Distributed Learning of Decentralized Control Policies for Articulated Mobile Robots

Guillaume Sartoretti, *Member, IEEE,* William Paivine, *Student Member, IEEE,*
Yunfei Shi, *Student Member, IEEE,* Yue Wu, *Student Member, IEEE,* and Howie Choset, *Fellow, IEEE*

***Abstract*—State-of-the-art distributed algorithms for reinforcement learning rely on multiple independent agents, which simultaneously learn in parallel environments[1] while asynchronously updating a common, shared policy. Moreover, decentralized control architectures (e.g., CPGs) can coordinate spatially distributed portions of an articulated robot to achieve system-level objectives. In this work, we investigate the relationship between distributed learning and decentralized control by learning decentralized control policies for the locomotion of articulated robots in challenging environments. To this end, we present an approach that leverages the structure of the asynchronous advantage actor-critic (A3C) algorithm to provide a natural means of learning decentralized control policies on a single articulated robot. Our primary contribution shows individual agents in the A3C algorithm can be defined by independently controlled portions of the robot's body, thus enabling distributed learning on a single robot for efficient hardware implementation. We present results of closed-loop locomotion in unstructured terrains on a snake and a hexapod robot, using decentralized controllers learned offline and online respectively, as a natural means to cover the different key applications of our approach. For the snake robot, we are optimizing the forward progression in unstructured environments, but for the hexapod robot, the goal is to maintain a stabilized body pose. Our results show that the proposed approach can be adapted to many different types of articulated robots by controlling some of their independent parts in a distributed manner, and the decentralized policy can be trained with high sample efficiency.**

***Index Terms*—Articulated Robots, Distributed learning, reinforcement learning, decentralized control, robotic learning**

I. Introduction

ARTICULATED mobile robots such as snake and legged robots have the potential to excel at locomoting through a large variety of environments by leveraging their ability to adapt their shape to their surrounding terrain [1]. In the case of snake robots in particular, recent results have shown that a decentralized control architecture can improve locomotion through unstructured and cluttered environments [2]–[4]. For articulated robots, in general, decentralized control is generally achieved by partitioning the robots' body into spatially distributed portions, which can then be coordinated by a set of coupled central pattern generators (CPGs) [5], [6]. In deep reinforcement learning (RL), new state-of-the-art methods are also based on the idea of distributing the learning task to several agents, which can then asynchronously update a common, global policy. The policy is generally represented by a function, mapping the current state of an agent to the optimal action the agent should enact; this function is very often approximated by a neural network [7]–[9]. In this paper, we present a learning approach that leverages the general structure of asynchronous distributed learning algorithms [9] as a natural means to learn decentralized control policies for a single articulated platform. In particular, we focus on the A3C algorithm among all asynchronous distributed learning algorithms [9] because it allows agents to learn a stochastic policy (rather than a deterministic one such as a Q-table). Stochastic policies are usually more robust to the many sources of noise and uncertainty that are integral to any hardware training and deployment [10].

Our primary contribution shows how a low-level agent in the A3C algorithm can be defined as one of the independently controlled portions of an articulated robot (Figure 1) in a distributed learning framework.

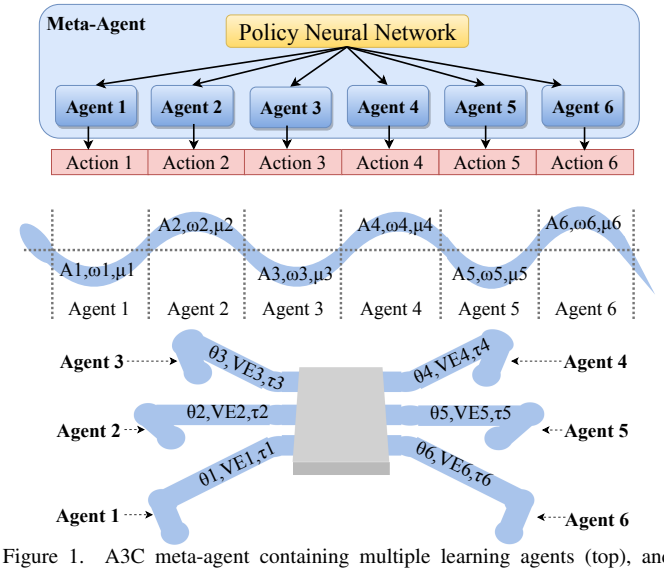


Figure 1. A3C meta-agent containing multiple learning agents (top), and (i) a snake robot (middle), as well as a hexapod robot (bottom). The A3C meta-agent stores a shared policy, that multiple worker agents implement and improve distributedly. Similarly, a single articulated robot can be separated into independent portions, each controlled by a trained agent learning – and ultimately implementing – a common policy in a decentralized manner. In our example, a snake robot is separated into 6 control windows, with local shape parameters A_i, ω_i, μ_i $(i = 1, .., 6)$. Similarly, we consider a hexapod robot whose legs are controlled by an agent each, with local parameters θ_i, VE_i, τ_i.

G. Sartoretti, H. Choset, W. Paivine, and Y. Wu are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213, USA. {gsartore,wjp,ywu5,choset}@andrew.cmu.edu.

Y. Shi is with the Department of Electrical Engineering at The Hong Kong Polytechnic University, Hong Kong. yunfei.shi@connect.polyu.hk.

Manuscript received October 28, 2018;

[1]Independent copies of the same learning environment, i.e., game simulation in which a reinforcement learning agent is allowed to explore its state-action space, collect rewards, and ultimately learn a policy.

This paper explores the fundamental link between distributed learning approaches and decentralized control architectures, our approach is able to quickly learn near-optimal decentralized policies on a single platform, for an efficient hardware implementation. Our previous works focused on offline learning of decentralized policies using A3C [11]; however, in the present paper, apart from the offline approach, we also extend our approach to online distributed learning of decentralized policies. We present a case of offline training for a snake robot, and a case of online training directly on hardware for a hexapod robot. This choice is motivated by our desire to show the feasibility of both training approaches on robots with different morphologies, thus covering the different applications of our approach, without the need to specifically study all the cross cases. In doing so, we note that online training raises two main challenges: first, since different agents share the same experiences on a single robot, the learning task can be destabilized by the resulting, highly-correlated learning gradients being pushed to the global neural network. To overcome this problem, we rely on experience replay [12] that helps decorrelating the agents' experiences with time during training, similar to our recent multi-robot construction work [13]. Second, an agent's actions may affect the other agents' states during learning, and rewards need to be adapted to truly reflect each agent's contribution to the robot's state. That is, in this work, we need to explicitly address *credit assignment* to enable consistent training.

The paper is outlined as follows: Section II provides a short overview of the A3C algorithm, as well as recent works in multi-agent reinforcement learning (MARL). In Section III, we first summarize the general shape-based controller for serpenoid locomotion, whose feedback controller, which adapts the amplitude and frequency of the snake based on proprioceptive sensory input, we seek to replace by a trained agent. We then outline the reinforcement problem structure, as well as the agent structure used in this work. After describing the offline training procedure of our agent, we finally validate the learned policy experimentally on a snake robot and discuss our results. Section IV follows the same outline as Section III, but focuses on learning a decentralized controller online, and directly on hardware, for the stabilized locomotion of a hexapod robot. The learned controller uses proprioceptive and inertial feedback to adapt the pose of an individual leg for the purpose of leveling the robot's body. We finally validate our learned policy on a hexapod robot, locomoting on inclined planes as well as in rocks, and show how the learned policy naturally scales to varying robot morphologies and applies to a quadruped without further training. An overall discussion of our approach is presented in Section V, and concluding remarks and future works in Section VI.

II. Background

A. Single-Agent Learning

Existing learning-based approaches often consider locomotive tasks at the whole robot level [14]–[16]. These approaches usually train an agent to select actions that will simultaneously affect all the degrees of freedom of the robot, most likely sampled from a large-dimensional action space. Therefore, these approaches usually require a large amount of experiences to converge to good/near-optimal policies.

Recent advances in the field of deep reinforcement learning (RL) have highlighted the benefits of relying on multiple independent agents to learn a *common*, optimal policy in a distributed – and thus more time-efficient – manner. Some works have focused on learning *homogeneous* decentralized controllers – i.e., a common policy is enacted in the different portions of the robot based on local feedback – for locomotion [17]–[19], but have only considered single-agent learning approaches where a single agent learns a controller that is then applied to different portions of the robot; others have focused on learning heterogeneous controllers in a distributed manner [20], [21]. In either case, however, these approaches did not take advantage of running multiple learning agents to speed up training, even when considering distributed learning.

Mnih et al. proposed a novel single-agent learning approach for deep reinforcement learning [9], taking advantage of multi-core architectures to obtain near-linear speed-up via distributed learning, where the learning task is distributed to several agents that asynchronously update a shared neural network, based on their individual experiences in independent learning environments. That is, these methods rely on multiple agents to distributedly learn a single-agent policy. A general meta-agent stores a global network, to which the worker agents regularly push their individual gradients during training, and then pull the most recent global weights that aggregate the lessons learned by all agents.

The increase in performance is mainly due to the fact that multiple agents, experiencing different situations in independent environments, produce more decorrelated gradients that are pushed to the global net. Additionally, agents in different environments can better cover the state-action space, and faster than a single agent (e.g., a deep Q-learning agent [7], [8], [22]). Among these asynchronous methods, the A3C algorithm, extending the standard Advantage Actor-Critic learner to asynchronous distributed learning, was shown to produce the fastest learning rate in a stable manner [9]. In this algorithm, the global network has two outputs: a discrete stochastic policy vector (the actor), and a value (the critic), estimating the long-term cumulative reward from the current state.

