Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра нелинейных динамических систем и процессов управления

Ван Юйфэн

Применение методов машинного обучения для управления многозвенными роботами

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
д.ф-м.н., Заведующий кафедрой НДСиПУ,
В.В. Фомичёв

Москва, 2024

Содержание

1 introduction

Inspired by real snakes, motion control of robotic snakes has received considerable attention in recent years. With long and narrow bodies and multiple degrees of freedom, snake robots can mimic the movements of biological snakes and realise flexible geometric changes[1]. As a result, they are able to perform various tasks not available to other types of robots and adapt to unknown and complex environmental conditions. Snake robots are used in a wide variety of applications, such as disaster relief, rescue, reconnaissance, medical surgery, and complex space operations[2].

Snakes experience resistance on the ground from objects like rocks and effectively use their abdominal scales to propel themselves forward through friction. Much of the current research has focused on wheeled robotic snakes, which are relatively easy to control, but cannot reproduce the movement patterns of biological snakes and adapt to complex and variable environments[3]. In this work, we focus on a wheel-less snake robot whose movement more closely resembles that of real snakes. The robot moves due to the friction between its connecting rods and the ground. There are two types of friction between the snake and the ground: Coulomb friction, which depends on mass, and viscous friction, which depends on velocity[4].

Рис. 1: Робот-змея

The snake robot consists of many connected rigid body links, its system has a large number of degrees of freedom, while the controller has a small dimensionality, which makes the system difficult to control. It is first necessary to define a mathematical model of the robot dynamics that accurately describes the relationship between the robot state, friction and controller outputs. The state of the robot should include the lengths of individual links, their masses, positions, velocities, angles between links, etc. In Chapter 2, we will present a mathematical model of the robot dynamics and detail the process of building this model, ending with the derivation of the dynamic equations of the system.

In this paper, we will consider two problems: the motion of the snake robot along a straight line and the motion of the snake robot towards a target point. For each problem, we will try to find a solution on both simulations.

We will investigate the feasibility of moving the snake robot by building a mathematical model on the Simulink simulator, and non-model control methods by building a model of the robot and an unknown environment more similar to reality on the MuJoCo simulator.

Control of bionic robots is often achieved by modelling animal gaits. One of the key movement mechanisms used by biological snakes to move forwards is lateral undulation. During this movement, the snake creates periodic undulating curves across its body from head to tail. To mimic this motion in a robotic snake, the angular changes of each of its segments are tuned according to a sinusoidal function. Demonstrating the feasibility of this technique involves investigating different control strategies and performing tasks such as moving forwards, changing speed and following a given trajectory. The results of these studies, presented in Section 5.1 and 5.2, confirm the feasibility of control using a simulator.

However, in real-world environments where characteristics such as friction patterns and friction coefficients may be uncertain and variable, the presented methods may not be sufficient. Therefore, the study considers the use of alternative control approaches, including the method of alternating between fast and slow control. For each mode, a genetic algorithm is used to optimise the control cycles, allowing the robot to achieve fast straight-line motion. The details of this method are outlined in Section 5.3.

To improve the adaptability and performance of the snake robot in unpredictable environments, a reinforcement learning algorithm, PPO (Proximal Policy Optimisation), is applied. This approach, validated in continuous control tasks, focuses on optimising the torque of the output oscillators controlling the motion of each joint. This paper focuses on analysing the capabilities of the PPO algorithm in the context of bionic robot control, validating its potential to improve control efficiency and robustness in dynamic environments.

The control system of the snake robot consists of a system of oscillators, each of which is responsible for generating a periodic torque to control the robot's motion. We have developed a reinforcement learning mechanism to create a neural network with a policy, whose inputs are the states of the robot over a period of time and whose outputs are certain parameters of the oscillators, which provide a series of signals that allow the robot to autonomously determine the amplitude of the oscillation at the output of each controller based on the current and previous states. We will use a PPO [5] algorithm to optimise the parameters of these oscillators so that the robot can effectively mimic the lateral undulating motion of a biological snake, and accurately change direction and move towards a target. In the experiments, the outputs of the oscillators are not simply fixed or set, but can be dynamically adjusted depending on the robot's real-time interaction with the environment. By solving the target point tracking problem, we can initially implement the motion trajectory tracking problem.

Section 5.4 of this paper details the results of applying this approach to the target tracking task, demonstrating the snake robot's ability to navigate the environment and adapt its behaviour when its own state changes. The experimental results will be used to analyse the specific impact of the PPO algorithm and oscillator parameter optimisation on the robot's performance.

2 Description of the robot snake model

2.1 Position and shape of the robot

The snake robot in the plane consists of N connecting links of length $2l$, so there are $N-1$ connection points. Each link has mass m, the mass distribution is uniform, so that the centroid of the link (CM) is at the centre point, assuming the snake robot moves on the plane, there are $N+2$ degrees of freedom in total, including N degrees of freedom describing the angle of the link and 2 degrees of freedom describing the position of the robot [4].

The angle of each link with respect to the x-axis:

$$\boldsymbol{\theta} = [\theta_1, \theta_2 ... \theta_N]^T \in \mathbb{R}^2 \tag{2.1.1}$$

The centre of mass of the robot (if there are coordinates of the centre of mass of each connecting link (x_i, y_i)):

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N} x_i \\ \frac{1}{N}\sum_{i=1}^{N} y_i \end{bmatrix} \in \mathbb{R}^2 \tag{2.1.2}$$

The x-axis coordinate vector and the y-axis coordinate vector:

$$\begin{aligned} \mathbf{X} &= [x_1, x_2, \ldots, x_N]^\top \in \mathbb{R}^N, \\ \mathbf{Y} &= [y_1, y_2, \ldots, y_N]^\top \in \mathbb{R}^N. \end{aligned} \tag{2.1.3}$$

The absolute angle (angle with respect to the x-axis) of the link is θ_i.

The angle between the i-th and $\{i+1\}$-th links and $\phi_i = \theta_{i+1} - \theta_i$. Thus, the relationship between the coordinates of the neighbouring links:

$$\begin{cases} x_{i+1} - x_i = l\cos\theta_i + l\cos\theta_{i+1}, \\ y_{i+1} - y_i = l\sin\theta_i + l\sin\theta_{i+1} \end{cases} \tag{2.1.4}$$

By matrix transformation:

$$\begin{cases} \mathbf{DX} + l\mathbf{A}\cos\boldsymbol{\theta} = 0, \\ \mathbf{DY} + l\mathbf{A}\sin\boldsymbol{\theta} = 0 \end{cases} \tag{2.1.5}$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 1 & \ldots & 0 \\ & & \cdots & & \\ 0 & 0 & \ldots & 1 & 1 \end{bmatrix} \in \mathbb{R}^{(N-1)\times N} \qquad \mathbf{D} = \begin{bmatrix} 1 & -1 & \ldots & 0 & 0 \\ 0 & 1 & -1 & \ldots & 0 \\ & & \cdots & & \\ 0 & 0 & \ldots & 1 & -1 \end{bmatrix} \in \mathbb{R}^{(N-1)\times N}$$

