Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Deep Reinforcement Learning for Snake Robot Locomotion

Junyao Shi [*] **Tony Dear** [*] **Scott David Kelly** [**]

[*] *Department of Computer Science, Columbia University, New York, NY 10027 USA (e-mail: {junyao.shi, tony.dear}@columbia.edu)*
[**] *Mechanical Engineering and Engineering Science, University of North Carolina at Charlotte, Charlotte, NC 28223 USA (e-mail: scott.kelly@uncc.edu)*

Abstract: The design of gaits for underactuated robots is often unintuitive, with many results derived from either trial and error or simplification of system structure. Recent advances in deep reinforcement learning have yielded results for systems continuous in either states or actions, which may extend to a variety of locomoting robots. In this work we employ reinforcement learning to derive efficient and novel gaits for both terrestrial and aquatic multi-link snake robots. Although such systems operate in different environments, we show that their shared geometric structure allows us to utilize the same learning techniques in both cases to find gaits without any human input. We present results learned and rolled out in simulation, and we describe preliminary efforts to implement the entire learning process on a physical system.

Keywords: Mobile robots, Autonomous robotic systems, Intelligent robotics

1. INTRODUCTION

Robot motion planning often turns to biology for inspiration of gaits or locomotion modes, particularly for those systems with unusual morphologies. For example, biological imitation is a powerful technique for enabling snake robot locomotion, which often exhibits some form of slithering or sidewinding motions. However, while biology provides an ideal baseline or starting point, there is no single prevailing method to improve or operate to begin with when there is a lack of data and observations. To address these challenges, we consider how many biological organisms acquire these locomoting skills in the first place—namely, learning.

Reinforcement learning (RL) describes a class of techniques that are perhaps not too different from the learning process that many organisms appear to follow. Actions that produce desired results are encouraged, while actions that are counterproductive are weeded out over time. Genetics and innate ability notwithstanding, organisms must learn to adapt to new environments all the time. In the same way, one way to approach the problem of robot locomotion is to have the system try different actions and build up a repertoire of "good" gaits over time, rather than the roboticist prescribing such a repertoire.

While such an approach is attractive in theory, reinforcement learning has not appeared as a viable technique for robot locomotion until recently. Real robots are nonlinear, high-dimensional, and continuous in their states and actions (controls), and classical reinforcement learning techniques do not scale well for such systems. This paper describes our work in alleviating these challenges in two complementary ways. The first is to turn to recent advances in *deep* reinforcement learning, which take advantage of deep neural networks to approximate the complex learned controls. For this we use established techniques in the literature and apply them to our specific systems.

The second part of our approach is to identify a class of locomoting robots that exhibit structural symmetries and to exploit these symmetries to reduce system complexity *before* the learning process. While we are likely not the first to take advantage of system symmetries in reinforcement learning, an extensive literature has been established specifically for locomoting systems under the purview of *geometric mechanics*, and the goal of our work is to show how such geometric structure can be beneficial to a reinforcement learning approach in addition to traditional motion planning.

Our paper is structured as follows. We first review relevant work in geometric mechanics for locomoting robots and deep reinforcement learning for general problems. We then formulate the geometric models for our example systems, a wheeled terrestrial robot and a multi-link swimming robot. With these models, we describe the learning techniques that we use, along with our setup of the problem of learning simple gaits from a blank slate. Finally, we present our simulated results and show that our formulation is well suited for implementation on a physical system.

2. PRIOR WORK

2.1 Geometric Mechanics

In recent decades, techniques and methods from geometric mechanics have been a popular way to model and control mechanical systems. A key idea is that of *symmetries* in a system's configuration space, which allow for the *reduction* of the equations of motion to a simpler form. This has been addressed for general mechanical systems by Marsden and Ratiu (2013), as well as nonholonomic systems by Bloch et al. (1996). For locomoting systems, geometric reduction is often leveraged in tandem with a decomposition of the configuration variables into actuated shape variables and unactuated position variables. If such a splitting is possible, then the configuration space often takes on a *fiber bundle* structure, whereby a mapping called the *connection* relates trajectories between each subspace. Analysis of the connection can then give us intuition into ways to perform motion planning and control of the system, as detailed by Kelly and Murray (1996) and Ostrowski and Burdick (1998). This mathematical structure also lends itself to visualization and design tools, detailed by Hatton and Choset (2011).

Much of the progress in the geometric mechanics of locomotion is predicated on the assumption that the symmetries of a system coincide exactly with the position degrees of freedom. The three-link robot of Fig. 1 with actuated joint inputs is one of the simplest examples of such a system, and it has received considerable attention from researchers such as Ostrowski (1999) and Shammas et al. (2007) treating it as a *kinematic* system, so named because its three constraints eliminate the need to consider second-order dynamics when modeling its locomotion. This allows for the treatment of the system's locomotion, and subsequent motion planning, as a result of geometric phase (see Murray and Sastry (1993), Mukherjee and Anderson (1993), Kelly and Murray (1995), Bullo and Lynch (2001)).

Geometric methods have also examined systems locomoting in fluids. As with terrestrial systems, such a description is most useful if the position degrees of freedom correspond to system symmetries and the rest to internal shape. For single bodies, motion may be achieved through temporal deformation of the body's shape. For articulated swimmers like a three-link robot, deformation occurs naturally when joints are moved relative to each other (see Melli et al. (2006), Hatton and Choset (2013), Burton et al. (2010)), analogous to the terrestrial version of the system.