B. Multi-Agent Reinforcement Learning (MARL)

The first and most important problem encountered when transitioning from the single- to multi-agent case is the curse of dimensionality: most joint approaches fail as the dimensions of the state-action spaces explode combinatorially, requiring an absurd amount of training data to converge [23]. In this context, many recent works have focused on decentralized policy learning [24]–[27], where agents each learn their own policy, which should encompass a measure of agent cooperation at least during training. One way to do so is to train each agent to predict the other agents' actions [25], [26]. In most cases, some form of centralized learning is involved, where the sum of experience of all agents is used toward training a common aspect of the problem (e.g., network output or

value/advantage calculation) [24], [26], [27]. When centrally learning a network output, parameter sharing can be used to let agents share the weights of some of the layers of the neural network and learn faster and in a more stable manner [24]. In actor-critic approaches, for example, the critic output of the network is often trained centrally with parameter sharing, and can be used to train cooperation between agents [24], [26]. Centralized learning can also help when dealing with partially-observable systems, by aggregating all the agents' observations into a single learning process [24], [26], [27]. Many of these approaches rely on explicit communication among agents, to share observations or selected actions during training and sometimes policy execution [25]–[27].

Second, MARL approaches struggle with stabilizing learning: multiple agents can fail at learning in an ever-changing environment where other agents constantly update their policies [24], [27]–[29]. Recently, actor-critic methods have been experimentally shown to be more robust to changing environments, and to lead to more stable MARL [24]–[26]. Additionally, for non-Q-learning-based approaches, experience replay can also help stabilize learning [24]–[26]. Finally, some recent works have proposed curriculum learning, where the complexity of the problem (number of agents in the system) is slowly increased during training, as a way to train numerous agents in a stable manner [24].

III. Decentralized shape-based locomotion

A. Background - Shape-based compliant control for articulated locomotion

In this section, we briefly summarize some results from serpenoid locomotion, as well as their use for low-dimensional *shape-based* compliant control of sequential mechanisms [30]. This is in contrast to non-*shape-based* control strategies, as given in [4], [31]–[34]. Previous works on shape-based control focused on sequential mechanisms as well as legged systems, but we here focus on results pertaining to snake robot locomotion.

1) Serpenoid curves: For an N-jointed snake-like robot, the slithering gait can be parameterized by a sine wave propagating through the lateral plane of the snake (i.e., parallel to the ground) [1], [2], [35].

$$\theta(t) = (\theta_i(t))_{i=1}^{N} \text{ , with } \theta_i(t) = \theta_0 + A\sin(\omega_S s_i - \omega_T t) \quad (1)$$

where θ_i are the commanded joint angles, θ_0 the angular offset, and A the amplitude of the curvature. ω_T is the temporal frequency of the wave, while ω_S describes its spatial frequency, which determines the number of sinusoidal periods on the snake robot's body. $s_i \in \{0, 2 \cdot l_s, \ldots, N \cdot l_s\}$ is the distance from the head to the module i, where l_s is the length of one module.

2) Shape-based Compliant Control: Shape-based control makes use of *shape functions* as a method to reduce the dimensionality of high-degree-of-freedom systems [3]. A shape function $h : \Sigma \to \mathbb{R}^N$ determines $\theta(t)$, which is parameterized by a small number of control parameters, namely the shape variables. Those shape variables define the *shape space* Σ, which is usually of a lower dimension than $\mathbb{R}^N$. In our case, we consider the case when the serpenoid curve Eq.(1)'s amplitude and spatial frequency are shape variables. The shape function $h(A(t), \omega_S(t)) = \theta(t)$ of our system is simply the serpenoid curve Eq.(1), with the two variables being the amplitude and spatial frequency, while the other parameters are fixed:

$$\begin{aligned} h: \quad & \Sigma = \mathbb{R}^2 \mapsto \mathbb{R}^N \\ h_i(A(t), \omega_S(t)) \quad &= \quad \theta_0 + A(t)\sin\left(\omega_S(t)\, s_i - \omega_T\, t\right). \end{aligned} \quad (2)$$

In the state-of-the-art compliant controller for locomotion [3], $A(t)$ and $\omega_S(t)$ (now time-dependent) are determined by the output of an admittance controller [36]. This controller allows the robot to adapt these two serpenoid parameters, with respect to a set of external torques $\mu_{ext}(t) \in \mathbb{R}^N$ that are measured at each joint along the snake's body. This controller enables the robot to comply with its surroundings, by adapting its shape to external forces. The admittance controller for $\beta = \big(A(t), \omega_S(t)\big)^T$ reads:

$$M\,\ddot{\beta}(t) + B\,\dot{\beta}(t) + K\,(\beta(t) - \beta_0) = F(t), \quad (3)$$

where $M, B, K \in \mathbb{R}^{2\times 2}$ respectively represent the effective mass, damping and spring constant matrices of the system; $F(t)$ is the external torques $\mu_{ext}(t)$ transformed from the joint space into the shape space (detailed below).

The considered admittance controller Eq.(3) enables the desired control parameters β to respond to an external forcing $F(t)$ with second-order dynamics, acting as a forced spring-mass-damper system. In the absence of external forces, the control parameters will converge back to their nominal values $\beta_0 = \big(A_0, \omega_{S,0}\big)^T$. In this (centralized) controller, where all joints share the same control parameters $A(t)$ and $\omega_S(t)$, the joint angles $\theta_i(t)$ are computed at each time step t from Eq.(1).

The shape function Eq.(2) is used to map the external torques measured along the snake's body by the use of the associated Jacobian matrix $J(h) \in \mathbb{R}^{2\times N}$:

$$J(h) = \begin{pmatrix} \frac{\partial h_1(A,\omega_S)}{\partial A} & \cdots & \frac{\partial h_N(A,\omega_S)}{\partial A} \\ \frac{\partial h_1(A,\omega_S)}{\partial \omega_S} & \cdots & \frac{\partial h_N(A,\omega_S)}{\partial \omega_S} \end{pmatrix} \quad (4)$$

External torques are mapped from the joint space to the shape space, via:

$$F(t) = J(h)|_{(A(t),\omega_S(t))} \cdot \mu_{ext}(t), \quad (5)$$

which is used to update Eq.(3).

In this paper, we seek to replace the admittance controller Eq.(3), which iteratively adapts the shape parameters based on proprioceptive sensory feedback, by a trained agent. In order to simplify the agent's policy and learning process, we leverage the dimensionality reduction offered by the use of shape functions to keep the agent's state space low-dimensional. That is, we train our agents to reason about and make action in the shape space of the considered robot.

3) Decentralized Control: Decentralized control extends previous works on shape-based locomotion [2]. This previous work relies on the use of *activation windows* (i.e., groups of successive joints), and only couples the control of those joints covered by the same window, isolating them from other joints in the snake. These windows are positioned along the backbone curve of the snake robot in order to create

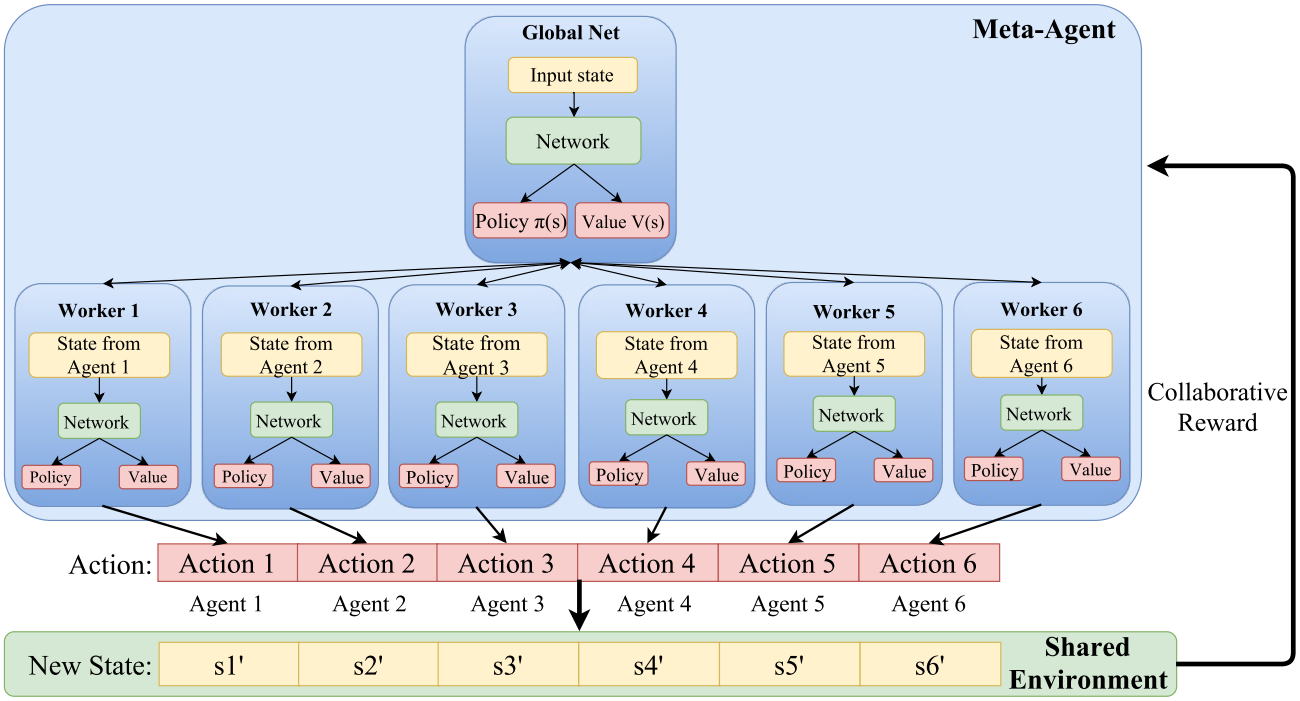


Figure 2. Structure of the A3C Meta-Agent, with global weights for the Actor-Critic network. Each internal agent (worker) updates its local weights based on shared rewards during local interactions with the environment, and regularly pushes its gradients to the global networks. Each worker draws its state from that of each portion of the articulated robot. Rewards are calculated in response to a vector of the agents' actions that is enacted on the robot.

independent groups of neighboring joints, in which torques are sensed and motion parameters are adapted. This framework allows each separate portion of the snake robot to react independently against local forcing. Any impact detected by torque sensors locally in a specific section of the body activates a reacting reflex in the individual section only, resembling biological reflexes where muscle contractions are produced by local feedback [37].