Therefore, the coordinate vectors of each connecting link:

$$\begin{cases} \mathbf{X} = \mathbf{T}^{-1}\begin{bmatrix} -l\mathbf{A}\cos\boldsymbol{\theta} \\ p_x \end{bmatrix} = -l\mathbf{K}^T\cos\boldsymbol{\theta} + \mathbf{e}p_x \\ \mathbf{Y} = \mathbf{T}^{-1}\begin{bmatrix} -l\mathbf{A}\sin\boldsymbol{\theta} \\ p_y \end{bmatrix} = -l\mathbf{K}^T\sin\boldsymbol{\theta} + \mathbf{e}p_y \end{cases} \tag{2.1.6}$$

where:

$$\mathbf{e} = \begin{bmatrix} 1 & 1 & \ldots & 1 \end{bmatrix}^T \in \mathbb{R}^n$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{D} \\ \frac{1}{N}\mathbf{e}^T \end{bmatrix} \in \mathbb{R}^{N\times N},$$

$$\mathbf{T}^{-1} = \left[\mathbf{D}^T\left(\mathbf{DD}^T\right)^{-1}\mathbf{e}\right] \in \mathbb{R}^{N\times N}$$

$$\sin\boldsymbol{\theta} = \begin{bmatrix} \sin\theta_1 & \sin\theta_2 & \dots & \sin\theta_n \end{bmatrix}^T \in \mathbb{R}^n$$

$$\cos\boldsymbol{\theta} = \begin{bmatrix} \cos\theta_1 & \cos\theta_2 & \dots & \cos\theta_n \end{bmatrix}^T \in \mathbb{R}^n$$

$$\mathbf{K} = \mathbf{A}^T \left(\mathbf{D}\mathbf{D}^T\right)^{-1} \mathbf{D} \in \mathbb{R}^{n\times n}$$

Assume that the snake robot is arranged in reverse order, i.e., the head is the Nth link.

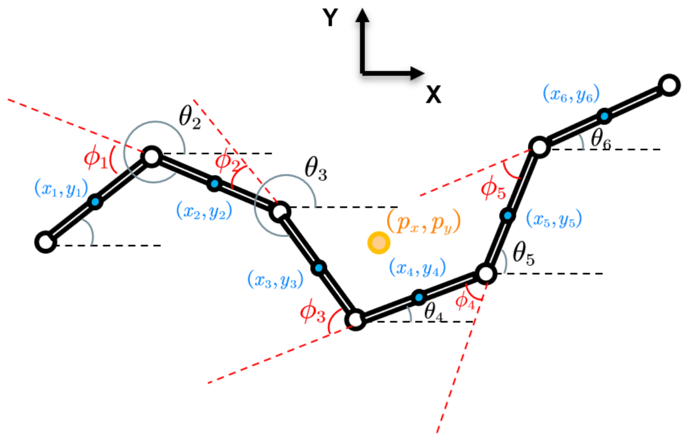


Рис. 2: Форма робота-змеи

2.2 Friction model

A snake robot moving on a flat surface constantly changes the shape of its body to induce ground friction forces that propel the robot forward. When travelling on a plane, it is very important for the motion of the snake robot that the ground friction forces on the links are anisotropic, which means that the friction coefficient describing the friction force in the tangential direction of the link c_t, μ_t(parallel to the link, i.e., along the x-axis of the link) is different. (i.e. along the x-axis of the link), is different from the friction coefficient describing the friction force in the direction normal to the link c_n, μ_n (perpendicular to the link, i.e. along the y-axis of the link). This property of friction is evident in biological snakes. We decided to use an anisotropic viscous friction model. We assume that the forces of viscous friction on the ground act only on the CM of the links. The friction acting on the i-th link in the frame is related to the direction of motion:

2.2.1 Anisotropic viscous friction model

First, we consider an anisotropic viscous friction model. We assume that the forces of viscous friction on the ground act only on the CMs of the links. The friction acting on the i-th link in the frame is related to the direction of motion:

$$\mathbf{f}_{R,i}^{i} = -\begin{bmatrix} c_t & 0 \\ 0 & c_n \end{bmatrix} \mathbf{v}_i \tag{2.2.1}$$

where $\mathbf{v}_i$ is the velocity of the i-th link in the direction of motion. Friction acting on the i-th link in the x-y frame:

$$\mathbf{f}_{R,i} = \mathbf{R}_i \mathbf{f}_{R,i}^{i} = -\mathbf{R}_i \begin{bmatrix} c_t & 0 \\ 0 & c_n \end{bmatrix} \mathbf{v}_i = -\mathbf{R}_i \begin{bmatrix} c_t & 0 \\ 0 & c_n \end{bmatrix} \mathbf{R}_i^T \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} \tag{2.2.2}$$

where:

$$\mathbf{v}_i = \mathbf{R}_i \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix}, \mathbf{R}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}$$

After transforming the matrix:

$$\mathbf{f}_{R,i} = \begin{bmatrix} \mathbf{f}_{R,x,i} \\ \mathbf{f}_{R,y,i} \end{bmatrix} = -\begin{bmatrix} c_t \cos^2\theta_i + c_n \sin^2\theta_i & (c_t - c_n)\sin\theta_i\cos\theta_i \\ (c_t - c_n)\sin\theta_i\cos\theta_i & c_t \sin^2\theta_i + c_n \cos^2\theta_i \end{bmatrix} \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} \tag{2.2.3}$$

Global frame viscous friction forces on all links:

$$\mathbf{f}_R = \begin{bmatrix} \mathbf{f}_{R,x} \\ \mathbf{f}_{R,y} \end{bmatrix} = -\begin{bmatrix} c_t (\mathbf{C}_\theta)^2 + c_n (\mathbf{S}_\theta)^2 & (c_t - c_n)\mathbf{S}_\theta\mathbf{C}_\theta \\ (c_t - c_n)\mathbf{S}_\theta\mathbf{C}_\theta & c_n (\mathbf{C}_\theta)^2 + c_t (\mathbf{S}_\theta)^2 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{X}} \\ \dot{\mathbf{Y}} \end{bmatrix} \in \mathbb{R}^{2N} \tag{2.2.4}$$

where:

$$\mathbf{S}_\theta = \mathrm{diag}(\sin\boldsymbol{\theta}) \in \mathbb{R}^{n\times n}$$

$$\mathbf{C}_\theta = \mathrm{diag}(\cos\boldsymbol{\theta}) \in \mathbb{R}^{n\times n}$$

2.2.2 Anisotropic Coulomb friction model

Next, we consider the model of anisotropic Coulomb friction.

