The Pennsylvania State University

The Graduate School

# OPTIMIZATION OF CENTRAL PATTERN GENERATOR PARAMETERS IN A SIMULATED UNDULATORY FISH ROBOT

A Thesis in

Mechanical Engineering

by

Matthew B. Ng

© 2020 Matthew B. Ng

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2020

The thesis of Matthew B. Ng was reviewed and approved by the following:

Bo Cheng
Associate Professor of Mechanical Engineering
Thesis Advisor

Anne Martin
Assistant Professor of Mechanical Engineering
Faculty Reader

Daniel Haworth
Professor of Mechanical Engineering
Chair of the Graduate Program

## ABSTRACT

This thesis investigates the implementation and optimization of a central pattern generator (CPG) in a biologically-inspired undulatory fish robot, known as the Modular Undulatory robot (MUBot). CPG is a mathematical model that generates periodic, oscillatory signals from a simple command input. Utilizing the Matsuoka CPG model allows for a state feedback signal to influence the locomotive behavior. However, the challenge in implementing the CPG lies in its laborious parameterization as there is no standard methodology to tune the large number of CPG parameters. In this thesis, the CPG parameters are optimized using Policy Gradient with Parameter Exploration (PGPE) method in a simulated MUBot environment, developed in Gym and Gazebo via the Robot Operating System (ROS). The PGPE method optimizes the CPG parameters to yield a fast, forward-swimming gait. The motivation for this thesis is to apply the learned CPG model to the physical MUBot in an effective manner such that fast swimming speeds and high maneuverability will be achieved without extensive experimental tuning or learning.

# TABLE OF CONTENTS

LIST OF FIGURES ............ v

LIST OF TABLES ............ vi

ACKNOWLEDGEMENTS ............ vii

Chapter 1 Introduction ............ 1

- Literature Review ............ 1
- Modular Undulatory Robot (MUBot) ............ 2

Chapter 2 Implementation of the Matsuoka CPG Model for the MUBot ............ 4

- Mathematical Model and Simulation for the Original Matsuoka CPG Model ............ 4
- Mathematical Model and Simulation for the Modular Matsuoka CPG Model ............ 6
- Methodology of CPG Model in the Online Simulation Environment ............ 8

Chapter 3 Reinforcement Learning Using Policy Gradient with Parameter Exploration ............ 10

- Policy Gradient with Parameter Exploration Learning ............ 10
- Application of Learning Strategy in Simulation ............ 11

Chapter 4 Training Methodology for CPG ............ 13

- Simulation Methodology ............ 13
- Learning Methodology ............ 14

Chapter 5 Results from Training Simulation ............ 17

- Mock Physical Experimentation Trial ............ 17
- Modified Reward with Penalty Trial ............ 19
- Gait Analyses for the Learned CPG Outputs ............ 22

Chapter 6 Conclusion ............ 27

References ............ 28

# LIST OF FIGURES


Fig. 1-1: Shortened MUBot robot used in simulation............................................................2

Fig. 1-2: Layout of custom-built actuator in each MUBot segment. ......................................3

Fig. 2-1: Sample network used to verify the original Matsuoka CPG model ..........................5

Fig. 2-2: Simulation result from sample network. ..................................................................6

Fig. 2-3: New sample network for the modular Matsuoka CPG model..................................7

Fig. 2-4: Simulation result for sample modular network. .......................................................8

Fig. 4-1: General structure of the MUBot simulation ............................................................14

Fig. 4-2: General structure of a PGPE learning episode. ........................................................15

Fig. 5-1: PGPE learning results for the first trial of simplified learning (μ±σ) ......................18

Fig. 5-2: History of each parameter's distribution for first trial (μ±σ) ..................................18

Fig. 5-3: Learned, full CPG output for the first trial..............................................................19

Fig. 5-4: Section of the learned, steady-state CPG output for the first trial............................19

Fig. 5-5: PGPE learning results using the modified reward with penalty...............................20

Fig. 5-6: History of each parameter's distribution for second trial (μ±σ)...............................21

Fig. 5-7: Learned, full CPG output for the second trial ..........................................................21

Fig. 5-8: Section of the learned, steady-state CPG output for the second trial ........................22

Fig. 5-9: Full, learned gait for the first trial ............................................................................24

Fig. 5-10: Section of learned gait for the first trial .................................................................24

Fig. 5-11: Full, learned gait for the second trial......................................................................25

Fig. 5-12: Section of learned gait for the second trial.............................................................26

Fig. 5-13: Snapshots of first trial's learned trajectory at two second intervals........................26

Fig. 5-13: Snapshots of second trial's learned trajectory at two three intervals ......................27

# LIST OF TABLES

Table 2-1: Parameters used in MATLAB simulation. ............................................................5

Table 2-2: Parameters used in MATLAB simulation. ............................................................7

Table 4-1: Initial parameters used to begin PGPE learning. ....................................................15

Table 5-1: Learning hyperparameters for the first trial..........................................................17

Table 5-2: Learning hyperparameters for the second trial. ......................................................20

## ACKNOWLEDGEMENTS

The author would like to acknowledge Dr. Bo Cheng for offering the opportunity to work on this project. Hankun Deng, and Donghao Li are acknowledged for their excellent teamwork and cooperation. Patrick Burke, and Jackson Sizer are acknowledged for their emotional and mental support that they provided. Finally, the author would like to acknowledge his family and loved ones for their endless support during the COVID-19 pandemic.

This material is based upon work supported by the National Science Foundation's Division of Computer and Network Systems under Award No. 1932130. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

## Chapter 1

# Introduction

Underwater robotics is an emergent study that pioneers a new way of studying aquatics, learning about underlying principles for locomotion, or discovering new functions for biological traits. With a robust robot platform, users could explore the ocean in revolutionary ways. A robot could covertly study marine life and environments. There are also defense applications. Equipped with the proper sensors and hardware, a robot could complete tasks such as explosive ordinance disposal, reconnaissance, and payload transportation. Since ground robotics can accomplish some of these tasks already, progress could be extended into underwater robotics to yield an amphibious concept.

## Literature Review

Combining this study with biologically-inspired design can help researchers understand more about aquatic animals, namely their different swimming gaits. With a better understanding of this feature, the design of an underwater robot will be functionally enhanced, making these types of robots more feasible and effective. Also, certain fluid dynamic phenomena can be investigated like the propulsive strategies of fish. For example, one investigated phenomenon is whether an eel propels itself by developing vortices along its body length [1] or generating high-pressure fields in the water [2]. New design criteria may be discovered and taken advantage of in order to develop an eel-like robot that swims faster.

Currently fish-like robots are being developed, and one of the success metrics is its swimming speed. As shown by Wen et al. [3], the Strouhal number is often used to characterize swimming performance, and its significance is emphasized by linking its value to a swimming robot's power efficiency. Another common measure of swimming speed is the value with respect to a robot's body length. Crespi et al. [4] developed a robotic fish that was able to swim up to 1.4 body-lengths-per-second using Central Pattern Generation (CPG) controller. This result is assumed to be from using a set of hand-tuned CPG parameters. However, hand-tuning CPG parameters is often considered difficult and not time efficient for the programmer [5-6]. Therefore, the literature suggests to use automated processes like machine learning to tune CPG parameters [6].

Applying machine learning has shown to be significantly successful. For comparison, the results in hand-tuned robot performances from two pieces of literature [7-8] were 0.16 and 0.955 body-lengths-per-second respectively. Using a similar robot with a learned parameter set yielded 1.15 body-lengths-per-second [9]. Similar success has been found in other robots using CPG control as well [10-12].

With new machine learning algorithms being developed, the option to learn the CPG controller becomes more attractive. One recently developed algorithm is known as Policy Gradient with Parameter Exploration (PGPE) [13], which boasts a faster convergence rate compared to previously used reinforcement learning strategies. However, PGPE has not been

applied frequently, perhaps due to how long it has been in the literature. Nonetheless, one application of PGPE was found to learn the CPG controller parameters for a robotic caterpillar [14]. Also comparing PGPE performance to that of other learning strategies, Ishige et al. showed successful results while other strategies failed to learn the task of climbing an obstacle of varying height. Therefore, PGPE could possibly be a major improvement for learning a CPG parameter set. This thesis seeks to develop the possibility by applying PGPE to learn a CPG-controlled swimming gait for an eel-like robot.

## Modular Undulatory Robot (MUBot)

The Modular Undulatory Robot (MUBot) is an underwater eel-like robot that is being developed in the Biological & Robotic Intelligent Fluid Locomotion Lab at The Pennsylvania State University. The robot consists of a series of identical body segments led by a head segment. Figure 1-1 shows a smaller version of the actual MUBot with a total body length of approximately 0.16 meters. The design of one segment allows for a range of ± 9.7 degrees.

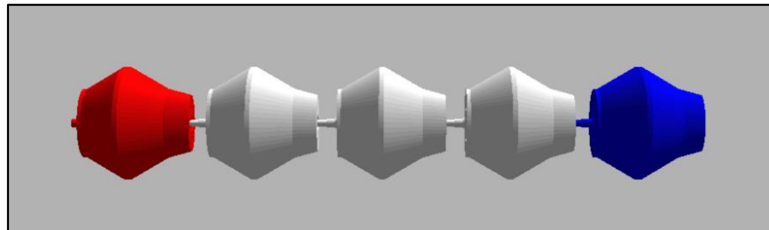

*Fig. 1-1: Shortened MUBot robot used in simulation*

In an attempt to conform to the morphology of an eel, these segments were designed to be as narrow and shallow as possible. Therefore, a custom-built actuator connects one segment to the next consecutive segment. Pictured in Figure 1-2, this actuator consists of two cylindrical permanent magnets, one coil of wire, and one rod. The magnets are mounted on opposite sides of a segment's interior with their northern sides facing each other. The rod acts as a moment arm by connecting the wire coil to the fulcrum. A voltage is applied across the wire coil, which induces a magnetic field like a solenoid. Modulating this voltage's magnitude and polarity makes the coil oscillate between the two magnets, which in turn exerts a torque that drives the MUBot's propulsive effort.