2.2 Reinforcement Learning

Reinforcement learning (Sutton and Barto (1998)), or RL, is a sub-field of machine learning that enables agents, such as robots, to learn good policies for sequential decision problems. A goal is typically quantified in the form of a reward function, and the agent aims to maximize its cumulative reward as it interacts with the environment. Mathematically, the RL approach can be represented using a *Markov Decision Process* (MDP): a set of states S, a set of actions A, a transition function $T : S \times A \rightarrow S$, and a reward function $R : S \times A \rightarrow \mathbb{R}$. A policy $\pi : S \rightarrow A$ is learned to maximize the cumulative reward $\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$, where $0 < \gamma < 1$ is a discount factor.

Q-learning (Watkins (1989)) is a popular algorithmic implementation of RL, as it does not require an explicit model of the system (*i.e.*, it is *model-free*), and it can solve problems with stochastic transitions and rewards. The goal of Q-learning is to learn the policy π by learning a real-valued *Q-mapping* $\pi_Q : S \times A \rightarrow \mathbb{R}$. The Q-value of a state-action pairing (s, a) indicates the expected cumulative future rewards received by taking action a at state s.

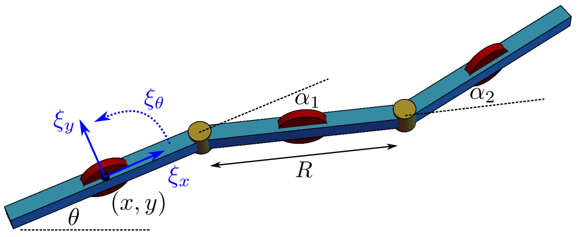


Fig. 1. A three-link nonholonomic snake robot with identical link lengths R. The coordinates (x, y, θ) denote the $SE(2)$ inertial configuration of the proximal link, which also has velocities $(\xi_x, \xi_y, \xi_\theta)$ relative to a body-fixed frame. The relative angles of the joints are denoted $\alpha = (\alpha_1, \alpha_2)$.

Each time a new sample (s_t, a_t, s_{t+1}, r_t) is received, where $s_{t+1} = T(s_t, a_t)$ and $r_t = R(s_t, a_t)$, the *Bellman update equation* is used to update $Q(s_t, a_t)$:

$$\begin{aligned} Q^{\text{new}}(s_t, a_t) =& (1 - \alpha) \cdot Q^{\text{old}}(s_t, a_t) \\ &+ \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)), \end{aligned}$$

where α is a tuned learning rate and γ is a discount factor.

While powerful, classical Q-learning lacks generality because it can only deal with discrete state-action pairings. Recent advances in artificial neural networks have enabled Q-learning to generalize to continuous spaces by combining reinforcement learning with deep neural networks. *Deep Q-Network* (Mnih et al. (2015)), or DQN, is able to solve RL tasks with continuous state spaces and discrete action spaces by leveraging deep neural networks to approximate the optimal action-value function Q^*. Previous algorithms that represented Q^* with a nonlinear functional approximator often diverged due to instabilities caused by correlations in the sequence of observations, correlations between Q^{old} and Q^{new}, and the fact that small updates to Q may significantly change the policy and the data distribution. To address these issues, DQN utilizes randomized experience replay to remove correlations between data, correlations in the observation sequence, and bias in the data distribution. It also uses iterative network updates to alleviate the effects of Q updates to the policy and reduce Q's correlation with its network update target. These improvements enable DQN to effectively solve a wide range of challenging tasks.

2.3 Motion Control of Robots

To achieve efficient snake-like movement in their robotic counterparts, a variety of kinematics-based methods (Hirose (1993), Ma (1999), Tesch et al. (2009)) have been used to simplify parametric representations of their trajectories. However, because these methods largely rely on parameter tuning, they are time-consuming and have limited gait efficiency. RL has recently been adapted to the robotic gait generation and motor control problems, with applications in bipedal robots (Peng et al. (2017)), quadruped robots (Tan et al. (2018)), Minitaur legged robots (Haarnoja et al. (2018)), and multi-fingered robotic hands (Zhu et al. (2019)). Recent studies have attempted to apply new RL techniques to design gaits for snake-like robots as well. Whereas Bing et al. (2019) used RL to learn slithering and energy-efficient gaits for a dynamics-based snake-like robot, Zhang et al. (2018) used RL to make an underwater gliding snake-like robot adapt to its environment and automatically learn gliding actions.

To the authors' knowledge no work has investigated the application of RL to snake robot models derived from geometric mechanics. Furthermore, most prior work in RL and robot locomotion have limited the training phase to simulation, as training is time-consuming in real-world settings. In this work, we use insights in geometric mechanics to both increase the efficiency of our training process and then subsequently analyze the learned gaits, both with an eye toward making the problem suitable for real-world implementation.

3. KINEMATIC ROBOT MODELS

In this section we briefly develop the models of the two representative robots in this paper: a wheeled three-link robot and a neutrally buoyant three-link robot swimming in a low Reynolds fluid. In doing so we will highlight the usefulness of a geometric formulation, as well as common visualizations that can help us better understand their system structure.

3.1 Wheeled Snake Robot

As shown in Fig. 1, the wheeled robot consists of three rigid links, each of length R, which can rotate relative to one another. Its configuration is defined by $q \in Q = G \times B$, where $g = (x, y, \theta)^T \in G = SE(2)$ specifies the position and orientation of the first link in an inertial frame; we measure a link's position at the center of the link. The joint angles $\alpha = (\alpha_1, \alpha_2)^T \in B$ specify the links' relative orientation. We can view Q as a principal fiber bundle, in which trajectories in the shape or base space B lift to trajectories in the group G (Kelly and Murray (1995)).