As in the viscous friction model, we define the Coulomb friction force acting on the i-th link as.

$$\mathbf{f}^i_{R,i} = -mg \begin{bmatrix} \mu_t & 0 \\ 0 & \mu_n \end{bmatrix} sgn(\mathbf{v}_i) \tag{2.2.5}$$

where $\mathbf{v}_i$ is the velocity of the i-th link in the direction of motion.Friction acting on the i-th link in the x-y frame:

$$\mathbf{f}_{R,i} = \mathbf{R}_i \mathbf{f}^i_{R,i} = -mg\mathbf{R}_i \begin{bmatrix} \mu_t & 0 \\ 0 & \mu_n \end{bmatrix} sgn(\mathbf{v}_i) = -\mathbf{R}_i \begin{bmatrix} \mu_t & 0 \\ 0 & \mu_n \end{bmatrix} \mathbf{R}_i^T sgn\left(\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix}\right) \tag{2.2.6}$$

Global frame viscous friction forces on all links:

$$\mathbf{f}_R = \begin{bmatrix} \mathbf{f}_{R,x} \\ \mathbf{f}_{R,y} \end{bmatrix} = -mg \begin{bmatrix} \mu_t \mathbf{C}_\theta & -\mu_n \mathbf{S}_\theta \\ \mu_t \mathbf{S}_\theta & \mu_n \mathbf{C}_\theta \end{bmatrix} sgn\left(\begin{bmatrix} \mathbf{C}_\theta & \mathbf{S}_\theta \\ -\mathbf{S}_\theta & \mathbf{C}_\theta \end{bmatrix} \begin{bmatrix} \dot{\mathbf{X}} \\ \dot{\mathbf{Y}} \end{bmatrix}\right) \in \mathbb{R}^{2N} \tag{2.2.7}$$

2.3 Robot dynamics

Now we give the equation of motion of the robot in terms of the angular acceleration $\ddot{\boldsymbol{\theta}}$ of the rotation angle of the link and the acceleration $\ddot{\boldsymbol{p}}$ of the position of the centre of mass. According to Newton's second law, the forces on each link in the x and y directions are as follows:

$$m\ddot{x}_i = f_{R,x,i} + h_{x,i} - h_{x,i-1} \quad , \quad m\ddot{y}_i = f_{R,y,i} + h_{y,i} - h_{y,i-1} \tag{2.3.1}$$

where $h_{z,i}, h_{z,i-1}$ represent the joint constraints of $\{i-1\}$-th link and $\{i+1\}$-th link, respectively, in the z direction.

$$m\ddot{\mathbf{X}} = \mathbf{f}_{R,x} + \mathbf{D}^T \mathbf{h}_x \quad , \quad m\ddot{\mathbf{Y}} = \mathbf{f}_{R,y} + \mathbf{D}^T \mathbf{h}_y \tag{2.3.2}$$

where $\mathbf{h}_x = [h_{x,1}, ..., h_{x,N}]^T \in \mathbb{R}^N$,$\mathbf{h}_y = [h_{y,1}, ..., h_{y,N}]^T \in \mathbb{R}^N$

Take the second derivative of the above formula :

$$\mathbf{D}\ddot{\mathbf{X}} = l\mathbf{A}(\mathbf{C}_\theta \dot{\boldsymbol{\theta}}^2 + \mathbf{S}_\theta \ddot{\boldsymbol{\theta}}) \quad , \quad \mathbf{D}\ddot{\mathbf{Y}} = l\mathbf{A}(\mathbf{S}_\theta \dot{\boldsymbol{\theta}}^2 - \mathbf{C}_\theta \ddot{\boldsymbol{\theta}}) \tag{2.3.3}$$

Since $\mathbf{e}^T\mathbf{D}^T = 0$, we can conclude by finding the second derivative of $\mathbf{p}$ – the acceleration of the robot as a whole in the x- and y-axis directions is related only to the friction vector $\mathbf{f}_R$:

$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \end{bmatrix} = \frac{1}{N}\begin{bmatrix} \mathbf{e}^T\ddot{\mathbf{X}} \\ \mathbf{e}^T\ddot{\mathbf{Y}} \end{bmatrix} = \frac{1}{Nm}\begin{bmatrix} \mathbf{e}^T\mathbf{f}_{R,x} \\ \mathbf{e}^T\mathbf{f}_{R,y} \end{bmatrix} \tag{2.3.4}$$

This equation simply states that the acceleration of the snake robot's CM is equal to the sum of the external forces acting on the robot divided by its mass. The torque balance for link i is defined as follows:

$$J\ddot{\theta}_i = u_i - u_{i-1} - l\sin\theta_i\left(h_{x,i} + h_{x,i-1}\right) + l\cos\theta_i\left(h_{y,i} + h_{y,i-1}\right) \tag{2.3.5}$$

where J is the moment of inertia.

Through the above equations (2.3.2) and (2.3.3), we can obtain the joint constraint force equation $\mathbf{h}_x$, $\mathbf{h}_y$:

$$\begin{aligned} \mathbf{h}_x &= \left(\mathbf{D}\mathbf{D}^{\mathbf{T}}\right)^{-1}\left(ml\mathbf{A}(\mathbf{C}_\theta\dot{\boldsymbol{\theta}}^2 + \mathbf{S}_\theta\ddot{\boldsymbol{\theta}}) - \mathbf{D}\mathbf{f}_{R,x}\right) \\ \mathbf{h}_y &= \left(\mathbf{D}\mathbf{D}^{\mathbf{T}}\right)^{-1}\left(ml\mathbf{A}(\mathbf{S}_\theta\dot{\boldsymbol{\theta}}^2 - \mathbf{C}_\theta\ddot{\boldsymbol{\theta}}) - \mathbf{D}\mathbf{f}_{R,y}\right) \end{aligned} \tag{2.3.6}$$

Combining the above equations, we can obtain the equations of motion:

$$\begin{cases} \mathbf{M}_\theta\ddot{\boldsymbol{\theta}} + \mathbf{W}\dot{\boldsymbol{\theta}}^2 - l\mathbf{S}_\theta\mathbf{K}\mathbf{f}_{R,x} + l\mathbf{C}_\theta\mathbf{K}\mathbf{f}_{R,y} = \mathbf{D}^{\mathbf{T}}\mathbf{u} \\ Nm\ddot{\boldsymbol{p}} = Nm\begin{bmatrix} \ddot{\boldsymbol{p}}_x \\ \ddot{\boldsymbol{p}}_y \end{bmatrix} = \begin{bmatrix} \mathbf{e}^T\mathbf{f}_{R,x} \\ \mathbf{e}^T\mathbf{f}_{R,y} \end{bmatrix} \end{cases} \tag{2.3.7}$$

where:

$$\begin{aligned} &\mathbf{M}_\theta = J\mathbf{I}_N + ml^2\mathbf{S}_\theta\mathbf{V}\mathbf{S}_\theta + ml^2\mathbf{C}_\theta\mathbf{V}\mathbf{C}_\theta, \quad \mathbf{W} = ml^2\mathbf{S}_\theta\mathbf{V}\mathbf{C}_\theta - ml^2\mathbf{C}_\theta\mathbf{V}\mathbf{S}_\theta, \\ &\mathbf{V} = \mathbf{A}^T\left(\mathbf{D}\mathbf{D}^{\mathbf{T}}\right)^{-1}\mathbf{A}, \end{aligned}$$

We express the angle, position, velocity and angular velocity of each link as the system state, and the moment input acting on the joint of each link as the control, and we obtain a nonlinear system that can accurately represent the motion of the snake robot.

$$\mathbf{x} = \left[\boldsymbol{\theta}^T, \mathbf{p}^T, \dot{\boldsymbol{\theta}}^T, \dot{\mathbf{p}}^T\right]^T \in \mathbb{R}^{2N+4} \tag{2.3.8}$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\boldsymbol{\theta}} \\ \dot{\mathbf{p}} \\ \ddot{\boldsymbol{\theta}} \\ \ddot{\mathbf{p}} \end{bmatrix} = \mathbf{F}(\mathbf{x}, \mathbf{u}) \tag{2.3.9}$$

3 Modelling environment

In this chapter, we will introduce two environments for modelling the motion of a snake robot, describe the specific parameters of the snake robot model and the environment, and their respective advantages.