Independent motion parameters for each windows are defined by using sigmoid functions:

$$\beta(s,t) = \sum_{j=1}^{W} \beta_{s,j}(t) \left[\frac{1}{1+e^{m(s_{j,s}-s)}} + \frac{1}{1+e^{m(s-s_{j,e})}} \right], \quad (6)$$

where m controls the steepness of the windows, W is the number of windows, and $\beta_{s,j}(t)$ the values of the serpenoid parameters in window j at position s along the snake's backbone. Window j spans the backbone of the robot over $[s_{j,s}, s_{j,e}] \subset [0;1]$. In this work, windows are anchored between zero curvature points of the serpenoid curve Eq. (1), as is standard practice for use in decentralized shape-based compliant control [2]. Note that the snake's spatial frequency (determining the number of sine periods along the snake's body) influences the number and positions of amplitude windows along the backbone [2].

Recent results have shown that some form of control decentralization is advantageous for a snake robot moving through unknown terrain by relying on local torque-feedback only [2], [38], or in combination with normal force-feedback [32]. That is, decentralization allows different portions of the snake robot to react independently based on local sensory feedback, in a manner reminiscent of most animals' muscular reflexes. In particular, these results have considered the case where windows are not fixed to the snake's backbone, but are moved along the snake robot at the same velocity as the serpenoid wave, in order to pass information down the robot's body. In this paper, we consider the state-of-the-art decentralized control of the snake robot with sliding windows [2], while shape parameters in each window are updated by a trained agent.

B. Policy Representation

In this section, we cast the problem of controlling the snake robot's shape in response to external torques as a Markov decision problem (MDP). To this end, we detail the state-action space of the learning agents, as well as the structure of the neural nets used to represent the policy agents learn. We look to the A3C algorithm to distributedly learn a common stochastic policy [9], which will then be followed by the joints in each independent window. In a sense, the A3C meta-agent can represent the whole robot, while the low-level worker agents act as the windows, each selecting local adaptation in the shape parameters and learning from local interactions with the environment, as depicted in Figure 1. Only the global reward, measuring the forward progression of the robot based on the workers' chosen actions, is shared among all workers. Alternative reward functions, such as energy efficiency or impact with environment could also result in viable control policies, but were not tested in this paper.

1) State Space: In reinforcement learning, the state of an agent is the information available to it in order to make its current action decision. In our case, since this decision is made via the use of a neural network, the state also represents the input to that neural network. More specifically,

the state η is a 7-tuple of shape-based quantities. It contains the current shape parameters $\beta(s,t)$ of a window on the snake robot (i.e., amplitude and spatial frequency), as well as their nominal values β_0. Additionally, the state features the current external torque readings $\mu_{ext}(s,t)$ for the same window, after projection into the shape space by the Jacobian of the system. Finally, the state needs to feature some information about the cyclic nature of the serpenoid gait. To this end, we include the *modular time* quantity $[0;1] \ni \tau(t) = \frac{t \mod T_s}{T_s}$, which is simply a normalized phase of the serpenoid's wave, with $T_s = \frac{2\pi}{\omega_T}$ the period of the serpenoid curve Eq.(1). The state vector $\eta_j(t)$ for window $j \in \{1,..,W\}$ reads:

$$\eta_j(t) = \langle \tau(t),\, \beta_j(t)^T,\, \mu_{ext}(j,t)^T,\, \beta_0^T \rangle. \tag{7}$$

2) Action Space: The admittance controller Eq.(3) lets the shape parameters evolve with continuous increments. In this work, we choose to discretize the possible increments in the shape parameters, as a means to reduce the dimensionality of the action space. Specifically, we let an action be a 2-tuple $a = \langle a_A, a_\omega \rangle$, with $a_A \in \{-\Delta_A, 0, +\Delta_A\}$ and $a_\omega \in \{-\Delta_\omega, 0, +\Delta_\omega\}$. The resulting action space contains 9 discrete actions (3×3), which simultaneously update both shape parameters by applying the selected increments:

$$\beta_j(t + dt) = \beta_j(t) + a(j,t) \cdot dt, \tag{8}$$

with $a(j,t) = \langle a_A, a_\omega \rangle_j$ the action selected by agent j and dt here is set to be $\frac{\pi}{160}$ seconds.

3) Actor-Critic Network: We use two deep neural networks with weights Ψ_A, Ψ_C to approximate the stochastic policy function (Actor) and the value function (Critic). Both networks are fully connected with two hidden layers (see Figure 3). The output of each layer of the Actor network passes through a rectifier linear unit (ReLU), except for the output layer that estimates the stochastic policy using a Softmax function. We also use a ReLU at the output of each layer of the Critic network to introduce some non-linearity. The nonstandard use of a ReLU at the output of the value network is explained by the fact that only positive advantages can be calculated in practice, since the snake robot can only move forward or get stuck (resulting in positive or zero rewards).

Six workers, which correspond to the six independent windows along the backbone of the snake[2], are trained concurrently and optimize their individual weights using gradient descent. Each worker calculates its own successive gradients during each episode. At the end of each episode, which typically lasts around $89\,dt$ (≈ 1.75 seconds), each worker updates the global network and then collects the new state of the global weights.

We used an entropy-based loss function introduced in [39] to update the policy, $\pi(a_t \,|\, s_t; \Psi_A)$:

$$\begin{aligned} f_\pi(\Psi_A) \;\; = \;\; & \log\left[\pi(a_t \,|\, s_t; \Psi_A)\right](R_t - V(s_t; \Psi_A)) \\ & + \kappa \cdot H\left(\pi(s_t; \Psi_A)\right), \end{aligned} \tag{9}$$

In Eq.(9), notice that the policy loss is calculated using the *advantage* $(R_t - V(s_t; \Psi_A))$, where R_t is the estimated discounted reward over time t, instead of only using the cumulative discounted reward R_t. The advantage measures how favorable a given action is in term of long-term expected reward, compared to the value of the current state acting as a baseline of the state's overall quality. This comparison has been shown to improve a learning agent's ability to not over-/under-estimate the quality of the available actions in very favorable/unfavorable states [9], and to improve its decision-making in such difficult states. $H\left(\pi(s_t; \Psi_A)\right)$ is the entropy factor used to encourage exploration in training and $\kappa \in \mathbb{R}$ helps manage exploration and exploitation, with higher κ favoring exploration. The value loss function reads:

$$f_v(\Psi_C) = \left(R_t - V(s_t; \Psi_C)\right)^2. \tag{10}$$

The stochastic policy is optimized using policy gradient. During training, we compute the gradients of both functions and optimize the functions using the Adam Optimizer [40].

4) Shared Rewards: Different low-level workers have individual input states, actions and new states corresponding to independent windows. However, we propose that workers learn based on shared rewards for the combination of their actions. That is, a shared reward is calculated based on the current progression of the robot, measured from its initial position at the beginning of the current episode $X_{0,i}$ as

$$r_t = \tanh\left(\lambda_r \cdot \left(\|X(t)\|_2 - \|X_{0,i}\|_2\right)\right), \tag{11}$$

with $X(t)$ the current position of the robot in the world frame and λ_r a scaling factor controlling the slope of the hyperbolic tangent. Note that the reward is normalized by using the $\tanh$ function, to avoid unwanted spikes in rewards, e.g., when the robot suddenly boosts forward due to dynamic effects.

We expect shared rewards to allow the joint actions of all agents to be valued simultaneously, in order to better train them for a collaborative locomotion task. It is also worth noting that we only design and test this shared reward based on the progression as our goal is to optimize the forward progression of the robot. The reward can be designed to optimize the other metrics such as energy and impact. The global structure of the A3C meta-agent in this paper is illustrated in Figure 2.

C. Learning

To effectively learn the policy $\pi(a_t \,|\, s_t; \Psi_A)$ using real robot hardware, the state-action space must ideally be densely sampled in the region of realistic parameters. However, using randomly seeded values to initialize π would likely induce the need for thousands of trials to collect enough data to thoroughly sample this space, which is generally not feasible when learning on hardware (as opposed to running thousands of simulated scenarios). In earlier works, replaying experience from a database to train agents has proven successful in both stabilizing and improving the learned policy [41]. We use elements of this idea to train our learning agents more effectively.

