Distributed Learning for the Decentralized Control of Articulated Mobile Robots

Guillaume Sartoretti[1], Yunfei Shi[2], William Paivine[1], Matthew Travers[1], and Howie Choset[1]

***Abstract*— Decentralized control architectures, such as those conventionally defined by central pattern generators, independently coordinate spatially distributed portions of articulated bodies to achieve system-level objectives. State of the art distributed algorithms for reinforcement learning employ a different but conceptually related idea; independent agents simultaneously coordinating their own behaviors in parallel environments while asynchronously updating the policy of a system- or, rather, meta-level agent. This work, to the best of the authors' knowledge, is the first to explicitly explore the potential relationship between the underlying concepts in homogeneous decentralized control for articulated locomotion and distributed learning. We present an approach that leverages the structure of the asynchronous advantage actor-critic (A3C) algorithm to learn decentralized control policies on a single platform. Our primary contribution shows an individual agent in the A3C algorithm can be defined by an independently controlled portion of the robot's body, thus enabling distributed learning on a single platform for efficient hardware implementation. We show how the system is trained offline using hardware experiments implementing a state-of-the-art controller. Our experimental results show that the trained agent outperforms the compliant control baseline by more than 40% in terms of steady progression through a series of randomized, highly cluttered evaluation environments.**

I. Introduction

Due to their high degrees of freedom, articulated mobile robots such as snake robots excel at locomoting through a large variety of environments by leveraging their ability to conform to the shape of surrounding terrain [1]. For such robots, recent results have shown that a decentralized control architecture can improve locomotion through unstructured and cluttered environments [2], [3]. Decentralized control is achieved by partitioning the robots' body into spatially distributed portions, which can then be coordinated by a set of coupled central pattern generators [4], [5]. In deep Reinforcement Learning (RL), new state-of-the-art methods are based on a similar idea and distribute the learning task to several agents which asynchronously update a common, global policy. In this paper, we present a learning approach that leverages the general structure of the state-of-the-art asynchronous distributed learning algorithms [6] as a natural means to learn decentralized control policies for a single platform. In particular, we focus on the A3C algorithm among all asynchronous distributed learning algorithms [6] because it allows us to learn a stochastic policy (rather than a deterministic one such as a Q-table), which is usually more robust to the many sources of noise and uncertainty that are integral to any hardware implementation [7]. Our primary contribution shows a low-level worker in the A3C algorithm can be defined as one of the independently controlled portions of the robot's body (Figure 1).

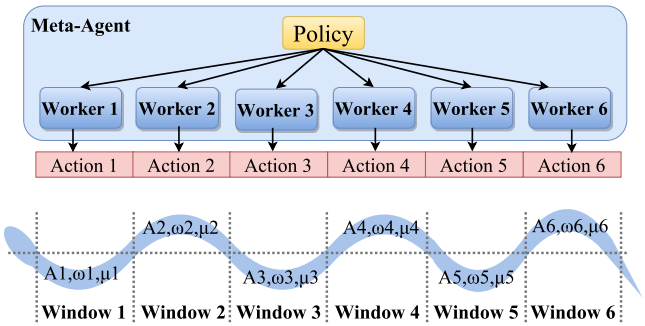

Fig. 1. Analogy between the A3C meta-agent containing multiple workers (top), and a snake robot divided into independent control windows (bottom).

Existing learning-based approaches often consider locomotive tasks at the robot-level [8], [9], [10]. These approaches usually train an agent to select actions that will simultaneously affect all the degrees of freedom of the robot, and which is most likely sampled from a large-dimensional action space. Therefore, these approaches usually require 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 for locomotion [11], [12], but have only considered centralized learning approaches; others have focused on learning heterogeneous controllers in a distributed manner [13], [14]. In either case, however, such learning approaches did not take advantage of running multiple agents to speed up learning. By exploring 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.

The paper is outlined as follows: In Section II, we summarize the general shape-based controller for locomotion, whose feedback controller we seek to replace by a trained agent. Section III outlines the reinforcement problem structure, as well as the agent structure used in this work. We

[1]G. Sartoretti, W. Paivine, M. Travers, and H. Choset are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213, USA. {gsartore,wjp,mtravers}@andrew.cmu.edu, choset@cs.cmu.edu

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

describe the training procedure of our agent in Section IV. We then validate the learned policy experimentally on a snake robot, and discuss the results in Section V. Concluding remarks and future works are finally presented in Section VI.