We will construct a snake robot with six links and five controllers. The length and mass of each link are equal, and a controller is located at each link connection point.

3.1 Simulation of the mathematical model based on Matlab with Simulink simulation software

Firstly, we will use the Simulink simulator to mathematically model the system described in Chapter II. The advantage of using Simulink is that the system can be accurately modelled based on physically derived mathematical models, which will help us to better verify the feasibility of controlling the motion of the snake robot and provide a theoretical basis for subsequent studies.

We construct the environment according to the model described above, and the parameters of the environment and the robot are listed in Table 1. In the process of solving the system of ordinary differential equations, we will use a solver, ode45 which is based on the Runge-Kutta method.

3.2 Model simulation based on MuJoCo simulator

We will then use the MuJoCo [6] modelling software to simulate the motion of a real snake robot. The advantages of using MuJoCo lie in its highly optimised physics engine, which provides fast and accurate physics simulation, which is extremely important for modelling complex dynamical systems such as snake robots. MuJoCo is designed to handle highly nonlinear systems, provides fast collision detection and friction modelling, and provides powerful visualisation tools, making it ideal for robotics applications.

We will conduct experiments on the MuJoCo simulator with control methods not based on exact physical models.

In developing the model, we represent the snake robot links by a series of rectangles of equal length and mass, set the Coulomb friction coefficient with the ground, set joint moments at the link connection points and add angular constraints, and the specific parameters are shown in Table 2.

Таблица 1: Параметры в Matlab

количество звена N	6
длина звена $2l$	0.14
масса звена m	1
c_t	0.5
c_n	3
момент инерции J	0.0016

Таблица 2: Параметры в MuJoCo

количество звена N	6
длина звена l	0.08
масса звена m	1
μ_t	3
μ_n	5
момент инерции J	0.0016

4 Problem statement

In this section, we describe in detail the experimental problem formulation and the methodology for solving them. This study focuses on two main problems of robot snake motion control :

4.1 Moving the snake robot in a straight line

First, the objective of the problem is the ability of the snake robot to move continuously and steadily along a straight trajectory without external interference [7]. This ability is a key metric for evaluating the performance of basic motion control algorithms for the snake robot .

To achieve this goal, we first applied a feedback-based control strategy on a Simulink simulator with known environmental parameters, in particular, we used a PD controller to control the angular changes between the robot links to realise the goal of straight-line motion.

We then investigated a hybrid control method based on fast and slow control transformations and genetic algorithms to find the optimal parameters that can make the robot move fast and linearly under unknown environmental conditions. We implement this algorithm on the MuJoCo simulator.

4.2 Movement of the snake robot to the target point

The second challenge was to develop an algorithm that would allow the snake robot to recognise a target point and automatically redirect its motion to reach that point[8]. The challenge of this task is that the robot must be able to accurately change direction while maintaining smooth and stable motion.

We first conduct a study based on a physical model in Simulink simulator, where the robot changes the direction of motion by continuously switching the target point and changing the output of the head controller, given known environmental parameters.

The above method cannot adapt to the situation of unknown environmental parameters, and changing only the output of the head link does not fully utilise the potential of the robot controllers to perform more complex tasks. Therefore, to perform the task of moving a snake to an arbitrary target in an unknown environment on the MuJoCo simulator, we will try to design a neural network based on the state of the robot to change the output of each controller. We will try to design a robot state-based neural network to change the outputs of each controller to perform the task of moving the snake to an arbitrary target, and then to perform the task of moving the robot along an arbitrary trajectory based on that.

5 Main approaches to solving

5.1 Solving the problem of robot movement along a straight line using the virtual constraint control method

5.1.1 Solution idea

The idea for the solution emerged from the literature [4] и [11]. We use $\bar{\boldsymbol{\phi}} = [\phi_1, \phi_2 ... \phi_{N-1}, \theta_N]^T \in \mathbb{R}^N$ for the complex representation of the angle between the robot links and the angle θ_N used to control the motion direction, where:

$$\phi_i = \theta_i - \theta_{i-1}, i = 1, ..., N-1$$

$$\boldsymbol{\phi} = [\phi_1, \phi_2 ... \phi_{N-1}]^T \in \mathbb{R}^{N-1}$$

We make the robot move forward by controlling the shape of the robot-snake body. Suppose that at time t the desired angle $\phi_{1,ref}, \phi_{2,ref} ... \phi_{N-1,ref}$ is given:

$$\phi_{i,ref} = \alpha \sin\left(\lambda + (i-1)\,\delta\right), i = 1, ..., N-1 \tag{5.1.1}$$

$$\boldsymbol{\phi}_{ref} = \left[\phi_{1,ref}, \phi_{2,ref} ... \phi_{N-1,ref}\right]^T \in \mathbb{R}^{N-1}$$

Thus, we can turn the robot's motion control into the task of tracking a series of given angles $\phi_{i,ref}$. These given angles act as virtual constraints that guide the robot towards the desired shape change, allowing it to perform the forward motion task.

Among them, the positive constant α defines the waviness of the robot. When the value of α becomes larger, the angle between two neighbouring links of the robot in the motion state becomes larger and the lateral wobble of the robot increases.

The positive constant δ is related to the number of links of the robot, which provides a wave-like state of the robot body shape, similar to the sine function.

The function $\lambda(t)$ controls the rate of change of the robot's link angle, thus can control the robot's forward speed. Ideally, when the function $\lambda(t)$ is some linear function $\lambda(t) = \tilde{\lambda} \cdot t$, the robot's velocity v_t also converges to a constant [4].