[2]As mentioned in Section III-A3, the current shape of the snake robot influences the number and placement of the independent windows along its backbone. However, for the implementation of the decentralized controller, we cap the number of windows to 6. This cap has been selected because the number of windows along the backbone of the snake robot has never experimentally exceeded 5. If less than 6 windows are positioned along the snake's backbone at a given time, the other windows are assumed to be composed of 0 modules and experience no external torque. Their shape parameters are additionally assumed to be the nominal ones β_0.

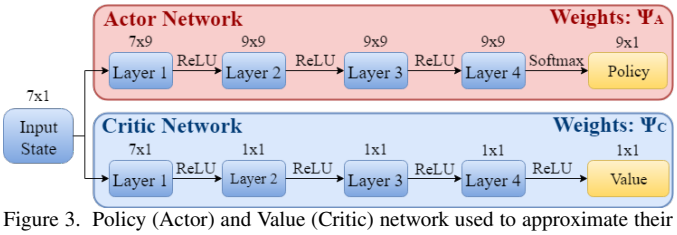


Figure 3. Policy (Actor) and Value (Critic) network used to approximate their respective functions, based on their weights Ψ_A and Ψ_C.

A3C is inherently an on-policy learning algorithm, which can be used to train agents by freely allowing it to sample its state-action space. If A3C is used off-policy, the gradient updates in the algorithm cannot be theoretically guaranteed to be correct in general. However, in this work, we experimentally show that A3C can be used to learn off-policy from data sampled by a near-optimal policy. We believe that this can be the case, since sampling the region of the state-action space close to such a near-optimal policy allows for approximate, yet beneficial updates to the policy. To this end, we propose to train the agents by replaying past experiences, collected from the state-of-the-art compliant controller (in a manner resembling imitation learning).

In order to build the experience replay database, we performed 310 trials of the snake robot in randomized peg arrays and with randomized control parameters. In doing so, we record actions that result in positive and negative reward values for different states (in practice, only positive or zero rewards, as explained in Section III-B3). In the compliant controller, three 2×2 diagonal matrices (i.e., a total of 6 parameters) M, B, and K, control the mass, damping, and spring constant of the system. For each trial, different values for the diagonal terms of M, B, and K are drawn at random from a uniform distribution over $[\frac{1}{2}; 5]$. The reward is calculated at each time step using Eq.(11), by relying on an overhead motion capture system to record the position of the snake robot from tracking beads placed along the snake's backbone. For each time step, the state and action is recorded as described in Sections III-B1-III-B2. Each trial is run for approximately 15 seconds in length which we then divided into several episodes. After collecting the 310 trials of the compliant controller in randomized peg arrays (e.g., Figure 5), we stored the data from each trial in a single experience database.

We then assigned a single window to each agent, which was kept constant across all training runs. During training, each agent asynchronously selected a random run, and then a random starting point in the run. The agent then learned from an episode of 89 action-state-reward steps before randomly selecting a new episode. Leveraging the symmetric nature of the serpenoid curve, we chose 89 steps to match the length of half of a gait cycle (given our choice of dt). That is, this choice of episode length should provide agents with enough temporally-correlated experiences to connect episodes in a smooth and meaningful manner, and thus learn an efficient global policy. Each of the agents then sampled episodes until $50,000$ had been drawn. At this point, we stopped the training process and obtained the learned policy.

The learning parameters used during offline training read:

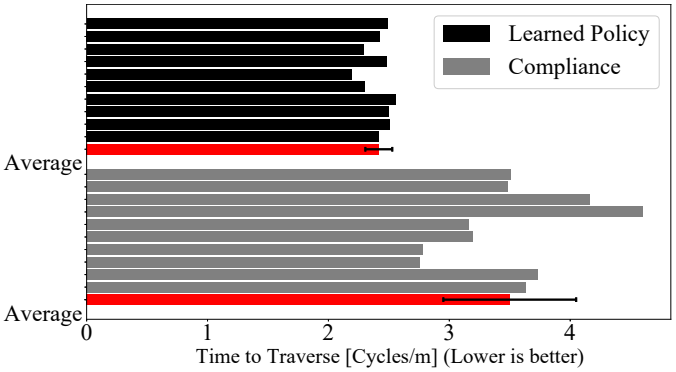


Figure 4. Experimental comparison between both controllers during the 15 trials, in terms of serpenoid gait cycles needed to progress one meter. Lower values are better. The learning-based controller (black) outperforms the state-of-the-art compliant controller [2] by more than 40%.

$$\Delta_A = 0.005 \qquad \Delta_\omega = 0.012 \qquad \lambda_r = 100$$
$$\gamma = 0.995 \qquad \alpha_A = \alpha_C = 0.0001 \qquad \kappa = 0.01,$$

with γ the discount factor for the policy update, and α_A, α_C the learning rates of the Adam optimization step for the Actor/Critic networks. The full offline learning code and database used in this work is available online at https://github.com/gsartoretti/Deep-SEA-Snake.

D. Experimental Validation

In order to test the validity of our learning approach, we trained a meta-agent to adapt the shape parameters of a snake robot in a decentralized manner. We implemented the decentralized controller described in Section III-A3 on a snake robot and ran experiments in randomized unstructured environments. Our goal was to compare the average forward progression of the robot when the shape parameters are updated by the trained agents or the compliant controller Eq.(3).

1) Experimental Setup: We implemented two versions of the decentralized controller Section III-A3, using 1) the learned policy, and 2) the compliant controller Eq.(3) to adapt the shape parameters. These controllers were tested on a snake robot, composed of sixteen identical series-elastic actuated modules [42]. The modules were arranged such that two neighboring modules were torsionally rotated 90 degrees relative to each other. The deflection between the input and output of a rubber torsional elastic element is measured using two absolute encoders, allowing us to read the torque measured at each module's rotation axis. Only the 8 planar modules were active, as only planar gaits were tested. A braided polyester sleeve covered the snake, reducing the friction with the environment, and forcing the snake to leverage contacts to locomote. Robot displacement data was collected with an overhead 4-camera OptiTrack motion capture system (NaturalPoint Inc., 2011).

2) Experimental Results: To compare the efficacy of both controllers, we calculate the number of gait cycles needed to traverse one meter and the variance of this number over 15 trial runs, as established in [2]. This metric removes the potential variations due to running the gaits with differing temporal frequencies, thus only scoring the controller's kinematic abilities. Additionally, the variance of the forward progression over the

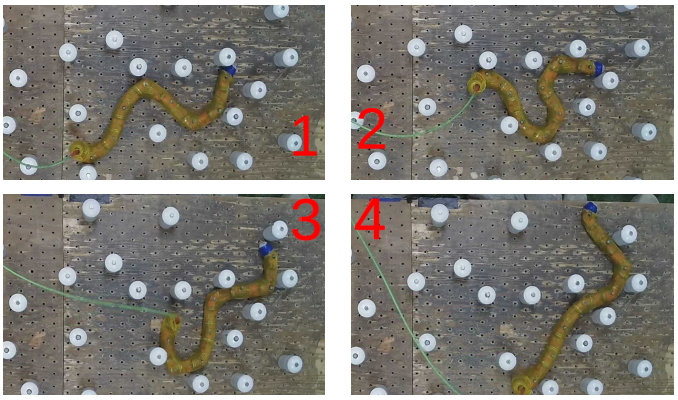


Figure 5. Example frames sampled from a trial, showing the behavior of the learning-based controller. Note how the amplitude and frequency of the waves in each of the windows of the snake robot change to fit the peg array.

runs serves as an indicator of the repeatability of the controller. For all the trials, the same unstructured peg array was used for both the compliant controller and the learning-based controller. Additionally, for each pair of runs, the snake robot was placed in identical starting location as to ensure that both controllers were tested with the same initial configurations. That is, initial configurations were randomized, but kept identical between both controllers.

Both controllers were run with the same temporal frequency and time step size. We used the following empirically established optimal parameters for the compliant controller, with time step dt.

$$M = \begin{bmatrix} 1.5 & 0 \\ 0 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \quad K = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\beta_0 = \left[\tfrac{\pi}{4}, 3\pi\right]^T \quad \omega_T^{lat} = 1.8 \quad dt = \tfrac{\pi}{160}[s]$$

Over the 15 trial runs, the learning-based controller achieves an average of $3.27\,[cycles/m]$ on average, with a variance of 0.65, compared to the compliant controller, which achieves $4.28\,[cycles/m]$ on average, with a variance of 1.54. The results of the compliant controller closely matches, or even slightly outperforms, previous results presented in [2], [3]. The results are compelling, as they show a distinct increase in the forward progression for the learned policy over the compliant controller. These results are summarized in Figure 4. The learned policy outperforms the compliant controller by a significant amount (over 40%), while the variance is decreased, implying that the learning-based controller is also more repeatable. Figure 5 shows how the learning-based controller modifies the shape parameters in each window during an example trial, in order to adapt to the local configuration of pegs by relying on local force sensing. Videos of the trials are available online at https://goo.gl/FT6Gwq.

