Sim-to-Real: Learning Energy-Efficient Slithering Gaits for a Snake-like Robot

Zhenshan Bing[1], Long Cheng[2], Kai Huang[3], and Alois Knoll[4]

***Abstract*— To resemble the body flexibility of biological snakes, snake-like robots are designed as a chain of body modules, which gives them many degrees of freedom on the one hand and leads to a challenging task to control them on the other hand. Compared with conventional model-based control methods, reinforcement learning based methods provide promising solutions to design agile and energy-efficient gaits for snake-like robots, since reinforcement learning based methods can fully exploit the hyper-redundant bodies of the robots. However, reinforcement learning based methods for snake-like robots have rarely been investigated even in simulations, let alone been deployed on real-world snake-like robots. In this work, we introduce a novel approach for designing energy-efficient gaits for a snake-like robot, which first learns a policy using a reinforcement learning algorithm in simulation and then transfers it to the real-world testing, thereby leveraging fast and economical gait generation process. We evaluate our reinforcement learning based approach in both simulations and real-world experiments to demonstrate that it can generate substantially more energy-efficient gaits than those generated by conventional model-based controllers.**

I. Introduction

Inspired by the versatile locomotion skills of biological snakes, researchers have designed many kinds of snake-like robots to mimic the structural and functional properties of their counterparts in nature. With distinctive skills, snake-like robots are expected to be deployed in special task scenarios, where other kinds of robots are incapable to be used, such as disaster rescue, military surveillance, and pipeline maintenance in factories. Since snake-like robots can only carry limited energy resources for field operations, it is essential to design energy-efficient gaits to reduce the impact of the power constrains. An energy-efficient movement, on the one hand, can maximize the service time of a robot while maintaining its performance. On the other hand, it can in return allow us to optimize the design of the robot by lowing its weight or adding other hardware [1]. However, it is difficult to design energy-efficient gaits for snake-like robots due to their redundant long mechanisms and complex dynamic interactions with the surroundings.

The most commonly adopted method for controlling a snake-like robot is to model its discrete body as a continuous curve with sideslip constraints in the planar environment, which is also widely known as the *serpenoid curve* [2]. This *serpenoid curve*, as a mathematical formation of the lateral undulation, parametrizes the angle of each body module to fit the shape of a sinusoidal curve and drives the robot forward using the external force generated from the side constraints. This kinematic-based method is also extended to combine two parameterized sinusoidal waves that propagate along the lateral and dorsal planes for modular snake-like robots moving in 3D space [1], [3], [4]. This parametrized kinematic method, also known as the *gait equation*, allows for the emergence of complex behaviors from low-dimensional representations with only few key parameters, greatly expanding their maneuverability and simplifying user control. With this method, researchers developed several biological gaits for snake-like robots to move in the indoor and outdoor environment. However, these simplified kinematic models also pose limitations on designing agile and energy-efficient gaits, since they are confined to those abstracted gait parameters, let alone the parameter tuning process is inefficient and time-consuming.

Designing a gait can be regarded as a parameter optimization problem, which aims at maximizing the objective functions, such as the velocity and power. For snake-like robots, it becomes even more challenging and the *gait equation* method mainly suffers from two reasons [5]. The extrinsic challenge comes from the complex dynamic interaction between the ground and the redundant mechanism with many degrees of freedom. The intrinsic challenge is how to synchronize and coordinate all the body joints to exhibit a proper motion pattern integrally, which is expected to be both robust and efficient.

Conventional control algorithms for snake-like robots require not only complex manual engineering to carefully tweak the analytical models of the robot, but also prior knowledge of the gaits for a specific robotic system. From the control perspective, conventional control methods often require accurate models of the system and expertise on other domain knowledge about the environment, which is usually complex, inaccurate, and difficult to be acquired in the real world. From the implementation perspective, specialized methods designed to control snake-like robots usually comprises complex software architectures, which are lengthy and error-prone. Therefore, it is promising to develop end-to-end learning methods that can greatly automate the gait design process. Such end-to-end learning methods should be able to remove the need for accurate dynamic models or domain knowledge, and can be applied to robotic systems without explicit system identification or too much manual engineering.

Instead of solving robotic tasks based on controllable kinematic or dynamic models, deep reinforcement learning (RL) methods solve similar tasks by mimicking the natural learning process in the trial-and-error paradigm. Agents can learn to master diverse and imaginative motor skills by only trying to maximize the reward signals. Although some robots

can be directly trained in the real world to learn motor skills, it is more feasible to learn complex skills from the simulation first, as it can greatly assist the development of RL algorithms and leads to a fast, cheap, and safe approach by alleviating the need of expensive and tedious real-world experiments.

On the basis of our previous work [6], this paper aims to explore the gait generation approach by learning a neural network controller via reinforcement learning for snake-like robots and transfer learned knowledge from simulations to real-world implementations. We largely extend our work from [6] and study our research in five steps. First, instead of using a simplified snake robot, we design a new snake robot prototype both in the simulation environment and the real world. Second, we design a reward signal that aims at encouraging energy-efficient locomotion. This reward function takes both the velocity and power consumption into consideration and integrates them after normalization. Third, instead of only training the controller using the proximal policy optimization (PPO) algorithm, we redesign the training pipeline by utilizing a dynamic randomization strategy, an informative prediction network, and the PPO algorithm to train the NN-based controller. The randomization includes the observation space and some of the crucial physical parameters. And the informative prediction network can predict unobservable features using the observable information via supervised learning. Fourth, to show the effectiveness of the proposed method in a more general context, we perform simulation tasks with different robot designs (joint number) and physical parameters (friction coefficient). We demonstrate that the proposed method can successfully leverage the dynamics of the robot in a general context and generate energy-efficient gaits in diverse scenarios. Last, we compare the gait results from the NN-based controller with results from the model-based method both in simulation and real-world experiments.

Our main contributions are summarized as follows. First, different from prior works that used RL-based methods to adapt the parameters of a low-level gait generator, our NN-based controller takes full advantage of the flexible body of the snake-like robot and learns sophisticated gaits simply from scratch. With the help of the reward signal and the iterative RL scheme, this method offers a new alternative for snake-like robots to study complicated motor skills without requiring any prior knowledge. Second, with the dynamic randomization strategy and informative prediction network, our NN-based controller successfully learns robust and adaptive behaviors in both simulation and real world, where some physical parameters and sensory measurements are either difficult to obtain or inaccurate. Existing solutions that are used for sim-to-real transfer, such as domain randomization [7], can only improve the controller's performance on the basis of a known and yet inaccurate domain. Our informative prediction network can predict the observations that can not be measured in a self-supervised manner. Third, by comparing the results, we demonstrate the superiority of our RL-based method against model-based methods in terms of energy efficiency.