Next, we assume that λ is a linear function, and derive the control equation for the uniform motion of the snake robot:

$$\mathbf{D}\mathbf{M}^{-1}\mathbf{D}^T\mathbf{u} = \left(\mathbf{D}\mathbf{M}^{-1}\right)\left(\mathbf{M}_\theta\ddot{\boldsymbol{\theta}} + \mathbf{W}\dot{\boldsymbol{\theta}}^2 - l\mathbf{S}_\theta\mathbf{K}\mathbf{f}_{R,x} + l\mathbf{C}_\theta\mathbf{K}\mathbf{f}_{R,y}\right) \tag{5.1.2}$$

$$\mathbf{u} = \left(\mathbf{D}\mathbf{M}^{-1}\mathbf{D}^T\right)^{-1}\left(\mathbf{D}\mathbf{M}^{-1}\mathbf{W}\dot{\boldsymbol{\theta}}^2 - l\mathbf{D}\mathbf{M}^{-1}\mathbf{S}\mathbf{C}_\theta^T\mathbf{f}_R + \mathbf{D}\ddot{\boldsymbol{\theta}}\right) \tag{5.1.3}$$

Let's assume that $\tilde{\boldsymbol{\phi}} = \boldsymbol{\phi} - \boldsymbol{\phi}_{ref}$. Let us write asymptotically stable ordinary differential equations for $\tilde{\boldsymbol{\phi}}$:

$$\ddot{\tilde{\boldsymbol{\phi}}} + k_d\dot{\tilde{\boldsymbol{\phi}}} + k_p\tilde{\boldsymbol{\phi}} = 0 \tag{5.1.4}$$

where k_d and k_p are positive constants. Thus, if the following conditions are fulfilled, $\boldsymbol{\phi}$ is stably equal to 0, and the body shape of the snake robot can match the desired angle and move forward at a uniform speed:

$$\mathbf{D}\ddot{\boldsymbol{\theta}} = \ddot{\boldsymbol{\phi}} = \ddot{\boldsymbol{\phi}}_{ref} - k_d(\dot{\boldsymbol{\phi}} - \dot{\boldsymbol{\phi}}_{ref}) - k_p(\boldsymbol{\phi} - \boldsymbol{\phi}_{ref}) \tag{5.1.5}$$

As a result, we can obtain the control law equation for a snake robot moving forward at a uniform speed:

$$\mathbf{u} = \left(\mathbf{D}\mathbf{M}^{-1}\mathbf{D}^T\right)^{-1}\Big(\mathbf{D}\mathbf{M}^{-1}\mathbf{W}\dot{\boldsymbol{\theta}}^2 - l\mathbf{D}\mathbf{M}^{-1}\mathbf{S}\mathbf{C}_\theta^T\mathbf{f}_R \\ +\ddot{\boldsymbol{\phi}}_{ref} - k_d(\dot{\boldsymbol{\phi}} - \dot{\boldsymbol{\phi}}_{ref}) - k_p(\boldsymbol{\phi} - \boldsymbol{\phi}_{ref})\Big) \tag{5.1.6}$$

5.1.2 Analysing the results

We define the virtual constraint management parameters in Table 3:

Таблица 3: Параметры управления виртуальными ограничениями

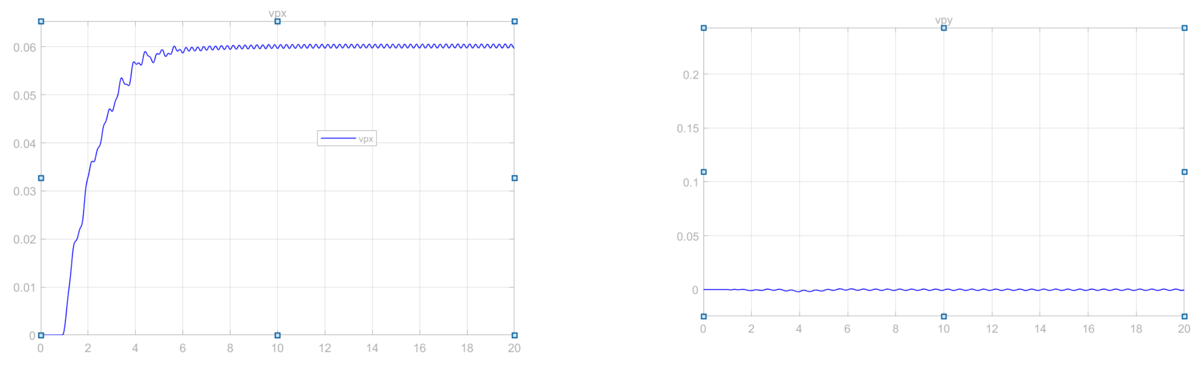
α	λ	δ
0.15	12.5838	1.2566

We set the initial state of the robot as follows: the centre of mass is at the origin and all links are parallel to the Ox-axis.

We limit the output torque range to $[-1, 1]$ and filter the high-frequency output signal using a low-pass [9] filter. This will allow the robot to form an effective serpentine position at the beginning of the motion.

With equation (5.1.6) controlling the motion of the snake robot, we obtain the following results:

First, the change in velocity of the robot's centre of mass in the x-axis v_x and y-axis v_y direction is shown (Figure 3):



(a)

Рис. 3: Скорость цетра массы роботы

It can be seen that after some time, the snake robot tends to a certain velocity value and performs the task of moving along the x-axis with uniform velocity.

Thus, the virtual constrained control is effective for this snake robot model, and it is able to perform the task of moving along a straight line with uniform speed.

Second, we show the numerical variation of the robot controller in Figure 4:

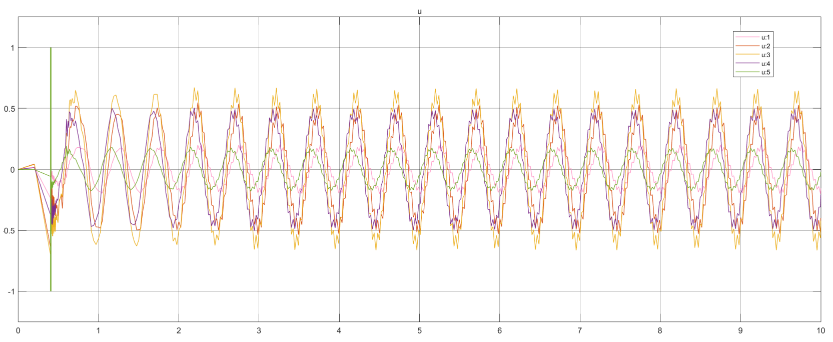


Рис. 4: Численная вариация регулятора робота

It can be seen that after the oscillation period, the output of the controller becomes periodic. At the same time, the closer the controller is to the head or tail of the robot, the smaller the maximum controller output is.

Figure 5 shows the different morphologies of the snake robot during motion:

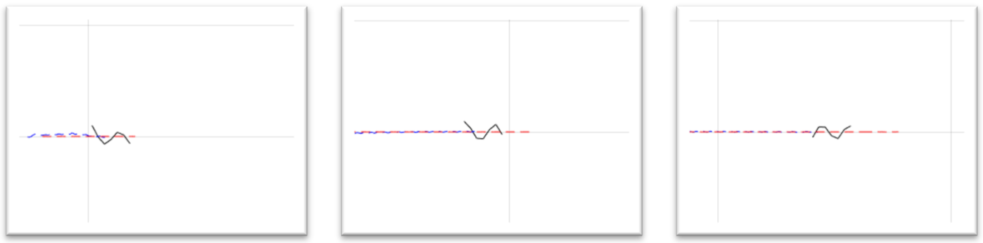


Рис. 5: Различные морфологии робота-змеи

5.2 Solving the problem of robot movement to the target point using sliding mode

5.2.1 Solution idea

We control $\ddot{\theta}_N$, control the torque at the joint point of the head, thus control the direction of the robot.