IV. CPG-BASED STABILIZED LEGGED LOCOMOTION

In the previous section, we considered the offline distributed learning of a decentralized policy, based on experience gathered from a state-of-the-art controller. Here, we now consider online learning of decentralized policies directly on hardware, by relying on the same distributed learning framework. To this end, we focus on the problem of stabilizing the body of a hexapod robot during locomotion, by considering each of its legs as one agent, which need to work with the others. We build upon our recent works on central pattern generator-based locomotion [43] to let each agent (leg) adapt a single shape parameter controlling the height of a shoulder. We detail the state-action space and the policy representation used when casting this problem in the RL framework, and propose a reward structure that enables collaborative training. The trained policy is validated on a series-elastic hexapod robot, while standing and walking on an inclined board and in rocks.

A. Background - CPG Model

We first summarize the central pattern generator (CPG) model, and highlight the model parameters that are adapted to stabilize the robot's body during standing and locomotion.

1) Robot Description: The robot used for our experimental validation is a hexapod made out of series-elastic joints, similar to [30]. These joints are identical to the modules of the snake robot in Section III. Each leg of the robot, shown in Figure 7 (right), is composed of three modular joints: a planar proximal joint, as well as intermediate and distal vertical joints. The non-joint aluminum portions of the hexapod body (the distal ends of the legs and the rectangular body) are made of hollow aluminum, with an ethernet network switch within the main body. In this work, we refer to the two proximal joints as the robot's shoulders.

2) Mathematical Modelling: We express our CPG model as a set of coupled oscillators, i.e., a set of coupled ordinary differential equations, directly in the joint space of the robot. Let $x(t) = [x_1(t), ..., x_n(t)]$ represent the shoulder joint angles of each leg in the axial plane, and $y(t) = [y_1(t), ..., y_n(t)]$ those in the sagittal plane. We consider a general family of parametric limit cycles $H(x, y)$ [43], [44], taken from the superellipse functions on $\mathbb{R}^2$ [45], [46], and enabling us to vary the shape of the robot's steps during locomotion (see [43] for details). The resulting CPG model is:

$$\begin{cases} \dot{x_i}(t) = -\omega \cdot \partial H_{y_i} + \gamma\Big(1 - H_{c_i}(x_i(t), y_i(t))\Big)\partial H_{x_i} \\ \dot{y_i}(t) = +\omega \cdot \partial H_{x_i} + \gamma\Big(1 - H_{c_i}(x_i(t), y_i(t))\Big)\partial H_{y_i} \\ \qquad\quad +(\lambda \textstyle\sum_j K_{ij}(y_j(t) - c_{y,j}), \end{cases} \tag{12}$$

where ω selects the angular frequency of the gait, γ the forcing to the limit cycle, λ the coupling strength between legs, and K, the coupling matrix [47], defines the gait by controlling the phase relationship between the robots' legs. The limit superellipse $H(x, y)$, with $\partial H_\zeta = \frac{\partial H_{c_i}}{\partial \zeta}(x_i(t), y_i(t))$, reads

$$H_c(x, y) = \left|\frac{x - c_x}{a}\right|^n + \left|\frac{y - c_y}{b}\right|^n. \tag{13}$$

where a and b select the major/minor axes of the limit ellipse and n defines its curvature level. For example, $n = 2$ provides a simple elliptic limit cycle; $n > 2$ gives a more rectangular limit cycle with sharper corners as n increases.

Similar to our previous works [43], we use the vertical shoulder offsets $c_{y,j}(t)$ to keep the robot's body leveled and at a constant desired height from the ground. These offsets allow us to modify the pose of the body in the null space of

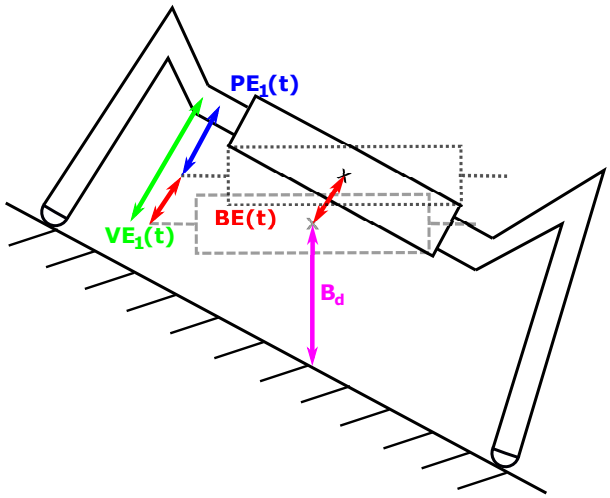


Figure 6. Body posture error **PE**(t), body height error **BE**(t), and overall vertical error **VE**(t) displayed for agent 1 (leg 1). The current configuration (i.e., shape) of the robot is displayed in solid lines, corrected body orientation (level orientation of the body) in dotted line and target body pose (level with body height $\mathbf{B}_d$ from the ground) in dashed line. The ground plane is estimated from the position of the three grounded feet of the robot (known from the current state of the CPG oscillators).

the proximal joint. That is, we are able to control the body's pitch/roll angles without affecting its yaw (heading).

To implement our CPG model on the robot, we simply output $x_i(t)$ and $y_i(t)$ to the two proximal joints of leg i. A closed-form inverse kinematics function is used to calculate $\theta_{3,i}(t)$, the angle of the distal joint of each leg, during locomotion. For example, when walking in a straight line, $\theta_{3,i}(t)$ needs to keep the end-effector at a constant, desired distance from the body as the proximal joint angles change.

B. Policy Representation

We now describe how stabilizing the hexapod robot is cast as a Markov decision problem (MDP). To this end, we detail the selected state and action spaces, the policy representation networks, as well as the collaborative reward structure used for distributed learning.

1) State Space: The state for each agent (i.e., leg) contains raw information from the sensors of each module of the associated leg, plus a higher-level kinematic information about the overall vertical position error of the same leg. For each of the three modules of leg i $(1 \leq i \leq 6)$, the state contains the measured joint angle $\theta_{j,i}(t)$ and external torque $\tau_{j,i}(t)$ $(1 \leq j \leq 3)$ at each time step. The last component of the state vector is the current estimated vertical error $VE_i(t)$ at the proximal joint level. The vertical error is a sum of the vertical offset induced by any pose error, and any overall body height error. Let $SP \in \mathbb{R}^{3\times 6}$ be a matrix containing the 6 column vectors giving the position of the center of rotation of each of the proximal joints in the robot's body frame. Given $T \in SO(3)$ the current body pose of the robot, the body posture error $PE(t) \in \mathbb{R}^{1\times 6}$ reads:

$$T \cdot SP = \begin{bmatrix} XE(t) \\ YE(t) \\ PE(t) \end{bmatrix}. \tag{14}$$

The body height error $BE(t)$ is estimated by first measuring the current height of the robot's body with respect to the ground plane, and then calculating the difference between the current and desired body heights. To this end, we estimate the ground plane by fitting a plane through the position of the end effector (foot) of the 3 grounded legs (known from the current state of the CPG oscillators). We note that this ground plane estimation method yields an accurate body height estimation on flat level/inclined ground. However, the body height may not always be meaningful when the robot locomotes on unstructured environments (such as rock piles), where the body height will be measured from the feet's position and not the actual ground details. We selected this approach, since there is no general method to estimate the body height of a legged robot locomoting over unstructured terrain, without access to a level map of the terrain, and we show that our approach works very well even on unstructured terrains. Relevant quantities are illustrated in Figure 6. With this, we express the estimated vertical error $VE(t) \in \mathbb{R}^{1\times 6}$ as

$$VE(t) = PE(t) + BE(t). \tag{15}$$

The 1×7 state-vector $s_i(t)$ of agent i at time t finally reads

$$\langle \theta_{1,i}(t), \theta_{2,i}(t), \theta_{3,i}(t), VE_i(t), \tau_{1,i}(t), \tau_{2,i}(t), \tau_{3,i}(t) \rangle. \tag{16}$$

This state space differs from that in Section III mainly in that direct joint angles are given to the agent, rather than coordinates in the shape space. In Section III, joint angles could not be used, as there was not a constant number of joints in each window, but the neural net needs a constant number of inputs. Here, since the number of joints considered by each agent is fixed (3 per leg/agent), we directly use those angles in the state space and let agents learn from raw information.

2) Action Space: The action state is composed of a set of discrete positive and negative increments to be applied to current value of the shoulder offsets $c_{y,i}(t)$, given in radians, from Eq.(12). Increasing (resp. decreasing) the value of a shoulder offset results in the associated robot's shoulder being moved upward (resp. downward) in the world frame, iff the associated foot is in contact with the ground.

In order to compare between centralized/distributed learning, we need to keep the individual agents' action space low-dimensional, in order to obtain a joint action space as low-dimensional as possible. For this reason, we propose to only use 3 actions: $a_i(t) \in \{-1, 0, 1\}$, given in radians, for agent i. For each agent, applying the selected action $a_i(t)$ updates the value of the associated shoulder offset, following:

$$c_{y,i}(t) = c_{y,i}(t) + a_i(t) \cdot dt, \tag{17}$$

with dt the duration of a time step. This closely mirrors the action space given in the first section, with the difference being that this action is directly in the joint space rather than in the shape space as in the first section.