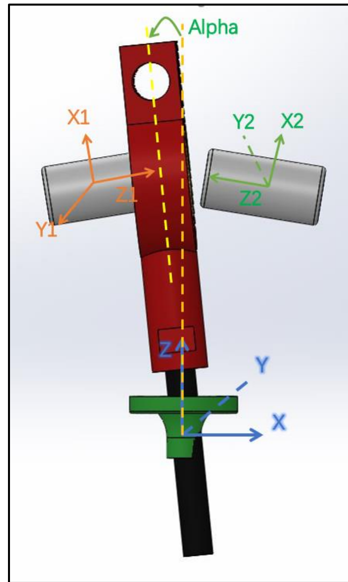


*Fig. 1-2: Layout of custom-built actuator in each MUBot segment*

## Chapter 2

# Implementation of the Matsuoka CPG Model for the MUBot

CPG networks are comprised of a series of ordinary differential equation sets. This set of equations changes based on the CPG model. From the connection pattern and model parameter values, the network will yield a varying periodic, oscillating signal. The Matsuoka CPG is a model that is favored for biologically-inspired robotics due to its similarities to biological systems, unlike other CPG models. Other models such as the Hopf oscillator were considered for their respective benefits such as easy implementation, or straightforward tuning methodology, but the Matsuoka model proved more beneficial for future research purposes. This chapter will introduce the design process for the network that was used in the MUBot, show offline simulations results in MATLAB, and explain the network's implementation in the learning simulation.

### Mathematical Model and Simulation for the Original Matsuoka CPG Model

The design process began with the original Matsuoka CPG model [15-16]. It defines a set of equations for an $i$-th set in a network comprised of $n$ sets:

$$\begin{gathered} \dot{x}_i + x_i = -\sum_{j=1}^{n} a_{ij} y_j + s_i - b x_i' \\ \tau \dot{x}_i' + x_i' = y_i \\ y_i = \max(0, x_i) \\ (i = 1,\ 2,\ \dots,\ n), \end{gathered} \tag{2.1}$$

where $x_i$ and $y_i$ are the state and output of the $i$-th set respectively. The parameter $a_{ij}$ is defined as the inhibition weight of the $i$-th set on the $j$-th set. The adaptation of the $i$-th set is defined as $x_i'$, which has adaptation constants $b$, and $\tau$. Finally, $s_i$ is a simple command input to the $i$-th set. Applying this model assumes no excitatory connections or self-inhibitory connections exist in the network. Thus, $a_{ij}$ must either be zero or positive, indicating either no relation exists or an inhibitory relation exists respectively.

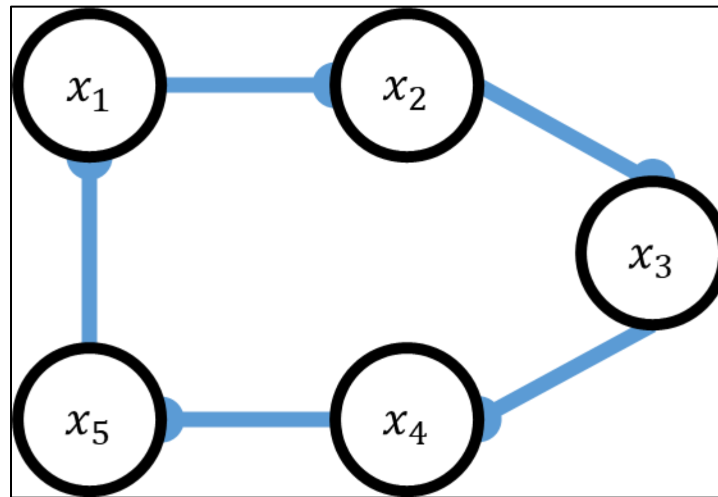


*Fig. 2-1: Sample network used to verify the original Matsuoka CPG model*

A network consisting of five sets of equations, shown in Figure 2-1, was simulated using MATLAB's ode45 numerical integrator with parameters shown in Table 2-1. The blue lines indicate inhibitory relations such that, for example, $x_1$ inhibits $x_2$. The simulation methodology will be discussed in detail later. The results seen from Figure 2-2 match the expected behavior from Matsuoka [15] as well as yield new observations.

*Table 2-1: Parameters used in MATLAB simulation*

| Parameter | Value |
|---|---|
| $a_{ij}$ | 2.7 |
| $b$ | 2.5 |
| $\tau$ | 12 |
| $s_i$ | [2.5,1,1,1,1] |
| $x_{i,0}$ | [1,0,0,0,0] |
| $x'_{i,0}$ | [0,0,0,0,0] |

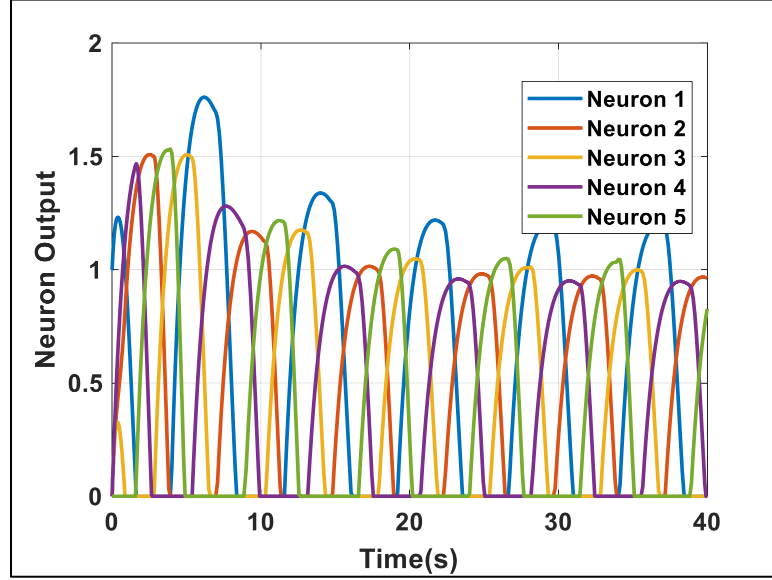


*Fig. 2-2: Simulation result from sample network*

The key observations are found in the activation pattern, the initial "transient" behavior, and non-inverting output. While the example network used was arbitrarily numbered, the inconsistency with the activation pattern and set numbering could be somewhat confusing when applied to the MUBot. The "transient" behavior may cause issues with the MUBot, but this behavior can be avoided by applying a classical PD controller [3] or better initial conditions for the network. However, the most important observation is that the original Matsuoka CPG model is unable to naturally invert its output, which is a critical feature for the MUBot's actuator. Therefore, the next modification to the control strategy necessitated a way to invert the CPG output.

### Mathematical Model and Simulation for the Modular Matsuoka CPG Model

The work from Wu et al. [17] was chosen to be followed due to its relative simplicity and suitability over other adaptations of the Matsuoka CPG model. This work involves adding an additional organizational layer, known as modules, that is a subset of set pairs in a network. Adding a new organizational layer between the entire network and the equation sets modifies the set of equations. Specifically for a network with $i$-th of $n$ sets per $k$-th of $m$ modules,

$$
\begin{gathered}
\tau_1 \dot{x}_{i,k} + x_{i,k} = \sum_{j=1}^{m} w y_{out,j} + u_i - b x'_{i,k} - a_{ij} y_{s,k} \\
\tau_2 \dot{x'}_{i,k} + x'_{i,k} = y_{i,k} \\
y_{i,k} = \max(0, x_{i,k}) \\
y_{out,k} = y_{1,k} - y_{2,k} \\
(i = 1,\ 2,\ \ldots,\ n;\ k = 1,\ 2, \ldots, m) \\
s = \begin{cases} n, & if\ i = 1 \\ i-1, & else \end{cases}
\end{gathered}
\tag{2.2}
$$

where most of the nomenclature remains to (2.1) except that the reference input $s$ is now represented as $u_i$. The new variables $y_{out,j}$, and $w$ represent the output of the $k$-th module, and the module-module inhibition constant respectively. Applying this model allows for each MUBot actuator to be represented with one module. Therefore, the important output becomes that of the module instead of an individual set.

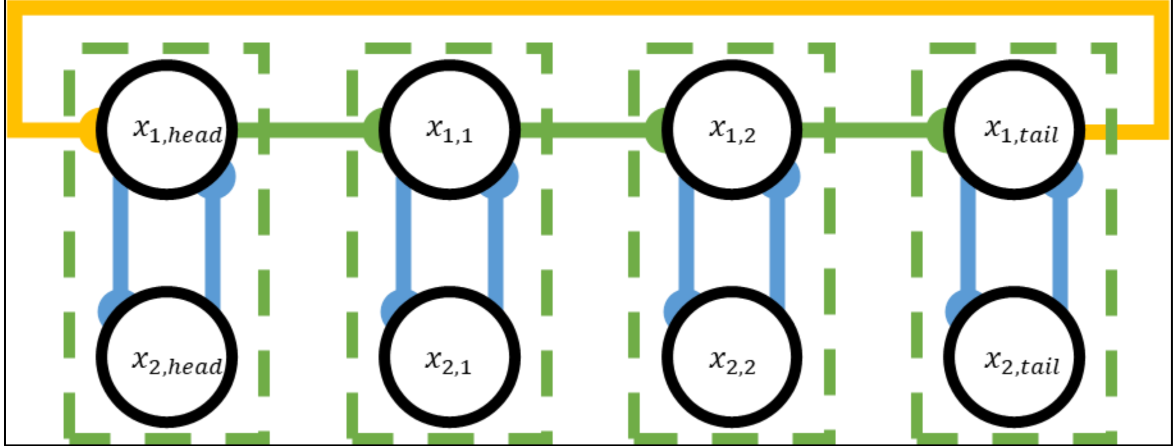


*Fig. 2-3: New sample network for the modular Matsuoka CPG model*

A new sample network with eight total sets grouped in four modules is shown in Figure 2-3. Each module contains a mutually-inhibiting pair of sets. To maintain clarity, the dashed green boxes represent each module in the network. The blue inhibitory connections represent relations within the module, and similarly the green inhibitory connections represent module-module relations within the network. The yellow inhibitory connection is a special module-module relation that defines the network's feedback. Changing the module that inhibits the head module affects the number of oscillations the network experiences at once [17]. Connecting the tail module to the head module allows the network to experience one oscillation at once, but this phenomenon is not explored further in this thesis. This network was computed similarly to the previous simulation using the parameters shown in Table 2-2, and the results are shown in Figure 2-4. Note that the amplitudes of the CPG output on Figure 2-4 were modified to reflect certain operational specifications for the MUBot. The legend indicates the actuator between each segment of the MUBot.