Let θ_{ref} be the expected angle, and $\tilde{\theta} = \theta_N - \theta_{ref}$, we use the ordinary differential equation [10]:

$$\ddot{\tilde{\theta}} + k_d\dot{\tilde{\theta}} + k_p\tilde{\theta} = 0 \tag{5.2.1}$$

, $\tilde{\theta}$ is asymptotically stable, so we can make the angle θ_N reach the desired value. Set the sliding surface:

$$S = \tilde{\theta} + K_n\dot{\tilde{\theta}} \tag{5.2.2}$$

In order to make $\tilde{\theta}$ and $\dot{\tilde{\theta}}$ exponentially stable, we use the exponential law of attainment of sliding mode control:

$$\dot{S} = -\epsilon_2 sgn(S) - k_2 S \tag{5.2.3}$$

Therefore:

$$u_2 = (-\epsilon sgn(S) - k_2 S - K_n\dot{\tilde{\theta}}) \tag{5.2.4}$$

Thus, the control law can be written as:

$$\begin{aligned}\mathbf{u} = \left(\mathbf{DM^{-1}D}^T\right)^{-1} \Big(&\mathbf{DM^{-1}W}\dot{\boldsymbol{\theta}}^2 - l\mathbf{DM^{-1}SC}_\theta^T\mathbf{f}_R \\ &+\ddot{\boldsymbol{\phi}}_{ref} - k_d(\dot{\boldsymbol{\phi}} - \dot{\boldsymbol{\phi}}_{ref}) - k_p(\boldsymbol{\phi} - \boldsymbol{\phi}_{ref}) \\ &+\mathbf{b}(u_2 + k_d\dot{\theta}_N + k_p\theta_N)\,)\end{aligned} \tag{5.2.5}$$

where: $\mathbf{b} = [0, 0, \ldots, 0, 1]^T \in \mathbb{R}^{n-1}$

5.2.2 Analysing the results

For trajectory tracking tasks, We can obtain the desired trajectory by specifying the desired trajectory $\bar{\mathbf{p}}$, and using the

$$\theta_{ref} = arctg(\frac{\bar{p}_x - p_x}{\bar{p}_y - p_y})$$

to obtain the target angle θ_{ref} so that we can perform some simple [11] trajectory tracking tasks.

Figure 6 shows a snake robot tracking the trajectory $x^2 + (y - 1.5)^2 = 2.25$, where the red curve captures the change in the desired trajectory $\bar{\mathbf{p}}$. The blue curve captures the change in the robot's centre of mass. It can be seen that the robot still does a good job of tracking trajectories with faster angle changes.

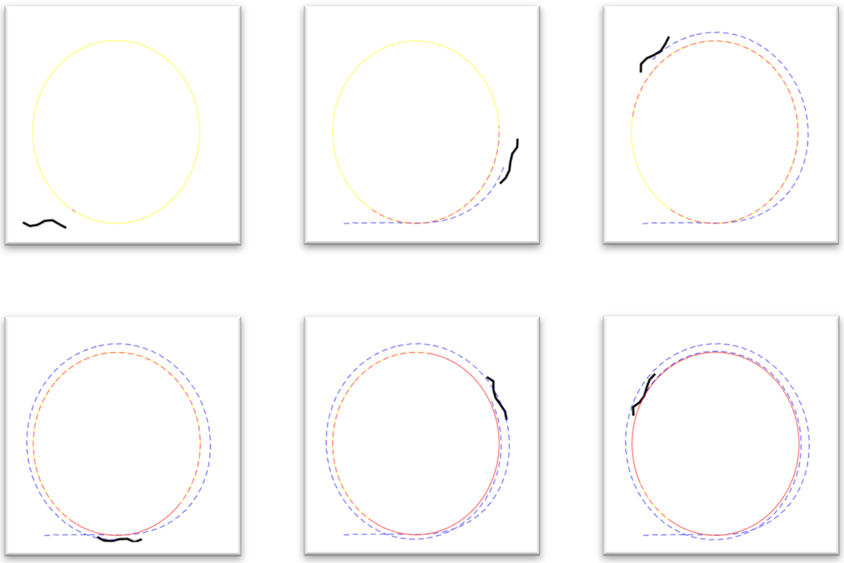


Рис. 6: Робот-змея, отслеживающий траекторию $x^2 + (y - 1.5)^2 = 2.25$

5.3 Solving the problem of robot motion in a straight line using the fast-slow control change method

5.3.1 Solution idea

Fast motion of a three-link snake robot is considered in the [12] literature. This approach is not based on a friction model, but on changing the state of the snake robot using a large control torque to perform the locomotion task. This approach is not possible on a multi-link robot model due to the complexity of the snake robot, which is a nonlinear system. However, this gives us a new idea: we will try to transform the control torque of each link to perform the straight line locomotion task with fast and slow.

Our solution is as follows:

We set five parameters for each controller, namely: control start time T_i^0, fast control period T_i^1, slow control period T_i^2, fast control torque u_i^1 and slow control torque u_i^2.

At the start of control, the controllers between each robot link begin to move periodically according to their respective start times, first producing a fast control torque in the fast control period and then producing a slow control torque in the slow control period. The parameters of the controllers are not related to each other.

The structure of this approach is shown in Figure 7.

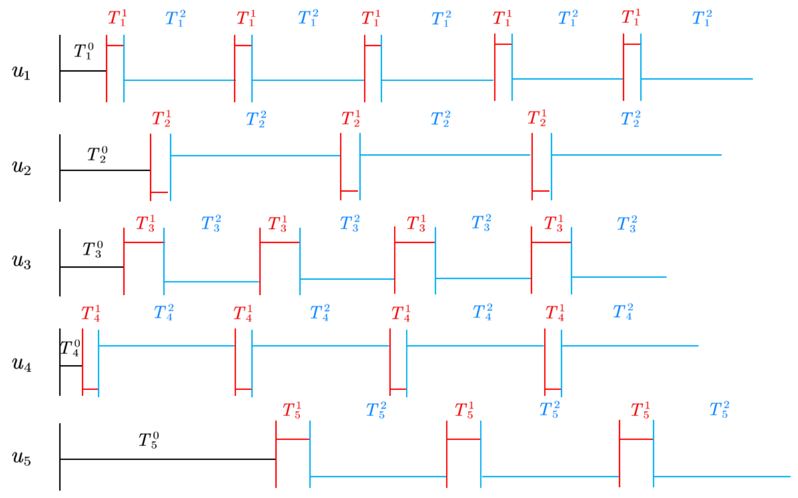


Рис. 7: Периодическое преобразование быстро-медленного управления

In the mode of alternating fast and slow control (angle change) at each joint point, you need to adjust the combination of slow and fast movements. For this purpose, a genetic algorithm is used to find the optimal parameters at each joint point for the forward motion. In this problem, the dimensionality of the parameters is 25.

