



國立台灣科技大學  
資訊工程系

---

碩士學位論文

生成用於互動性游泳者動畫的神經網路控制

Learning Neural Motion Control for Interactive  
Swimmer Animations

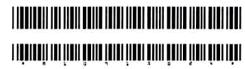


研究生：葉定豪

學號：M10915024

指導教授：戴文凱教授

中華民國一百一十一年九月二十六日



 碩士學位論文指導教授推薦書  
Master's Thesis Recommendation Form

M10915024

系所： 資訊工程系  
Department/Graduate Institute Department of Computer Science and Information Engineering

姓名： 葉定豪  
Name YEH, TING HAO

論文題目： 生成用於互動性游泳者動畫的神經網路動作控制  
(Thesis Title) Learning Neural Motion Control for Interactive Swimmer Animations

係由本人指導撰述，同意提付審查。

This is to certify that the thesis submitted by the student named above, has been written under my supervision. I hereby approve this thesis to be applied for examination.

指導教授簽章：  
Advisor's Signature

共同指導教授簽章（如有）：  
Co-advisor's Signature (if any)

日期：  
Date(yyyy/mm/dd)

2022/09/26



# 碩士學位考試委員審定書

Qualification Form by Master's Degree Examination Committee

M10915024

系所： 資訊工程系  
Department/Graduate Institute Department of Computer Science and Information Engineering

姓名： 葉定豪  
Name YEH, TING HAO

論文題目： 生成用於互動性游泳者動畫的神經網路動作控制  
(Thesis Title) Learning Neural Motion Control for Interactive Swimmer Animations

經本委員會審定通過，特此證明。

This is to certify that the thesis submitted by the student named above, is qualified and approved by the Examination Committee.

## 學位考試委員會

Degree Examination Committee

委員簽章：

Member's Signatures

王學武  
彭致軒

葉定豪

指導教授簽章：

Advisor's Signature

彭致軒

共同指導教授簽章（如有）：

Co-advisor's Signature (if any)

彭致軒

系所（學程）主任（所長）簽章：  
Department/Study Program/Graduate Institute Chair's Signature

日期：

Date(yyyy/mm/dd)

2022/09/26

## 摘要

本論文提出了一種程序化生成動作控制器的方法。我們使用了計算流體力學 (Computational fluid dynamics) 方法，模擬流體與水下游泳者間的相互作用，並使用深度強化學習，學習出真實的游泳者的運動動作。論文中提出的目標函數 (Loss function) 以及為其設計的課程 (Curriculum learning)，大幅提升了最佳化的效率和探索出的動作的有效性。然後，我們實施了策略蒸餾，將學到的游泳策略轉移到神經網路動作控制器上。最後產出的動作控制器能夠從學習出的游泳策略所產生的資料集中，學習到流體動力學的資訊，並且能夠透過連續當前狀態預測下一幀的狀態產生真實且具互動性的游泳運動。我們能夠經由此學習出的動作控制器，生成出足夠敏捷且可控制的游泳動作，使其能快速應對使用者所期望的輸入。



# Abstract

We propose an approach to generate the motion controllers procedurally. The simulation framework uses computational fluid dynamics (CFD) to simulate the two-way coupling interactions with the underwater swimmer and deep reinforcement learning to learn the realistic locomotion of the swimmer. Our proposed loss function and curriculum learning method significantly improve the efficiency of the learning process and the motions' effectiveness during exploration. In the second stage, we enforce a policy distillation to transfer the learned swimming policy to a neural motion controller. The resulting motion controller learns the dynamics implicitly from the dataset generated through the learned swimming policy and can synthesize realistic interactive animations by predicting the next state at each frame according to the current state. With the learned motion controller, we are able to generate swimming movements that are more agile and interactable with given user inputs.

# Acknowledgements

誠摯感謝指導教授戴文凱教授，給予我許多指點，獲益匪淺，在教授的幫助下使得本論文能夠更加完善、嚴謹。也感謝實驗室裡的學長姊、同學、學弟妹在學術上的共同砥礪，讓我能拓展視野，受益良多。



# Contents

Abstract in Chinese . . . . .	iii
Abstract in English . . . . .	iv
Acknowledgements . . . . .	v
Contents . . . . .	vi
List of Figures . . . . .	viii
List of Tables . . . . .	ix
List of Algorithms . . . . .	x
1 Introduction . . . . .	1
2 Preliminaries . . . . .	3
2.1 Physics-based Character Control . . . . .	3
2.2 Deep Motion Controller . . . . .	5
2.3 Swimming Simulation . . . . .	6
3 Method . . . . .	8
3.1 Gait Policy Learning . . . . .	8
3.1.1 Policy Representation . . . . .	9
3.1.2 Reward Function . . . . .	10

3.1.3	Network Training . . . . .	12
3.2	Controller Learning . . . . .	14
3.2.1	Data Generation . . . . .	14
3.2.2	Direct Distillation Controller . . . . .	15
3.2.3	Motion-Matching Controller . . . . .	16
4	Experiments . . . . .	19
4.1	Motion Distribution . . . . .	20
4.2	Curriculum Learning Effectiveness . . . . .	22
4.3	Test on Simplified Hydrodynamics . . . . .	25
5	Conclusions . . . . .	27
5.1	Future Work . . . . .	27
5.2	Limitations . . . . .	28
References	. . . . .	29
Letter of Authority	. . . . .	31

# List of Figures

3.1	Direct Distillation Controller Network . . . . .	15
3.2	Motion-Matching Controller Network . . . . .	18
4.1	Velocity and time-to-converge features of controllers . . .	21
4.2	5-link turning . . . . .	21
4.3	Alien turning motion . . . . .	21
4.4	Comparison of baseline and curriculum learning . . . .	22
4.5	5-link simulated motion . . . . .	24
4.6	Goldfish simulated motion . . . . .	24
4.7	Alien simulated motion . . . . .	24
4.8	4-legged swimmer . . . . .	24
4.9	Alien shape motion . . . . .	25
4.10	Alien shape in simplified simulation . . . . .	25
4.11	5-link in simplified simulation . . . . .	26
4.12	4-link in Unity sequence . . . . .	26
4.13	4-link in Unity screenshot . . . . .	26

# List of Tables

- 4.1 Statistics of rewards with baseline and curriculum learning 23



# List of Algorithms



# Chapter 1     Introduction

Interactive character animation is essential in the entertainment industry, such as in video games and virtual worlds. Traditionally created from sets of motion capture data, underwater motion capturing clips and the setup for capturing are hard to acquire. The requirement for real-world references makes it hard to animate unseen shapes or adapt to changes in configurations such as body weights or a dynamic environment.

On the other hand, physically based simulated model can yield convincingly realistic results and is applicable to wide variety of shapes, but Computational Fluid Dynamics (CFD) methods are computationally expensive for interactive applications and massive swarm simulations. There exists several methods using simplified fluid dynamics models to simulate swimming motions, but the lack of flow information will make it lose a bit of realism and can't apply to those swimmers whose movement heavily depend on the flow interaction. Because of the complicity and chaotic nature of fluid dynamics, and numerical errors in physic simulation models.