II. Related Work

A. Gait Optimization With Model-Based Methods

To mimic the body structure and motion functions of biological snakes, snake-like robots are usually designed as a chain of body modules. These modules are used to interact with the environment to propel the robot, thereby resulting in a highly complex system to be controlled with. A biomorphic approach, *gait equation* [1], [2], is widely used to control the locomotion of snake-like robots, since it can easily mimic snake-like motions by shaping its body as a sinusoid curve. The *gait equation* can be regarded as a process to simplify parametric representations of snake-like trajectories. Crespi et al. adopted a heuristic optimization algorithm to rapidly adjust the travel speed of an amphibious snake robot [8]. Tesch et al. used the Bayesian optimization approach to regulate open-loop gait parameters for snake robots, which made the robot move faster and more reliable [9]. However, all these works were confined by the parameterized gait generation system, and have very limited effect on further improving the energy efficiency of a gait. Moreover, this open-loop method is also time-consuming and inefficient [10] to fine tune the gait parameters to achieve expected control objectives.

B. Gait Optimization With RL-Based Methods

As an intelligent trial-and-error learning method, RL brings new solutions for free gait generation tasks without knowing precise models or prior knowledge. RL technologies were used to fine-tune parameters for motor skills in a model-based manner, which can either be the controllable model of the system [11] or another abstracted gait generator such as the central pattern generator [12]. Shi et al. [13] developed a simplified snake-like robot with two actuated joints and implemented a controller based on deep Q-networks (DQN) algorithm. They demonstrated that their method can produce meaningful gaits. Similar to the concept in [14], Liu et al. proposed a RL-based controller to modulate the activity of a CPG controller for generating goal-reaching and tracking behaviors for a soft snake-like robot in simulations [15]. Chatterjee et al. proposed a policy improvement method to choose different control parameters for a snake-like robot with screw-drive units [16]. Sartoretti et al. presented an RL-based approach to adapte the shape parameters of a snake-like robot in response to external sensing [17].

C. Sim-To-Real Transfer

To narrow the simulation-reality gaps, researchers attempted to improve the simulation fidelity by building up accurate models and refine them using real data. For instance, to match the performance of the real system, an actuator can be modeled and refined using its real data [18]. Researchers also work on improving the robustness of a learned policy to variations of system properties and perception information, thereby enabling it to be feasible and adaptive in real-world systems [19]. The task to transfer a policy learned from simulation to real world is treated as an instance of domain adaption, where the source domain (simulation) is modeled

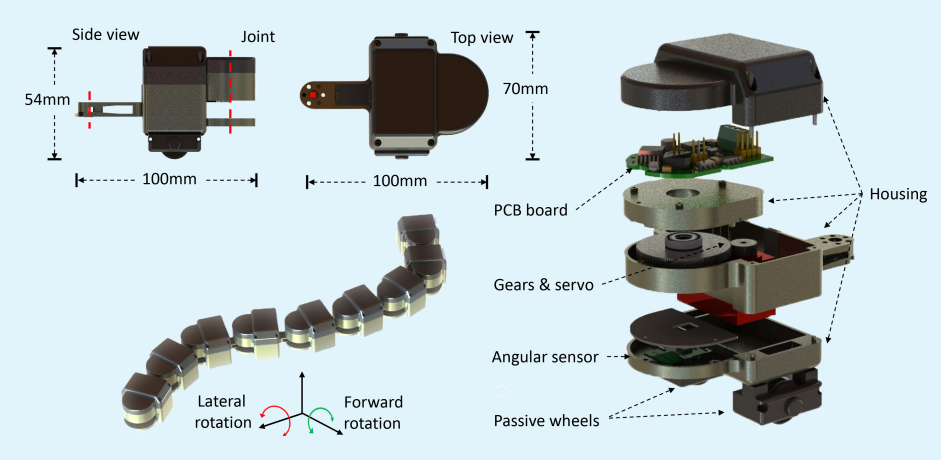


Fig. 1: The snake-like robot and its module dimensions.

as close as possible to the target domain (real world) [7]. Robotic arms are the most common agents to deploy learned policy from simulations, since the kinematics and dynamics can be accurately modeled. Christiano et al. controlled a robotic arm using learned policy from simulation, of which the inverse dynamics (from observations to actions) of that robot was learned from the data generated by the forward dynamics (from actions to observations) in simulations [20].

Domain randomization is another technique for sim-to-real transfer based on the introduction of higher variance in the domain-specific, but task-irrelevant features of the training data. Peng et al. propose introducing variability in the dynamics of the simulation by sampling values for strategic features (e.g., friction, mass, damping coefficients) during the training phase [21]. They argued that, while modeling the simulation in accordance with the real environment is important for the sim-to-real policy transfer, a complete and accurate model is often impossible due to many reasons, such as unforeseen forces, left-out environmental characteristics, calibration issues and so on. In the meantime, Wulfmeier et al. argued that unfavourable domanin randomization can lead to poor performance of the policy that is transferred from simultion [22]. They proposed another idea to tackle the systematic model discrepancies by aligning the distributions over visited states between the simulated agent and the real-world agent. They demonstrated their argument by performing experiments between two simulations with either different parameterizations or completely different simulation engines to create situations of misaligned and unknown system dynamics.

III. Robot and Model

A. Snake-like Robot

Our planar snake-like robot adopts a modularized design manner, which consists of eight identical actuated body modules and one head module to slither forward (See Figure 1). Each module is connected to the adjacent module by an actuated joint, which can rotate 90° in both left and right directions. Each body module is composed of an actuation system, the control system, housing components, and a pair of passive wheels to imitate the anisotropic friction property of the snake skin. The actuation system consists of a servo, a gearbox, and an angular sensor to feedback the angular position of the output joint. The DC servo has a maximum torque of 12.8 Kg·cm and drives a gearbox with a reduction factor of 3.71. The angular sensor is a Hall effect encoder that is used to measure the angular position of the output shaft of each module. The control system is a customized STM32 micro-controller. The STM32 runs three tasks: controlling the servo, reading the joint angle, and communicating with the other modules. A real-time operation system is executed on the control system and the control loop is set as 20 Hz. Table I summarizes the technical specifications of the robot.

B. Simulation Model

We model the snake-like robot as close as the real-world counterpart and simulate it in MuJoCo. As the fundamental component of a motion behavior, the dynamic model plays a great role in narrowing the gap between the simulation and real world. There are several key parameters that directly determine the accuracy of the dynamic model, such as the mass of the robot, inertia, friction, and other physical parameters.