5.3.2 Генетический алгоритм

A genetic algorithm is an optimisation algorithm that simulates natural selection[13]. They solve optimisation problems by simulating the mechanisms of inheritance, mutation and natural selection during biological evolution. In genetic algorithms, a solution is abstracted as an "individual" and the entire set of solutions constitutes a "population" . Each individual is represented by a string of "chromosomes" (i.e., parameters). Each iteration of the algorithm simulates the cycle of biological evolution, including the processes of selection, crossingover (mating and recombination) and mutation.

The basic steps of the genetic algorithm are as follows:

Initialisation: an initial population is randomly generated.

Evaluation: each individual in the population is evaluated to determine its fitness function.

Selection: individuals are selected based on their fitness and individuals with high fitness have a higher probability of being selected for the next generation.

Crossbreeding: selected individuals are recombined by crossing over chromosomes to produce new individuals.

Mutation: changing certain genes in certain individuals with a certain probability to introduce new genetic diversity.

New generation: new individuals form a new population that replaces the old one.

Repetition: repeat steps 2-6 until the termination condition is reached.

5.3.3 Modelling process

The parameter ranges of the controllers in this problem are summarised in Table 3.

For each "individual"according to the parameters of its "chromosome"a method for controlling the snake robot is developed and a complete simulation is performed on the MuJoCo simulator.

We will use the following fitness function to evaluate the advantages and disadvantages of the "individual".

$$f = e^{-c \cdot p_y} \tag{5.3.1}$$

If the robot's centre of mass is closer to the x-axis, its fitness function will be higher.

The optimisation process will be carried out according to the algorithm described in Section 5.3.2. Some parameters of the algorithm are given in Table 4.

The values of the parameters are given in Table 5.

Таблица 4

T_i^0	[0.0,1.0]
T_i^1	[0.0,0.3]
T_i^2	[0.3,1.5]
u_i^1	[-0.3,0.3]
u_i^2	[-0.03,0.03]

Таблица 5

Количество итераций	400
Количество особей в популяции	200
Количество выбранных особей	40

Таблица 6

T_i^0	[0.50,0.85,0.58,0.31,0.43]
T_i^1	[0.09,0.38,0.02,0.21,0.02]
T_i^2	[1.13,0.48,1.05,0.32,0.47]
u_i^1	[-0.03,-0.26,-0.12,0.24,0.34]
u_i^2	[0.023,0.038,0.001,-0.155,-0.027]

5.3.4 Analysing the results

The parameters obtained by the genetic algorithm are applied to the MuJoCo simulator and Figure 8 will show the trajectory of the robot's centre of mass.

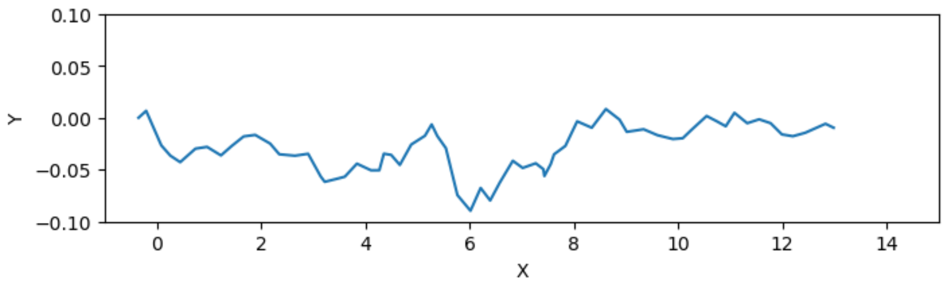


Рис. 8: Траектория движения по прямой

It can be seen that the robot can mainly move in the positive direction of the Ox-axis.

Figure 9 shows different models of the robot during the motion.

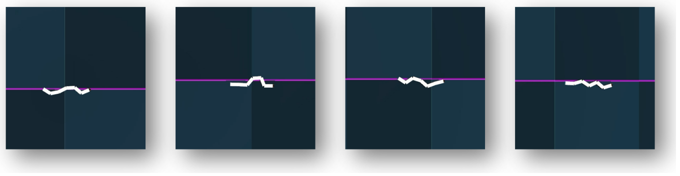


Рис. 9: Различные формы робота во время движения

This proves the effectiveness of our solution.

5.4 Solving the problem of robot movement to the target point using reinforcement learning and oscillator method

5.4.1 CPG Oscillator

A Central Pattern Generator (CPG) is a type of biological neural network (meaning, for example, a specific region in the spinal cord) whose distinguishing feature is its ability to produce rhythmically ordered motor signals without feedback and without constant input excitation[14].

The CPG-based control method, which has recently been widely applied in bionic robots, is a novel bionic control strategy that mimics the principle that the vertebrate central nervous system can spontaneously generate rhythmic signals, and achieves high degree-of-freedom signals by using a small number of control parameters in the upper layers to provide motion control signals to distributed robot joint controllers. The CPG-based robot gait model avoids dependence on the accuracy of dynamics modelling and is easier to implement in complex control.

To provide control signals to the snake robot, we chose the Matsuoka [15] oscillator model, which can provide smoother, more stable, and more energy-efficient motion control than other CPG-based oscillator models. In the following, the structure of the Matsuoka oscillator is briefly described.

In the Matsuoka oscillator model, each neuron consists of two parts that model excitatory and inhibitory neuronal activity, respectively. Each neuron in the oscillator can be represented by a set of nonlinear difference equations:

$$\begin{aligned}
\tau_r \dot{x}_i^e &= -x_i^e - a z_i^f - b y_i^e - \sum_{j=1}^{N} w_{ji} y_j^e + u_i^e \\
\tau_a \dot{y}_i^e &= z_i^e - y_i^e \\
\tau_r \dot{x}_i^f &= -x_i^f - a z_i^e - b y_i^f - \sum_{j=1}^{N} w_{ji} y_j^f + u_i^f \\
\tau_a \dot{y}_i^f &= z_i^f - y_i^f ,
\end{aligned} \tag{5.4.1}$$

where:

$$z_i^q = g(x_i^q) = \max(0, x_i^q)$$

Variables in the model:

x_i^e and x_i^f: represent the state of (i)-th excitatory and inhibitory neuron, respectively.

y_i^e and y_i^f: represent the output of the (i)-excitatory and inhibitory neuron, respectively.

z_i^e and z_i^f: represent the self-excitation state (i) of the excitatory and inhibitory neuron, respectively.

Parameters in the model:.

τ_r: the time constant of the neuron's state (x), which determines the rate at which the neuron's state is updated. A smaller value of (τ_r) makes the state more responsive to changes in the input signal.

τ_a: the time constant of the self-motivated state (z), which controls the rate of dynamic change of the self-motivated state.

a: the strength of the coupling between the self-motivated state and the neuronal state, which determines how strongly the self-motivated state suppresses the neuronal state.

b: the strength of the feedback from the output of neuron (y) to the state of neuron (x), which affects the inhibitory effect of the output on the state.

w_{ji}: the synaptic weight connecting the output of neuron (j) to neuron (i). This weight determines how the outputs of other neurons affect the state of the current neuron.

u_i^e and u_i^f: the external inputs of the excitatory and inhibitory neurons, respectively, which are used to change the amplitude of the neuron's output.