II. Background - Decentralized Shape-Based Control for 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. Additionally, we overview recent results that highlight the benefits of decentralized control of articulated robots during locomotion. In this paper, the fact that the considered decentralized control mechanism is common across the independent parts of the robot is leveraged to teach agents a common policy in a distributed, time-efficient manner.

A. Serpenoid curves

For an N-jointed snake-like robot, locomotive gaits can be parameterized by two sine waves propagating through the two planes (lateral and dorsal) of the snake [15], [1], [2]. One wave controls joints that rotate in the dorsal plane of the snake, while the other wave controls the lateral joints:

$$\begin{cases} \theta_i^{lat}(t) = \theta_0^{lat} + A^{lat}\sin(\omega_S^{lat} s_i^{lat} - \omega_T^{lat} t) \\ \theta_i^{dor}(t) = \theta_0^{dor} + A^{dor}\sin(\omega_S^{dor} s_i^{dor} - \omega_T^{dor} t + \phi), \end{cases} \tag{1}$$

where θ_i^{lat} and θ_i^{dor} are the commanded joint angles on the lateral and the dorsal plane, θ_0^{lat} and θ_0^{dor} the angular offset, A^{lat} and A^{dor} the amplitude of the curvatures, ϕ the phase shift between the two planes sine waves. ω_T^{lat} and ω_T^{dor} are the temporal frequencies, while ω_S^{lat} and ω_S^{dor} describe the spatial frequencies of the waves, which determines the number of waves on the snake robot's body. $s_i^{lat} \in \{0, 2\cdot l_s, \dots, N\cdot l_s\}$ and $s_i^{dor} \in \{l_s, 3\cdot l_s, \dots, (N-1)\cdot l_s\}$ are the distances from the head of the module i, where l_s is the length of one module.

These parameters are used to define different open-loop joint trajectories for locomotion, i.e., gaits [16]. In this work, we consider the planar slithering gait, for which $A^{dor} = \theta_0^{dor} = 0$ in Eq.(1).

B. 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 $\theta(t)$ as a function of a small number of control parameters. Those parameters lie in 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 can be dynamically set. The shape function $h(A, \omega_S) = \theta(t)$ of our system is simply the serpenoid curve Eq.(1), with the two control parameters being the amplitude and spatial frequency, while the other parameters are fixed:

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

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 [17]. 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:

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

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 joints' angles $\theta_i(t)$ are then computed at each time 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 $J(h) \in \mathbb{R}^{2\times N}$:

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

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

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

which is used to update the two-dimensional shape-based compliant controller Eq.(3).

In this paper, we seek to replace the admittance controller Eq.(3), which iteratively adapts the shape parameters based on external readings, 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 agent to reason about and make action in the shape space of the considered robot.

C. Decentralized Control

Decentralized control extends previous works on shape-based locomotion [2]. It uses activation windows, positioned along the backbone curve of the snake, in order to create independent groups of neighboring joints, in which torques are sensed and motion parameters are adapted. That is, only the joints covered by the same window are coupled, isolating them from other joints in the snake. This framework allows each separate portion of the snake to react independently against local forcing. Any impact occurring 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 [18].

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 backbone of the snake. 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 node points of the serpenoid curve Eq. (1). 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 blind snake that has to move through an unknown terrain, so that different portions of the snake can react independently [2]. Moreover, these results have considered the case where windows are not fixed to the snake's backbone, but are moved along the snake at the same velocity as the serpenoid wave, in order to pass information back down the body. In this paper, we consider the state-of-the-art decentralized control of the snake with sliding windows, while shape parameters in each window are updated by a trained agent.

III. Policy Representation

In this section, we detail the structure of the deep learning meta-agent, as well as that of the lower-level worker agents. Specifically, we look to the A3C algorithm to distributively learn a common stochastic policy [6], which will then be followed by the group of 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 set of actions selected by all the workers, is shared among all workers.