1) Modeling Mass: The mass can be weighted exactly by measuring the module of the prototype, while the moment of inertia is estimated from its CAD assembly, in which each

TABLE I: Technical details of the robot module.

Parameters	Quantity
Dimensions	Width: 70mm, Length: 100mm, Height: 54mm
Mass	Body: 202g, One wheel: 2.2g
Servo	Max torque: 12.8kg·cm, Max speed: 0.07s/60°
Gearbox	Gear ratio: 151:43
Communication	CAN bus
Sensors	AS5047D (resolution:0.022°)
Power	Voltage: 24V, Max current: 10A
Control rate	20 Hz

component is given its material.

2) Modeling Actuation: The simulated servo system consists of a position controlled motor, a feedback signal of the joint position, and a gear ratio. For the reason of simplicity, by setting the gear ratio as 1, we directly map the motor position to the output position of the servo system. The torque is limited to a range of $[-4.6, 4.6]$ in newton-meter based on the specifications of the servo.

3) Modeling Friction: To imitate this anisotropic friction property, each robot module is equipped with two passive wheels. Those passive wheels enable a minimum friction in the direction of rotation and a high friction in the lateral direction (See Figure 1). To obtain realistic movement, we directly measured the friction coefficients for these two directions in the real world.

C. Energy Efficiency Metric

We aim to design gaits that can make snake-like robots move energy efficiently and in the meantime keep a steady speed of the movement. In this work, we define the power efficiency metric to evaluate the performance of all gaits.

The total power P of the robot can be calculated by adding up the power of all the N joints, which can be calculated as $P = \sum_{i=1}^{N} \left|\tau_i \dot{\phi}_i\right|$, where $\dot{\phi}_i$ is the velocity of joint i. The torque τ_j is the product of its applied force f_i and its gear constant parameter h_i (the length of the actuator). The model uses actuators with a limited force of f_{max} as the maximum force in both directions. With this property, the normalized power consumption $\hat{P}$ is calculated as

$$\hat{P} = \frac{1}{N} \sum_{i=1}^{N} \frac{\left|f_i h_i \dot{\phi}_i\right|}{f_{max} h_i \dot{\phi}_{max}} \ .$$

This normalized power consumption $\hat{P}$ will be further used for reward definition.

With the power P and the velocity v, several efficiency metrics can be calculated. The usual way is to calculate the Cost of Transport (COT), which is the power P divided by the mass m, the gravity g and the velocity v:

$$COT = \frac{E}{mgd} = \frac{P}{mgv} \tag{1}$$

This unit-less measure is able to compare the efficiency between different mobile systems. Since the same robot and environmental properties are compared, those constants only scale the results and would not provide any additional insight for the comparison. Thus, a simplified metric is to calculate the Averaged Power per Velocity (APPV) as $APPV = \frac{P}{v}$.

IV. Baseline Controller

In this section, we first introduce the widely used method for generating gaits for snake-like robots, which is the *gait equation* controller. Then we use the grid search algorithm to explore the energy efficient gaits at different velocities by brutally searching the parameter space with fine steps.

A. Gait Equation Controller

As the most widely adopted method, the *gait equation* controls the locomotion gaits of a snake-like robot by shaping its back curve as a moving sinusoid wave that propagates along the body. This method or similar ideas (e.g., central pattern generator based methods) have been used for decades by many kinds of snake-like robots to generate different gaits [2]. We utilize the *gait equation* from [9], which is modeled as

$$\phi(n,t) = (\frac{n}{N}x + y) \cdot A \cdot \sin(\omega t + \lambda n) \ . \tag{2}$$

$\phi(n,t)$ presents the joint angle value at time t, where n is the joint index and N is the total joint number. λ and ω are the spatial and temporal frequency of the movement, respectively. The spatial frequency represents the cycle number of the wave and the temporal frequency represents the traveling speed of the wave. A is the serpentine amplitude and x and y are the constants for shaping the body curve.

B. Grid Search Optimization

We take the grid search algorithm to explore the parameter space, which can lead to the optimum gaits as long as the searching intervals are small enough. We use the grid search method to generate a variety of gaits and determine those parameter combinations with the best power efficiency at different velocities. The grid search method generates a Cartesian product from the parameters in Table II, resulting in 10080 parameter sets. Then, each motion parameter set gets tested by running 1000 steps in the simulation environment. For each run, the first 200 time steps are ignored and the remaining 800 time steps are evaluated for collecting experimental data. It is because it has been observed that the snake robot needs about 200 time steps to accelerate and then moves at a steady speed.

V. NN-Based Controller

A. Observation Space

The full observation space consists of all the joint position ϕ_j, the joint velocity $\dot{\phi}_j$, the actuator torque τ_j, the head module velocity v_1, and the target velocity v_t. The joint position ϕ_j and its joint velocity $\dot{\phi}_j$ are required to learn the locomotion and represent the proprioceptive awareness of the robot. The head module velocity v_1 helps to sense its global velocity, which offers better movement awareness. To learn

TABLE II: The gait parameters used for the grid search.

Params.	Descriptions	Values
ω	Temporal frequency	0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1,8, 2.0
A	Amplitude (in degrees)	30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85
y	Linear reduction	0.1, 0.2, 0.3, 0.4
x	Linear reduction	(1-y)
λ	Spatial frequency (in degrees)	40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140

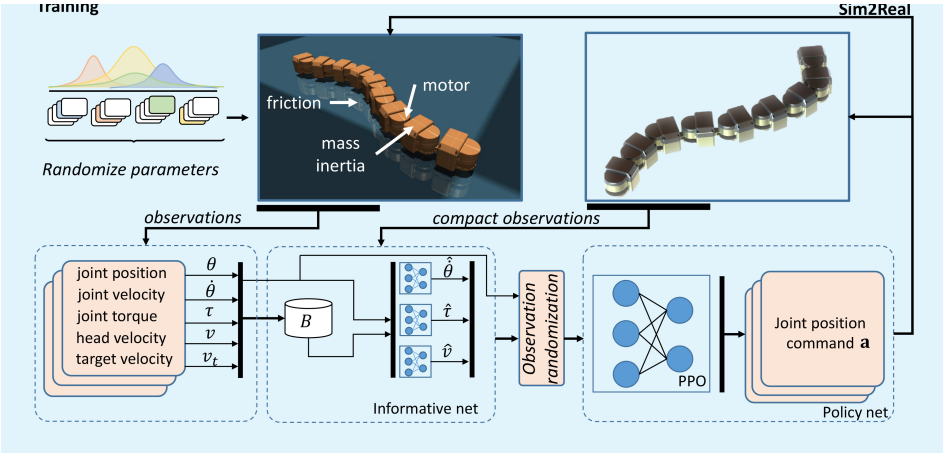


Fig. 2: The training pipeline for the sim-to-real transfer of the NN-based controller. There are three steps in this pipeline. First, the physical parameters are randomized in each episode during training. Second, the observation space is used to train an informative neural network to predict the observations that can not acquired directly from real-world environments. Third, the randomized observations are fed to the NN-based controller, which is trained using PPO algorithm.

an energy efficient gait, the sense of energy consumption is necessary. Therefore, the actuator torque of each joint τ_j is provided and can be interpreted in combination with $\dot{\phi}_j$ to determine the total power usage. The specified target velocity v_t is passed to the environment and can be changed, which is required to control the velocity of the robot.