*Table 2-2: Parameters used in MATLAB simulation*

| Parameter | Value |
|---|---|
| $a_{ij}$ | 2.7 |
| $w$ | 0.1 |
| $b$ | 2.5 |
| $\tau_1, \tau_2$ | 0.02, 0.06 |
| $u_i$ | [25,25,25,25] |
| $x_{i,k0}$ | [1,0,0.5,0] |
| $x'_{i,k0}$ | [0.25,0,0.1,0] |

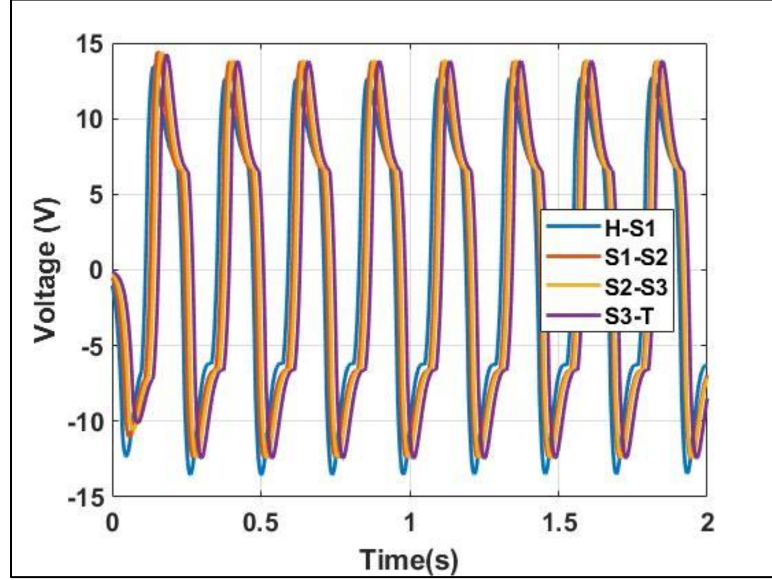


*Fig. 2-4: Simulation result for sample modular network*

Applying this modular network yields favorable results for the MUBot's design. Through proper parameterization, this network can not only achieve an inverting output, but also generate high-frequency operation. The activation sequence matches the sequence of modules, which makes the network's design more intuitive. Following the work of Lu et al. [20], adding complexity within the module can change the shape of the oscillations. However, other shapes were not considered.

## Methodology of CPG Model in the Online Simulation Environment

Now the offline CPG calculations will be explained in order to show how they are adapted for the simulation environment to find the commanded actuator voltages. In the MATLAB simulations, ode45 is used to numerically simulate the network. Specifically, a system of first-order, ordinary differential equations is built following the commonplace expression,

$$\dot{x} = \boldsymbol{A}x + \boldsymbol{b}u, \tag{2.3}$$

with state and input vectors for the original and modular models respectively being

$$x_{orig} = [x_1\ x_2\ \dots x_n, x_1'\ x_2'\ \dots\ x_n']^T$$
$$u_{orig} = [y_1\ y_2\ \dots\ y_n, s]^T \tag{2.4}$$

$$x_{mod} = \left[x_{1,1}\ x_{2,1}\ x_{1,2}\ x_{2,2} \dots x_{n,m}, x_{1,1}'\ x_{2,1}'\ x_{1,2}'\ x_{2,2}'\ \dots\ x_{n,m}'\right]^T$$
$$u_{mod} = \left[y_{1,1}\ y_{2,1}\ y_{1,2}\ y_{2,2}\ \dots\ y_{n,m}, y_{out,1}\ y_{out,2}\ \dots\ y_{out,m}, u\right]^T. \tag{2.5}$$

However, the activation function must also be included to account for the inherent nonlinearity in CPG. The sizes of matrices $\boldsymbol{A}$, and $\boldsymbol{b}$ depend on the model used. For the original model denoted in (2.4), the respective sizes are $[(2n)\ x\ (2n)]$, and $[(2n)\ x\ (n+1)]$, comprised of the terms found for the original Matsuoka model shown in (2.1). For the modular model denoted in (2.5), the respective sizes are $[(2mn)\ x\ (2mn)]$, and $[(2mn)\ x\ (mn+m+1)]$, comprised of the terms found in (2.2). To conform to the format in (2.3), the differential equations are isolated for the $\dot{x}$ term. While large, the matrices are sparse and comprised of the aforementioned CPG parameters,

and smaller diagonal matrices. For a simpler model comprised of two modules and two sets per module, the matrices $\boldsymbol{A}$, and $\boldsymbol{b}$ are

$$\boldsymbol{A} = \begin{bmatrix} \frac{-1}{\tau_1} & 0 & 0 & 0 & \frac{-b}{\tau_1} & 0 & 0 & 0 \\ 0 & \frac{-1}{\tau_1} & 0 & 0 & 0 & \frac{-b}{\tau_1} & 0 & 0 \\ 0 & 0 & \frac{-1}{\tau_1} & 0 & 0 & 0 & \frac{-b}{\tau_1} & 0 \\ 0 & 0 & 0 & \frac{-1}{\tau_1} & 0 & 0 & 0 & \frac{-b}{\tau_1} \\ 0 & 0 & 0 & 0 & \frac{-1}{\tau_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-1}{\tau_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{\tau_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{\tau_2} \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{b} = \begin{bmatrix} 0 & \frac{-a_{ij}}{\tau_1} & 0 & 0 & 0 & \frac{w}{\tau_1} & 1 \\ \frac{-a_{ij}}{\tau_1} & 0 & 0 & 0 & 0 & \frac{w}{\tau_1} & 1 \\ 0 & 0 & 0 & \frac{-a_{ij}}{\tau_1} & \frac{w}{\tau_1} & 0 & 1 \\ 0 & 0 & \frac{-a_{ij}}{\tau_1} & 0 & \frac{w}{\tau_1} & 0 & 1 \\ \frac{1}{\tau_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\tau_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\tau_2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\tau_2} & 0 & 0 & 0 \end{bmatrix} \tag{2.7}$$

where the nomenclature in the matrix elements follows that given in (2.2).

Using a similar strategy, the modular model is written for the simulation environment using four modules with two sets per module as shown in Figure 2-3. Initially, the code was written in Python 2 class to accommodate for the ROS distribution used. Even though most of the code remains the same, Python 2 does not directly have the ode45 integrator. Therefore a new function, odeint, from the numerical Python library, numpy, replaced the MATLAB function. Originally the code would be executed at every timestep to compute the voltages to command until the next timestep. However, the code was not suitable for that application, so it was modified to execute once at the beginning of each simulation to calculate the voltages for the entire simulation at 20 Hz.

## Chapter 3

# Reinforcement Learning Using Policy Gradient with Parameter Exploration

One of the most significant challenges with using CPG control is tuning the parameters for each set. While the literature offer some guidelines to hand-tuning parameters, there is no definitive strategy to the best of our knowledge. For example with the modular Matsuoka CPG model, changing a singular parameter can affect multiple aspects of the output behavior as detailed by Wu et al. [17]. Also, proper parameterization is necessary for a functional CPG network. Therefore, hand-tuning CPG parameters can be unreliable and difficult to accomplish.

Recently, machine learning became the main strategy for CPG parameterization. Various techniques like, genetic algorithm, and actor-critic reinforcement learning, have been applied to successfully optimize CPG parameters [7-12], which in turn optimizes the robot's performance. Using different machine learning algorithms can improve the convergence time, and the optimal result found, and numerous algorithms are still being developed that may further improve these aspects. This chapter will introduce the algorithm used to train the MUBot's CPG network: policy gradient with parameter exploration (PGPE) [13]. Its implementation to the simulation environment is also presented.

### Policy Gradient with Parameter Exploration Learning

PGPE is a relatively new reinforcement learning strategy that boasts a rapid convergence rate, and better optimum search technique. Its distinct benefit allows parameter exploration through sampling a normal distribution that describes a range of valid values for a given parameter. When a set of parameters, or policy, is found this way, it is then evaluated in a simulation that is quantified as a reward. The parameters' distributions are improved based on this performance relative to a baseline. Using this methodology, the learning process experiences less noise between each learning episode [13].

For a given episodic history, $h$, of robot states, $s_t$, and robot actions, $a_t$, that takes $T$ long, a cumulative reward is defined simply as

$$r(h) = \sum_{t=1}^{T} r_t, \tag{3.1}$$

with the instantaneous reward at time $t$ is defined as $r_t$. For the first part of the MUBot study, the instantaneous reward was designated to be

$$r_t = v_{x,head}, \tag{3.2}$$

where $v_{x,head}$ is the forward velocity of the MUBot's head segment, and the denominator is the sum of the power input to all actuators.

The specified expected reward function is

$$J(\theta) = \int_H p(h|\theta) r(h) dh, \tag{3.3}$$

where $p(h|\theta)$ is the probability of eliciting a history from a given set of CPG parameters, $\theta$. Differentiating (3.3) via finding its gradient with respect to $\theta$ allows for the expected reward to be maximized; thus, an optimized set of values for $\theta$ is found. After simplifications, and sampling application, the gradient is approximately

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_\theta p(a_t^n | s_t^n, \theta) r(h^n), \tag{3.4}$$

where

$$h^n = p(h^n|\theta). \tag{3.5}$$

At this point, PGPE's nuance is applied in order to predict the output of $p(a_t^n|s_t^n, \theta)$, which is the determination of instantaneous action given the current state and policy for the $n$-th history. The parameters in $\theta$ are fitted to normal distributions as

$$p(a_t|s_t, \rho) = \int_\theta p_G(\theta|\rho)\delta_{F_\theta(s_t),a_t} d\theta. \tag{3.6}$$

where $\rho$ are the means and standard deviations that define each parameter's distribution in $\theta$. $F_\theta(s_t)$ is defined as the chosen action for $s_t$ with a given $\theta$, which is the CPG trajectory. Substituting $\rho$ for $\theta$ in (3.3) and finding its new gradient yields a similar expression:

$$\nabla_\rho J(\rho) \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\rho \log p_G(\theta|\rho)\, r(h^n). \tag{3.7}$$

