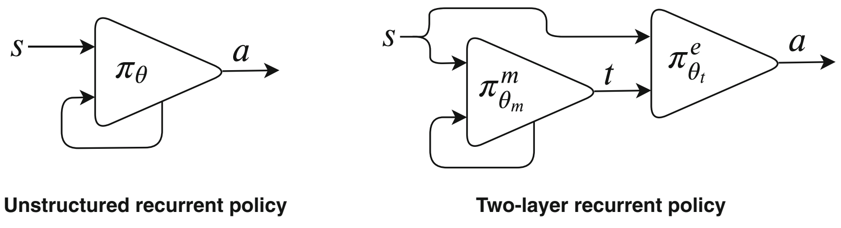

Fig. 2 Policy structure: Left general recurrent policy π_θ, Right proposed two-level structure for terrain locomotion consisting of a recurrent multiplexer policy $\pi^m_{\theta_m}$ and a set of reactive expert policies $\pi^e_{\theta_t}$

recurrent policy $\pi^m_{\theta_m}$, which switches between terrain-expert policies $\pi^e_{\theta_t}$.

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 $t \in \mathcal{T}$ by learning parameter vectors θ_t using reinforcement learning for each terrain type t independently.

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

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 betweenthe predictions and terrain labels.

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

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 (Figs. 3, 4, 5, 6 and 7).

5.1 Environment creation

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 [23] 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.

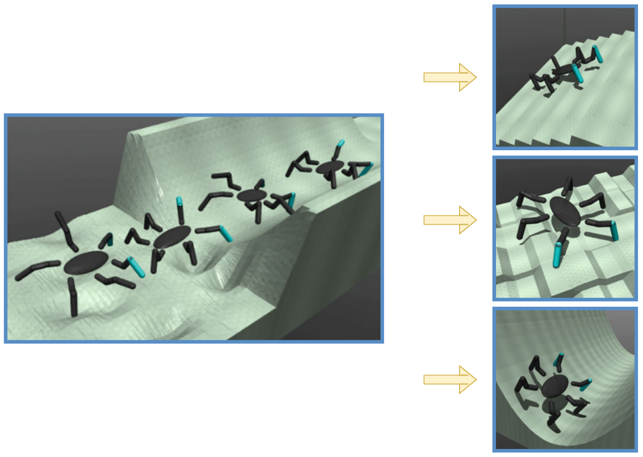

Fig. 3 A scenario of a complex structured terrain which is decomposed into several distinct terrains

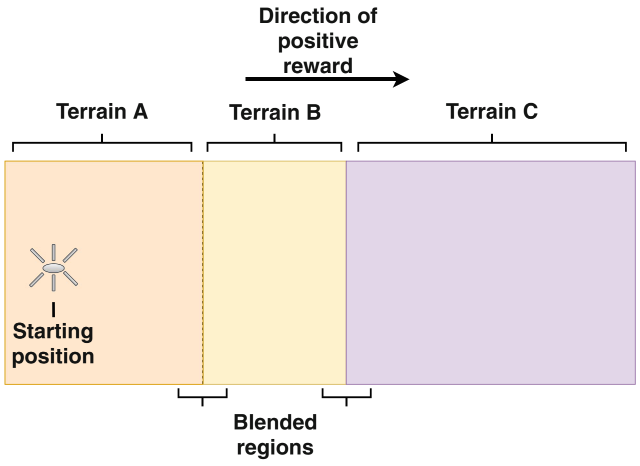

Fig. 4 Top-down illustration of compound terrain consisting of 3 joined instances

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 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.

5.2 Training expert policies

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 networks 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 [16]. The term $A(a_t, o_t)$ can be any estimate of the advantage, or *goodness* of action a_t.