B. Action Space

The action space has 8 DoFs with finite continuous values in the range of $[-1.5, 1.5]$, which linearly translates to a corresponding joint angle ϕ in the range of $[-90°, 90°]$. Each action represents eight actuator angle positions of the servo motors. In the real-world setup, the joint can turn around with the motor speed as $0.07\text{s}/60°$ and the gear ration as $151 : 43$. The control frequency for the snake-like robot is set as 20 Hz. Thus, the maximum turning angle in 0.05s is set as $12°$ in the simulation. The simulation time step is also set as 0.05s. It should be noted that 20 Hz is selected to ensure the real-time calculation of the NN-based controller.

C. Reward Function

The objective of our task is to learn an energy efficient gait for a variety of specified velocities. Therefore, the energy consumption and the difference between the actual model velocity and the target velocity are the main criteria to find a successful behavior. The challenge is to combine and weigh the variables into one numerical reward for each time step. Therefore, we first split the power efficiency and velocity criteria into two normalized reward function components.

First, a normalized reward is defined to maintain the specified velocity. The objective is to reach and maintain the target velocity v_t by comparing it with the head velocity v_1. The following function represents the velocity reward:

$$r_v(t) = (1 - \frac{|v_t - v_1|}{a_1})^{\frac{1}{a_2}}. \tag{3}$$

$a_1 = 0.2$ influences the spread of the reward curve, by defining the x-axis intersections with $x = v_t \pm a_1$. $a_2 = 0.2$ affects the changes of the curve's gradient. If $|v_t - v_1| = 0.0$, the velocity reward r_v reaches the maximum value 1.0.

Second, the normalized value of the total power usage $\hat{P}$ in (III-C) is used to determine the power efficiency reward component r_P, which is represented by

$$r_P(t) = r_{max}|1 - \hat{P}|^{b_1{}^{-2}}. \tag{4}$$

Here, r_{max} is the maximum reward value and $b_1 = 0.6$ is the slope of the curve. The power efficiency is influenced by the desired target velocity. Thus, the normalized r_{max} represents this influence by limiting the maximum value of r_P.

Last, the rewards from the velocity r_v and the power efficiency r_P are combined to form the overall reward r:

$$r(t) = (1 - \frac{|v_t - v|}{a_1})^{\frac{1}{a_2}}|1 - \hat{P}|^{b_1{}^{-2}} \tag{5}$$

This equation replaces the r_{max} in (4) with r_v. With that, the maximal power efficiency depends on the absolute value of the difference between the desired velocity and the robot velocity and the total power consumption.

D. Network Architecture

Given the input (observation $\mathbf{o}_i^t$) and the output (action $\mathbf{a}_i^t$), we now elaborate the policy network mapping $\mathbf{o}_i^t$ to $\mathbf{a}_i^t$. We design a fully connected 2-hidden-layer neural network as a non-linear function approximator to the policy π_θ. The input layer has the same dimension as the action space $\mathbf{o}_i^t$. Both hidden layers each have 200 neurons and are each followed by a ReLU layer. The final layer outputs the joint position commands for the robot. To train the network, the PPO algorithm adapted from [23] is used.

We train our policy network on a computer with an i7-7700 CPU and a Nvidia GTX 1080 GPU. A total of six million time steps (about 7500 episodes) are used for training. The maximum number of timesteps in an episode is 2000. With the environment settings of 50 ms per time step, the training takes about 42 hours in total simulation time and 2 hours in wall clock time for the policy to converge.

VI. Sim-To-Real Transfer

We present two approaches that are designed for robust transfer of the policy learned in simulation to the real world, namely, the parameter randomization and informative network.

A. Parameter Randomization

Parameter randomization is an effective approach to improve the robustness of the system, especially taking the gap between simulation and real-world experiments into consideration. By randomizing the physical parameters and observations during training, the learned policy can be more robust to be deployed in real-world experiments.

1) Physical Parameter Randomization in Simulation: The physical parameters given in Section III are either measured or estimated, which may lead to inaccuracy. And different settings of physical parameters will directly impact the performance of the generated gaits. When the agent is trained in a stable environment with fixed dynamic parameters, it usually leads to an over-fitting controller, which will not work properly in the real-world environment. We randomly sample physical parameters as listed in Table III.

2) Observation Space Randomization: In a real-world setup, the joint angles can not be perfectly measured due to several uncertain factors, such as the noise of the encoder, the data errors or delay caused by the communication. To eliminate the impact of this inaccuracy, we add a Gaussian noise $\mathcal{N}(\mu, \sigma)$ on every joint angle positions in the observation space during training, where $\mu = 0$ rad and $\sigma = 0.05$ rad.

B. Informative Prediction Network

The observation space used in the simulation is only partially observable or measurable in the real world. To solve this problem, we propose an informative prediction network to predict the joint torque, joint velocity, and the head velocity under the framework of supervised learning. We present three steps to construct such an informative prediction network.

1) Compact Observation Space: In the real world, some of the observations are difficult to acquire, such as the joint velocity $\dot{\phi}_j$ and the joint torque τ_j. We can calculate the joint velocity by differentiating two consecutive joint positions at each timestep. However, this is infeasible due to the noise of the joint position. To solve this issue, we also design a compact observation space that only contains the joint position ϕ_j and the target velocity v_t. The informative network is used to infer the corresponding joint velocity $\dot{\phi}_j$, joint torque τ_j, and head velocity v_1 according to the given joint position ϕ_j . The input of the network is the joint position ϕ_i. The informative network is trained using supervised learning, where the dataset is directly sampled from simulation.

2) Data Generation: With a well-trained NN-based controller, the agent is controlled to slither in the simulation environment with a varying target velocity to sample training data. Each episode contains 400 timesteps and the target velocity starts from 0.05 m/s, and slightly increases $1.33e^{-3}$ m/s at each episode until 0.25 m/s. One logged data point contains the target velocity, joint position of the timestamp, and values of the last action of the NN-based controller. The generated data is shuffled and split, where 80% of the dataset is used for training and the other 20% is used for testing.