The wheels at the centers of the links provide a set of nonholonomic constraints that restrict the system's motion. Each of the constraints can be written in the form

$$-\dot{x}_i \sin\theta_i + \dot{y}_i \cos\theta_i = 0, \tag{1}$$

where $(\dot{x}_i, \dot{y}_i)$ is the velocity and θ_i is the orientation of the ith link. These quantities can be found via the system's geometry and written in terms of the configuration coordinates and velocities. We note that the constraints are symmetric with respect to the group in that they do not explicitly depend on where the system is positioned or how it is oriented in space. We can rewrite the constraints into a reduced Pfaffian form (Shammas et al. (2007)) as

$$\omega_\xi(\alpha)\xi + \omega_\alpha(\alpha)\dot{\alpha} = 0, \tag{2}$$

where $\omega_\xi \in \mathbb{R}^{3\times 3}$ and $\omega_\alpha \in \mathbb{R}^{3\times 2}$. The variables $\xi = (\xi_x, \xi_y, \xi_\theta)^T$ give us the body velocity of the system, as shown in Fig. 1. In $SE(2)$, the mapping that takes body velocities to inertial velocities is given by $\dot{g} = T_e L_g \xi$, where

$$T_e L_g = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

Since the number of independent constraints is equal to the dimension of the group, these equations are sufficient to derive a kinematic connection for the system (Shammas et al. (2007)). In other words, the constraint equations

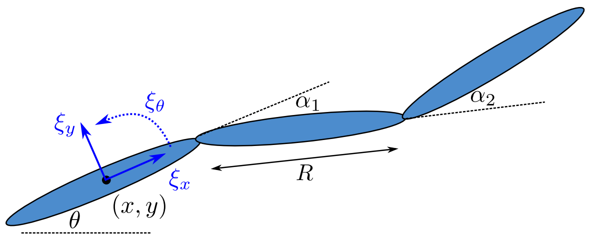


Fig. 2. A swimming snake robot comprised of three articulated slender bodies. The inertial coordinate configuration, body velocity configuration, and relative joint angles are all identical to that of the robot in Fig. 1.

fully describe the first-order dynamics of the group variables in terms of the shape variables only. Thus, Eq. (2) can be rearranged to show this explicitly as the *kinematic reconstruction equation*:

$$\xi = \underbrace{-\frac{1}{D}\begin{pmatrix} \frac{R}{2}(\cos\alpha_1 + \cos(\alpha_1 - \alpha_2)) & \frac{R}{2}(1 + \cos\alpha_1) \\ 0 & 0 \\ \sin\alpha_1 + \sin(\alpha_1 - \alpha_2) & \sin\alpha_1 \end{pmatrix}}_{\mathbf{A}(\alpha)} \dot{\alpha}, \tag{4}$$

where $D = \sin\alpha_1 + \sin(\alpha_1 - \alpha_2) - \sin\alpha_2$. $\mathbf{A}(\alpha)$ is called the local connection form, a mapping that depends only on the shape variables, in this case α_1 and α_2. Note that this quantity is not well defined when $\alpha_1 = \alpha_2$, which corresponds to a singular configuration of the system.

3.2 Swimming Snake Robot

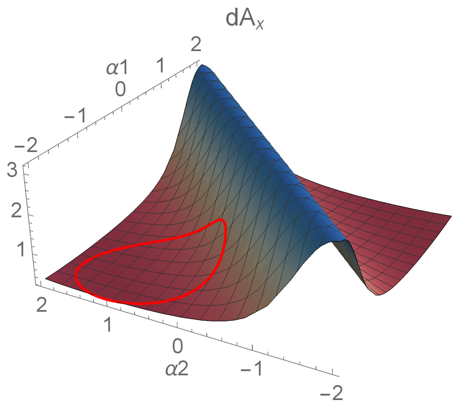
The presence of singularities is a challenge for motion planning for the nonholonomic snake robot, and a partial treatment is detailed by Dear (2018). In this paper we explicitly prescribe joint limits during learning to avoid computational difficulties during simulation. Since this also prevents the robot from using its full range of motions, we also demonstrate learning results on a swimming version of the three-link robot, which has no singularities. This robot, as shown in Fig. 2, is physically very similar to the wheeled version, but the kinematic model is derived in a different manner.

Following the treatment of Hatton and Choset (2013), we assume that the swimmer is comprised of three slender bodies and suspended in a planar low Reynolds number fluid, in which viscous drag forces dominate inertial forces. This allows us to approximate the drag forces as linear functions of the system's body and shape velocities ξ and $\dot{\alpha}$; we also assume that net forces acting on the system are zero for all time due to damping out by drag forces. Just as with the nonholonomic robot, we can derive a *Pfaffian constraint* on the swimming system's velocities as

$$F = (F_x, F_y, F_\theta)^T = (0, 0, 0)^T = \omega_1(\alpha)\xi + \omega_2(\alpha)\dot{\alpha}, \tag{5}$$

where $\omega_1 \in \mathbb{R}^{3\times 3}$ and $\omega_2 \in \mathbb{R}^{3\times 2}$ as before.