Expanding $\rho$ to their constituent means and standard deviations for the $i$-th parameter in $\theta$ yields

$$\nabla_{\mu_i} \log p_G(\theta|\rho) = \frac{\theta_i - \mu_i}{\sigma_i^2} \tag{3.8}$$

$$\nabla_{\sigma_i} \log p_G(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{3.9}$$

where $\mu_i$ and $\sigma_i$ are the mean and standard deviation of the $i$-th parameter in $\theta$ respectively. The equations (3.8, 3.9) are the basic equations used in order to approximate the reward gradient. To enforce the numerical constraints from using the CPG model, modifications to these equations were made to sample a Truncated Gaussian distribution. A Truncated Gaussian distribution allows for constraining a normal distribution without affecting its statistical integrity. This change in sampling method is reflected in the PGPE equations (3.8, 3.9)

$$\nabla_{\mu_i} \log p_{TG}(\theta|\rho) = \frac{\theta_i - \mu_i}{\sigma_i^2} + p_{TG}(b_i|\rho, a_i, b_i) - p_{TG}(a_i|\rho, a_i, b_i) \tag{3.10}$$

$$\nabla_{\sigma_i} \log p_{TG}(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3} + \frac{p_{TG}(b_i|\rho, a_i, b_i)(b_i - \mu_i) - p_{TG}(a_i|\rho, a_i, b_i)(a_i - \mu_i)}{\sigma_i}, \tag{3.11}$$

where $p_{TG}(b_i|\rho, a_i, b_i)$ and $p_{TG}(a_i|\rho, a_i, b_i)$ are the probability density functions for a Truncated Gaussian distribution with constraints defined for $\theta \in [a, b]$. The density function is defined as

$$p_{TG}(\theta|\rho, a_i, b_i) = \begin{cases} \frac{p_G(\theta|\rho)}{P_G(b|\rho) - P_G(a|\rho)} & \theta \in [a, b] \\ 0 & \text{otherwise} \end{cases}, \tag{3.12}$$

where $p_G$ and $P_G$ are the probability density function and the cumulative density function respectively for a Gaussian distribution.

### Application of Learning Strategy in Simulation

Similar to the implementation of the modular Matsuoka CPG model, PGPE is coded in Python 2 as a class. The Numerical Python library included the necessary function to sample from a Truncated Gaussian distribution. The episodic loop consists of sampling a set of CPG parameters, $\theta$, and finding the cumulative reward, $r(h)$. This process is repeated several times where each iteration is referred to as a rollout so that the current policy's performance is better known. Using multiple rollouts also allows for the application of an episodic baseline that is used to compare each rollout relative to one another. An optimized baseline definition, $b$, from the work of Zhao et al. [21]

$$b = \frac{\boldsymbol{E}[R(h)||\nabla_\rho \log p_{TG}(\theta|\rho)||^2]}{\boldsymbol{E}[||\nabla_\rho \log p_{TG}(\theta|\rho)||^2]}. \tag{3.13}$$

The numerator is defined as the expected value of the rollout rewards multiplied by the respective squared magnitudes of the characteristic eligibility, $\nabla_\rho \log p_{TG}(\theta|\rho)$. The baseline could simplify

to the expected average of the rollout rewards, but only if the two aforementioned terms are independent of each other. However, this is not the case as observed through experimentation.

Once enough rollouts are recorded, the policy, $\rho$, updates based on the resulting gradients using

$$\mu_{i,new} = \mu_i + \alpha_\mu \nabla_{\mu_i} \log p_{TG}(\theta|\rho, a_i, b_i)\,(r(h) - b) \tag{3.14}$$

$$\sigma_{i,new} = \sigma_i + \alpha_\sigma \nabla_{\sigma_i} \log p_{TG}(\theta|\rho, a_i, b_i)\,(r(h) - b), \tag{3.15}$$

where $\alpha_\mu$ and $\alpha_\sigma$ are the learning rates for the mean and standard deviation respectively. The learning rates for each parameter can be tuned separately to enhance the learning. Also, increasing the number of simulated rollouts per episode allows for better learning because more information on the policy's performance is known. However, too many rollouts per episode will lead to infeasibly longer episodes.

# Chapter 4

# Training Methodology for CPG

Now that the main parts of the simulation are described, the rest of the supporting material for the simulation can be explained in order to entirely present the simulation protocol. Then, the work flow in relation to the simulation is explained.

## Simulation Methodology

The simulation is executed in a middleware known as the Robotic Operating System (ROS) Melodic, which is a popular, open-source software based for Linux. ROS is the foundation for articulating the MUBot using the programmed joint state, effort-driven CPG controller, and this controller is being learned using PGPE. In order to learn the controller, another middleware is introduced known as Gym, which is also an open-source software with ROS compatibility. Because there is no physical experimentation, the simulation environment, Gazebo, not only visualizes the MUBot's movements, but also computes most of the physics using its native physics engine, Open Dynamics Engine (ODE). The remainder of the physics must be computed using customized pieces of code, known as plugins, to fit the specific MUBot.

Using ODE reduces the amount of work required to compute the MUBot's dynamics. However, ODE cannot determine more complex interactions, namely the hydrodynamics. Therefore, an improved version of the Lighthill Model was selected [18-19]. The Lighthill Model is one of the few hydrodynamic models that approximates the added-mass force experienced on a swimming body that propels it forwards. This model applies to swimming bodies that comply with the Large-Amplitude Elongated Body Theory (LAEBT), which is suitable for the MUBot. LAEBT requires the body to be both long and slender, which corresponds to a large thickness-to-length aspect ratio.

With these prerequisites, the improved Lighthill Model specifies two types of forces: reactive, and resistive. Each type generates a longitudinal force in the global frame, a lateral force in the global frame, and a torque about the segment's joint. The reactive forces are responses from the robot due to the fluid, and the resistive forces are frictional interactions with the robot and the fluid. The reactive forces are

$$F_{x,react} = \omega v_y M_{total} + \omega^2 MS + \frac{v_y^2 \bar{M}(0)}{2} - \frac{(v_y + \omega l)^2 \bar{M}(l)}{2} \tag{4.1}$$

$$F_{y,react} = v_x(v_y + \omega l)\bar{M}(l) - v_x v_y \bar{M}(0) - (a_y + 2 v_x \omega) M_{total} - \dot{\omega} MS \tag{4.2}$$

$$T_{z,react} = v_x l (v_y + \omega l) \bar{M}(l) \tag{4.3}$$

where $\omega$, $v_x$, and $v_y$ are one segment's joint angular velocity, global x-direction velocity, and global y-direction velocity respectively. $M_{total}$ is the total body mass tensor. $\bar{M}(x)$ is the cross-sectional added-mass tensor at a position $x$ along the length of a segment where zero is the cranial end and $l$ is the caudal end. $MS$ is the first inertia moments tensor. Meanwhile, the resistive forces are

$$F_{x,res} = -\frac{\rho_f c_f |v_x| v_x}{2} \int_0^l P(x) dx \tag{4.4}$$

$$F_{y,res} = -\frac{\rho_f c_d}{2} \int_0^l |v_y + \omega x| (v_y + \omega x) h(x) dx \tag{4.5}$$

$$T_{z,res} = -\frac{\rho_f c_d}{2}\int_0^l \left|v_y + \omega x\right|\left(v_y + \omega x\right)h(x)x dx, \tag{4.6}$$

where $c_f$, and $c_d$ are the coefficients for friction and drag, and $\rho_f$ is the fluid medium's density. The functions $h(x)$, and $P(x)$ are cross-sectional depth, and perimeter at a given $x$ position along a segment. Donghao Li was responsible for calculating the terms for the MUBot.

Another aspect of the MUBot that requires further specification is an actuator model that converts a given actuator voltage to a torque. Because the actuator was independently developed, it needed to be characterized independently as well. A physical version of the actuator was built and tested within a range of ±15 Volts. However even with a high-precision force sensor, the actuator's torque output was difficult to measure. An alternative approach was taken that uses the analytical model and known parameters of the constituent actuator parts. Through evaluating the analytical model, a numerical relationship was established between the voltage and output torque. Donghao Li found that

$$I_k = \frac{V_k - \omega_k(-0.01271 + 0.004490\theta_{head}^2)}{90}, \tag{4.7}$$

where $I_k$ is the current to the actuator that corresponds to the $k$-th CPG module, $V_k$ is the voltage, which is known from the CPG output, $\omega_k$ is the angular velocity of the $k$-th actuator, and $\theta_{head}$ is the angular position of the $k$-th actuator. The angular measurements are known in radians from virtual sensors in Gazebo, so no additional work is needed to find this data. Additionally, the power is found using Joule's Law. Knowing the current, the output torque is calculated using

$$\tau_k = I_k\left(-0.01271 + 0.00449\theta_{head}^2\right), \tag{4.8}$$

where $\tau_k$ is the torque of the $k$-th actuator. Finally, Figure 4-1 shows the structure of the MUBot simulation in Gazebo.

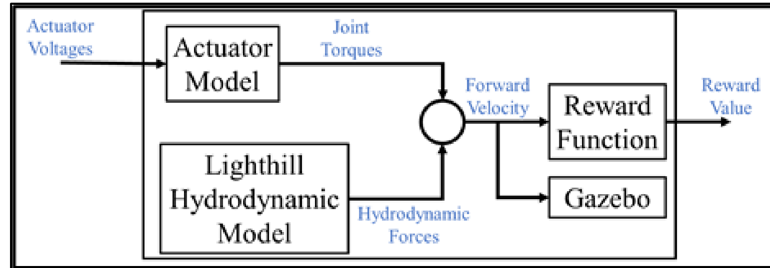


*Fig. 4-1: General structure of the MUBot simulation*

## Learning Methodology

Finally, Gym incorporates the CPG and PGPE's aforementioned Python classes. Gym divides its domain into three essential parts: the training script, the robot environment, and the task environment. This separation allows the user to easily modify one aspect, like the robot or the learning algorithm, without affecting the rest of the simulation. The training script part contains one file that consists of a Python class of the necessary functions for the learning algorithm, and another file that dictates how the algorithm is applied on an episodic and step-by-step basis. The robot environment part contains the files that describe the learning robot. The task environment part contains one file that details how the robot can execute the task that it is learning. Relating this to the MUBot, the CPG and PGPE classes are placed in the task environment and training script respectively. The MUBot is described using the Unified Robot

Description Format, which is compatible with both ROS and Gazebo, and this file is considered to be the robot environment.