Under this setup, it is presumably difficult for RL-based methods to explore the simulation space effectively since the sampling-based exploration often fails to produce accurate controls and unsatisfying results, such as noisy, meaningless motions that are unnatural or inefficient enough. Ultimately, RL-based methods are still relevant in optimization problems like this and flexible to deploy on diverse simulation environments.

In this thesis, we aim to optimize swimming motions under a fluid simulation environment and create a motion controller suitable for applications

that require more responsive and precise manipulations. We present a system to learn the interactive motion controllers for different shape designs unsupervisedly. The learning process is realized by training a gait policy network to generate plausible swimming patterns for the given shape, then using the trained network to generate a motion dataset and create a motion control model that can capture the essence of those generated motions and guarantee the motions are responsive to the control input. We evaluate our RL training pipeline with various swimmer shapes and compare the results of different motion controllers on the reproducibility of the dataset and the control responsiveness.

The main contributions of our research are:



- We present the learning framework that improves the effectiveness of training a deep reinforcement learning agent to explore the physically simulated environment.
- We introduce our realistic swimming motion dataset to deep motion matching methods and evaluate the realism and agility of different motion control schemes.

# Chapter 2 Preliminaries

Our work is built on multiple fields, and we will briefly introduce those researches related to ours on topics from gait optimization to motion control and underwater swimmer simulation.

## 2.1 Physics-based Character Control

Recent motion controllers often use the high-low-level controller scheme [1], [2]. The low-level controller learned a motion manifold with local features such as joint positions, joint velocities and local environment information to map control input to proper motion and maintain body balance. Low-level controllers are usually learning a motion manifold by imitating motion capture data. While the high-level controllers respond to path planning to achieve various control tasks by giving orders to the low-level controller.

In contrast to the model-free RL used in the works mentioned above, there are model-based methods [3]. Instead of sampling only by interacting with the world, they trained a world model to learn the dynamics explicitly, which will be further used to train the control policy.

Unlike the aforementioned high-low level controller schemes, V. Tsounis et al. [4] first trained their high-level controller using the Convex Resolution Of Centroidal dynamics trajectories method (CROC) [5] mentioned in their work. This way, they separate the high-level controller from interacting with the simulation and the low-level controller, allowing both

controllers to be trained independently, thus mitigating the explore complexity. But the simplified dynamic model results in the burden of finding the fully dynamic and optimal motions to the low-level controller.

Other works tackle the controller training problem by combining trajectory optimization to optimize the motion controller in fashions like V. Tsounis et al. [4]. Trajectory optimization methods can be categorized into forward shooting methods and collocation methods. The former optimizes the trajectory through forward integration in time. Systems like fluid dynamics that are highly dependent on the previous states will be hard to converge and prone to suffer from stacking in the local maximum, similar to the exploration of model-based RL methods. The latter solves the trajectory problem for all states simultaneously and usually converges much faster, but it does not guarantee feasibility since since feasibility is considered to be a soft constraint. Moreover, collocation trajectory optimization methods require solving the inverse dynamics problem, which might be complicated to compute depending on the system.

I. Mordatch et al. [6] generated sets of optimized paths by trajectory optimization and then trained the network through imitation, producing fascinating results with optimal motions. L. Liu and J. Hodgins [7] learned the basketball dribbling control efficiently by separating the task into locomotion control and arms control since dribbling skills require the arm to do maneuvers with much more precision. To learn a detailed control policy for arm motions, they combine trajectory optimization with deep reinforcement learning.

## 2.2 Deep Motion Controller

H. Zhang et al. [8] introduced an architecture using the motion features to learn the phase transitions with mixture-of-experts for controlling quadruped characters. Their system contains a motion prediction network to compute the next state and a gating network to dynamically blend the experts' weights of the motion prediction network according to the current gait cycle.

For generating a character animation controller with the additional constraint of the control responsiveness, K. Lee et al. [9] used a teacher-student framework. They first train the teacher policy to optimize motions for time-critical responsiveness by predicting the trajectory and creating the corresponding motion with the traditional motion matching method. Then use the trained policy as a data generator to produce the training data with various conditions for the student policy. They use LSTM layers for the student policy to learn with time-series data.

The model proposed by D. Holden et al. [10] is inspired by the traditional motion-matching algorithm. The traditional motion-matching algorithm searches the database and finds the best-fit motion at runtime. The database scales linearly with the motion clips, resulting in the trade-off between the diversity of actions and the runtime performance. They improve the quality of the motion matching result, runtime performance, and memory usage by replacing the algorithm's key stages with their neural networks to exploit the scalability of neural network models. Thus can store more motion clips without excessive memory usage and loss of performance when searching the database.

S. Starke et al. [11] presented a deep periodic autoencoder to learn the periodic features for expressing the spatial-temporal structure of motions. By combining various downflow tasks, such as motion matching with the learned features, they demonstrate the effectiveness of the periodic features in improving the details and the transitions between different types of motions.

## 2.3 Swimming Simulation

J. Tan et al. [12] proposed a method to generate realistic swimming animations, they first present a method to model the two-way coupling simulation, then find the optimal open-loop swimming gait control of the articulated shape with the Covariance Matrix Adaptation (CMA) optimization method. Once the respective maneuvers of swimming straight, turning direction, and pitch up and down are learned, the path-following task can be performed by switching to the best-suited maneuver. S. Min et al. [13] and P. Ma et al. [14] proposed using differentiable simulation to find the optimal swimming motion, P. Ma et al. [14] combined the simulation with differentiable modeling of the swimmer's shape and control using a distribution model, thus achieves to optimizing swimming gait and shapes simultaneously to find the optimal swimmer design, but they both build upon the phenomenological fluid simulation model, also mentioned by S. Min et al. [13] and J. Tan et al. [12] that the simplified model produces unrealistic results and cannot express interactions of the fluid field, a more sophisticated fluid model is necessary to simulate the incompressible fluid for simulating the motions like generating thrust by spouting out the water

or the passive motion caused by the flows pushing the body.

Swimming simulations are also used for the studying of biological behaviors. M. Gazzola et al. [15] and S. Verma et al. [16] both solved the optimization by applying reinforcement learning methods to explore the swimming strategy of multiple fish when swimming together.

Regarding the methods of finding the optimal swimming motion, there are also methods focusing on the simulation of schooling behaviors or swarm simulations. D. Satoi et al. [17] utilized periodic functions to produce swimming-like motions and make the fish behave more naturally by adding randomness in the motion planner, sampled by a probabilistic distribution considering the schooling behaviors with pipe following, avoidance, aggregation, and directions.



Q. Chen et al. [18] presented the parametric periodic function to model the flapping animation, and combining the phenomenological aerodynamics to model the lift forces from the wind, and use the random vortex method to add chaotic actions into the system and reproduce the randomness of insect flying.