3) Informative Network Model: Three observation features, namely, the joint velocity $\dot{\phi}_j$, joint torque τ_j, and the head velocity v_1, used for training the NN-based controller in the simulation are not available in the real world. Since each feature has its own characteristic, we design three separate prediction models to predict them with the same architecture. For one prediction network, the input consists of the action values and the joint positions for each joint in the last two timesteps, together with the target velocity.

As explained in [24], the dynamics of the actuators are independent to each other and accordingly, they trained their models for predicting actuator forces separately. However, for the snake-like robot, we believe the action of one joint may affect the other joints, since the snake robot is a class of serially connected active cord mechanism. Therefore, we adopted two approaches for sampling the input data, namely, the joint-wise model and non-joint-wise model. For a joint-wise model, each value of the feature is strongly related to one's specific joint index, e.g., the torque of the first joint is predicted only using the data from the first joint. The joint-wise prediction models split the input features of the input layer by joint and process them separately. The processed values are concatenated at the output layer. Oppositely, the non-joint-wise models process values from all existing joints together at once as input and predict output features for all joints at once. The main assumption of this is that a target feature depends on the entire kinematics of the snake-like robot.

In this work, prediction models with two layers of MLP,

TABLE III: Technical details of the robot module. During simulations, these physical parameters are randomly selected from the randomization range of the baseline value of each parameter.

Parameters	Baselines	Unit	Randomization ranges
Ground friction	0.6	–	$90\% \sim 100\%$
Mass	0.206	kg	$90\% \sim 100\%$
Armature inertia	0.01	kg·m^2	$80\% \sim 120\%$
Motor damping	0.3	N·s/m	$90\% \sim 110\%$

TABLE IV: Training configurations for prediction. The length of the input sequence means the number of past timesteps.

	Torque	Joint velocity	Head velocity
ANN	MLP	LSTM	
Input feature	Joint position, action, and target velocity		
Input length	2	2	8
Non-joint-wise/Joint-wise	joint-wise	non-joint-wise	-

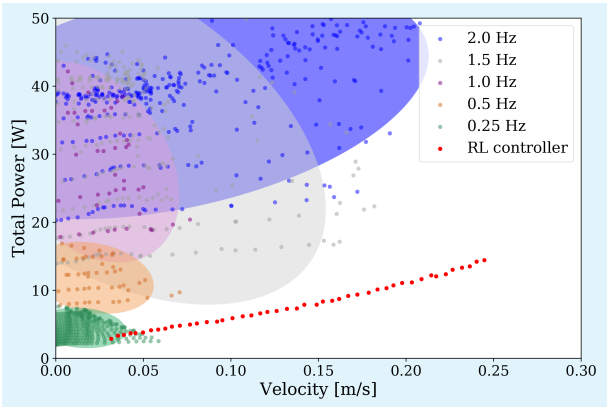


Fig. 3: This scatter plot directly shows the energy consumption results of the controllers at a range of velocities in the simulation. The red points represent the performance of the NN-based controller. For the *gait equation* controller, we select five temporal frequencies w, namely, 0.25 Hz, 0.5 Hz, 1.0 Hz, 1.5 Hz, and 2.0 Hz. We then run the grid search algorithm with each frequency and plot the scatters and its confidence region.

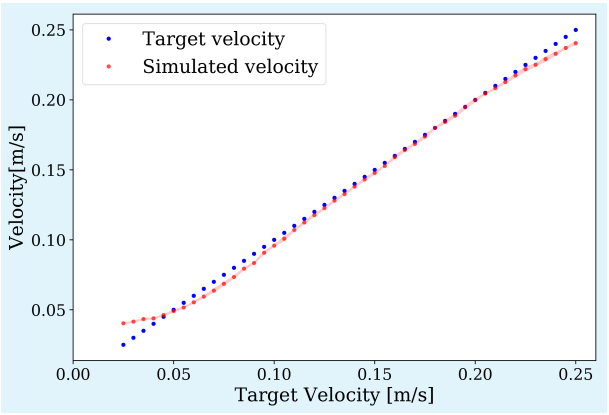


Fig. 4: The actual velocities (red dots) measured from simulations and the corresponding target velocities (blue dots). The error interval of the red line represents the standard deviation over the five random seeds

LSTM, or a single TCN layer (which consists of multiple convolution layers) were tested. For the length of input sequences (the number of past timesteps), 2, 4, and 8 were used for training to observe how the input sequence length affects model performance. The mean squared error (MSE) was used as the loss function for training models, which is a commonly used for regression tasks. We skip the training and testing details for choosing proper models or the length of input sequences and present the final setup for each feature in Table IV. To predict the torque, we choose multi-layer perceptron as the architecture of the prediction network, which is trained via the joint-wise style. The length of the sequential input is 2. To predict the joint velocity, we choose LSTM as the network architecture, which is trained via the non-joint-wise style. The input length is also 2. To predict the head velocity, we choose LSTM as the network architecture as well. The input length of this prediction network is 8. Since the head velocity is one overall property of the robotic system, the joint-wise or non-joint-wise style is not applicable here. The inputs for predicting these three states are the joint position, action, and the target velocity.

To enable the learning of different velocities, the parameter v_t is changed by iterating over 0.05, 0.1, 0.15, 0.20, and 0.25 for each episode while training. It should be noted that, in theory, we can also select the target velocity from a uniform distribution to train the network. But empirical results show the training is much more successful and stable when using discretized target velocities. Meanwhile, to simplify the beginning of the learning process, the first 100 episodes are trained with a fixed target velocity of 0.1 m/s.

VII. Simulation Results and Comparisons

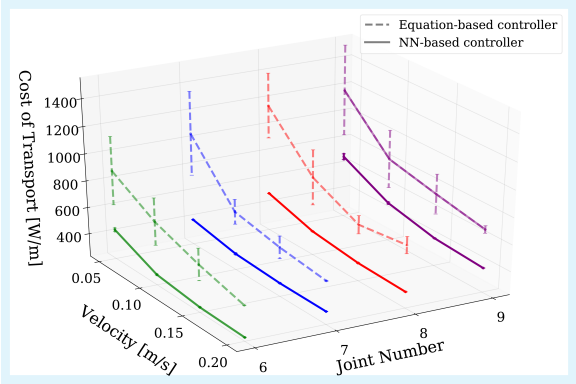
A. Baseline Performances

The power consumption and the corresponding velocity results from the grid search algorithm are shown in Figure 3 as a point cloud of parameter sets using dot markers. The lowest points at different velocities in the point cloud have the highest energy efficiency. To reduce the number of equation parameters, we select five temporal frequencies to show the pattern of gaits generated by the gait equation, since the temporal frequency is the most direct factor to impact the velocity. Second, instead of connecting the scatter points with lines, we show the confidence region of the gaits generated with each frequency. We can observe that low frequency (0.25 Hz) leads to very slow gaits, while high frequency (2.0 Hz) leads to energy-intensive gaits. Gaits with 1.5 Hz are more promising to generate energy-efficient gaits. We also demonstrate that the grid search method suffers from being inefficient to search proper parameters for energy-efficient gaits, since most of the parameter sets in the searching space are distributed in the low velocity area ($0 - 0.1$ m/s). Note that we show all the gaits from the grid search algorithm as we want to depict the inefficiency of the baseline method.