A. State Space

The state s is a 7-tuple of shape-based quantities. It contains the current shape parameters $\beta(s,t)$ of a window on the snake (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 $[0;1] \ni \tau(t) = \frac{t \mod T_s}{T_s}$, with $T_s = \frac{2\pi}{\omega_T}$ the period of the serpenoid curve Eq.(1). The 7-dimensional state vector $\eta_j(t)$ for window j reads:

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

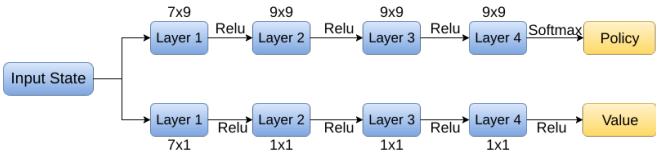

Fig. 2. Policy (Actor) and Value (Critic) network used to approximate their respective functions. Each deep network is fully connected with two hidden layers and ReLU's at the output, except for the output of the actor layer which uses a softmax function to estimate the stochastic policy.

B. Action Space

The admittance controller Eq.(3) lets the shape parameters evolve with continuous increments. In this work, we look to discretizing 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:

$$\beta(j, t+dt) = \beta_j(t) + a_j(t), \quad (8)$$

with $a_j(t)$ the action selected by agent $j \in 1, ..., W$.

C. 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 Actor and Critic networks are fully connected with two hidden layers. 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 network structure is shown in Figure 2.

Six workers, which correspond to the six independent windows along the backbone of the snake[1], 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, each worker updates the global net and collects the new global weights.

We used an entropy-based loss function introduced in [19] to update the policy:

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

The advantage is calculated using $(R_t - V(s_t; \Psi_A))$ where R_t is the estimated discounted reward over time t. $H\left(\pi(s_t; \Psi_A)\right)$ is the entropy factor used to encourage exploration in training and $\kappa \in \mathbb{R}$ helps manage exploration and exploitation. The value loss function reads:

$$f_v(\Psi_C) = \left(R_t - V(s_t; \Psi_C)\right)^2, \quad (10)$$

The stochastic policy is optimized using policy gradient. During training, we calculate the gradients of both functions and optimize the functions using the Adam Optimizer [20].

[1] As mentioned in Section II-C, the current shape of the snake influences the number and placement of the independent windows along the backbone of the snake. 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 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.