# Chapter 3      Method

Our system consists of two stages. First, we use deep reinforcement learning to find the optimal swimming motion for the given swimmer. Next, we enforce teacher-student training to adapt the learned policy to two types of control inputs. We train a motion-matching network using generated motions from the simulation environment with the transformed control inputs. By changing the control input from low-level objectives used for RL to high-level input as root motion trajectory, in this stage, the student network is trained using learned result motions of the simulation. Therefore we can moderate the undesirable noisy behaviors exhibited in the first stage since the student network doesn't have to interact with the chaotic and complex fluid simulation environment and exploit those effective motions in the training data.



## 3.1 Gait Policy Learning

We use reinforcement learning to find each swimmer type's effective swimming motions under the simulation environment. The goal is swimming toward the target direction, which requires learning suitable propelling and turning controls. The optimal control problem is modeled as a Markov Decision Problem (MDP) that aims to maximize the expected trajectory reward by finding the parameters of the policy  $\pi_\theta(a_t|s_t)$  that outputs the probability of choosing an action  $a_t$  at state  $s_t$ , defined with states  $s_t \in \mathcal{S}$ , actions  $a_t \in \mathcal{A}$ , a dynamic function  $\mathcal{P}(s_{t+1}|s_t, a_t)$  denoting the probability distribution of transition to the next state  $s_{t+1}$  given the current state and

action.  $T$  is the number of steps in each episode,  $\gamma \in [0, 1)$  is the discount factor and  $R_t$  is the reward function respect to  $s_t$  which returns a reward.

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^T \gamma^t R_t \right] \quad (3.1)$$

where  $p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$  represents the probability distribution over the trajectories  $\tau = (s_0, \alpha_0, s_1, \alpha_1, \dots, s_{T-1}, \alpha_{T-1}, s_T)$ .

We choose the Proximal Policy Optimization (PPO) [19] method with clipped loss to train our gait policy network for each swimmer.

As for the simulation environment, We use the method described by J. Tan et al. [12] to solve the two-way coupling interactions between the incompressible fluid and the swimmer expressed in an articulated body. The two-way coupling is realized by integrating the articulated body and fluid independently to intermediate states  $u^*$  and  $\dot{q}^*$  where  $u^*$  and  $\dot{q}^*$  are the intermediate fluid velocity field and joint velocity expressed in generalized coordinate. Then solve for the pressure field to satisfy the in-compressible fluid condition combining multi-body dynamics on the coupled surface simultaneously. And the actuation control of the articulated body uses the Stable Proportional-Derivative(SPD) controller. For more details on this method, we refer to their original work.

### 3.1.1 Policy Representation

The gait policy  $\pi_\theta(\tilde{s}_t, \hat{c}_t)$  takes the concatenated vector of current state  $\tilde{s}_t$  and the target direction control  $\hat{c}_t$  as the input observation and outputs an action vector  $\alpha_t^* \in [0, 1]^n$ . The action vector is then mapped to the joint

actuation control  $\alpha_t$  with a linear mapping  $f_{actuator}$  from  $[0, 1]$  to each actuator's limit  $b_j \in [bound_j^{lower}, bound_j^{upper}]$ , and sent to the SPD controller.

$$\pi_\theta(\tilde{s}_t, \hat{c}_t) \rightarrow \alpha_{t+1}^* \quad (3.2)$$

$$\alpha_{t+1} = f_{actuator}(\alpha_{t+1}^*; b_0, b_1, \dots, b_n) \quad (3.3)$$

We parameterize the continuous output action space as a multivariate Gaussian distribution with the diagonal covariance matrix of the standard deviation  $\sigma_\theta$ , and the mean values output by the Neural Network.

The state is represented as  $\tilde{s} = \{r_v, r_\omega, v_{com}, j_p, j_v, \alpha\}$ , where  $r_v \in \mathbb{R}^2$  and  $r_\omega \in \mathbb{R}^1$  are the root linear and angular velocity respectively, and  $v_{com}$  is the center of mass velocity.  $j_p \in \mathbb{R}^n$  and  $j_v \in \mathbb{R}^n$  are the joint angles and joint velocities. And  $\alpha \in \mathbb{R}^n$  is the last actuator control vector.

The target control  $\hat{c} \in \mathbb{R}^2$  is a unit vector pointing to the target direction.

### 3.1.2 Reward Function

The reward for the gait policy network includes two terms.

$$r = r_{target} + r_{regularization}, \quad (3.4)$$

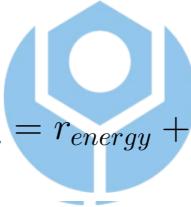
$$r_{target} = r_{velocity} * r_{direction}.$$

where

$$r_{velocity} = \vec{v}_{avg} \cdot \hat{forward} + \vec{r}_{vel} \cdot \hat{target}, \quad (3.5)$$

$$r_{direction} = \left| \frac{\hat{forward} \cdot \hat{target} + 1}{2} \right|^2.$$

The target direction term  $r_{target}$  rewards the agent when moving and facing towards the target direction, defined as the product of  $r_{velocity}$  and  $r_{direction}$ . Where  $\vec{v}_{avg}$  is the average center of mass velocity over 30 frames(1 second) duration, and  $forward$  and  $target$  are the swimmer's forward direction and target direction represented as unit vectors. It amplifies the penalties along with the training process as the policy becomes better at the task. Since the movement of the swimmer is unknown at the beginning, enforcing the multiplicative reward can automatically modulate different reward terms without manually tuning the weights. V. Tsounis et al. [4] also mention the effect of multiplicative reward in their work on balancing among reward terms.



$$r_{regularization} = r_{energy} + r_{exploration}. \quad (3.6)$$

The regularization term contains rewards to keep the swimmer's motion behaving naturally and encourages policy exploration.

$$r_{energy} = \begin{cases} -\omega_{energy} * (W - W_{bound}), & \text{if } W \geq W_{bound} \\ 0, & \text{otherwise} \end{cases}, \quad (3.7)$$

$$W = \sum_i \tau_i \dot{q}_i.$$

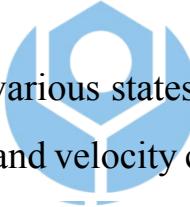
$r_{energy}$  penalizes excessive energy usage, calculated by summing up the works done by the actuated joints.  $W_{bound}$  is estimated using the torque limits defined by the user.

$$r_{exploration} = \omega_{exploration} \sum_i |q_{i,t} - q_{i,t-1}|. \quad (3.8)$$

$r_{exploration}$  encourages the exploration by rewarding the agent for changing the joint positions.

We set  $\omega_{exploration} = 0.1$ , and  $\omega_{energy} = 1.0$ , for all swimmers in our experiments.

### 3.1.3 Network Training



We train the agent to adapt to various states and targets by initializing the episode with different position and velocity configurations. And use a retry mechanism to randomly continue the episode from the previous terminated point and reset the fluid field when the reward is below average. In this way, the agent can have a vision of long-term rewards and deal with more challenging initial states requiring extra exploration.