After initial experimentation with this software, one major issue was found that invalidated the simulation. Originally, the CPG code was written such that the voltages were calculated and commanded in real time in order to resemble physical operation of the MUBot. However, rollouts were not repeatable such that one sampled policy would result in vastly different behaviors. This inconsistency was due to the CPG code not being computationally efficient enough to calculate voltages fast enough before the next timestep. Also due to software limitations with Gazebo, simply pausing the simulation to wait for the CPG code to complete was not possible. Therefore, the simulation needed to be restructured to allow the CPG code to calculate the entire voltage output before each rollout. An additional modification was made in order to ensure that simulation would only progress when expected. This modification was a service client that granted additional control over Gazebo's functionality, which is normally inaccessible to users. A service client is a code script for ROS that allows for persistent calls to ROS functions, known as services, which encompasses Gazebo. Specifically for this application, the service client grants access to Gazebo's world control that has the capability to mandate a specific number of timesteps for the simulation to stay active. Finally with this added control of Gazebo, the effort is exactly applied over the course of one timestep, and the observed behavior of one policy is exactly repeatable.

Figure 4-2 shows the general work flow within an episode of learning. Beginning with the "Reinforcement Learning" block, the PGPE sampled one value from each parameter's distribution to generate the episode's parameter set. This set was applied to (2.2, 2.3, 2.5), yielding the actuator voltages for each MUBot actuator. Once simulated, the cumulative reward was found according to (3.2), and the parameter distributions were updated using (3.14, 3.15) once enough rollouts were recorded. The initial parameter distributions were defined using a set of known functional values and arbitrarily making the standard deviations to be 25% of the mean value, and the exact numbers are listed in Table 4-1.

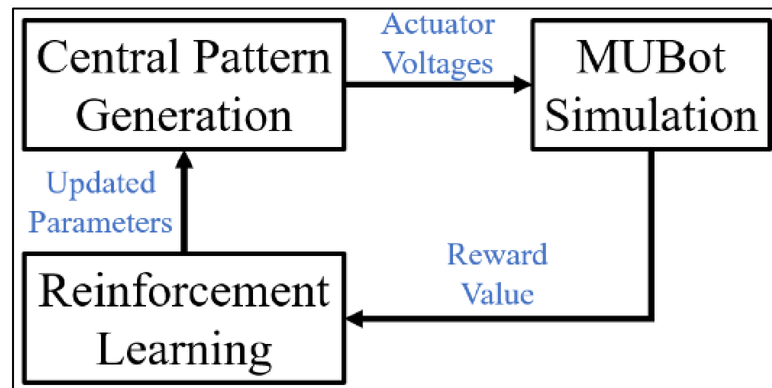


*Fig. 4-2: General structure of a PGPE learning episode*

*Table 4-1: Initial parameters used to begin PGPE learning*

| Parameter | $\mu_{initial}$ | $\sigma_{initial}$ |
|---|---|---|
| $b$ | 2.5 | 0.625 |
| $\tau_1$ | 0.02 | 0.005 |
| $\tau_2$ | 0.06 | 0.015 |
| $a_{ij}$ | 2.7 | 0.675 |

| $w$ | 0.01 | 0.0025 |
|---|---|---|
| $u$ | 12.0 | 3.0 |

## Chapter 5

# Results from Training Simulation

Two different trials of training were conducted to learn the MUBot's optimal CPG parameters. Physical experimentation inspired the first trial in order to observe how realistic the virtual representation of the MUBot is. The second trial uses a complex reward definition that considers additional measurements which are more difficult to measure physically such as the MUBot's power consumption. The second trial's goal was to learn a policy that accounts for lateral drift, power efficiency, and speed.

### Mock Physical Experimentation Trial

The physical experimental design inspired the first virtual design. The parameters that define how the learning was conducted, referred to as hyperparameters, are shown in Table 5-1 below. The reward definition, number of rollouts, and simulation time were selected based on the preceding design. The other hyperparameters were found via hand-tuning until acceptable learning behavior was achieved. The instantaneous reward was defined as

$$r_t = v_{x,head}, \tag{3.2}$$

where $v_{x,head}$ is the forward velocity of the head segment in millimeters-per-second. In accordance with the physical experimental design, the reward was only recorded for the final three seconds of each rollout to capture the steady state behavior of the MUBot alone.

*Table 5-1: Learning hyperparameters for the first trial*

| Hyperparameter | Value |
|---|---|
| Number of rollouts per episode | 15 |
| Rollout simulation time | 11 seconds |
| Learning rate for policy average, $\alpha_\mu$ | 0.1 |
| Learning rate for policy standard deviation, $\alpha_\sigma$ | 0.05 |
| Weight on velocity reward term | 0.01 |

The first trial of learning ran for 67 episodes, which took 10 hours to complete. The average rollout cumulative reward after each episode is shown in Figure 5-1. The learning was successful because the reward improved and became consistent. Also, each parameter converged to suitable values. The average swimming speeds for the MUBot before and after learning were 0.0291 and 0.163 meters-per-second respectively. The swimming speed from the learned result was faster than that learned during physical experimentation. This discrepancy could be due to modeling inaccuracies in the fluid dynamics and the MUBot's physical properties.

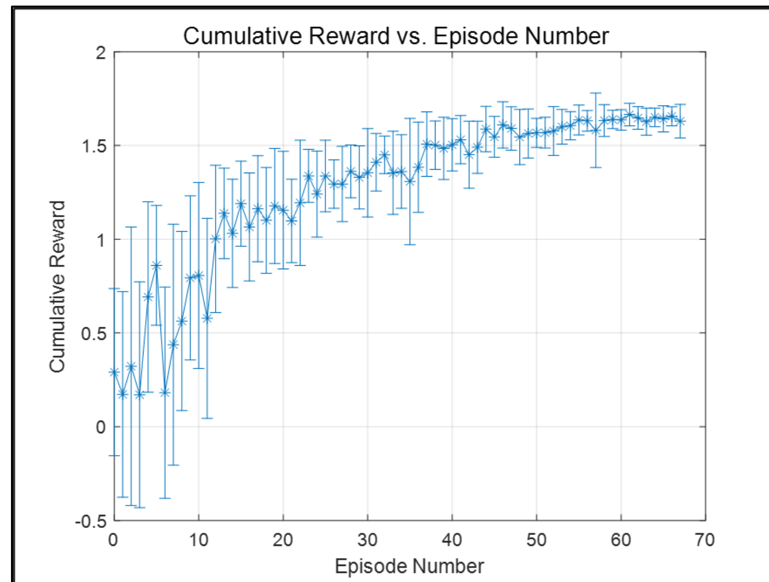


*Fig. 5-1: PGPE learning results for the first trial of simplified learning ($\mu \pm \sigma$)*

The parameter distributions were also tracked, and the histories are shown in Figure 5-2. The histories were divided into three plots based on the relative magnitudes of the parameters for clarity. The hyperparameters were tuned to avoid overconfidence in the learning process.

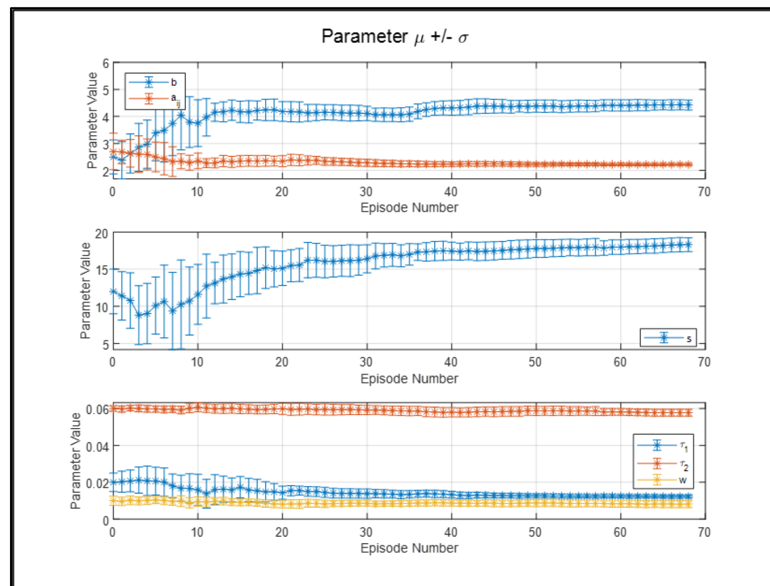


*Fig. 5-2: History of each parameter's distribution for first trial ($\mu \pm \sigma$)*

Figures 5-3 and 5-4 show the CPG output voltages of the best performing rollout from the final episode. Figure 5-4 is a section of Figure 5-3 for clarity. One interesting observation is that the learned CPG output for each joint actuator has practically no phase delay between one another. This behavior was unexpected, and it was not similar to the physical experimentation results.

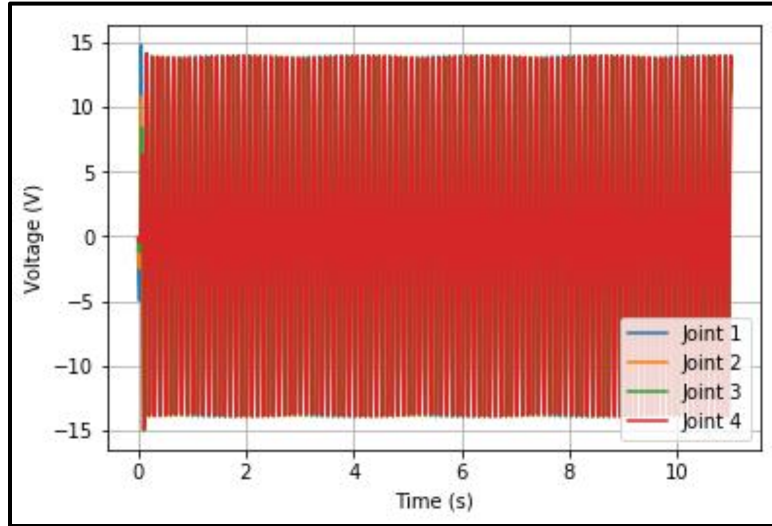


*Fig. 5-3: Learned, full CPG output for the first trial*

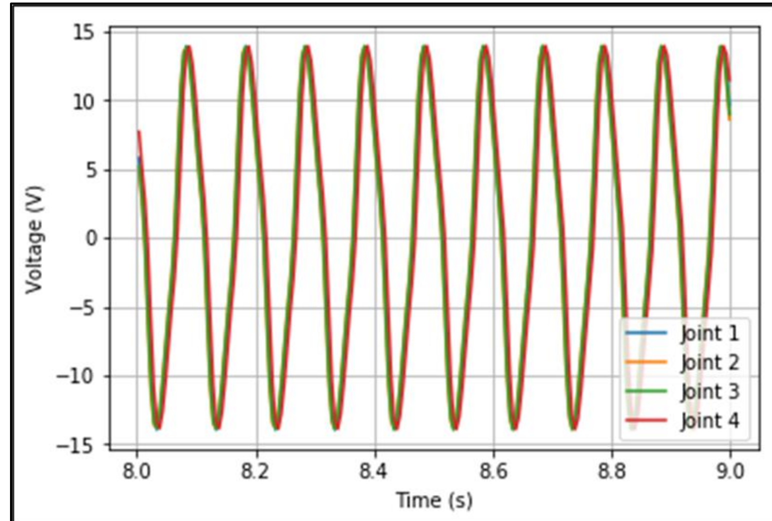


*Fig. 5-4: Section of the learned, steady-state CPG output for the first trial*

**Modified Reward with Penalty Trial**

In the next trial, a new reward expression was defined as

$$r(h) = \frac{\bar{v}_{x,head}}{\bar{P}_{in,total}} - |\bar{v}_{y,head}| + x_{fin,head}, \tag{5.1}$$