The full forms of these components can be found in Hatton and Choset (2013). The general approach would be to first compute local drag forces on each link, and then combine them to find the total force components for each of the body frame directions. In addition to the system link length R, the kinematics also utilize the drag



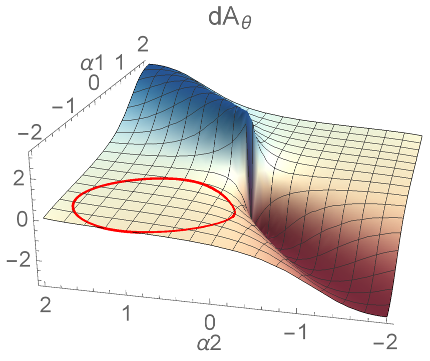


Fig. 3. Visualizations of the x and θ components of the connection exterior derivative for the wheeled three-link snake robot.

constant of the fluid, characterized by k. Finally, once the Pfaffian equations are derived, a kinematic reconstruction equation and a connection form can be written down for the swimmer in exactly the same form as that of the wheeled robot: $\xi = -\mathbf{A}(\alpha)\dot{\alpha}$.

3.3 Visualization of Kinematics

The structure of the connection form in Eq. (4) can be visualized in order to understand the response of ξ to input trajectories without regard to time, according to Hatton and Choset (2011). We can first integrate each row of Eq. (4) over time to obtain a measure of displacement corresponding to the body frame directions. In the world frame, this measure provides the exact rotational displacement, *i.e.*, $\dot{\theta} = \xi_\theta$ for the third row, and an approximation of the translational component for the first two rows. If our input trajectories are periodic, we can transform this "body velocity integral" into one over the trajectory $\psi : [0, T] \to B$ in the joint space, since the kinematics are independent of input pacing. Stokes' theorem can then be applied to perform a second transformation into an area integral over β, the region of the joint space enclosed by ψ:

$$-\int_0^T \mathbf{A}(\alpha(\tau))\dot{\alpha}(\tau)\, d\tau = -\int_\psi \mathbf{A}(\alpha)\, d\alpha = -\int_\beta \mathrm{d}\mathbf{A}(\alpha). \tag{6}$$

The integrand in the rightmost integral is the exterior derivative of $\mathbf{A}$, computed as the curl of $\mathbf{A}$ in two dimensions. For example, the connection exterior derivative of Eq. (4) has three components, one for each row i given by

$$\mathrm{d}\mathbf{A}_i(\alpha) = \frac{\partial \mathbf{A}_{i,2}}{\partial \alpha_1} - \frac{\partial \mathbf{A}_{i,1}}{\partial \alpha_2}, \tag{7}$$

where $\mathbf{A}_{i,j}$ is the element corresponding to the ith row and jth column of $\mathbf{A}$.

The magnitudes of the connection exterior derivative over the joint space are depicted in Fig. 3[1], along with a gait trajectory shown as a closed curve on the surfaces. The area integral over the enclosed region is the geometric phase, a measure of the expected displacement in the body x and θ directions (the body y plot is not shown because it is zero everywhere). The x plot is positive everywhere, meaning that any closed loop will lead to net displacement along the ξ_x direction. In particular, a trajectory that advances in a *counter-clockwise* direction over time in joint space will yield positive body-x displacement, since that corresponds to a positive area integral; negative body x-displacement is achieved with a *clockwise* trajectory. The θ plot is anti-symmetric about $\alpha_1 = -\alpha_2$, meaning that gaits symmetric about this line will yield zero net reorientation while simultaneously moving the robot forward. Note that the magnitudes in both plots become unbounded closer to the singular configurations $\alpha_1 = \alpha_2$.

Finally, since we have derived a connection form for the swimming robot as well, we can also visualize the exterior derivative plots for this system. Contour versions of these plots are shown overlapped with simulation results of learned gaits in Fig. 5. We first note that unlike the connection form of the wheeled robot, these plots (and the connection form itself) are continuous over all joint angles; no singularity region appears at any joint angle configuration. Secondly, we see that unlike the wheeled robot the y plot is not null; it is indeed possible for the swimmer to move laterally to its body frame over a complete gait cycle.

4. REINFORCEMENT LEARNING FOR LOCOMOTION

While the derived geometric models give us an intuition behind locomotion of robots *given* a gait, it may not be easy to *design* an optimal or efficient gait, particularly for robots with high-dimensional joint spaces. While the robots presented in this paper are both low-dimensional, our objective is to use reinforcement learning to help the robots learn good gaits, which are then verified and improved upon via our geometric analysis. Since our reinforcement learning approach is model-free, it can be easily extended to robots with higher dimensions.

We begin this section by introducing the key components of the RL setup for our particular robots, followed by the reinforcement learning algorithm itself. We also describe throughout our approach of utilizing the intuition of geometric mechanics to simplify the RL components so as to make the problem feasible in a real-world setting. Training in the real world, as opposed to in simulation,

[1] We plot a scaled arctangent of these functions in order to visualize the singular portions. Instead of $\mathrm{d}\mathbf{A}_i(\alpha)$, we plot $\frac{1}{c}\arctan(c\,\mathrm{d}\mathbf{A}_i(\alpha)))$, where c is positive.

may be more desirable so as to account for effects like friction or wheel-slip.

4.1 Reinforcement Learning Setup

State Space For both the wheeled and swimming robots, we use a three-dimensional state space S, parameterized by $(\alpha_1, \alpha_2, \theta)$, where α_1, α_2 corresponds to the robot's relative angles of the robot's two joints, and θ is the orientation of the robot in the inertial frame. The first two quantities, α_1 and α_2, describe the robot's joint angle configuration, on which the RL agent applies joint velocity "actions" to transition into another angle configuration.