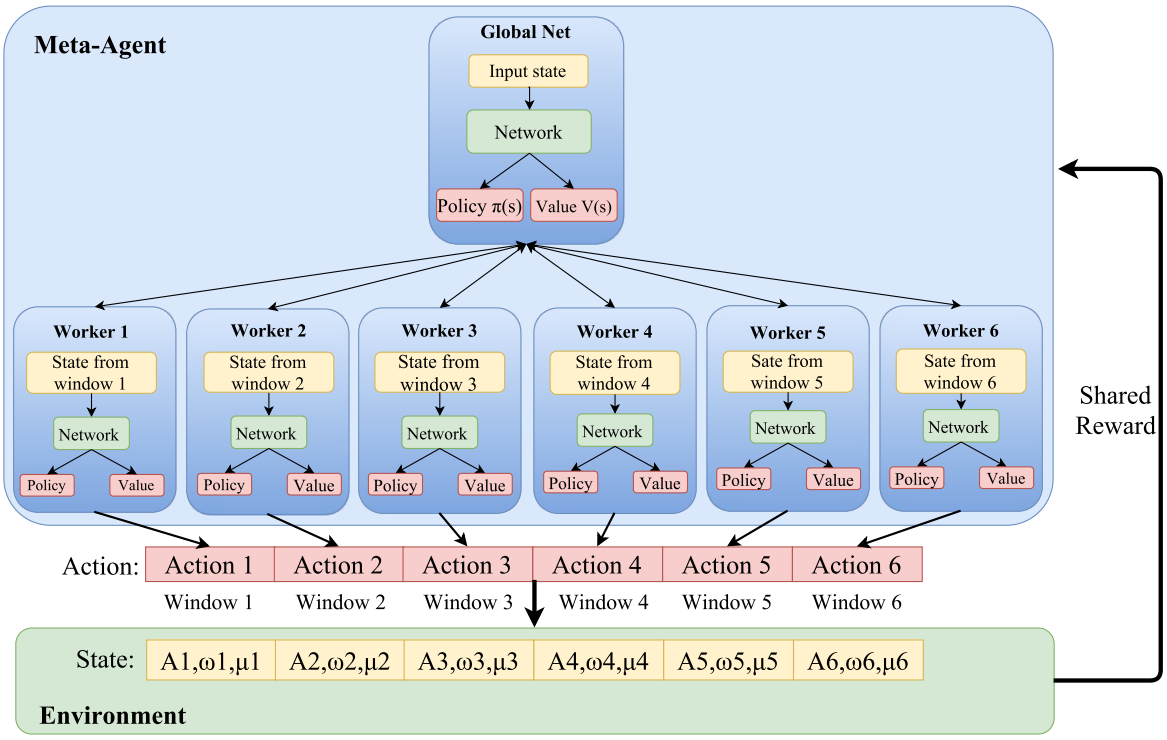

Fig. 3. Structure of the A3C Meta-Agent, with global weights for the Actor-Critic network. Each internal agent (worker) updates its local weights based on shared rewards during local interactions with the environment, and regularly pushes its gradients to the global networks. Each worker draws its state from that of each window on the snake robot. Rewards are calculated in response to a vector of the 6 agents' actions that is enacted on the robot.

D. 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 $X_{0,i}$:

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

with $X(t)$ the current position of the robot in the world and λ_r a scaling factor. Note that the reward is normalized by using the $\tanh$ function, in order to avoid unwanted spikes in rewards, e.g., when the robot suddenly boosts forward due to dynamics 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. The global structure of the A3C meta-agent is illustrated in Figure 3.

IV. Learning

To effectively learn the policy Ψ 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 Ψ 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 the learning agent has proven to be successful in both stabilizing and improving the learned policy [21]. We use elements of this idea to train our learning agent more effectively.

A3C is inherently an on-policy learning algorithm, which can be used to train an agent 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 the near-optimal policy generally allows for approximate, yet beneficial updates to the policy. To this end, we propose to train the agent by replaying past experiences, collected from the state-of-the-art compliant controller.

In order to build the experience replay database, we performed 310 trials of the snake robot in a randomized peg array, in order to record both actions which result in positive and negative reward values for each state. 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 step using Eq.(11), by relying on an overhead motion capture system to record the position of the snake from tracking beads placed along the snake's backbone. For each time step, the state and action is recorded as described in Sections III-A-III-B. Each trial is run for approximately 15 seconds in length.

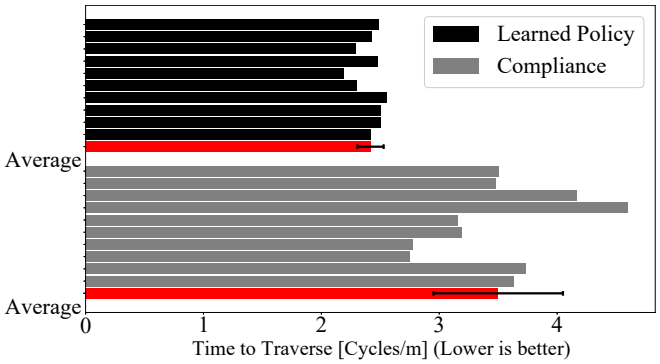

Fig. 4. Experimental results, comparing the performance of both controllers during the 10 trial runs in terms of serpenoid gait cycles needed to progress one meter. Lower values are better (i.e., larger forward progression per gait cycle). The learning-based controller (in black) can be seen to outperform the state-of-the-art compliant controller [2] by more than 40%.

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:

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

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 used in this work is available online `https://github.com/gsartoretti/Deep-SEA-Snake`, along with the database used for offline learning.

V. 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 of Section II-C on a snake robots and ran experiments in randomized unstructured environments, in order to compare the average forward progression of the robot when the shape parameters are updated by the trained agent or the compliant controller Eq.(3).

A. Experimental Setup

We implemented two versions of the decentralized controller Section II-C, 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 [22]. 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).

B. 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 10 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 runs serves as an indicator of the repeatability of the controller. For each run, the same peg array was used for both the compliant controller and the learning-based controller. Additionally, for each pair of runs, the snake was placed in identical starting location as to ensure that both controllers were tested with the same initial configurations.