where $\bar{v}_{x,head}$ is the average longitudinal velocity, $\bar{P}_{in,total}$ is the average power input, $\bar{v}_{y,head}$ is the average lateral velocity, and $x_{fin,head}$ is the final longitudinal position of the head. These terms were averaged over the entire history of a given rollout, unlike the first trial. The additional penalty term was expected to encourage learning straight-swimming gaits. Without the position term, initial experimentation would occasionally result in failure because the learned result would be motionless in order to minimize power input. Therefore, the position term incentivizes actual

movement as well to ensure successful learning. The hyperparameters for this trial are shown in Table 5-2 below.

*Table 5-2: Learning hyperparameters for the second trial*

| Hyperparameter | Value |
|---|---|
| Number of rollouts per episode | 5 |
| Rollout simulation time | 20 seconds |
| Learning rate for policy average, $\alpha_\mu$ | 0.01 |
| Learning rate for policy standard deviation, $\alpha_\sigma$ | 0.01 |
| Weight on longitudinal velocity reward term | 0.001 |
| Weight on power reward term | 1.0 |
| Weight on lateral velocity penalty term | 3.0 |
| Weight on final longitudinal position reward term | 5.0 |

This trial contained 166 episodes, which took 34 hours to complete. Shown in Figure 5-5, the reward showed similar, successful learning results to that of the previous trial. However, the behavior appeared noisier most likely due to the reduced number of rollouts per trial. Also, the penalty was successful in correctly quantifying the MUBot's deviation from a straight trajectory. The learned CPG output resulted in an average swimming speed of 0.160 meters-per-second, which is 1.0 body-length-per-second, and an average power consumption of 1.29 Watts per actuator. This result is a mild success relative to the reviewed literature. While the result was not faster than others found in the literature, the body length of the simulated MUBot was too short to take full advantage of undulatory swimming. Therefore, additional MUBot segments may improve the body-length-per-second swimming speed.

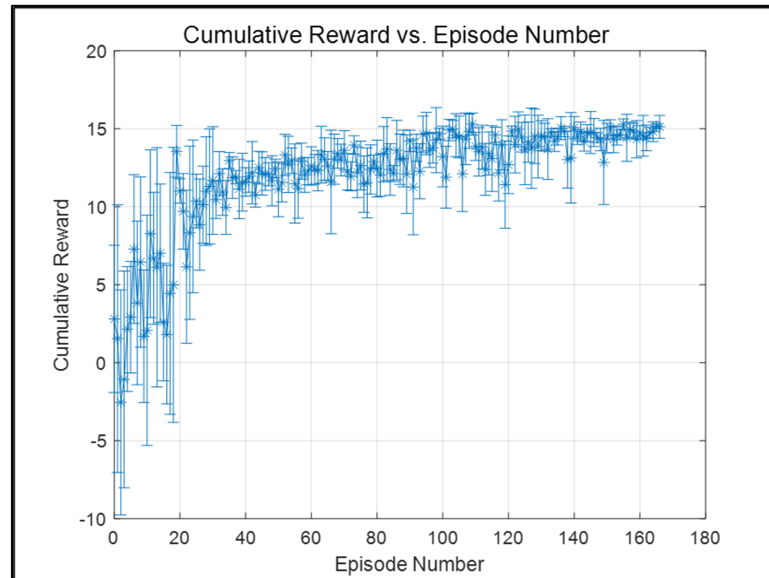


*Fig. 5-5: PGPE learning results using the modified reward with penalty*

From the parameter histories in Figure 5-6, the learned optimal policy is changed from that learned in the previous trial. The changed policy is also reflected in the differing CPG output from the best rollout in the final episode shown in Figure 5-7 and Figure 5-8. Comparing this

output behavior to the behavior learned in the previous trial, this behavior has a higher amplitude and frequency. However, both behaviors show the lack of phase delay.

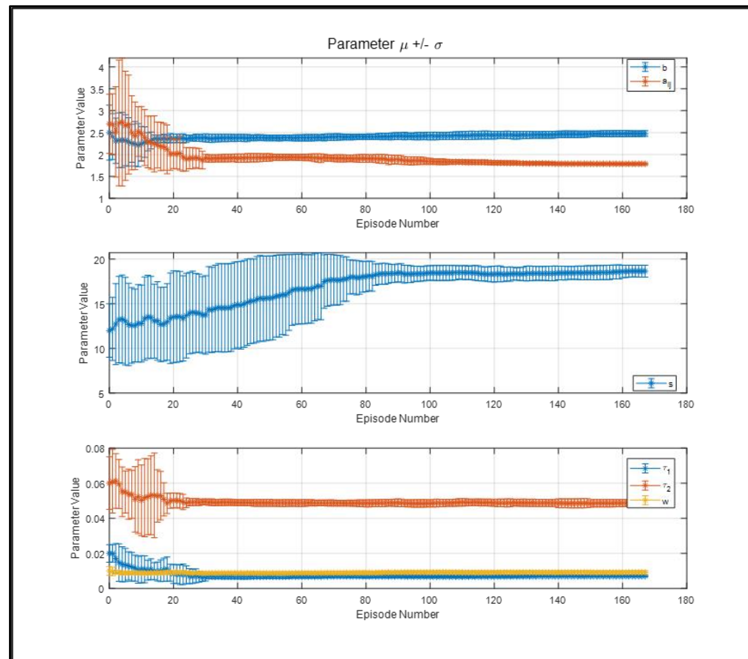


*Fig. 5-6: History of each parameter's distribution for second trial ($\mu \pm \sigma$)*

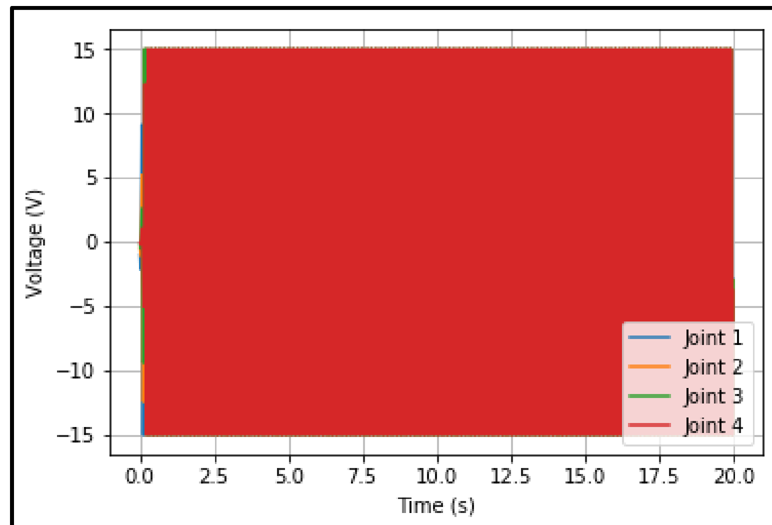


*Fig. 5-7: Learned, full CPG output for the second trial*

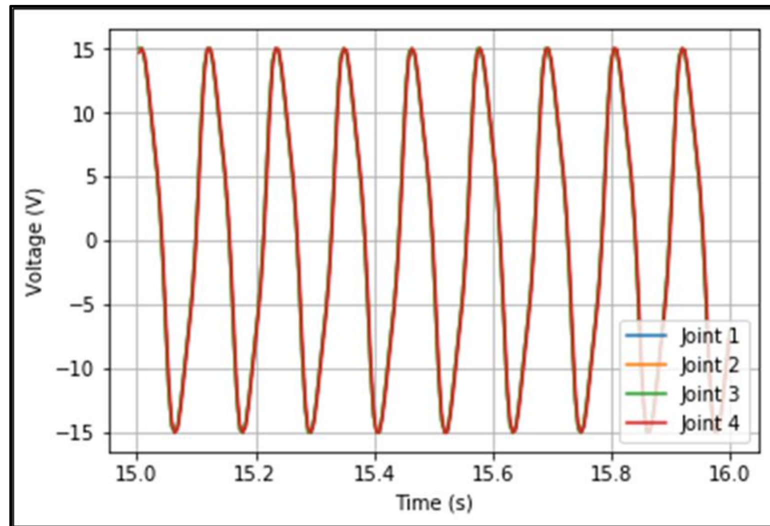


*Fig. 5-8: Section of the learned, steady-state CPG output for the second trial*

**Gait Analyses for the Learned CPG Outputs**

Figures 5-9 and 5-10 depict the resulting joint angles in the MUBot from the learned CPG output from the first trial. Joint 1 corresponds to the joint closest to the head segment, and Joint 4 corresponds to the joint closest to the tail segment. For the first trial's gait, Joint 1 and Joint 4 are approximately identical while Joint 2 and Joint 3 appear to act erratically. From the full gait, the angle limit for each joint of roughly ± 0.17 radians appears to be reached consistently in Joint 1 and Joint 4. The sequence of joints reaching their peak values is mostly Joint 1, Joint 4, Joint 2, and Joint 3. Even though the MUBot performance improves, the evidence of undulatory motion is lacking due the unclear periodic behavior of Joint 2, and Joint 3, along with Joint 4 behaving out of sequence. This behavior is unexpected because an undulatory motion was expected to be ideal.