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{o_t, a_t \in \tau} \nabla_\theta log\pi_\theta(a_t|o_t) \cdot A(a_t, o_t) \quad (1)$$

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

$$A(a_t, o_t) = \sum_{t=k}^{h} \gamma^{t-k} r(a_t, o_t) \quad (2)$$

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

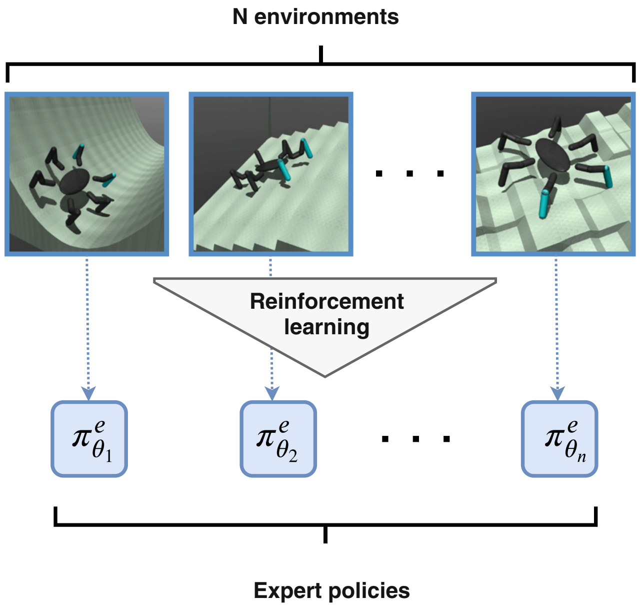

Fig. 5 Use RL to train expert policy for each environment

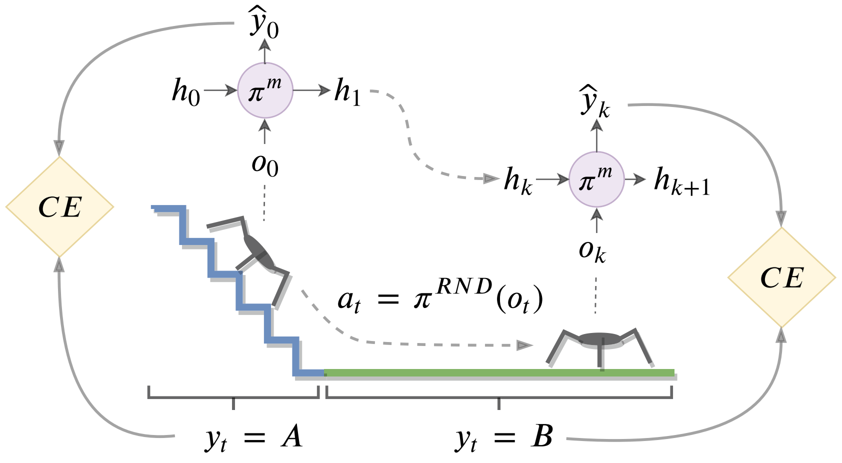

Fig. 6 Training of a recurrent terrain type classifier that is then used as a multiplexer. Observations o_t are seen at every time-step by the classifier, and a label $\widehat{y}$ is predicted. The predictions are then compared against ground truth labels using a cross entropy loss, marked as *CE* in the illustration

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:

$$r_t = \lambda_1 \cdot r_v \lambda_2 \cdot r_\theta - \lambda_3 \cdot \phi_t^2 - \lambda_4 \cdot \psi_t^2 - \lambda_5 \cdot \dot{z}_t^2 - \lambda_6 \cdot \tau_t^2 \tag{3}$$

where

$$r_v = \frac{1}{abs(\dot{x_t} - v_{tar}) + 1} - \frac{1}{v_{tar} + 1} \tag{4}$$

$$r_c = |\theta_{t-1}| - |\theta_t| \tag{5}$$

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.

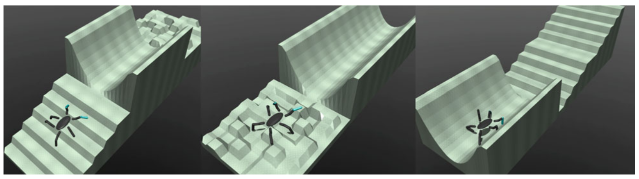

Fig. 7 Examples of generated compound environments from three terrain types

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:

$$\nabla_\theta \frac{1}{|D|} \sum_{(h_t, y_t) \in D} \mathcal{L}(\pi_{obs}(h_t), l_t) \tag{6}$$

Training is done on batches of episodes. It is essentially a sequence to sequence task where 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 ison a different terrain without first having interacted with it.