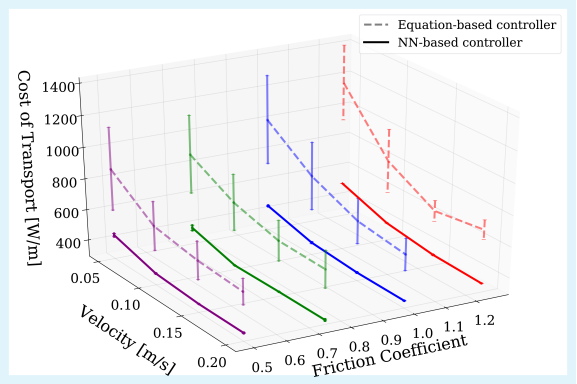
B. NN-Based Controller Performance

In this study, target velocities in the range of $[0.025, 0.25]$ m/s with a step interval of 0.005 m/s are used for the evaluation. Simulation results demonstrate that the NN-based controller has succeeded in learning a series of gaits from scratch without knowing any prior locomotion skills.

First, the NN-based controller can perform very accurate locomotion gaits in terms of velocity, even though the NN-based controller is trained with only five target velocities (0.05 m/s, 0.10 m/s, 0.15 m/s, 0.20 m/s, 0.25 m/s). As shown in Figure 4, the targeted velocities are represented with blue solid dots and the measured velocities from the simulation are marked with red solid dots. We can observe that the measured velocities almost match the target velocities from



(a) The gaits are generated with robots with different designs (number of the body module).



(b) The gaits are generated from scenarios with different friction coefficients.

Fig. 5: Simulation results from generalized scenarios. (a) shows the cost of transport (COT) performance of gaits generated from different robot design (body modules). (b) shows the COT performance of gaits generated from environments with different friction coefficients. The solid line shows the gaits generated from the NN-based controller. The dashed line shows the gaits generated from the *gait equation* controller. The error bars show the standard deviation of the energy consumption at each velocity.

0.05 m/s to 0.20 m/s with only very small errors close to zero. In the low velocity range (less than 0.05 m/s) and the high velocity range (higher than 0.20 m/s), the measured velocities differ from the target velocities. This is because, under extreme conditions, the NN-based controller is difficult to maintain target velocities while keeping optimizing the power consumption. In the low velocity range, the NN-based controller tends to move faster, since a minor modification of the motion will lead to the speed change. In the high velocity range, the NN-based controller is limited by the physical configuration of the snake-like robot to reach the target velocities.

Second, the power consumption results of the learned gait are marked with red points in Figure 3. The data depicts a linear relationship between the travel velocity and the power consumption, which is in line with the physical law $Power = Force \times Speed$. There are only a few points with higher power consumption when the velocity is below $0.08\ m/s$. Importantly, this also reveals the adaptability of the learning approach for generating gaits in a range of velocities. It can also be observed that the mean velocities do not exactly align with the specified interval of 0.005, especially at higher target velocities. The reason for this is the difficulty to achieve the exact ratio between holding the right velocity and performing the corresponding power-efficient locomotion.

C. Comparison

We first have a look at the energy metric based on the averaged power per velocity. After putting the power efficiency data of the *gait equation* controller (See Figure 3) and the NN-based controller together, we can clearly conclude that the NN-based controller has a much better power efficiency at a range of velocities. As the velocity grows, the advantage of the NN-based controller for saving energy is even more obvious.

To show the proposed method can generate energy-efficient gaits in a more general context, we also evaluated both controllers in different environments (friction coefficient) and with different robot designs (number of body modules) in simulation. The results are visualized in Figure 5. In Figure 5(a), we change the robot design by adding or reducing the body modules to test the proposed method. For a snake robot with joint number range from six to nine (the default number is eight), the proposed method shows consistent performance superiority to generate much energy-efficient gaits across different traveling speeds. In Figure 5 (b), we change the friction coefficient of the environment to test the effectiveness of the proposed method. On the one hand, the results clear show that NN-based controller can generate gaits that consume less energy than the gaits generated from the equation based controller. On the other hand, the NN-based controller also shows much more stable performance in generating energy-efficient gaits, while the equation based method exhibits fluctuating energy consumption results.

The NN-based controller improves on the traditional kinematic-based controller in two ways. On the one hand, the traditional *gait equation* controller is based on the kinematics and describes the gait movement with no influence of physical forces such as friction or damping. It only extracts several critical parameters to represent the gait, while the interaction between the snake-like robot and the environment is highly complex. Although the parameterized gait equation simplifies the control task, it inevitably brings difficulties for designing more sophisticated gaits despite of using traditional optimization technologies (the grid search algorithm), especially when the parameter space will grow exponentially with the increasing joint numbers. On the other hand, the RL method shows its effectiveness in the ability for solving this kind of complex control problem, since it

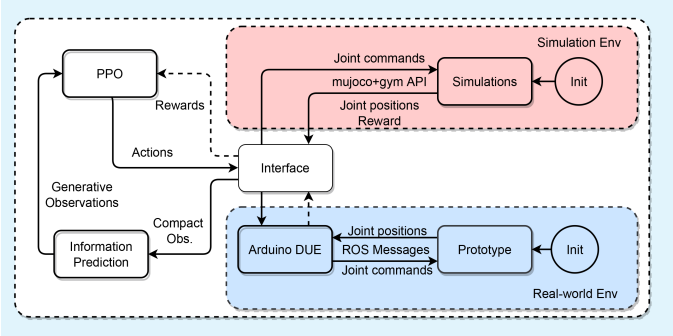


Fig. 6: The software framework for transferring learned policy from simulations to the real-world controller. The left part of the figure visualizes how the NN-based controller is trained and the right part illustrates the concept of sim-to-real transfer.

is trained directly in the dynamic environment. It is able to generate undulation gaits that not only imitate the movement of real snakes, but also explore its limitations and keep improving its behavior under its hardware constraints.