3) Actor-Critic Network: We use a 5-layer fully-connected neural network, each layer with output size 128, followed by a standard long short-term memory (LSTM) layer with 128 outputs, and a residual shortcut [48] between the input layer and the fourth hidden layer (see Figure 7). The output layer consists of the policy neurons with softmax activation, and a value output predicting the reward. We also added two single

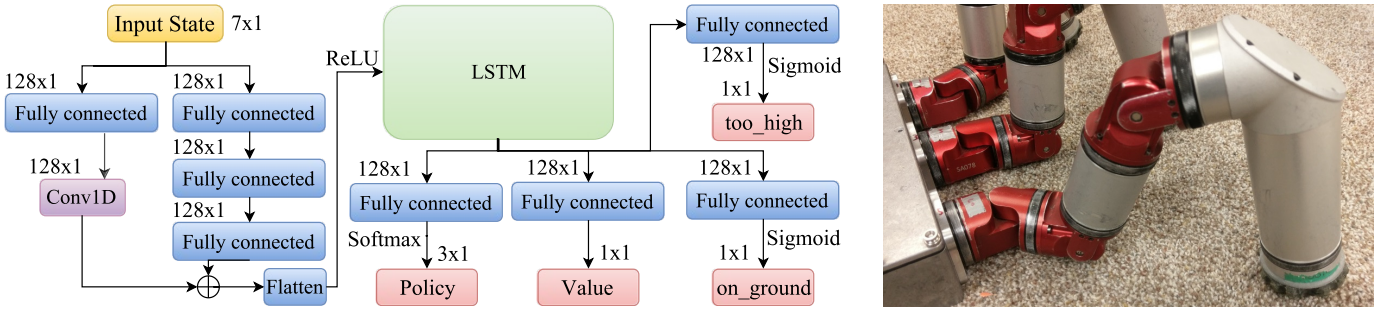


Figure 7. Left: 5-layer deep neural network used for policy estimation. Right: Hexapod leg configuration, showing the three joints (red modules).

output neurons, which enable feature augmentation [49], to further train the agent on its current state. More specifically, we let each agent estimate if the robot's body is above the desired body height, as well as whether the associated leg rests on the ground. Unlike proposed in the ViZDoom task [49], we connected this layer to the output of the LSTM along with the output layers. Since the input from the sensors is noisy, training instead the LSTM on the features makes the model more robust against noise. The primary difference from the network architecture used in Section III is the inclusion of an LSTM, which gives the agent the ability to better coordinate its actions across the time domain. This was not necessary in the network architecture for the snake robot because we made the assumption that the snake robot is purely kinematic, while the hexapod has unmodelled internal dynamics.

4) Collaborative Reward Structure: For each agent, we define the reward $r_i(t)$ resulting from enacting the joint actions $a(t) = \langle a_i(t) \rangle_{i=1}^{6}$, as negative the time-derivative of $PE_i(t)$. In other words, we favor decreasing the agent-specific shoulder vertical errors with time. However, simply training agents with this reward structure does not yield a working collaborative policy at the robot level.

During training, some of the legs are sometimes not in contact with the ground. At these times, the associated agent cannot correct its shoulder height, since increasing/decreasing the shoulder offset of a leg in the air does not change the robot's body pose. Therefore, the reward of an agent whose associated leg is in the air should not allow any misinterpretation of its actions (since they have no effects). In this work, we propose to simply set the reward value to 0 for any agent whose associated leg is currently in the air. With $\delta_i(t)$ a binary value stating whether leg i contacts the ground, the reward of agent i at time t finally reads:

$$r_i(t) = r_0 - \delta_i(t) \cdot \frac{VE_i(t) - VE_i(t - dt)}{dt}, \qquad (18)$$

where r_0 is negative (in practice, $r_0 = -0.02$), incentivizing agents to stabilize the robot as quickly as possible. This reward structure is fundamentally different from the reward structure used on the snake, as the task is also fundamentally different. However, both reward functions leverage the fact that a majority of the actors performing well correlates with the overall robot performing well.

C. Experimental Validation

In this section, we describe the experimental learning and validation of our approach on a series-elastic hexapod robot.

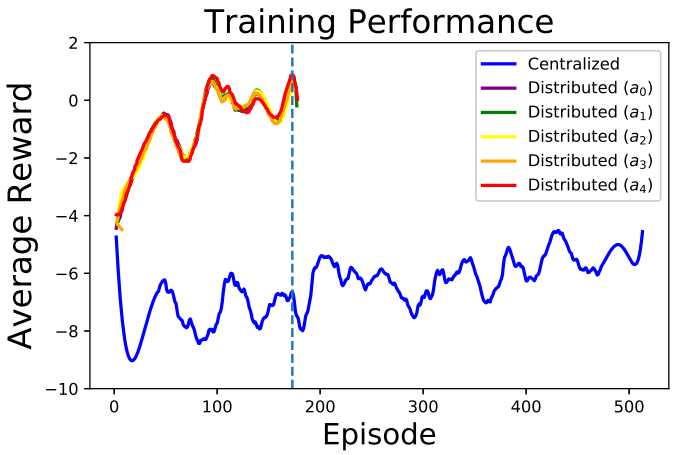


Figure 8. Reward profile during centralized (blue) and distributed (other colors) learning. The centralized approach, despite a network twice as large, did not show any improvement after 500 episodes due to its 729 actions. The distributed approach converges to a smooth policy in 176 episodes.

We first describe the experimental setting and our learning methodology, and then present and discuss our results.

1) Experimental Setting: The experimental setting is composed of a $1.5m \times 1.5m$ flat wooden board covered by a friction-rich carpet, on which the robot stands during training. The plank is tilted, so as not to let the robot overfit in the presence of a flat environment. At the beginning of each episode, the robot is initialized with a random pose, with some legs in the air and at least 3 on the ground. The robot is then given 250 time steps ($dt = 0.02\,s$) to stabilize; an episode is terminated early if the robot stabilizes within 250 iterations. The robot is stabilized if its posture error and its body height error are below a threshold. Additionally, in order to further vary the challenges given to the agent(s), we rotate the robot by $\frac{\pi}{4}$ between successive episodes, to offer varied torque readings to the different legs. The experimental setting is pictured in Figure 9. To compare the decentralized policy with a centralized policy, we trained a centralized policy in the exact same experimental setting. As the state space is six times larger, we doubled the neural network's width (number of neurons at each layer) of the decentralized case to still allow for a good function approximation. The centralized reward structure is such that the average of all legs' individual rewards (Eq.(18)) is given to the central learning agent.

2) Robot Learning: Since the neural network we proposed is relatively shallow, we trained the agents on a simple desktop computer (6-core AMD Phenom II X6 1055T processor

clocked at 2.8GHz with 8Gb of RAM). To decorrelate the gradients from each of the legs, we implement experience replay [12] for each agent. We create an experience pool of the most recent 6 episodes for each agent. To train an agent, we randomly draw an experience sample from a randomly selected episode of its pool. Experiences stored in the pool are different (since we vary the robot's initial poses as explained in Section IV-C1), thus further reducing over-fitting. We use the standard Boltzmann action-selection strategy [50] for the agents during training to encourage exploration. We stop the training when a smooth policy is found (i.e., the stochastic policy nearly always outputs a single certain action). The full code used for online learning and offline policy evaluation is available at https://github.com/gsartoretti/TRO2018-hexapodLearning.

3) Results: The distributed approach converged to a satisfactory policy after episode 176, while the centralized policy never managed to converge (within 500 episodes) as shown in Figure 8. With distributed learning, the performance (reward per episode) immediately started following an increasing trend. The reward per episode for the distributed policy increased up to episode 176, when it reached 1 reward per episode, which corresponded to the point at which a smooth policy was found. With centralized learning, no significant improvement was found even after nearly three times more training episodes.

After training, the distributed policy performs as expected, and stabilizes the robot's body. During execution of the trained policy, we used a slightly different action space $a_i(t) \in \{-0.3, 0, 0.3\}$. This proved necessary to avoid oscillations around the stable position of the body (i.e., goal body orientation and height), which arose when we allowed the larger increments $a_i(t) = \pm 1$. We believe that this is due to the fact that an action space composed of two extreme increments and an idle action does not always allow the legs to reach the goal pose of the body, thus creating oscillation patterns around the goal pose. Note that these oscillations never appeared during training, since episodes were ended when the robot's body reached (close to) the goal pose. With the action set $\{-0.3, 0, 0.3\}$, stabilization was generally slightly slower, but no oscillations were present when continuously executing the policy. Figure 9 shows starting and ending frames of two example trials of the distributed learned policy, when the robot is initialized in a random pose on the sloped testing environment and given time to level its body. All the videos of the training and trials are available at https://goo.gl/Ty7Sta, showing additional trials in rocky, unstructured terrain.

Once trained, the distributed policy is not limited to a specific number of limbs, and can scale/generalize to a different robot morphology, such as a quadruped or a hexapod where a leg has failed. Example static stabilization trials on a quadruped robot can be found at https://goo.gl/Ty7Sta. The center limbs of each side of the robot body are removed when switching the hexapod to a quadruped. We kept the same CPG parameters but only execute joint angle commands on the four limbs that are attached to the robot body. This results in static stabilization of the quadruped, but would not allow stable locomotion, for which a new CPG would likely need to be devised (including a new coupling matrix K).