To prevent the RL agent from getting stuck in the local maximum caused by the dramatic target direction difference between the agent's forward direction and the target direction. We ease out the input target direction using spherical linear interpolation to ensure a smooth transition for the RL agent.

We also enforce a curriculum learning scheme inspired by W. Yu et al. [20] to actively apply an aid force to the swimmer to simplify the gait learning process. Learning motions without the guidance of motion exam-

ples is considerably difficult.

The aid force is implemented by apply external velocity  $v_{ext,t}$  to the swimmer's root body directly until reaching the desired velocity  $v_\varepsilon$  towards the forward direction. We calculate the magnitude with the proportional and derivative control.

$$v_{ext,t} = \begin{cases} 0 & , \text{if } v_\varepsilon \leq v_{fwd,t} \\ -k_p(v_{fwd,t} - v_\varepsilon)\Delta t - k_d(v_{fwd,t} - v_{fwd,t-1}), & \text{otherwise} \end{cases} \quad (3.9)$$

where  $v_{fwd,t} = r_{v,t} \cdot \hat{forward}_t$ .



We schedule the curriculum to tune down the effectiveness of  $v_{ext}$  according to the average reward earned to gradually increase the difficulty of the curriculum for the agent. The exerted power is modified by setting the coefficients  $k_p$  and  $k_d$  of the PD controller.

The curriculum schedule is defined as follows. It will proceed to the next curriculum and decay the  $v_\varepsilon$  the multiplier of 0.5 to reduce the assistant force if the final average reward reaches 60% of the previous curriculum after fixed amounts of simulation steps which we consider the policy under a weaker. Otherwise, if it fails to pass the threshold, we will continue the current curriculum to let the agent does additional explorations.

The max and min values of  $v_\varepsilon$  are  $\frac{h}{5}$  and 0.0.  $h$  represents the swimmer's total length from head to tail.

$k_p$  and  $k_d$  are set to 1.0 and 0.1 in our experiments.

## 3.2 Controller Learning

The controller is trained with supervised learning where the motion sequence data are the outcomes generated using the previously trained policy described in Section 3.1.

We introduce two types of controller networks to learn with the teacher-student learning scheme, we consider the policy distillation as a motion-matching problem.

The first controller uses Recurrent Neural Network (RNN) to process the time-series data with input states defined as the same as the gait policy. And the motion-matching controller takes trajectories as extra states and control indicators to capture the sequential information in motion.



### 3.2.1 Data Generation

As mentioned in Section 1, the fluid flows act chaotically. The quality of motion will decrease as the complexity of fluid flows escalate through time due to the overwhelming complex fluid flows that are challenging for the agent to comprehend compared to the training stage. We want to sample from steadier states because the motions are more under control, given that the flow effect in those samples hasn't grown massively. It is also more accessible for the controller network to learn the motions without turbulence induced by previous states.

The generation step is by uniformly sampling the target direction and the episode length. The RL agent generates the motion sequences with the

simulation environment. The next episode uses the last state from the previous episode as its initial state and repeats this process to generate sequences of motion data.

At the beginning of each episode, we reset the fluid field to remove the fluid flows and limit the sequence length to avoid building an over-complicated environment for the RL agent and the controller.

We will discard the episodes whose overall rewards and increase rate are below average, and start over with a random initial state.

### 3.2.2 Direct Distillation Controller



We use an RNN model defined in Figure 3.1 for the first controller to imitate the behaviors of the motions in the dataset. The network architecture is proposed by K. Lee et al. [9]. It consists of an encoder-decoder with stacked LSTM layers in the middle. The input state at time  $t$  is defined as  $x_t^{rnn} = \{r_t^v, r_t^\omega, v_t^{com}, j_t^p, j_t^v, \hat{c}_t\}$  and the output state is  $y_t^{rnn} = \{r_{t+1}^v, r_{t+1}^\omega, v_{t+1}^{com}, j_{t+1}^p, j_{t+1}^v\}$ . Which is the same as the teacher policy input.

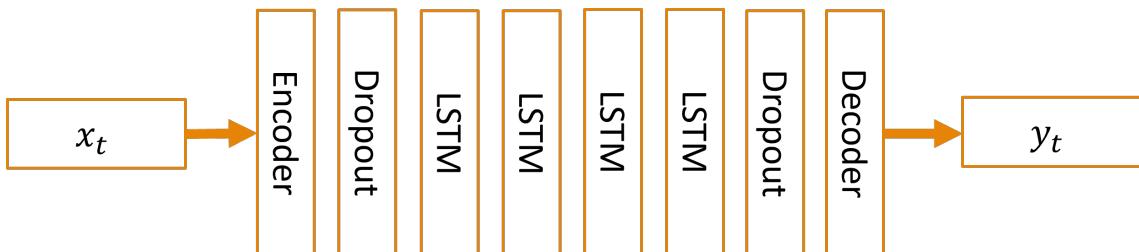


Figure 3.1: The network architecture used to model the direct distillation controller.

### 3.2.3 Motion-Matching Controller

Unlike the previous controller only performs policy distillation with an identical input scheme to the teacher policy. The motion-matching controller takes high-level states containing past and future trajectories as the new input state

$$x_t^{cross} = \{r_t^v, r_t^\omega, v_t^{com}, j_t^p, j_t^v, t_t^{past}, t_t^{desired}, t_t^{phase}\}$$

, and the output state is

$y_t^{cross} = \{r_{t+1}^v, r_{t+1}^\omega, v_{t+1}^{com}, j_{t+1}^p, j_{t+1}^v, t_{t+1}^{future}\}$ . The additional term  $t_{t+1}^{prev} \in \mathbb{R}^{4f_{prev}}$  is the past trajectory of root positions and velocities relative to the base frame t from frame  $t - f * samplerate$  to  $t - 1$ .  $t_{t+1}^{desired} \in \mathbb{R}^{4f_{desired}}$  is the desired trajectory according to the user control.  $t_{t+1}^{phase} \in \mathbb{R}^{4n \cdot w}$  is the phase trajectory for the phase encoder, contains joint angles  $j_p$ , joint velocities  $j_v$  and the joints' translational velocities relative to the root where  $n$  is the number of joints and  $w$  is the window size of the phase encoder. And  $t_{t+1}^{pred} \in \mathbb{R}^{4f_{pred}}$  is the future trajectory predicted by the controller network.

We use a similar network architecture as H. Zhang et al. [8] for our motion-matching controller where the inputs to the gating network are the phase features learned by the periodic encoder trained using the method introduced in [11] by S. Starke et al..

The phase features significantly help to capture the periodic features and align the motion sequences in time, keeping the detailed movements exhibited during transitions.

The desired trajectory  $t_t^{desired}$  is created by smoothly interpolating the target direction and base velocity with the current state. An additional process is added to transform the linearly interpolated trajectory into a trajec-

tory  $t_t^*$  that is in favor of the motion controller's training data. Which is critical for the controller to generate reasonable movements.