VIII. Prototype Experiments

A. Sim-Real Transfer Framework

To make it feasible to transfer a learned policy from the simulation to the real world, we should design a framework to easily connect the NN-based controller to the simulation environment or the real world with the same configuration. As shown in Figure 6, the NN-based controller has an interface to switch between the simulation environment and the real-world environment. For running both simulation and real-world environments, the compact observations are the joint positions $\hat{\phi}$. The compact observations are first fed into the informative network to predict the full observations $\mathbf{o}_i^t$. Then, the NN-based controller outputs the actions to be executed either in the simulation or the real world. For real-world experiments, the NN-based controller is computed on a desktop computer and the actions are sent to an Arduino DUE via ROS messages. Then the Arduino DUE serves as a host to transfer the ROS messages into CAN messages and then send them to the STM32 controller on each module via CAN bus.

B. Experiment Setup

The power consumption can be calculated with its running current and voltage. In our experiments, we use a 24 V constant voltage power supply to actuate the robot. The current can be acquired by measuring the voltage of a high-load resistor (50W/0.1Ω), which is stringed into the circuit. Thus, the power consumption of the prototype snake-like robot can be calculated as $P = \frac{U_p}{R_p}(24 - U_p)$, where $R_p = 0.1\ \Omega$ and $U_p = 24$ V. The voltage U_p is measured using an oscilloscope, which samples at 1000 Hz.

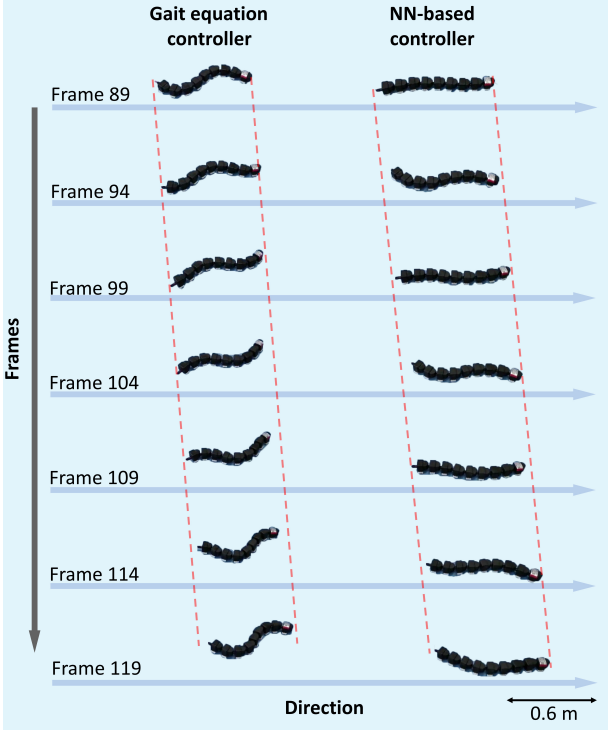
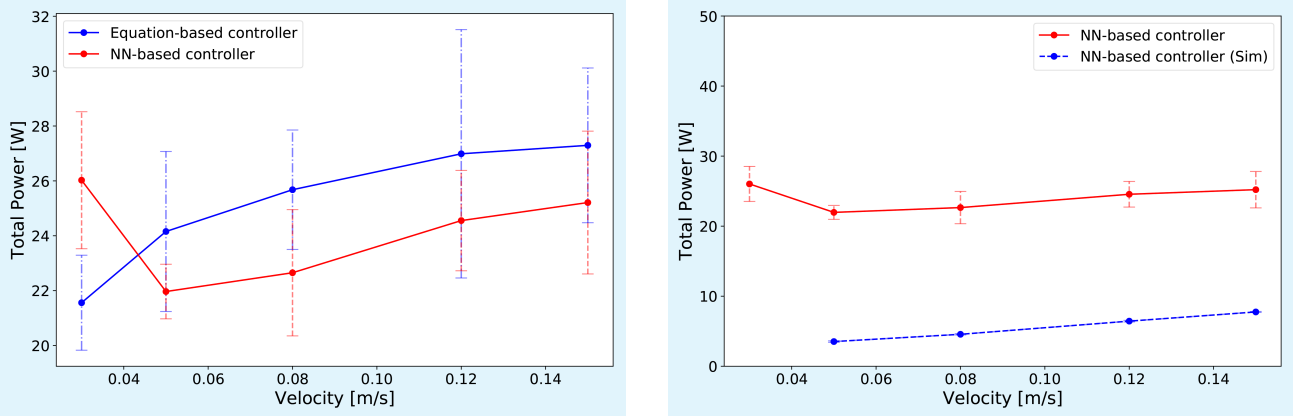


Fig. 7: The montages of the snake-like robot performing slither gait at 0.15 m/s, controlled with the NN-based controller and the *gait equation* controller. The video is recorded at 25 frames per second.

C. Results

Due to the time and hardware durability, we only tested five target velocities for the NN-based controller in our prototype experiments, namely, 0.05, 0.10, 0.15, 0.2, and 0.25 m/s. We controlled the robot to run for ten seconds to measure the power consumption and the actual velocity. The actual velocities for the gaits generated by the NN-based controller were 0.03, 0.05, 0.08, 0.12, 0.15 m/s, approximately, which were all less than the target velocities. This is because the reality gap between the simulated environment and the real world. For the same reason, the performance of the *gait equation* controller is also impacted by this reality gap. In order to fairly compare the energy efficiency, we chose parameters that lead to gaits traveling at these five actual velocities with the best energy efficiency in the simulation.

First, we present the montages of the slithering gaits at 0.15 m/s from the gait equation controller and the NN-based controller (See Figure 7). To show a full cycle of the movement, seven video frames (25 FPS) are selected to visualize the pattern of the gaits generated by both controllers. For the gait generated by the *gait equation* controller, frame 89 to frame 119 show one full cycle of the gait. For the gait learned by the NN-based controller, about two motion cycles are shown. As we can see, the body curve of the *gait equation* gait is shaped as a sinusoid wave. The body curve of the RL gait is slightly slender than the body curve of the parameterized gait. This slider



(a) (b)

Fig. 8: (a) Real-world power comparison of the slithering gaits from the *gait equation* controller and the NN-based controller. The target velocities are $[0.05, 0.10, 0.15, 0.2, 0.25]$ m/s. The actual velocities are $[0.03, 0.05, 0.08, 0.12, 0.15]$ m/s, approximately. The solid points represent the averaged power over 10 s and the error bars depict the standard deviation. Note the standard deviation is calculated based on the data over the duration of the experiment (10 s). (b) Power comparison of the NN-based controller in the simulation and the real world. In simulation, the NN-based controller cannot achieve a gait that is 0.03 m/s.

body curve can propel the robot more smoothly and avoid joint power consumption as much as possible, which is in accordance with the simulation results. We can also observe that the learned gait moves at a higher frequency than the parameterized gait and moves straight towards the forward direction, while the parameterized gait tends to deviate from the forward direction.