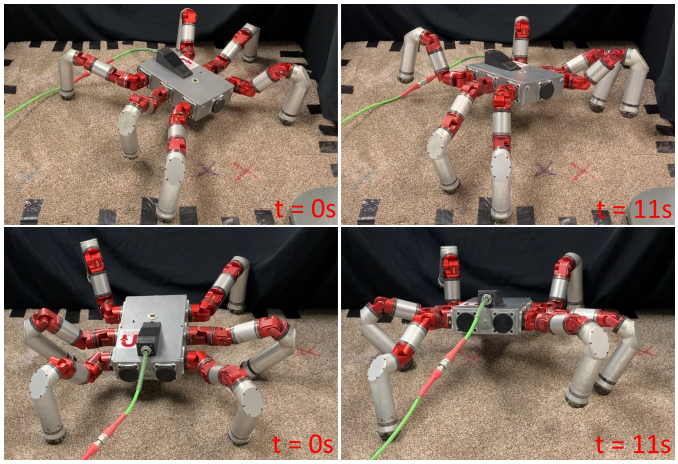


Figure 9. Example start/end robot configuration (up/down rows), showing body stabilization from two random initial poses and respective times to stabilize, using the trained distributed policy.

V. Discussion

In general, we observe that our distributed approach generates control policies which are more scalable and robust to failures (i.e. removing legs from the hexapod robot), and easy to extend by making small modifications to the network architecture. Additionally, we observe at least equivalent performance with conventional controllers, and in some cases even improved performance. Due to the nature of deep and convolutional neural networks, our approach should easily extend to incorporating higher dimensional sensors and camera inputs, by simply modifying the network architecture. By varying the environment in training, a network with enough capacity should also be able to better adapt to unseen, possibly time-varying, more complex environments, but it remains to be seen if training is still viable when accounting for this additional complexity. Distributed reinforcement learning approaches seem to offer a way to create control strategies which allow for robust coordination of complex robotic systems without needing to excessively hand-tune the algorithm for a specific robot topology. Robotic systems which exhibit symmetries should be able to leverage this approach, and therefore benefit from this approach in general, either matching or exceeding performance of classical algorithms while increasing robustness and adaptability.

Our results suggest that our distributed reinforcement learning approach creates efficient snake robot locomotion, meeting or exceeding performance of classical control algorithms such as those given in [1]–[4], [30], [33], [35]. Future work will also compare our learning-based results with other, non-shape-based control approaches, such as [31], [32]. In our experiments, we observed that our learned controller reacts to the environment in a similar way to the compliant controller, but visually is able to react more quickly to obstacles, and to hold the frequency and amplitude at set points that allow it to more easily navigate through obstacles. Additionally, the learned controller seems to better utilize the environment, pushing more strongly against the pegs to create a larger propelling force.

Regarding our hexapedal locomotion results, we first note that the task of leveling the hexapod's body is not only nonlin-

ear, but also very highly dimensional, with multiple paths for achieving stability, greatly complicating the task for multiple agents to reach consensus and perform beneficial actions to stabilize the body. Since a single agent has no control over the other portions of the robot, and cannot instruct other agents to do any specific action, the difficulties are amplified by the agents' need to collaborate without communication. Despite these challenges, the proposed framework manages to reach a satisfactory result after relatively few training episodes (corresponding to around 45 minutes of wall clock time).

In the centralized policy, the challenge of coordination between legs/agents is removed, as a single agent controls the entire robot. However, due to the inherent difficulty of the task, primarily as a result of the immense action space (729 actions), standard approaches using a centralized controller simply cannot train the agent fast enough. This is shown in Figure 8, where even after 500 episodes–corresponding to more than three hours of real world training–the centralized policy had not shown any signs of improvement). By breaking the state-action space into small regions in which each single agent of the shared environment can learn, we leverage the task homogeneity to distribute the learning, decreasing the total training time and increasing performance.

The learned controller for the hexapod robot is able to closely match the performance of a hand tuned controller given in [43], quickly flattening the robot body. While the performance of the learned controller does not visually seem to exceed that of the hand tuned controller, the learned controller can be more easily modified to account for additional sensory inputs or alternative goals, through the modification of the reward function (such as minimizing the shaking of a mounted camera tightly attached to the robot's body).

VI. Conclusion

This paper this work explicitly explores the potential relationship between the underlying concepts in distributed learning and decentralized control of articulated robots. In particular, we consider the problem of adapting the shape parameters of an articulated robot in response to external sensing, in order to optimize its locomotion through unstructured terrain, by casting this problem as a Markov decision problem (MDP). To solve this MDP, we propose to use a multi-agent extension of the A3C algorithm, a distributed learning method whose structure closely matches the decentralized nature of the underlying locomotion controller. That is, our approach leverages the decentralized structure of the underlying robot control mechanism, in order to learn a homogeneous policy where a centralized approach would usually not be tractable/trainable.

We first detail how a snake robot can be trained offline using hardware experiments implementing an autonomous decentralized compliant control framework. There, our experimental results show that the trained policy outperforms the conventional control baseline by more than 40% in terms of steady progression through a series of randomized, highly cluttered evaluation environments (Section III). Note that, for this case of serpenoid locomotion, a comparison with a centralized controller would be more difficult, as the number of control windows on the snake changes with time, making the creation of a centralized controller non-trivial. Second, we identify learning agents to each of the legs of a hexapod robot, and train agents online to stabilize the robot while standing and locomoting inclined and unstructured terrain. There, we show how distributed learning enables time-efficient, online learning, whereas a centralized learning approach could not converge due to the associated large-dimensional state-action space (Section IV). Our results suggest that this approach can be adapted to many different types of articulated robots, by controlling some of their independent parts (e.g., groups of joints, limbs, etc.) in a distributed manner, while fundamentally implementing the same control mechanism in each part.

In general, our distributed approach allows us to learn decentralized policies which naturally exhibit interesting scalability properties. For example, the policy learned for each leg of the hexapod in Section IV naturally adapts to a quadruped without any additional training. We envision that this natural scalability could benefit an autonomous legged robot experiencing joint/limb failures in the field, by allowing the other joints/limbs to keep the robot operational. Based on this observation, we believe that our approach can pave the way to the time-efficient learning of robust, decentralized policies for a wide variety of articulated robots, for use in real-world deployments such as search-and-rescue or inspection.

Since our current approach generates a homogeneous policy, we believe it can easily generalize to robots whose body segments are not tightly coupled (like the snake robot in this paper) or can be easily decoupled using existing methods (like the hexapod robot in this paper). However, we acknowledge that the proposed approach cannot be directly applied to some tightly coupled cases where such decoupling strategy may not exist, or where communication of information between limbs/portions of the robot may be necessary. We further note that, in this work, the way an articulated robot is decoupled into limbs/portions has to be performed manually, based on intuition/experience about the task at hand. In future work, we will both look at generating potentially heterogeneous policies for different robot segments as well as at learning how to decouple the robot into portions that can be controlled independently. Additionally, we note that our approach should be able to leverage the use of additional sensory input, and, depending on the task and robot, could improve performance with the addition of cameras, tactile sensors, LiDAR, or other sensors, and we are also interested in exploring these additions in future works. While explored briefly with the hexapod, we are also interested in maximizing the ability of trained policies to transfer to new, unseen environments, and we plan on exploring how the choice of environment in training and deployment affects the agent.

Finally, in future works, we are also interested in applying this approach to other tasks, such as static or mobile manipulation. We believe that we can build upon our approach, which relies on centralized learning but decentralized execution, to help address such problems for high-DoF articulated robots. However, we envision that for such tasks, portions of the robots may need to be more tightly coupled at specific time steps or for specific maneuvers. In this context, the main

challenge will likely be to learn and adapt the level of learning/control decentralization with time, based on the state of the system.

ACKNOWLEDGMENT

This work was supported by NSF grant 1704256, and by the 2017 CMU RISS and SURF programs. Detailed comments from anonymous referees contributed to the presentation and quality of this paper.

REFERENCES

[1] M. Tesch, K. Lipkin, I. Brown, R. Hatton, A. Peck, J. Rembisz, and H. Choset, "Parameterized and Scripted Gaits for Modular Snake Robots," *Advanced Robotics*, vol. 23, no. 9, pp. 1131–1158, 2009.

[2] J. Whitman, F. Ruscelli, M. Travers, and H. Choset, "Shape-based compliant control with variable coordination centralization on a snake robot," in *CDC 2016*, December 2016.

[3] M. Travers, C. Gong, and H. Choset, "Shape-constrained whole-body adaptivity," in *International Symposium on Safety, Security, and Rescue Robotics*, 2015.

[4] T. Kano, T. Sato, R. Kobayashi, and A. Ishiguro, "Local reflexive mechanisms essential for snakes' scaffold-based locomotion," *Bioinspiration & biomimetics*, vol. 7, no. 4, p. 046008, 2012.

[5] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural networks*, vol. 21, no. 4, pp. 642–653, 2008.

[6] A. Crespi and A. J. Ijspeert, "AmphiBot II: An amphibious snake robot that crawls and swims using a central pattern generator," in *Proceedings of the 9th international conference on climbing and walking robots (CLAWAR 2006)*, 2006, pp. 19–27.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *AAAI*, 2016, pp. 2094–2100.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[10] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement Learning with Deep Energy-Based Policies," *arXiv preprint arXiv:1702.08165*, 2017.

[11] G. Sartoretti, Y. Shi, W. Paivine, M. Travers, and H. Choset, "Distributed Learning for the Decentralized Control of Articulated Mobile Robots," in *ICRA 2018 - IEEE International Conference on Robotics and Automation*, 2018, pp. 3789–3794.