The orientation θ keeps track of the robot's relative orientation to the inertial frame, which is necessary if our objectives concern locomotion relative to the inertial frame (*e.g.*, motion going to the right). Alternatively, one can choose to prescribe the robot's tasks in the body frame, for which θ would no longer be needed. Note that in either case, we are explicitly taking advantage of symmetries present in the system by disregarding the need to include x and y in the state space. While seemingly trivial, this design decision significantly improves the efficacy of the learning procedure and is hugely beneficial in systems with obscure symmetries.

When necessary, the state space of the robot can be constrained within certain values. For example, physical snake robots typically have joint limits at $-\pi$ and π to avoid overlapping links. Moreover, we noted that the wheeled nonholonomic snake robot has a singularity at the $\alpha_1 = \alpha_2$ configuration, requiring us to avoid such a scenario in simulation. In practice, we constrain one joint angle to positive values and the other to negative values to ensure that the learning agent never achieves such a configuration.

Action Space We use a discrete two-dimensional action space A, parameterized by $(\dot{\alpha_1}, \dot{\alpha_2})$, each of which corresponds to a joint velocity of the robot. For simplicity, we assume that each joint has the same max action magnitude $\dot{a}_{\max}$ and action discretization interval $\dot{a}_{\text{interval}}$. The reason for discretizing the action space in simulation is twofold: first, we aim to translate our simulation results to physical robots, and the motors of the robot (shown in Fig. 6) require discrete joint velocities inputs; second, a discrete action space allows us to easily exploit the system's symmetries (see section 5.1), drastically reducing the size of the action space. That DQN has to search a smaller space for a good policy is crucial for achieving results comparable to simulation when training physical robots in the real world. Therefore, given $\dot{a}_{\max}$ and $\dot{a}_{\text{interval}}$, the action space can be expressed as $A = A_1 \times A_2$, where $A_i = \{\dot{a}_{\max}, \dot{a}_{\max} - \dot{a}_{\text{interval}}, \ldots, 0, \ldots, -\dot{a}_{\max} + \dot{a}_{\text{interval}}, \dot{a}_{\max}\}$. The action $(0, 0)$ is removed from A to prevent the robot from staying still. This last design choice also derives from the robots' kinematics, as we know that a null input leads to no movement.

Reward Function The reward function of the RL agent depends on the objective of the RL task. The advantage of the similarity between the two robots' kinematic models is that if we have identical or similar objectives for both robots, then the same reward function can be applied to them or other variations of these robots.

To train the neural networks for moving the robot forward and backward, we apply the reward function for executing action a at state s

$$R(s, a) = c_1 \cdot \Delta x - c_2 \cdot P_0 + c_3 \cdot R_\theta, \tag{8}$$

where c_1, c_2, and c_3 are positive constants used to weight each reward or penalty variable, Δx is the robot's x-displacement in the desired direction (positive for forward, negative for backward), $P_0 = 1$ is the penalty for having zero x-displacement, and R_θ is the reward variable for the robot's orientation after executing action a. Based on θ_{new}, the orientation of the robot after each step is defined as

$$R_\theta = \begin{cases} 1 & \text{if } -\dfrac{\pi}{4} \le \theta_{\text{new}} \le \dfrac{\pi}{4}, \\ \dfrac{\pi}{4} - |\theta_{\text{new}}| & \text{otherwise.} \end{cases} \tag{9}$$

The penalty variable P_0 helps accelerate the learning process by preventing the robot from staying still or converging at a local minimum. The reward variable R_θ is a positive constant when θ_{new} is within a desired range, otherwise a linearly increasing penalty is incurred. It is included to encourage the robot to stay in the desired orientation, thereby maximizing its displacement in the laboratory frame x-direction.

To train the neural networks for reorienting the robot left and right, we apply the reward function for executing action a at state s

$$R(s, a) = c_1 \cdot \Delta\theta - c_2 \cdot P_0, \tag{10}$$

where c_1, c_2 are positive constants used to weight each reward or penalty variable, $\Delta\theta$ is the robot's θ-displacement in the desired direction, and $P_0 = 1$ is the penalty for having zero θ displacement. Again, the rationale to include P_0 is to achieve better and faster learning convergence.

Simulation Method To simulate the transition of the robot from one state to another as a result of taking an action, as well as to obtain the values of parameters for associated rewards, we perform integration of ordinary differential equations described in Eq. 4 for the wheeled system and in Eq. 5 or the swimming system.

4.2 Reinforcement Learning Algorithm

We have chosen to use Deep Q-Network algorithm (Mnih et al. (2015)) to train our snake robot RL agents for several reasons: it has been proven to perform well for problems with a discrete action space and continuous state space; it is model-free, so we can apply our algorithm to a physical robot without knowledge of its model; it is off-policy, so the agent can efficiently utilize past experiences generated by a policy different from the current one to learn, which helps the agent to achieve high performance with fewer samples. This is shown in Algorithm 1.

We first run Algorithm 1 with a small t_{interval}(number of seconds per action) value and high number of total iterations, which allows the robot to learn granular gaits very slowly. Then we overlap the gaits in joint space atop the connection exterior derivative plots described in Fig. 3 (see Fig. 5 for example). t_{interval} is increased based on the periodic patterns of the plot to avoid repetitive actions within a single period while still maintaining roughly the same area enclosed (and thus amount of displacement). We verify the robots can learn faster by increasing t_{interval} value accordingly and decreasing the number of total iterations, but still achieving the same amount of displacement in policy rollout.