Second, we show the results of the power consumption of the gaits from both the *gait equation* controller and the NN-based controller, which are shown in Figure 8a. All the power consumption data was measured in 10 second for fair comparison. The solid line represents the exponentially weighted moving average of the power consumption and the error bar depicts the standard deviation of the raw power consumption measured by the oscilloscope. Note that we first process the raw data by removing outliers that are outside the range of five standard deviations. Note that the outliers are caused by the noisy measurement of the current signal. From the results of the *gait equation* controller, we can observe that the power consumption increases with the movement velocity, which ranges from 21.5 Watt to 28 Watt, approximately. Since the velocity of the gait generated from the *gait equation* controller is directly related to the frequency of the sinusoid wave, it moves much slower at the low speed and faster at the high speed, thereby consuming more power. We can see that the NN-based controller consumes more power at a low speed. This is because the controller generates a concertina gait, which only moves parts of the body to inch forward and those static joints require the locked-motor current, thereby consuming more power. Another potential reason is that the NN-based controller has difficulties to maintain low and accurate velocities. Small adjustments of each joint will impact the overall velocity observably and thus consumes more energy. Comparing with the power consumption of the *gait equation* controller, we can find that the NN-based controller can increase the energy efficiency by 5% to 10%, especially at the intermediate and high velocities. For both controllers at one velocity, the power consumption shows obvious fluctuation, which can be interpreted as the periodic movement of the robot. Figure 8b shows the power profile comparison of the simulation and real-world results . We can find that there is still a performance gap between the simulation and the real-world experiment due to some potential reasons. For instance, it is challenging to simulate the environment with perfect fidelity and therefore leads to the performance drop. Inaccurate sensory measurements or noisy observation stats can also cause the performance drop.

IX. Conclusion

Designing energy-efficient gaits for snake-like robots remains a challenging task, since they come with redundant degrees of freedom and have complicated interactions with the environment. The most widely used method for generating gaits for snake-like robots is the *gait equation*, which mimics the snake's body shape using sinusoid-like curves. In this paper, we present a novel gait design method based on reinforcement learning. Compared with the optimized gaits generated by the *gait equation*, the learned gaits have shown better energy efficiencies at medium and high velocities. Although the learned gaits perform similarly or slightly worse than the parameterized gaits at low speeds, the proposed method shows great efficiency in discovering energy-efficient gaits automatically and maximizing the performance by removing the limitations of predefined models. Our work contributes and serves as an exploration for designing sophisticated moving patterns for snake-like robots. Future work will aim at designing gaits using reinforcement learning for snake-like robots without passive wheels.

ACKNOWLEDGMENT

This project/research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No.945539 (Human Brain Project SGA3). This work was also supported by the National Natural Science Foundation of China (Grant Number: 61902442) and Pazhou Lab PZL2021KF0020.

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] S. Hirose, *Biologically inspired robots: snake-like locomotors and manipulators*. Oxford University Press Oxford, 1993, vol. 1093.

[3] Z. Bing, L. Cheng, G. Chen, F. RÃ¶hrbein, K. Huang, and A. Knoll, "Towards autonomous locomotion: CPG-based control of smooth 3d slithering gait transition of a snake-like robot," *Bioinspiration & Biomimetics*, vol. 12, no. 3, p. 035001, apr 2017. [Online]. Available: https://doi.org/10.1088/1748-3190/aa644c

[4] Z. Bing, L. Cheng, K. Huang, M. Zhou, and A. Knoll, "Cpg-based control of smooth transition for body shape and locomotion speed of a snake-like robot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4146–4153.

[5] P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, *Snake robots: modelling, mechatronics, and control*. Springer Science & Business Media, 2012.

[6] Z. Bing, C. Lemke, Z. Jiang, K. Huang, and A. Knoll, "Energy-Efficient Slithering Gait Exploration for a Snake-Like Robot Based on Reinforcement Learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, QLD: IEEE, May 2018, pp. 3803–3810.

[8] A. Crespi and A. J. Ijspeert, "Online optimization of swimming and crawling in an amphibious snake robot," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 75–87, Feb 2008.

[9] M. Tesch, J. Schneider, and H. Choset, "Using response surfaces and expected improvement to optimize snake robot gait parameters," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 1069–1074.

[10] S. Ouyang and W. Wei, "Flexible adaptive control of snake-like robot based on lstm and gait," in *Journal of Physics: Conference Series*, vol. 1487, no. 1. IOP Publishing, 2020, p. 012049.

[11] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7559–7566, iSSN: 2577-087X.

[12] Y. Nakamura, T. Mori, M.-a. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a CPG-actor-critic method," *Neural Networks*, vol. 20, no. 6, pp. 723 – 735, 2007.

[13] J. Shi, T. Dear, and S. D. Kelly, "Deep reinforcement learning for snake robot locomotion," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9688–9695, 2020, 21th IFAC World Congress.

[14] S. Fukunaga, Y. Nakamura, K. Aso, and S. Ishii, "Reinforcement learning for a snake-like robot controlled by a central pattern generator," in *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, vol. 2, 2004, pp. 909–914 vol.2.

[15] X. Liu, R. Gasoto, Z. Jiang, C. Onal, and J. Fu, "Learning to locomote with artificial neural-network and cpg-based control in a soft snake robot," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 7758–7765.

[16] S. Chatterjee, T. Nachstedt, F. Worrgortter, M. Tamosiunaite, P. Manoonpong, Y. Enomoto, R. Ariizumi, and F. Matsuno, "Reinforcement learning approach to generate goal-directed locomotion of a snake-like robot with screw-drive units," in *2014 23rd International Conference on Robotics in Alpe-Adria-Danube Region (RAAD)*, 2014, pp. 1–7.

[17] G. Sartoretti, Y. Shi, W. Paivine, M. Travers, and H. Choset, "Distributed learning for the decentralized control of articulated mobile robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3789–3794.

[18] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to Walk via Deep Reinforcement Learning," in *Robotics: Science and Systems 2019*, Freiburg, Jun. 2019.

[19] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abk2822

[20] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model," *arXiv:1610.03518 [cs]*, Oct. 2016, arXiv: 1610.03518.

[21] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017. [Online]. Available: http://arxiv.org/abs/1710.06537

[22] M. Wulfmeier, I. Posner, and P. Abbeel, "Mutual alignment transfer learning," in *Conference on Robot Learning*. PMLR, 2017, pp. 281–290.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[24] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, Jan. 2019.