We achieve this in two ways. In the first method, we increase the time vacancy between samples in both training data and the desired trajectory since the input contains fewer dimensions and leaves more flexibility to the motion controller, but it requires some fine-tuning to find a balance between the control accuracy and the motion naturalness. In contrast to the first method, the second method outputs a detailed desired trajectory explicitly. It uses a fully connected network to infer a new trajectory given the linearly interpolated one and the periodic features.

Both controllers are trained with the MSE loss between the predicted states  $X$  and the training data  $Y$ .



$$L = MSE(X, Y). \quad (3.10)$$

We will discuss the parameters and comparison of the controllers on agility in the next section.

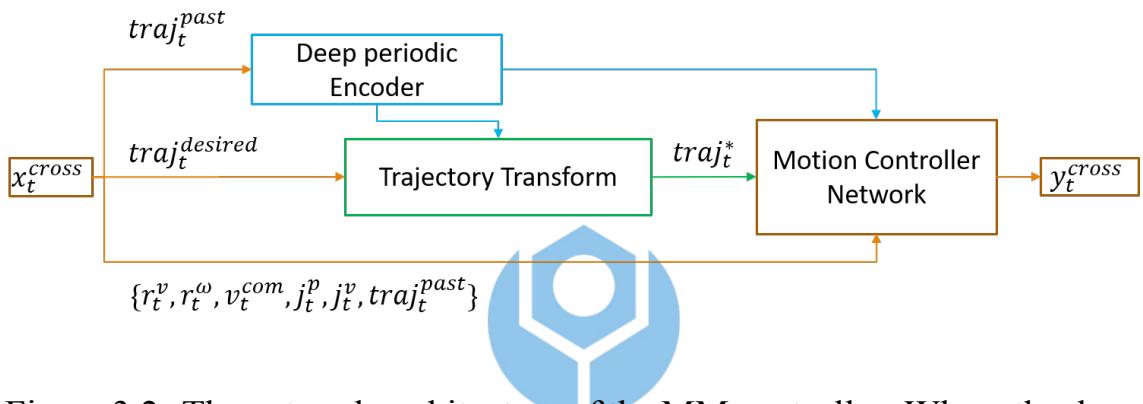


Figure 3.2: The network architecture of the MM controller. Where the deep periodic encoder [11] is used to extract the periodic features. The trajectory transform block then takes the linearly interpolated trajectory as input and outputs a new trajectory  $t_t^*$  with more realistic behavior. And the motion controller network [8] takes the periodic feature as the motion feature, the desired trajectory  $t_t^*$ , and a state vector to predict the future state  $y_t^{cross}$ .

# Chapter 4 Experiments

We implement the deep networks using Pytorch, and the simulation environment is built on C++ using the Pinocchio library for rigidbody dynamic algorithms.

We use the AdamW optimizer for all training processes. The gait policy takes at least 16 hours to train about 5,000,000 time steps, the computation is mostly cost by the simulation and depends on the grid resolution, 80 x 80 grid is used in our experiments. The simulation runs single thread on a desktop with a Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz and training with a NVIDIA Geforce GTX 1060 6GB graphic card.

The learning rate for the gait policy is  $1e^{-4}$  and the weight decay is  $5e^{-5}$ . The discount factor is 0.99 and the clip range is 0.2. We use the batch size 32 episodes with steps size 256, and sample the time steps into mini-batches of the size 64.

The RNN controllers take 12 hours on average to train, the motion-matching controller takes 8 to train and the periodic encoder takes 4 to 8 hours.

At runtime, we use the new state base on the controller output state to integrate the swimmer's root and joint positions and velocities, then store the results for the next prediction. And run an online process projecting the state back to feasible domain form by the training dataset to ensure the network can perform stably on the fly.

Due to the simulation environment using voxels to compute the fluid

dynamics, there will be stronger drags for the swimmer on the voxels' front sides. Thus, the resulting motion will depend on which direction it is swimming.

We reduce the effects brought by the drag difference by only changing the orientation of the swimmer and keeping the target direction at 45 degrees since the input states are in local space. In that fashion, we can ensure minimal fluid resistance in the target direction, and the drag difference won't govern the velocity term in the reward function.

## 4.1 Motion Distribution



We compute the distribution of the motions' velocity properties and compare the difference between the simulated motion produced by the gait policy and the deep controllers. We consider the results generated by the simulator with the gait policy as the baseline to examine the realism and agility of the RNN controller and the motion-matching controller.

Fig 4.1 shows the average velocities and standard deviations when applying different target directions and the time to converge to a stable swimming state. The direct distillation controller (middle) behaves similarly to the referenced gait policy (left) in comparison to the motion-matching controller (right). It also replicates those inaccurate behaviors from the training data since they share the same input states, the performance of the resulting motion generated by the direct distillation controller heavily depends on the gait policy. Results of the motion-matching controller indicate its motions converge towards the stable swimming state faster than the others, and the

velocity distribution is smoother across different target directions, as it was trained using information from the motions instead of the gait policy for the control input.

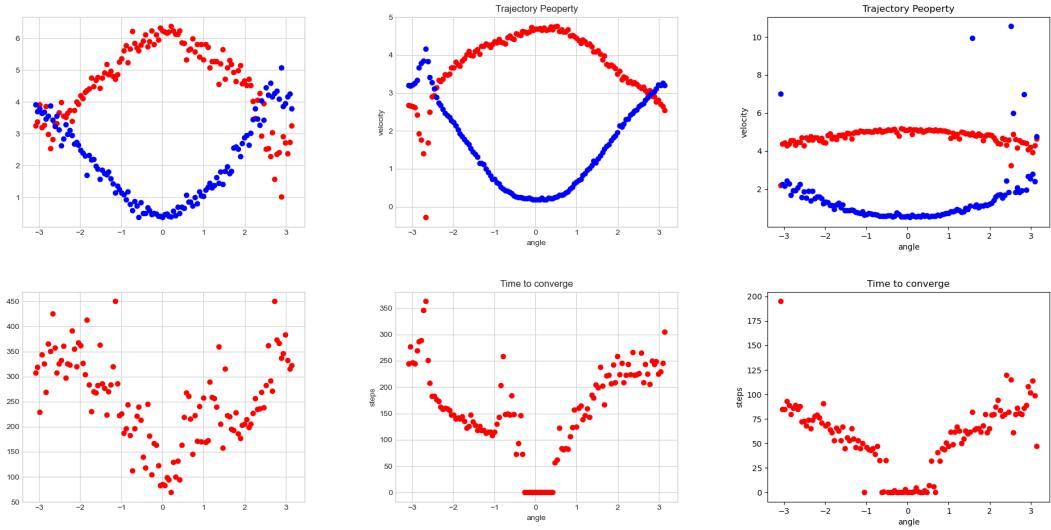


Figure 4.1: Top: Velocity means (red) and standard deviations (blue) to different target directions. Bottom: The time-to-converge steps to different target directions.