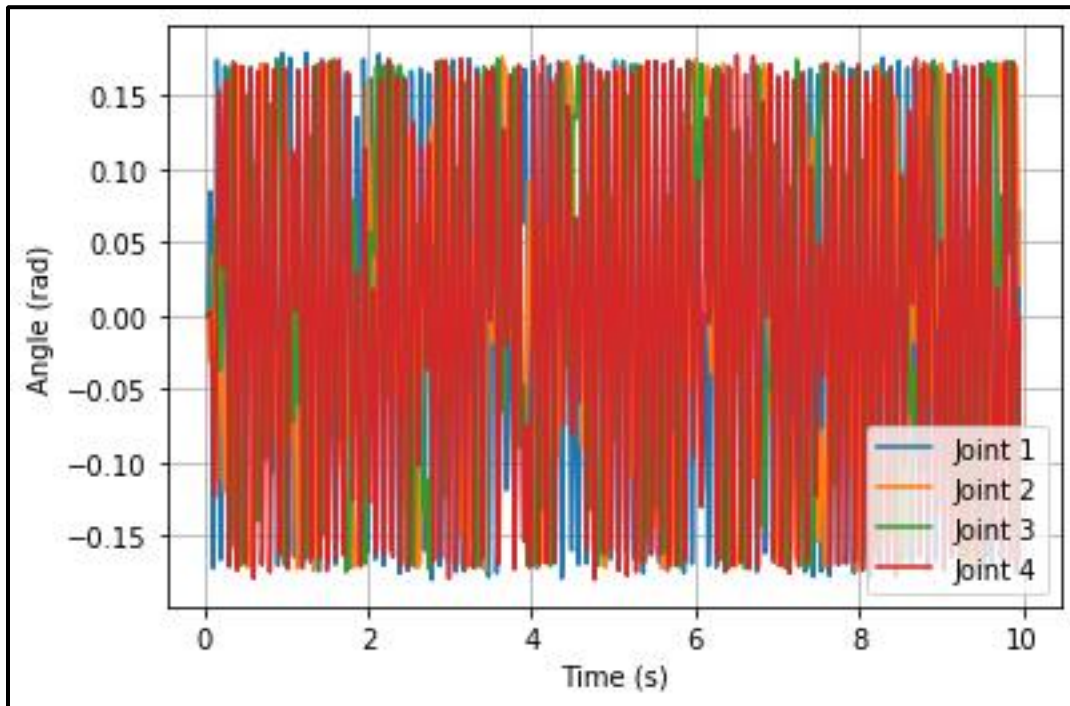


*Fig. 5-9: Full, learned gait for the first trial*

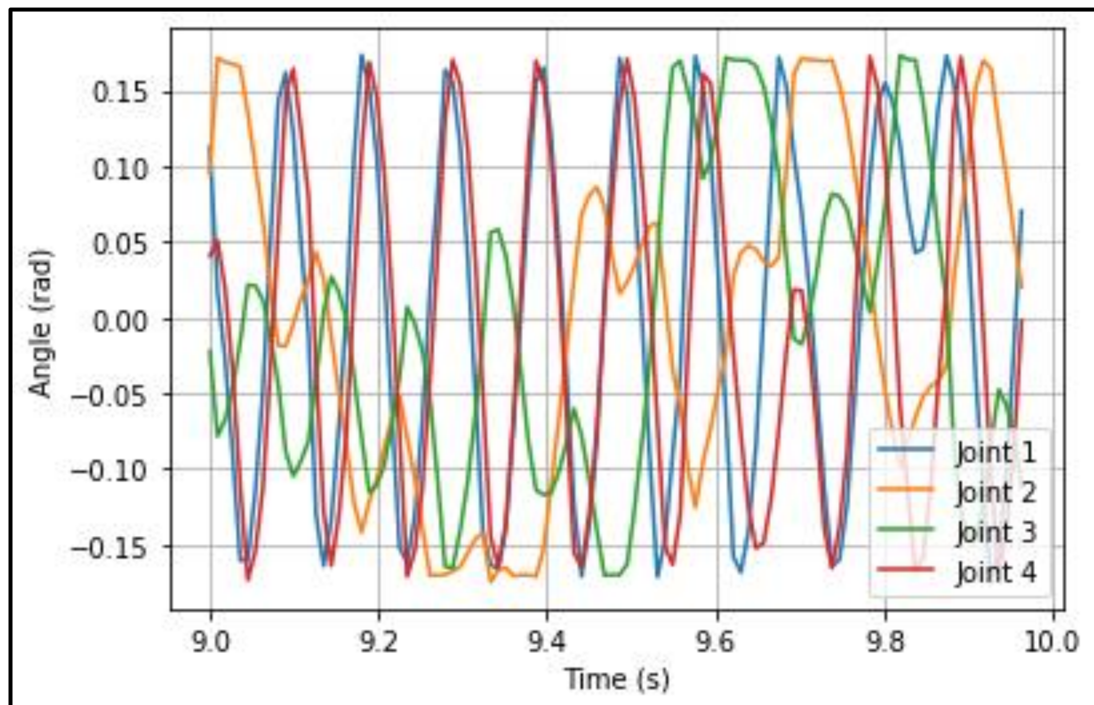


*Fig. 5-10: Section of learned gait for the first trial*

The learned gait for the second trial is similar to that of the first trial. Figure 5-11 is the full gait for the second trial, which broadly appears more consistent compared to Figure 5-9. Upon further inspection, the behavior for Joint 2 and Joint 3 appear less erratic, as shown in

Figure 5-12. Therefore, this gait shows undulatory behavior more clearly, even though Joint 4 still behaves out of sequence. The difference in CPG output frequency is believed to be the key factor to this improvement. Again with this learned gait, Joint 1 and Joint 4 joint angles are nearly identical. Accounting for the Lighthill Model equations, the MUBot appears to be maximizing the angular velocities for the segments in order to increase the hydrodynamic thrust forces, which in turn increases the reward. Figures 5-13 and 5-14 are snapshots of the trajectories starting from the beginning. Further lack of undulatory motion can be observed due to an absence of a traveling wave moving along the MUBot's body length.