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 selectingthe 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.

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 [1]. 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.

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 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 and 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 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

Table 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 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

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.

6.3 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 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 3 shows several examples.

Tables 3 and 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 8. This, however, comes with a 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.

6.4 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

Table 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

Table 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

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 timeslower 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 [24, 25].

6.5 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 [26]. The authors of [26] also use curriculum learning by using varying levels of assistance during training time that guides the agents and prevents bad local minima.

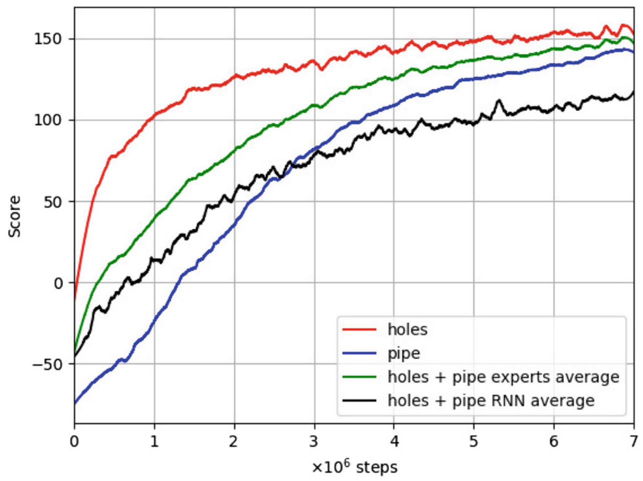

Fig. 8 Average performance comparison of expert policies versus RNN policy trained end-to-end. Graphs show filtered mean learning curves of 30 training sessions in each scenario. We can see that on terrains on contrasting terrains, the end-to-end policy learns slower and does not attain the performance of individual experts on each terrain

6.6 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 (Fig. 9).

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

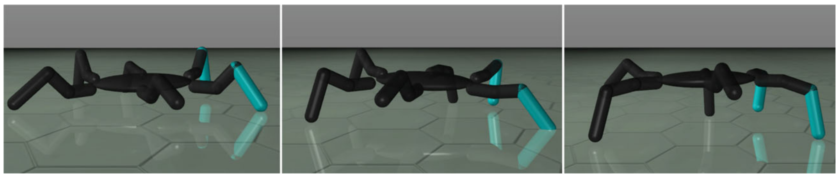

Fig. 9 Hexapod gait at three different body-height levels

this method. The disadvantage is that the policy has to be retrained if the user wishes to add additional criteria.

6.7 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 (Fig. 10).

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.

7 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 [27], 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.

8 Discussion

We presented a data-driven approach to hexapod locomotion that requires almost no domain knowledge and no handcrafting 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,

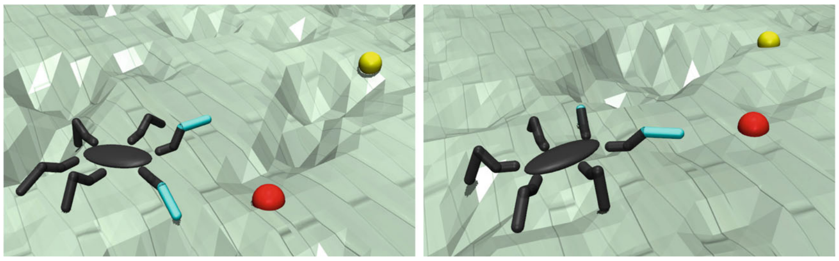

Fig. 10 Double goal-based locomotion. Red is the immediate goal and yellow is the pending goal. The agent receives as the coordinates of the red goal relative to its own and the yellow goal relative to the red

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 7, 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 terrain 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 [18, 28] 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) [29]. This is arguably the most flexible and scalable approach, and as argued by [30], scalability is key for the future of any almost AI technique.