Algorithm 1 Deep Q-learning for snake robots

```
Initialize replay memory $D$ to capacity $N_{\text{memory}}$
Initialize $Q$ and $\hat{Q}$ with random weights $\theta$ and $\theta^- = \theta$
for episode $e = 1, M$ do
    Randomly initialize $s_1 \in S$
    for iterations $i = 1, T$ do
        Select an action $a_t \in A$ by $\epsilon$-greedy approach
        Execute $a_t$; observe reward $r_t$ and next state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random minibatch of $N_{\text{minibatch}}$ transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
        Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
        Perform gradient descent step on
        $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
        Every $C_{\text{update}}$ steps reset $\hat{Q} = Q$
    end for
end for
```

4.3 Network Architecture

Our Q-network is a nonlinear function approximator for the true action-value function Q^*. Our implementation consists of an input layer, two fully connected hidden layers, and an output layer. The input layer has five dimensions, comprised of the concatenation of the state and action. We have found that a relatively simple network suffices for our task, with 50 ReLU units in the first hidden layer and 10 in the second. The final output layer uses a linear activation function, and it outputs the Q-value associated with the inputted state-action pairing. We employ the DQN algorithm adapted from Mnih et al. (2015) to train the network.

4.4 Training Configuration

The training parameters for the physical and wheeled robots are shown in Table 1. The values of all the hyper-parameters were selected by performing a systematic grid search on the wheeled system. The parameters are then fixed for both systems, since the only difference between them, structurally, is their kinematics.

5. RESULTS AND DISCUSSION

We present our results of learning gait primitives for the tasks of moving forward and reorientation for both the wheeled and swimming robots. Once learned, such primitives can easily be combined into more sophisticated gaits by simply querying the corresponding neural networks. We will also show that the speed and effort of the training process for our robot models are very efficient, which lends well to physical implementation in future work.

Table 1. Hyperparameters used in the RL training process.

Hyperparameter	Value	Description
N_{memory}	250	replay memory size
M	10	total number of episodes
T	500	iterations per episode
ϵ_0	1	initial value of ϵ
τ_{decay}	0.99954	rate of decay of ϵ
C_{memory}	50	replay start size
$N_{\text{minibatch}}$	8	minibatch size
γ	0.99	discount factor
C_{update}	20	network update frequency
$\dot{a}_{\max}$	$\pi/8$	maximum action magnitude
$\dot{a}_{\text{interval}}$	$\pi/8$	action discretization interval
lr	0.0002	RMSProp learning rate
c_1	10	used to weight Δx or $\Delta\theta$
c_2	10	used to weight P_0
c_3	1	used to weight R_θ
t_{interval}	4	seconds per action

5.1 Gait Generation

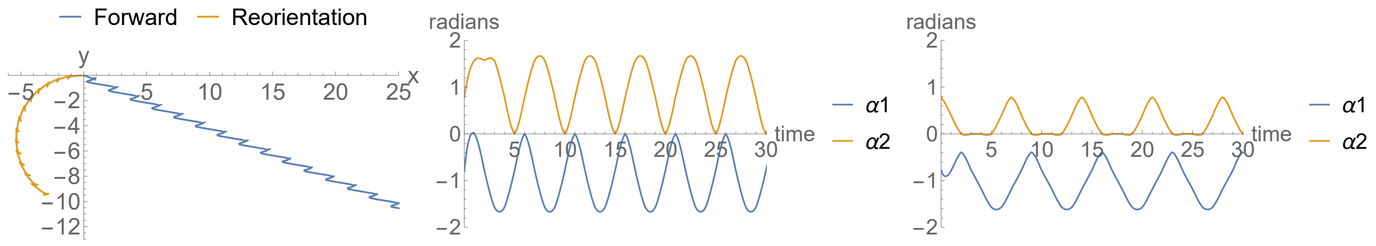
The gaits and locomotion of our snake robot agents are shown in Fig. 4, the wheeled robot in Fig. 4a and the swimming robot in Fig. 4b. For each robot, we assign four tasks in the laboratory frame: translating forward, translating backward, rotating left, and rotating right. From the robot's kinematics, we know (and have verified) that the gaits for forward and backward translation are equal and opposite, and similarly for positive and negative reorientation. We therefore only show results for forward and positive reorientation.

To achieve forward locomotion, we found that about 5,000 iterations were sufficient to achieve convergence for both robots. They are then evaluated by executing the learned forward gaits for 200 seconds and rotation gaits for 400, where the maximum magnitude of the action is $\dot{a}_{\max} = \frac{\pi}{8}$ radians per second. Each action takes $t_{\text{interval}} = 4$ seconds for the swimming robot and $t_{\text{interval}} = 2$ seconds for the wheeled robot. As shown in the plots, both robots are able to learn gaits that are sinusoidal with various differences in phase to achieve the different locomotion primitives. While the swimming robot manages to minimize displacement in components other than the desired one for all tasks, the wheeled one does not due to limitations placed on its joints. These results are consistent with prior analyses of these robots' locomotion.