Figure 4.2: Generated motion sequence of a five-link swimmer turning. The swimmer turns by bending and undulating to create a smooth turn.



Figure 4.3: Learned turning motion of an alien shape swimmer.

## 4.2 Curriculum Learning Effectiveness

We demonstrate the effectiveness of our curriculum design in exploring the fluid environment with reinforcement learning by comparing it with the baseline PPO learning method. We compare the effectiveness of the easiest and the hardest swimmer design in our experiments, the learning curves and statistics of rewards can be seen in Fig 4.4 and Table 4.2.

Our approach reaches higher rewards and produces higher rewards in earlier phases, making the policy learning steadier than the baseline PPO method, the effect is distinguishable as the difficulty grows up. Without curriculum learning, the gait policy tends to stay still and stuck on the local maximal.

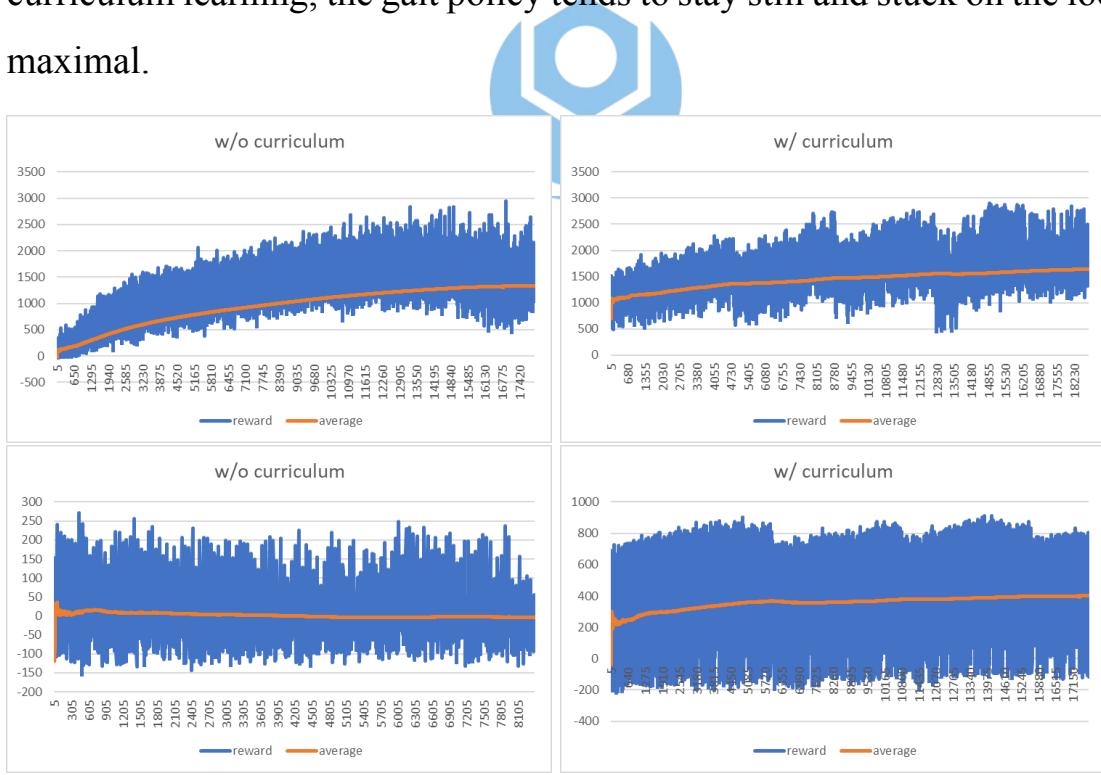


Figure 4.4: Comparison between baseline PPO and the proposed curriculum learning on different shapes and difficulties. Top: 4-link swimmer (easy). Bottom: 4-legged swimmer (hard)

	average	median	max	min
4-link w/o	1339.157	1358.581	2954.283	-27.374
4-link w/	1640.748	1631.555	2895.585	463.5683
4-legged w/o	-4.157	-21.382	270.321	-153.938
4-legged w/	400.809	428.647	911.536	-215.054

Table 4.1: The final reward statistics of each gait policy listed in Fig 4.4. There is a significant improvement in the average score and max score after applying our curriculum learning method.

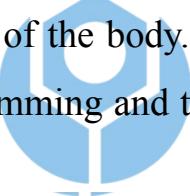
Besides the previously mentioned ones, there are other shapes and swimming motions explored by us. The goldfish is similar to the five-link swimmer with two fins on the sides of the body. These fins give it the ability to balance the body during swimming and to make sharper turns than the five-link swimmer.4.6.

Fig.4.7 shows an alien shape swimmer which has two long arms and a short tail. The symmetric motions of the arms were produced by enforcing additional constraints to synchronize the arms' movements. And Fig.4.8 shows the four-legged swimmer that also requires gait priors such as the symmetric constraint to learn swimming motions more effectively. Given that random sampling is not robust to generate such precise maneuvers without the guidance of prior knowledge.

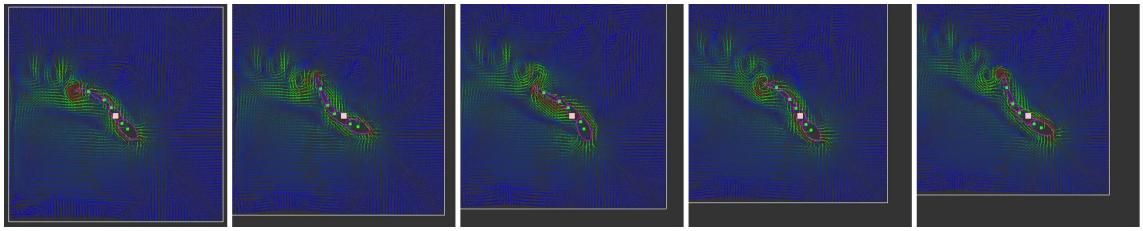


Figure 4.5: Motion sequence of a swimming five-link swimmer. This type of swimmer swims forward by undulating its lower body.

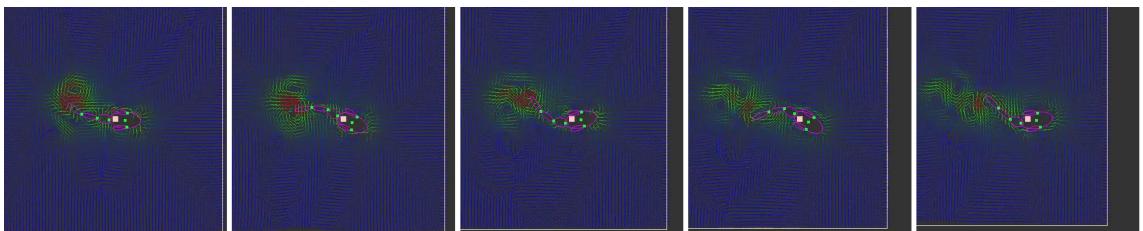


Figure 4.6: A goldfish-like swimmer with four links and two fins. Note the extra two fins help the swimmer to balance the body movement.