Acknowledgements The research leading to these results has received funding from the Czech Science Foundation under Project 17-08842S.

References

1. Vespignani, M., Friesen, J.M., SunSpiral, V., Bruce, J.: Design of superball v2, a compliant tensegrity robot for absorbing large impacts. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2865-2871 (2018). DOI 10.1109/IROS.2018.8594374
2. Lample, G., Chaplot, D.S.: Playing FPS games with deep reinforcement learning. CoRRabs/1609.05521 (2016). URL http://arxiv.org/abs/1609.05521
3. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347 (2017). URL http://arxiv.org/abs/1707.06347
4. Wang, T., Liao, R., Ba, J., Fidler, S.: Nervenet: Learning structured policy with graph neural networks. In: International Conference on Learning Representations (2018). URL https://openreview.net/forum?id=S1sqHMZCb
5. Peng, X.B., Berseth, G., van de Panne, M.: Terrain-adaptive locomotion skills using deep reinforcement learning. ACM Transactions on Graphics (Proc. SIGGRAPH 2016) 35(4) (2016)
6. Bjelonic, M., Kottege, N., Homberger, T., Borges, P., Beckerle, P.: Chli, M.:Weaver: Hexapod robot for autonomous navigation on unstructured terrain. Journal of Field Robotics. **35**, 1063–1079 (2018). https://doi.org/10.1002/rob.21795
7. Yu, W., Turk, G., Liu, C.K.: Learning symmetry and low-energy locomotion. CoRRabs/1801.08093 (2018). URL http://arxiv.org/abs/1801.08093
8. Boston dynamics, spot. https://www.bostondynamics.com/spot. Accessed: 16-10-2019
9. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural networks: the official journal of the International Neural Network Society. **21**(4), 642–653 (2008)
10. Trossen robotics. https://www.trossenrobotics.com/. Accessed: 22-05-2010
11. Isvara, Y., Rachmatullah, S., Mutijarsa, K., Prabakti, D.E., Pragitatama, W.: Terrain adaptation gait algorithm in a hexapod walking robot. In: 2014 13th International Conference on Control Automation Robotics Vision (ICARCV), pp. 1735-1739 (2014). DOI 10.1109/ICARCV.2014.7064578
12. Open AI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J.W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., Zaremba, W.: Learning dexterous in-hand manipulation. CoRR abs/1808.00177 (2018). URL http://arxiv.org/abs/1808.00177
13. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. **31**(5), 855–868 (2009). https://doi.org/10.1109/TPAMI.2008.137
14. Kruijff, G., Kruijff-Korbayová, I., Keshavdas, S., Larochelle, B., Janícek, M., Colas, F., Liu, M., Pomerleau, F., Siegwart, R., Neerincx, M., Looije, R., Smets, N., Mioch, T., van Diggelen, J., Pirri, F., Gianni, M., Ferri, F., Menna, M., Worst, R., Linder, T., Tretyakov, V., Surmann, H., Svoboda, T., Reinštein, M., Zimmermann, K., Petříćek, T., Hlaváč, V.: Designing, developing, and deploying systems to support human-robot teams in disaster