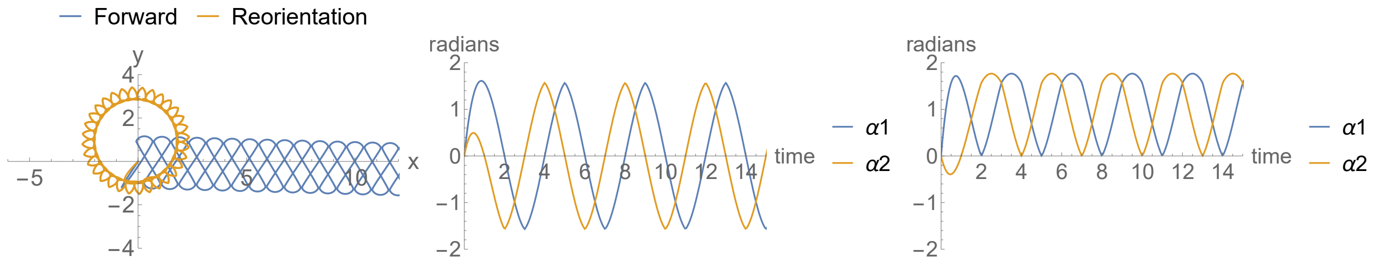
5.2 Implications for Physical Experimentation

Our ultimate goal is to produce feasible learning algorithms for under-actuated and high-dimensional physical robots. Whereas the difference between thousands and millions of iterations may be negligible in simulation, these can mean a difference between months and days when dealing with physical motors and encoders. We believe that our results, achieved in about 5,000 learning iterations at their best, can be realistically implemented on a physical robot, and this is a focus of our ongoing work.

While we have already described our simplification of the state space using geometric insights, we have found that our specifications of the action space have a large impact as well. Since we expect that emergent gaits will take a periodic form and enclose areas of the connection exterior



(a) Left: Forward and reorientation workspace trajectories of the wheeled robot. Middle and right: Corresponding joint trajectories.



(b) Left: Forward and reorientation workspace trajectories of the swimming robot. Middle and right: Corresponding joint trajectories.

Fig. 4. Resultant gaits learned for the wheeled and swimming robots.

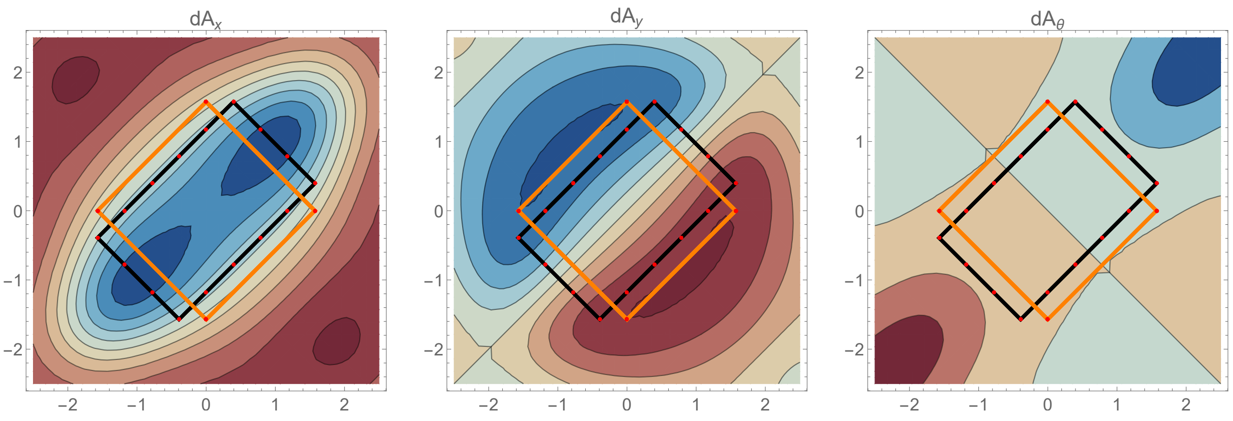


Fig. 5. Two forward gaits learned on the swimming robot and overlaid atop the connection exterior derivative plots, one after about 2M iterations (black) and the other after 5k iterations (orange).

derivative landscape of high volume (shown as blue areas in the x contour plot of Fig. 5), given a fixed maximum action (velocity) magnitude, we can vary the number of seconds per action (t_{interval}) that the agent is able to try in each iteration to better match the landscape.

The gait shown in black in Fig. 5 was learned in about 2M iterations, in contrast to the gait in orange corresponding to Fig. 4b and learned in about 5k iterations. The key difference between the trials was the choice of t_{interval}, or number of seconds per action. In the former, we chose $t_{\text{interval}} = 1$; in the latter, $t_{\text{interval}} = 4$. The first case gives rise to gaits that are much more granular than the gaits actually learned, which we can also see from the geometric plots. This results in a large number of repeated actions and therefore a much longer learning period. In contrast, allowing time intervals up to four times larger allowed the robot to execute the optimal gait in fewer overall timesteps and thus learn it much faster. Both gaits are equally effective in maximizing displacement; while the gaits are qualitatively different, they encircle roughly the same x volume under the exterior derivative plots.

6. CONCLUSIONS AND FUTURE WORK

We have described our efforts in using deep reinforcement learning (DRL) to achieve locomotion for a class of robots modeled using geometric mechanics. DRL has been applied successfully in a number of other fields, and recently robotics, but challenges still remain in making it feasible for physical systems. Systems described by the language of geometric mechanics naturally reveal symmetries and structure that can allow for powerful simplification of their complexities. These two observations have motivated us to combine contributions from the two fields to learn gaits for snake robots using DQN, all of which also match the geometric intuition of prior models.