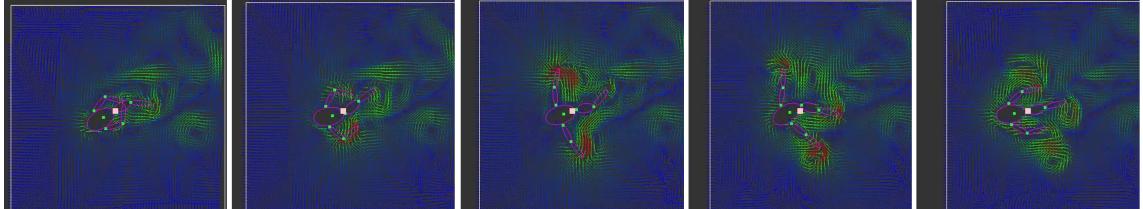


Figure 4.7: An alien shape swimmer which swims mainly by waving its two arms to create forward thrust and oscillating its tail to maintain direction and momentum.

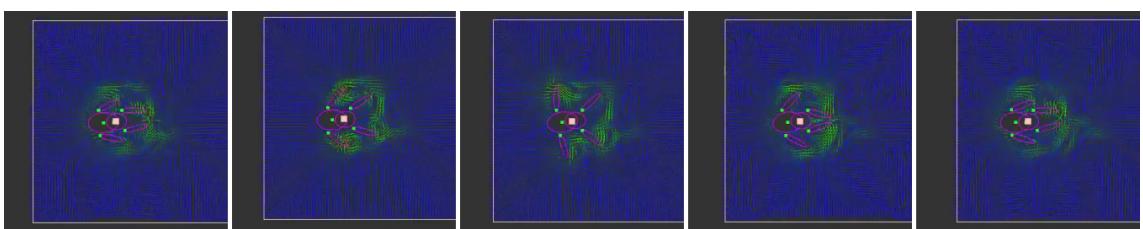


Figure 4.8: The four-legged swimmer moves through the fluid using a breaststroke-style gait. Note there is an extra constraint applied to force the lower legs in sync.

### 4.3 Test on Simplified Hydrodynamics

We test our system in a simplified hydrodynamics simulation environment to compare the effects on optimizing swimming motions. The simplified hydrodynamics model calculates the forces acting on the body depending on the velocity and normal vector on the surface. Without including the effects from the fluid flows, passive motions like hitting by the fluid flow from previous steps are missing. And the swimmers exploit the most out of the simplified model by maximizing the contact surface normal and velocity with the fluid to gain large forces towards the target direction. These effects result in unnatural swimming motions for the swimmers. However, with higher simulation resolution, simple shapes such as the five-link swimmer can still learn sufficiently good results using this simulation model.

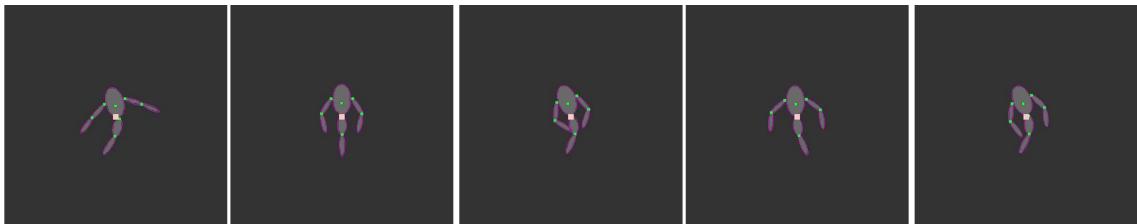


Figure 4.9: Generated swimming motion of an alien shape swims upward.

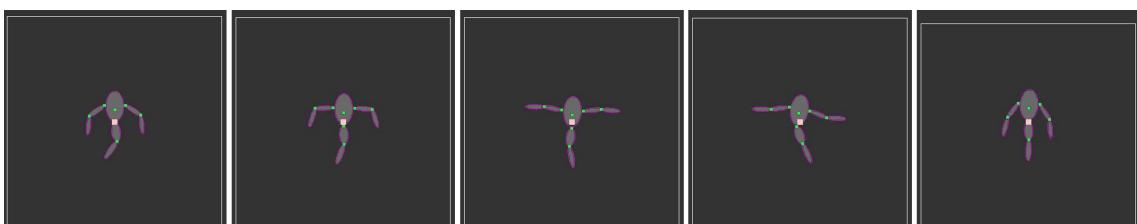


Figure 4.10: Motion sequence of an alien shape with simplified simulation.

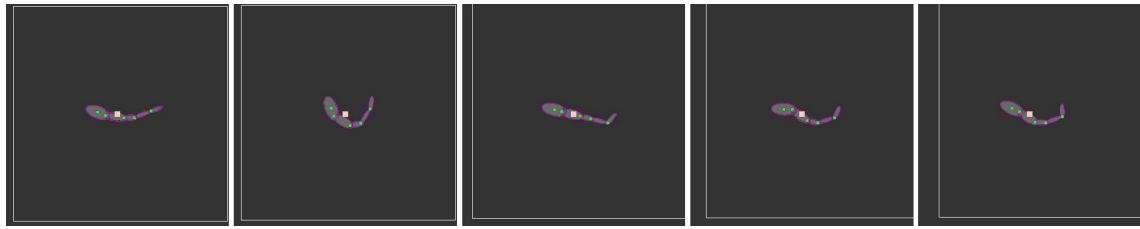


Figure 4.11: Motion sequence of a 5-link with simplified simulation. Note both Fig.4.10 and Fig.4.11 demonstrate the artifact of which the body movement is optimized to best utilize the simplified physic model by maximizing the surface and velocity of the body parts perpendicular to the moving direction.



Figure 4.12: Motion sequence of 4-link swimmer attached to a 3D model in Unity.

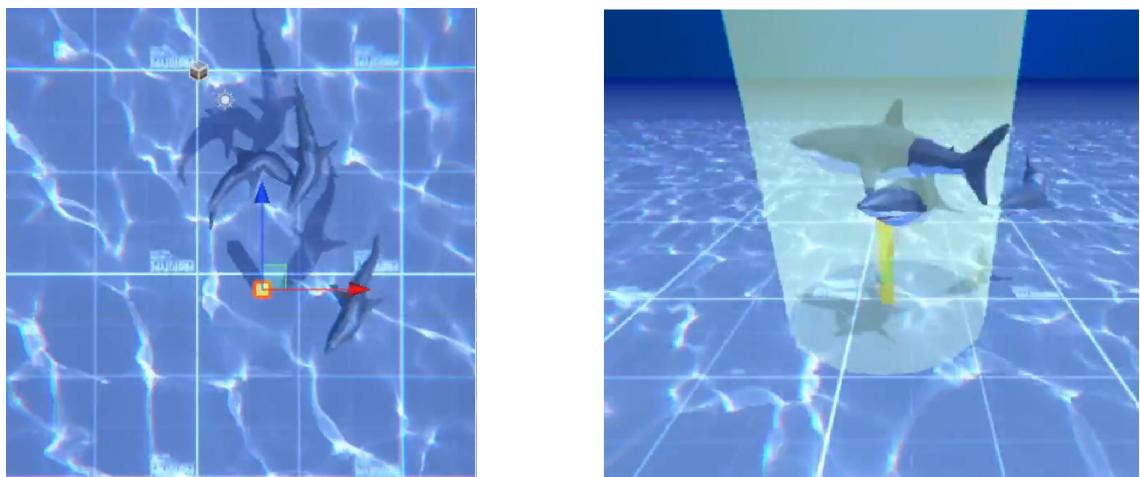


Figure 4.13: Screenshots from Unity editor of three sharks swimming towards a glowing pillar. Taken from left: editor view, and right: camera view.