response. Advanced Robotics. **28**(23), 1547–1570 (2014). https://doi.org/10.1080/01691864.2014.985335
15. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: S.A. Solla, T.K. Leen, K. Muller (eds.) Advances in Neural Information Processing Systems 12, pp. 1057-1063. MIT Press (2000). URL http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf
16. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. CoRRabs/1703.06907 (2017). URL http://arxiv.org/abs/1703.06907
17. Perlin, K.: Improving noise. ACM Trans. Graph. **21**(3), 681–682 (2002). https://doi.org/10.1145/566654.566636
18. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural evolution strategies. Journal of Machine Learning Research 15, 949-980 (2014). URL http://jmlr.org/papers/v15/wierstra14a.html
19. Hutter, M., Gehring, C., Lauber, A., Gunther, F., Bellicoso, C.D., Tsounis, V., Fankhauser, P., Diethelm, R., Bachmann, S., Bloesch, M., Kolvenbach, H., Bjelonic, M., Isler, L., Meyer, K.: Anymal - toward legged robots for harsh environments. Advanced Robotics. **31**(17), 918–931 (2017). https://doi.org/10.1080/01691864.2017.1378591
20. Graves, A., Mohamed, A., Hinton, G.E.: Speech recognition with deep recurrent neural networks. CoRR abs/1303.5778 (2013). URL http://arxiv.org/abs/1303.5778
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
22. Manoonpong, P., Parlitz, U., Wörgötter, F.: Neural control and adaptive neural forward models for insect-like, energy-e_cient, and adaptable locomotion of walking machines. Frontiers in neural circuits. **7**, 12 (2013). https://doi.org/10.3389/fncir.2013.00012
23. Ross, S., Gordon, G.J., Bagnell, J.A.: No-regret reductions for imitation learning and structured prediction. CoRR abs/1011.0686 (2010). URL http://arxiv.org/abs/1011.0686
24. Pecka, M., Zimmermann, K., Reinstein, M., Svoboda, T.: Controlling robot morphology from incomplete measurements. CoRR abs/1612.02739 (2016). URL http://arxiv.org/abs/1612.02739
25. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems pp. 5026-5033 (2012)
26. Čížek, P., Faigl, J.: On locomotion control using position feedback only in traversing rough terrains with hexapod crawling robot. IOP Conference Series: Materials Science and Engineering. **428**, 012065 (2018). https://doi.org/10.1088/1757-899X/428/1/012065
27. Sanchez-Gonzalez, A., Heess, N., Springenberg, J.T., Merel, J., Riedmiller, M.A., Hadsell, R., Battaglia, P.: Graph networks as learnable physics engines for inference and control. CoRR abs/1806.01242 (2018). URL http://arxiv.org/abs/1806.01242
28. Saranli, U.: Rhex: A simple and highly mobile hexapod robot. The International Journal of Robotics Research. **20**, 616–631 (2001). https://doi.org/10.1177/02783640122067570
29. Xie, Z., Berseth, G., Clary, P., Hurst, J.W., van de Panne, M.: Feedback control for cassie with deep reinforcement learning. CoRR abs/1803.05580 (2018). URL http: //arxiv.org/abs/1803.05580
30. Bitter lesson, rich sutton. http://www.incompleteideas.net/IncIdeas/BitterLesson. html. Accessed: 2019-04-18

Teymur Azayev is currently a PhD student at the Czech Technical University in Prague. He started his studies in 2013 and successfully obtained his Bachelor's degree in a Robotics and cybernetics programme at the end of 2016. He spent the following few months working at Spaceknow doing computer vision for satellite images. In 2018 he successfully obtained a Master's degree in Artificial Intelligence. He is currently the head teaching assistant of the Vision for robotics course offered in the Czech Technical Univeristy in Prague. His current interests are learning algorithms for robotics with the aim of practical application

Karel Zimmermann is associate professor at the Czech Technical University in Prague. He received his PhD degree in cybernetics in 2008. He worked as postdoctoral re-searcher with the Katholieke Universiteit Leuven (2008-2009) in the group of prof Luc van Gool. His current Hindex is 13 (google-scholar) and he serves as a reviewer for major journals such as TPAMI or IJCV and conferences such as CVPR, ICCV, IROS. He received the best lecturer award in 2018, the best reviewer award at CVPR 2011 and the best PhD work award in 2008. His journal paper has been selected among 14 best research works representing Czech Technical University in the goverment evaluation process (RIV). Since 2010 he has been chair of Antonin Svoboda Award (http://svobodovacena.cz. He was also with the Technological Education Institute of Crete (2001), with the Technical University of Delft (2002), with the University of Surrey (2006). His current research interests include learnable methods for robotics.