The next step is to transfer our implementation onto a physical robot. We believe that our results of reducing the complexity of the learning process are sufficient for a robot to learn and execute similar gaits in real time. We have already built a prototype of a wheeled snake robot, shown in Fig. 6, that can perform repeated gait actions as part of a learning process, along with the necessary sensors to detect displacement and computing power to

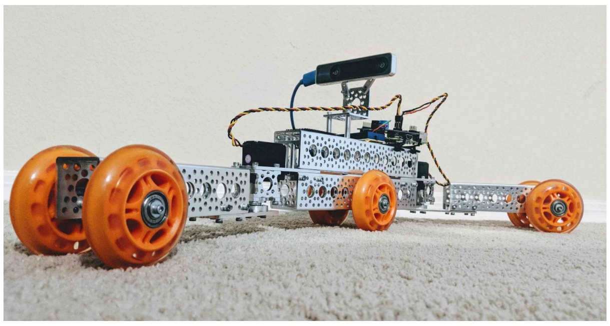


Fig. 6. Prototype of a wheeled snake robot with capabilities for deep reinforcement learning experimentation.

maintain and update a neural network, and we expect to have replicated our simulation results in the short term. Beyond that, we hope to generalize our results to greater varieties of locomoting robots and other physical systems, such as those with a combination of actuated and passive joints detailed by Dear (2018).

REFERENCES

Bing, Z., Lemke, C., Jiang, Z., Huang, K., and Knoll, A. (2019). Energy-efficient slithering gait exploration for a snake-like robot based on reinforcement learning. *arXiv preprint arXiv:1904.07788*.

Bloch, A.M., Krishnaprasad, P.S., Marsden, J.E., and Murray, R.M. (1996). Nonholonomic mechanical systems with symmetry. *Archive for Rational Mechanics and Analysis*, 136(1), 21–99.

Bullo, F. and Lynch, K.M. (2001). Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems. *IEEE Transactions on Robotics and Automation*, 17(4), 402–412.

Burton, L., Hatton, R.L., Choset, H., and Hosoi, A. (2010). Two-link swimming using buoyant orientation. *Physics of Fluids*, 22(9), 091703.

Dear, T. (2018). *Extensions of the Principal Fiber Bundle Model for Locomoting Robots.* Ph.D. thesis, Carnegie Mellon University.

Haarnoja, T., Zhou, A., Ha, S., Tan, J., Tucker, G., and Levine, S. (2018). Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*.

Hatton, R.L. and Choset, H. (2011). Geometric motion planning: The local connection, stokes' theorem, and the importance of coordinate choice. *The International Journal of Robotics Research*, 30(8), 988–1014. doi: 10.1177/0278364910394392.

Hatton, R.L. and Choset, H. (2013). Geometric swimming at low and high reynolds numbers. *IEEE Transactions on Robotics*, 29(3), 615–624.

Hirose, S. (1993). Biologically inspired robots. *Snake-Like Locomotors and Manipulators.*

Kelly, S.D. and Murray, R.M. (1995). Geometric phases and robotic locomotion. *Journal of Robotic Systems*, 12(6), 417–431.

Kelly, S.D. and Murray, R.M. (1996). The geometry and control of dissipative systems. In *Proceedings of the 35th IEEE Conference on Decision and Control*, volume 1, 981–986. IEEE.

Ma, S. (1999). Analysis of snake movement forms for realization of snake-like robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 4, 3007–3013. IEEE.

Marsden, J.E. and Ratiu, T.S. (2013). *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*, volume 17. Springer Science & Business Media.

Melli, J.B., Rowley, C.W., and Rufat, D.S. (2006). Motion planning for an articulated body in a perfect planar fluid. *SIAM Journal on applied dynamical systems*, 5(4), 650–669.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

Mukherjee, R. and Anderson, D.P. (1993). Nonholonomic motion planning using stoke's theorem. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, 802–809. IEEE.

Murray, R.M. and Sastry, S.S. (1993). Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5), 700–716.

Ostrowski, J.P. and Burdick, J.W. (1998). The geometric mechanics of undulatory robotic locomotion. *The International Journal of Robotics Research*, 17(7), 683–701.

Ostrowski, J. (1999). Computing reduced equations for robotic systems with constraints and symmetries. *Robotics and Automation, IEEE Transactions on*, 15(1), 111–123. doi:10.1109/70.744607.

Peng, X.B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4), 41.

Shammas, E.A., Choset, H., and Rizzi, A.A. (2007). Geometric motion planning analysis for two classes of underactuated mechanical systems. *The International Journal of Robotics Research*, 26(10), 1043–1073. doi: 10.1177/0278364907082106.

Sutton, R.S. and Barto, A.G. (1998). *Introduction to reinforcement learning*, volume 2. MIT press Cambridge.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*.

Tesch, M., Lipkin, K., Brown, I., Hatton, R., Peck, A., Rembisz, J., and Choset, H. (2009). Parameterized and scripted gaits for modular snake robots. *Advanced Robotics*, 23(9), 1131–1158.

Watkins, C.J.C.H. (1989). Learning from delayed rewards.

Zhang, X.L., Li, B., Chang, J., and Tang, J.G. (2018). Gliding control of underwater gliding snake-like robot based on reinforcement learning. In *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, 323–328. IEEE.

Zhu, H., Gupta, A., Rajeswaran, A., Levine, S., and Kumar, V. (2019). Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, 3651–3657. IEEE.