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.

$$M = \begin{bmatrix} 1.5 & 0 \\ 0 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\beta_0 = \begin{bmatrix} \frac{\pi}{4} \\ 3\pi \end{bmatrix} \qquad \omega_T^{lat} = 1.8 \qquad dt = \frac{\pi}{160}$$

Over the 10 trial runs, the learning-based controller achieves an average of $2.43\,[cycles/m]$ on average, with a variance of 0.092, compared to the compliant controller, which achieves $3.41\,[cycles/m]$ on average, with a variance of 0.42. The results of the compliant controller closely match previous results presented in [2], [3]; in fact, our scoring of the compliant controller seems to be slightly better. The results are compelling in that they show a distinct increase in the forward progression for the learned policy over the admittance controller. These results are summarized in Figure 4. The learned policy outperforms the compliant controller by a significant amount, while the variance is decreased, thus implying that the learning-based controller is also more repeatable. Figure 5 shows how the learning-based controller adapts 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. The videos of the presented experimental results are available online `https://goo.gl/FT6Gwq`.

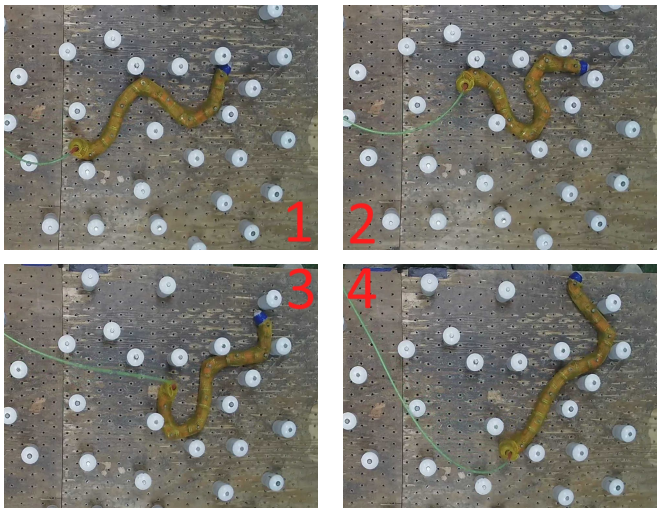

Fig. 5. Example frames sampled from one of the trials, showing the behavior of the learning-based controller in the peg array. Note how the amplitude and frequency of the waves in each of the windows of the snake change to fit the peg array.

VI. Conclusion

In this paper, we consider the problem of adapting the shape of an articulated robot in response to external sensing, in order to optimize its locomotion through unstructured terrain. To this end, we cast the problem of adapting the shape parameters of the robot in response to external torque feedback as a Markov decision problem (MDP). To solve this MDP, we propose to use the A3C algorithm, a distributed learning method whose structure closely matches the decentralized nature of the underlying locomotion controller. We train an agent offline based on experience gathered from a state-of-the-art controller, in order to guide the sampling of the state-action space toward regions of efficient locomotion. The learned policy is then implemented on a snake robot, and the performance is shown to outmatch the current state of the art by more than 40%, on a set of randomized environments.

The proposed approach leverages the decentralized structure of the underlying robot control mechanism, in order to learn a common policy in a more time-efficient manner. This suggests that many articulated robots can be controlled efficiently 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 this context, we propose to leverage the sum of local experience that can be gathered from each part, and rely on a distributed learning algorithm such as A3C to quickly obtain a global policy for the whole robot.

In future works, we will investigate the possibility to implement the proposed approach to online learning. Experience can be gathered by all the workers (i.e., one per independent part of the robot), and stored in an experience database, that can then be used to regularly improve the global policy online. Second, we are interested in applying this approach to other types of robots. In particular, a similar approach could be used to distributively learn a policy for a walking robot, by matching low-level A3C workers to the different limbs of a legged robot.

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] L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," *IEEE International Conference on Robotics and Automation*, pp. 819–824, 2008.

[5] 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)*, no. BIOROB-CONF-2006-001, 2006, pp. 19–27.

[6] 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.

[7] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *arXiv preprint arXiv:1702.08165*, 2017.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

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

[16] D. Rollinson and H. Choset, "Gait-based compliant control for snake robots," in *IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5138–5143.

[17] C. Ott, R. Mukherjee, and Y. Nakamura, "Unified impedance and admittance control," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 554–561.

[18] 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.

[19] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," 2016.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[21] 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. [Online]. Available: https://pdfs.semanticscholar.org/091d/1712b97c0353b9a79e8563e7f174a8fd5450.pdf{%}5Cnrll.berkeley.edu/deeprlworkshop/papers/database{_}composition.pdf

[22] 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 *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 4630–4636.