# Chapter 5 Conclusions

Our proposed system can improve controllability through neural motion control to process the motion and learn motions beyond the designated target directions. Moreover, the motion controllers run in real-time since the computationally expensive fluid simulation is out of the table. And we introduced a curriculum learning algorithm to help the PPO agent explore more robustly and smoothly with the fluid simulation environment.

The efficiency and stability of our RL-based method are not general enough to apply to arbitrary given shapes. Constraints of motion priors are still required in our work to reduce the training difficulty.

For fast iterating the design of our deep network and reward function and testing other different optimization methods, we only implemented the system on a 2D simulation environment. However, a 3D environment is crucial for simulating an accurate motion closer to reality with more surface area and fluid field to interact. But the training stage of the gait policy needs a large number of simulations to learn properly with reinforcement learning. Training with a 3D configuration will be too slow for our system, so we leave the optimization of the simulation as another future work.

## 5.1 Future Work

We consider replacing the reinforcement learning with a faster converging and less unstable algorithm like trajectory optimization or differentiable dynamics. I. Mordatch et al. [6] and L. Liu and J. Hodgins [7] both pro-

pose methods that utilized the trajectory optimization and can significantly improve the performance of the learning process. Differentiable simulation methods are also proven applicable to physic simulation optimization problems demonstrated by P. Ma et al. [14] and B. Ramos et al. [21].

Also, passing local flow information to the agent lets it fully sense its surroundings instead of only depending on the kinesthetic states. And applying the peripheral inputs to the motion controllers can also improve the results. A motion controller that responds to flow interference is essential to capture the interaction with a dynamic environment or reproduce accurate schooling behaviors.

And finally, the training dataset can have motions for different tasks to create a fully controllable character that behaves realistically.



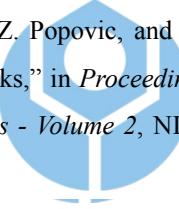
## 5.2 Limitations

The current fluid simulator could not fully model a realistic underwater environment, as described in the previous section the use of voxel grids is not optimal for fluid dynamic optimization.

Our gait optimization does not include various velocity control. The deep motion controller can only give some levels of velocity control with the trajectory input and will produce unrealistic results regarding the lack of motion dataset.

The fluid field is not considered as an input to the agent, the motion controller is incapable to react to external forces.

# References

- [1] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning,” *ACM Trans. Graph.*, vol. 36, Jul 2017.
- [2] J. Won, D. Gopinath, and J. Hodgins, “Physics-based character controllers using conditional vaes,” *ACM Trans. Graph.*, vol. 41, Jul 2022.
- [3] L. Fussell, K. Bergamin, and D. Holden, “Supertrack: Motion tracking for physically simulated characters using supervised learning,” *ACM Trans. Graph.*, vol. 40, Dec 2021.
- [4] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, “Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning,” 2019.
- [5] P. Fernbach, S. Tonneau, and M. Taïx, “Croc: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, 2018.
- [6] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. Todorov, “Interactive control of diverse complex characters with neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, (Cambridge, MA, USA), p. 3132—3140, MIT Press, 2015.
- [7] L. Liu and J. Hodgins, “Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning,” *ACM Trans. Graph.*, vol. 37, Jul 2018.
- [8] H. Zhang, S. Starke, T. Komura, and J. Saito, “Mode-adaptive neural networks for quadruped motion control,” *ACM Trans. Graph.*, vol. 37, Jul 2018.
- [9] K. Lee, S. Min, S. Lee, and J. Lee, “Learning time-critical responses for interactive character control,” *ACM Trans. Graph.*, vol. 40, Jul 2021.
- [10] D. Holden, O. Kanoun, M. Perepichka, and T. Popa, “Learned motion matching,” *ACM Trans. Graph.*, vol. 39, Jul 2020.
- [11] S. Starke, I. Mason, and T. Komura, “Deepphase: Periodic autoencoders for learning motion phase manifolds,” *ACM Trans. Graph.*, vol. 41, Jul 2022.
- [12] J. Tan, Y. Gu, G. Turk, and C. K. Liu, “Articulated swimming creatures,” *ACM Trans. Graph.*, vol. 30, Jul 2011.
- [13] S. Min, J. Won, S. Lee, J. Park, and J. Lee, “Softcon: Simulation and control of soft-bodied animals with biomimetic actuators,” *ACM Trans. Graph.*, vol. 38, Nov 2019.
- [14] P. Ma, T. Du, J. Z. Zhang, K. Wu, A. Spielberg, R. K. Katzschatmann, and W. Matusik, “DiffAqua,” *ACM Transactions on Graphics*, vol. 40, pp. 1–14, Aug 2021.

- [15] M. Gazzola, B. Hejazialhosseini, and P. Koumoutsakos, “Reinforcement learning and wavelet adapted vortex methods for simulations of self-propelled swimmers,” *SIAM Journal on Scientific Computing*, vol. 36, no. 3, pp. B622–B639, 2014.
- [16] S. Verma, G. Novati, and P. Koumoutsakos, “Efficient collective swimming by harnessing vortices through deep reinforcement learning,” *Proceedings of the National Academy of Sciences*, vol. 115, pp. 5849–5854, May 2018.
- [17] D. Satoi, M. Hagiwara, A. Uemoto, H. Nakadai, and J. Hoshino, “Unified motion planner for fishes with various swimming styles,” *ACM Trans. Graph.*, vol. 35, Jul 2016.
- [18] Q. Chen, T. Lu, Y. Tong, G. Luo, X. Jin, and Z. Deng, “A practical model for realistic butterfly flight simulation,” *ACM Trans. Graph.*, vol. 41, Mar 2022.
- [19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.
- [20] W. Yu, G. Turk, and C. K. Liu, “Learning symmetric and low-energy locomotion,” *ACM Transactions on Graphics*, vol. 37, pp. 1–12, Aug 2018.
- [21] B. Ramos, F. Trost, and N. Thuerey, “Control of two-way coupled fluid systems with differentiable solvers,” 2022.