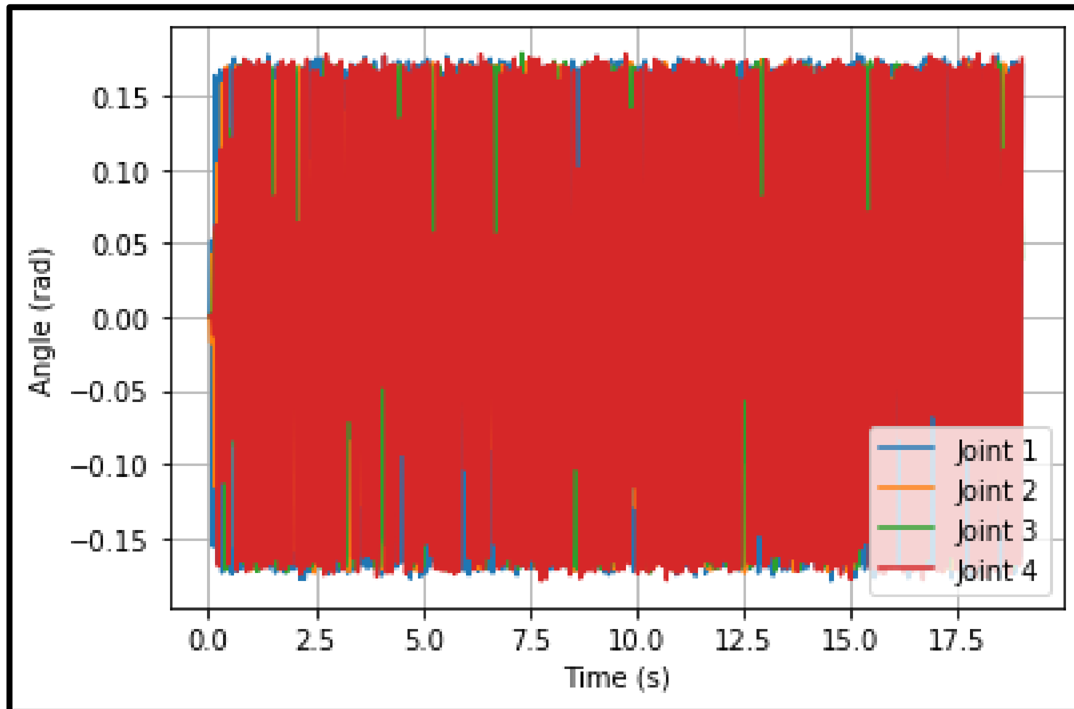


*Fig. 5-11: Full, learned gait for the second trial*

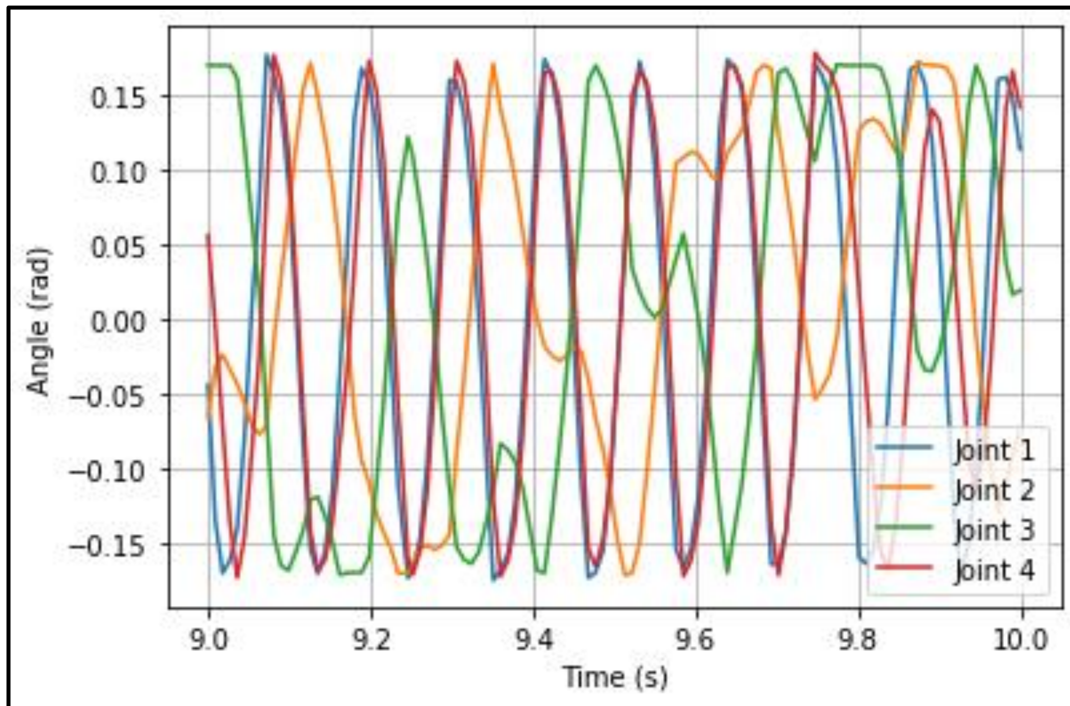


*Fig. 5-12: Section of learned gait for the second trial*

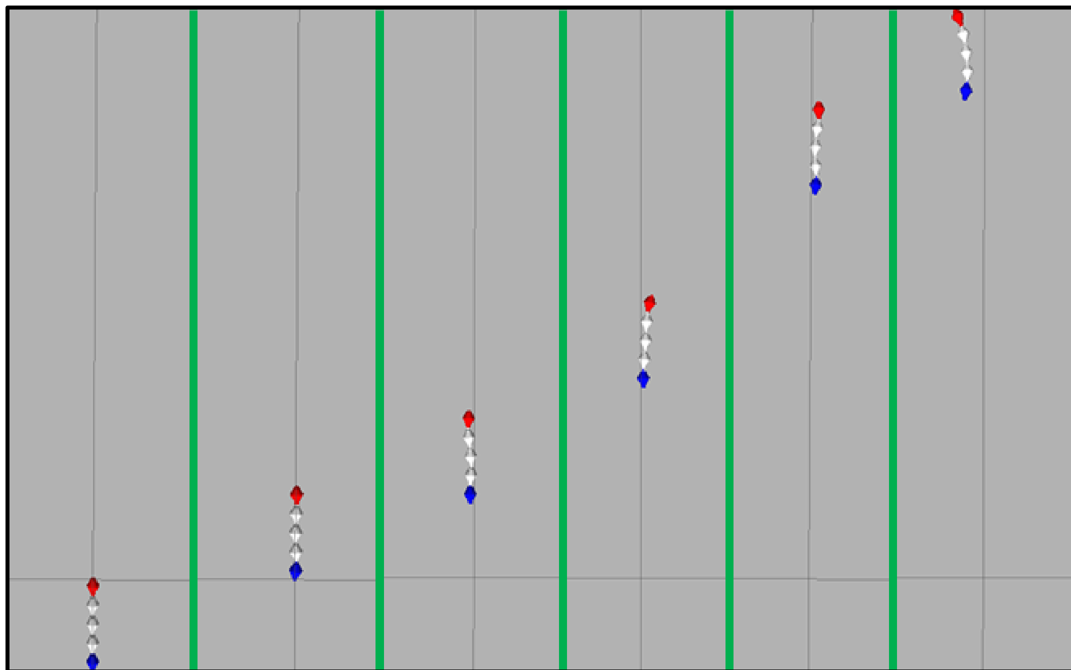

*Fig. 5-13: Snapshots of first trial's learned trajectory at two second intervals*

*Fig. 5-14: Snapshots of second trial's learned trajectory at three second intervals*

## Chapter 6

# Conclusion

A new reinforcement learning strategy, PGPE, was applied to optimize the Matsuoka CPG effort controller for the actuation of an eel-like, undulatory robot. The improved Lighthill Model simulated the hydrodynamic interactions between the MUBot and its fluid environment. The controller was optimized for a fast swimming trajectory, and PGPE showed promising results, according to the literature. Simulating the learning using a combination of Gazebo and Gym, the results were successful. Comparing the performance of the learned CPG controller to that from the literature, the best learned result was a forward velocity of 1.0 body-length-per-second, and the most similar controller found in the literature yielded 1.15 body-lengths-per-second. There were two different trials of simulations.

The first trial was the easiest to implement and utilized the average of instantaneous reward values to find the cumulative reward for rollouts in an episode. The first trial also used a truncated Gaussian distribution strategy to sample viable parameter values with hand-tuned hyperparameters. In the next trial, the reward expression was modified to include a penalty for drifting in unwanted directions, and for power consumption of the MUBot's actuators in the learning process. The resulting gaits for these trials showed little evidence of undulatory motion. However, the second trial showed clearer signs of undulatory motion, which was attributed to the differing frequency from the CPG output. From the learned gaits, the MUBot appears to exploit the angular velocity term in the Lighthill Model's hydrodynamic thrust in order to achieve faster swimming.

Improvements can still be made to the learning. From the source literature for PGPE, alternative sampling methods may be applied to further improve the learning process. Further automation can be applied in order to avoid hand-tuning the learning hyperparameters using the EM method, for example. This automation would assist in avoiding unsuccessful, or poor learning, which were common issues when hand-tuning these parameters.

To improve the learning outcome, the MUBot's intended design should be modeled, which modifies the head and tail segments to conform to an eel's body morphology. Making the MUBot longer in length would allow it to resemble an eel more accurately, and this resemblance may allow the MUBot to further exploit an undulatory swimming gait. Instead of having one set of CPG parameters describe all sets of equations in the network, the learning space can be expanded such that multiple sets of parameters are unique to specific modules or sets of equations. While this modification would greatly impact the learning process, each actuator in the MUBot could be uniquely tuned to have different behaviors, which is more versatile, and realistic among biological systems. Also, this performance improvement has been demonstrated through Hankun Deng's work. Currently, Hankun Deng is investigating the effects of body morphology variations on the learned CPG output, and the swimming performance. The effects of body mass, tail segment shape, and head segment shape have been observed. Using physical experimentation, Hankun Deng has been able to learn undulatory swimming gaits for the MUBot. Donghao Li seeks to extend these results to the simulated MUBot by allowing for longer learning trials, since physical experimentation is slower and more laborious.

## References

[1] Ijspeert, A. J., 2014, “Biorobotics: Using Robots to Emulate and Investigate Agile Locomotion,” Science (80-. )., **346**(6206), pp. 196–203.
[2] Gemmell, B. J., Colin, S. P., Costello, J. H., and Dabiri, J. O., 2015, “Suction-Based Propulsion as a Basis for Efficient Animal Swimming,” Nat. Commun., **6**, pp. 1–8.
[3] Crespi, A., Lachat, D., Pasquier, A., and Ijspeert, A. J., 2008, “Controlling Swimming and Crawling in a Fish Robot Using a Central Pattern Generator,”
[4] Wen, L., Wang, T., Wu, G., and Liang, J., 2013, “Quantitative Thrust Efficiency of a Self-Propulsive Robotic Fish: Experimental Method and Hydrodynamic Investigation,” IEEE/ASME Trans. Mechatronics, **18**(3), pp. 1027–1038.
[5] Conradt, J., and Varshavskaya, P., 2003, “Distributed Central Pattern Generator Control for a Serpentine Robot,” Int. Conf. …, **341**, pp. 2–5.
[6] Ijspeert, A. J., 2008, “Central Pattern Generators for Locomotion Control in Animals and Robots : A Review,” Neural Networks, **21**, pp. 642–653.
[7] Niu, X., Xu, J., Ren, Q., and Wang, Q., 2013, “Locomotion Generation and Motion Library Design for an Anguilliform Robotic Fish,” J. Bionic Eng., **10**(3), pp. 251–264.
[8] Wang, C., Xie, G., Wang, L., and Cao, M., 2011, “CPG-Based Locomotion Control of a Robotic Fish: Using Linear Oscillators and Reducing Control Parameters via PSO,” Int. J. Innov. Comput. Inf. Control, **7**(7 B), pp. 4237–4249.
[9] Yu, J., Wu, Z., Wang, M., and Tan, M., 2016, “CPG Network Optimization for a Biomimetic Robotic Fish via PSO,” IEEE Trans. Neural Networks Learn. Syst., **27**(9), pp. 1962–1968.
[10] Gay, S., Santos-Victor, J., and Ijspeert, A., 2013, “Learning Robot Gait Stability Using Neural Networks as Sensory Feedback Function for Central Pattern Generators,” IEEE Int. Conf. Intell. Robot. Syst., pp. 194–201.
[11] Nakamura, Y., Mori, T., Sato, M. aki, and Ishii, S., 2007, “Reinforcement Learning for a Biped Robot Based on a CPG-Actor-Critic Method,” Neural Networks, **20**(6), pp. 723–735.
[12] Matsubara, T., Morimoto, J., Nakanishi, J., Sato, M., and Doya, K., 2005, “Learning CPG-Based Biped Locomotion with a Policy Gradient Method,” pp. 208–213.
[13] Sehnke, F., Graves, A., Osendorfer, C., Schmidhuber, J., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J., 2010, “Parameter-Exploring Policy Gradients,” Neural Networks, **23**(4), pp. 551–559.
[14] Ishige, M., Umedachi, T., Taniguchi, T., and Kawahara, Y., 2019, “Exploring Behaviors of Caterpillar-Like Soft Robots with a Central Pattern Generator-Based Controller and Reinforcement Learning,” Soft Robot., **6**(5), pp. 579–594.
[15] Matsuoka, K., 1985, “Sustained Oscillations Generated by Mutually Inhibiting Neurons with Adaptation,” Biol. Cybern., **52**(6), pp. 367–376.
[16] Matsuoka, K., 1987, “Mechanisms of Frequency and Pattern Control in the Neural Rhythm Generators,” Biol. Cybern., **56**(5–6), pp. 345–353.
[17] Wu, X., and Ma, S., 2009, “CPG-Based Control of Serpentine Locomotion of a Snake-like Robot,” IFAC.
[18] Lighthill, M. J., 1960, “Note on the Swimming of Slender Fish,” J. Fluid Mech., **9**(2), pp. 305–317.
[19] Porez, M., Boyer, F., and Ijspeert, A. J., 2014, “Improved Lighthill Fish Swimming Model for Bio-Inspired Robots: Modeling, Computational Aspects and Experimental Comparisons,” Int. J. Rob. Res., **33**(10), pp. 1322–1341.

[20] Lu, Z., Ma, S., Li, B., and Wang, Y., 2004, “Serpentine Locomotion of a Snake-like Robot Controlled by Cyclic Inhibitory CPG Model *.”
[21] Zhao, T., Hachiya, H., Niu, G., and Sugiyama, M., 2012, “Analysis and Improvement of Policy Gradient Estimation,” Neural Networks, **26**, pp. 118–129.