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[13] G. Sartoretti, Y. Wu, W. Paivine, T. K. S. Kumar, S. Koenig, and H. Choset, "Distributed Reinforcement Learning for Multi-Robot Decentralized Collective Construction," in *DARS 2018 - Inter. Symp. on Distributed Autonomous Robotic Systems*, 2018.

[14] X. B. Peng, G. Berseth, and M. V. D. Panne, "Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning," vol. 35, no. 4, pp. 1–12, 2016.

[15] R. Yamashina, M. Kuroda, and T. Yabuta, "Caterpillar robot locomotion based on Q-Learning using objective/subjective reward," *2011 IEEE/SICE International Symposium on System Integration, SII 2011*, pp. 1311–1316, 2011.

[16] K. Ito and F. Matsuno, "A Study of Reinforcement Learning for the Robot with Many," no. May, pp. 3392–3397, 2002.

[17] S. Ha, J. Kim, and K. Yamane, "Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot," in *2018 15th International Conference on Ubiquitous Robots (UR)*. IEEE, 2018, pp. 348–354.

[18] E. Haasdijk, A. A. Rusu, and A. Eiben, "HyperNEAT for locomotion control in modular robots," in *International Conference on Evolvable Systems*. Springer, 2010, pp. 169–180.

[19] G. Baldassarre, S. Nolfi, and D. Parisi, "Evolution of collective behavior in a team of physically linked robots," *Applications of evolutionary computing*, pp. 207–212, 2003.

[20] D. J. Christensen, U. P. Schultz, and K. Stoy, "A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots," *Robotics and Autonomous Systems*, vol. 61, no. 9, pp. 1021–1035, 2013.

[21] P. Maes and R. A. Brooks, "Learning to coordinate behaviors." in *AAAI*, vol. 90, 1990, pp. 796–802.

[22] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR, abs/1507.06527*, 2015.

[23] L. Buoniu, R. Babuška, B. De Schutter, D. Srinivasan, L. C. Jain, L. Buoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *of Studies in Computational Intelligence*, vol. 310, pp. 183–221, 2010.

[24] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[25] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6382–6393.

[26] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," *arXiv preprint arXiv:1705.08926*, 2017.

[27] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[28] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the eleventh international conference on machine learning*, vol. 157, 1994, pp. 157–163.

[29] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: a critical survey," *Web manuscript*, 2003.

[30] M. Travers, J. Whitman, and H. Choset, "Shape-based coordination in locomotion control," *The International Journal of Robotics Research*, p. 0278364918761569, 2018.

[31] M. Ishige, T. Umedachil, T. Taniguchi, and Y. Kawahara, "Learning Oscillator-Based Gait Controller for String-Form Soft Robots Using Parameter-Exploring Policy Gradients," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6445–6452.

[32] T. Kano, R. Yoshizawa, and A. Ishiguro, "Tegotae-based decentralised control scheme for autonomous gait transition of snake-like robots," *Bioinspiration & biomimetics*, vol. 12, no. 4, p. 046009, 2017.

[33] T. Kamegawa, R. Kuroki, M. Travers, and H. Choset, "Proposal of EARLI for a Snake Robot Obstacle-Aided Locomotion," in *International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, October 2012.

[34] P. Liljeback, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, "Experimental investigation of obstacle-aided locomotion with a snake robot," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 792–800, 2011.

[35] S. Hirose, *Biologically Inspired Robots: Serpentile Locomotors and Manipulators*. Oxford University Press, 1993.

[36] C. Ott, R. Mukherjee, and Y. Nakamura, "Unified Impedance and Admittance Control," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 554–561.

[37] A. Prochazka, D. Gillard, and D. J. Bennett, "Positive force feedback control of muscles," *Journal of neurophysiology*, vol. 77, no. 6, pp. 3226–3236, 1997.

[38] F. Ruscelli, G. Sartoretti, J. Nan, Z. Feng, M. Travers, and H. Choset, "Proprioceptive-Inertial Autonomous Locomotion for Articulated Robots," in *ICRA 2018 - IEEE International Conference on Robotics and Automation*, 2018, pp. 3436–3441.

[39] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," in *Proceedings of the fifth International Conference on Learning Representations*, 2017.

[40] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014.

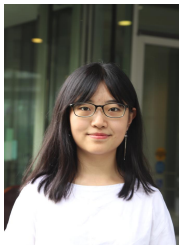
[41] T. de Bruin, J. Kober, K. Tuyls, and R. Babuska, "The importance of experience replay database composition in deep reinforcement learning," *Deep Reinforcement*, pp. 1–9, 2015.

[42] D. Rollinson, Y. Bilgen, B. Brown, F. Enner, S. Ford, C. Layton, J. Rembisz, M. Schwerin, A. Willig, P. Velagapudi, and H. Choset, "Design and architecture of a series elastic snake robot," in *Proceedings of IROS 2014*, 2014, pp. 4630–4636.

[43] G. Sartoretti, S. Shaw, K. Lam, N. Fan, M. Travers, and H. Choset, "Central Pattern Generator with Inertial Feedback for Stable Locomotion and Climbing in Unstructured Terrain," in *ICRA 2018 - IEEE International Conference on Robotics and Automation*, 2018, pp. 5769–5775.
[44] G. Sartoretti, M.-O. Hongler, M. E. de Oliveira, and F. Mondada, "Decentralized Self-Selection of Swarm Trajectories: From Dynamical System Theory to Robotic Implementation," *Swarm Intelligence*, vol. vol. 8, no. no. 4, pp. 329–351, 2014.
[45] N. T. Gridgeman, "Lamé ovals," *The Mathematical Gazette*, vol. 54, no. 387, pp. 31–37, 1970.
[46] G. Loria, "Spezielle algebraische und transzendente ebene Kurven: Theorie und Geschichte. Bd. 1, Die algebraischen Kurven," *BG Teubners Sammlung von Lehrbüchern auf dem Gebiete der Mathematischen Wissenschaften mit Einschluss ihrer Anwendungen*, 1910.
[47] L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 819–824.
[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
[49] G. Lample and D. S. Chaplot, "Playing FPS Games with Deep Reinforcement Learning," in *AAAI*, 2017, pp. 2140–2146.
[50] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction. 1998," *A Bradford Book*, 1998.

Guillaume Sartoretti Guillaume Sartoretti (M'15) is a Postdoctoral Fellow in the Robotics Institute at Carnegie Mellon University, where he works with Prof. Howie Choset. He received his Ph.D. in robotics from EPFL (Switzerland) in 2016 for his dissertation on "Control of Agent Swarms in Random Environments," under the supervision of Prof. Max-Olivier Hongler. He also holds a B.S. and an M.S. degree in Mathematics and Computer Science from the University of Geneva (Switzerland). His research focuses on the distributed/decentralized coordination of numerous agents, at the interface between stochastic modelling, conventional control, and artificial intelligence. Applications range from multi-robot systems, where independent robots need to coordinate their actions to achieve a common goal, to high-DoF articulated robots, where joints need to be carefully coupled during locomotion in rough terrain.

William Paivine William Paivine is an undergraduate student studying computer science at Carnegie Mellon University. He works with Dr. Guillaume Sartoretti and Prof. Howie Choset, with his research interests focusing on the application of distributed reinforcement learning for control of legged robots, artificial intelligence, and planning.



Yunfei Shi Yunfei Shi is a Master student in the Robotics Institute at Carnegie Mellon University, where she works with Dr. Guillaume Sartoretti and Prof. Howie Choset. She received her B.Eng. degree in Electrical Engineering from the Hong Kong Polytechnic University. Her research interest lies at the intersection of distributed reinforcement learning, control, and planning.

Yue Wu Yue Wu is a Sophomore Undergraduate student at Carnegie Mellon University, where he worked with Dr. Guillaume Sartoretti and Prof. Howie Choset. He is studying towards a major in Computer Science and a minor in Machine Learning. His research interest lies in on applied distributed reinforcement learning, computer vision, and probabilistic graphical models.

Howie Choset (F15) received the B.S.Eng. degree in computer science, and the B.S.Econ degree in entrepreneurial management from University of Pennsylvania (Wharton), PA, USA, in 1990, the M.S. and Ph.D. degrees in mechanical engineering from California Institute of Technology (Caltech), CA, USA, in 1991 and 1996, respectively. He is a Professor of robotics with Carnegie Mellon University, Pittsburgh, PA, USA. His research group reduces complicated high-dimensional problems found in robotics to low-dimensional simpler ones for design, analysis, and planning. He directs the Undergraduate Robotics Major at Carnegie Mellon and teaches an overview course on robotics, which uses series of custom developed Lego Labs to complement the course work. He has authored a book entitled *Principles of Robot Motion* (MIT Press, 2015).

Prof. Choset was elected as one of the top 100 innovators in the world under 35, by the MIT Technology Review, in 2002. In 2014, Popular Science selected his medical robotics work as the Best of Whats New in Health Care. Finally, his students received the Best Paper awards at the Robotics Industry Association in 1999 and International Conference on Robotics and Automation (ICRA) in 2003. His groups work has been nominated for Best Paper awards at ICRA in 1997, International Conference on Intelligent Robots and Systems in 2003, 2007, and 2011, and International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines in 2012; won the Best Paper award at IEEE Bio Rob in 2006, International Symposium on Safety, Security and Rescue Robotics 2012 and 2015; won the Best Video award at International Society for Minimally Invasive Cardiothoracic Surgery 2006 and ICRA 2011; and was nominated for the Best Video award in ICRA 2012.