The output equation for each oscillator neuron is as follows:

$$u_i = z_i^e - z_i^f = g(x_i^e) - g(x_i^f) \tag{5.4.2}$$

A schematic of the oscillator model is shown in Figure 10.

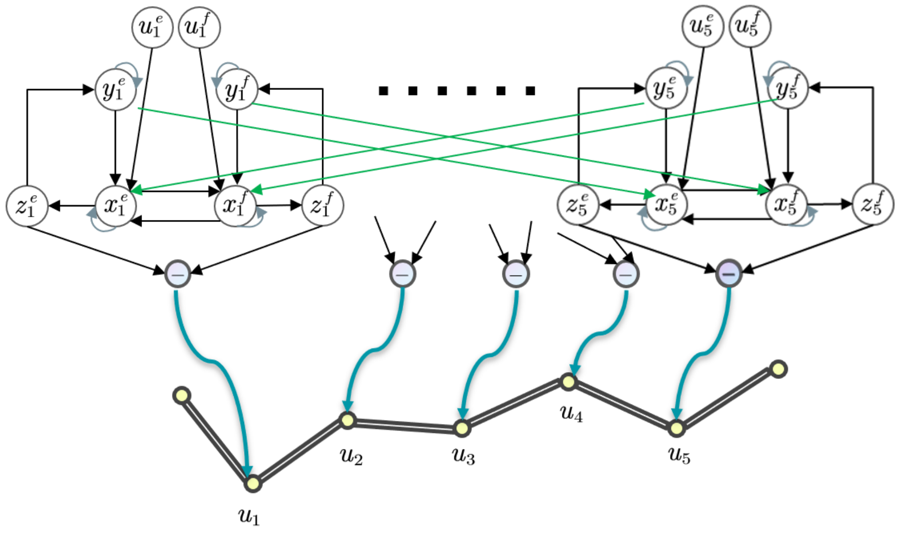


Рис. 10: Осциллятор Мацуока для робота-змеи

By adjusting the parameters in the oscillator model, we can obtain a set of periodic signals. By adjusting the parameters u_i^e and u_i^f, we can change the amplitude of the oscillator output signal without changing the frequency, which provides a theoretical basis for controlling the robot to change the direction of motion.

If the parameters u_i^e and u_i^f are the same for each controller, then the amplitudes will be the same and the output of the oscillator (in Figure 11) will be similar to the output obtained from the PD controller in 5.1 (Figure 4).

We will try to use the output of this oscillator as a controller for the connection points between the links of the snake robot, and the results show that the robot can move forwards by friction against the ground, but cannot control the direction. However, this experiment demonstrates that it is possible to use Matsuoka's oscillator to control the robot's motion, and if we can adjust the values of the signals u_i^e and u_i^f for each neuron (in Figure 12), we can solve the problem of moving the robot in any direction.

The next task is to find a way to change the oscillator signal depending on the state of the robot and thus control the direction of its movement.

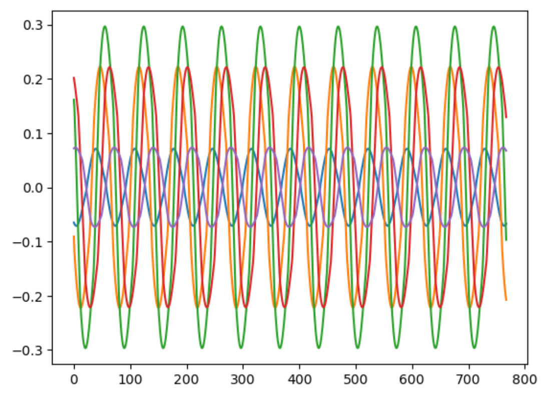


Рис. 11: Выходы осциллятор Мацуока при условия $u_i^e = u_i^f$

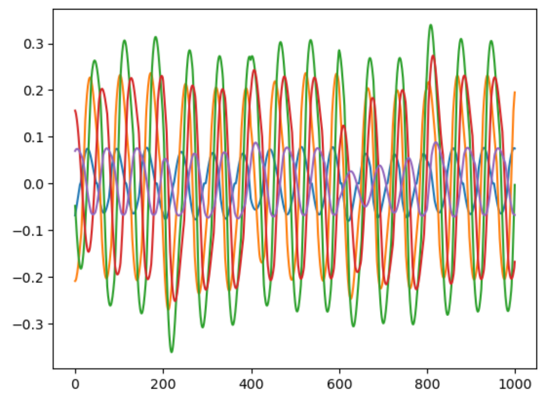


Рис. 12: Выходы осциллятор Мацуока при условия $u_i^e \neq u_i^f$

5.4.2 Algorithm Policy Proximal Optimization

Reinforcement Learning is a machine learning technique that allows an agent to learn the optimal policy through trial and error in the environment. The reinforcement learning process is iterative and recursive, in which the intelligence constantly tries different actions, receives feedback from the environment, learns which actions are more rewarding, and thus gradually improves its action strategy[16] .

The RL problem is modelled as a discrete-continuous Markov decision-making process (MDP) consisting of (S, A, P, R, γ), where

S is a continuous state space,

A is a continuous action space,

P is the dynamics of transitions in the environment, which defines the transition probabilities from one state to another for a given action $p(s_{t+1}|s_t, a_t)$.

R is the reward function, $R(s_t, a_t) = r_t \in \mathbb{R}$, associated with the current state and action

$\gamma \in [0, 1]$ is a discount factor that determines the importance of future awards relative to the current award.

The policy π is stochastic and is mapped from states to a distribution of actions $S \to A$. The agent's goal is to learn a policy $\pi(a_t|s_t)$ that maximises the expected reward:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right]. \tag{5.4.3}$$

where $(\tau \sim \pi)$ denotes the entire MDP trajectory (τ) generated by the policy (π), and (r_t) denotes the rewards at step (t).

In deep reinforcement learning, the policy is represented by an approximation of a nonlinear function, i.e., a $\pi(a_t|s_t; \boldsymbol{\theta})$ neural network parameterised by $\boldsymbol{\theta}$. The input of this neural network is the current state of the agent and the output is the probability of choosing a particular action value. Policy search method use a parameterised policy and directly search for the optimal parameters of that policy.

One of the RL methods is the actor-critic method. The key idea of actor-critic approaches is to simultaneously train a value function and a policy. The critic evaluates the actor's actions based on the learned value function and provides feedback by which the actor updates its policy parameters. The proposed approach follows the actor-critic algorithm and learns both the policy parameters and the value function approximator. This means that the value function also requires a neural network $v(s_t; \mathbf{w})$ parameterised by $\mathbf{w}$.

The policy is trained using the Proximal Policy Optimisation (PPO) algorithm [5]. PPO is one of the most successful policy-based RL algorithms. The PPO algorithm is an approximated version of Trust Region Policy Optimisation (TRPO) with first order gradients. The goal of the PPO algorithm improves the robustness of learning by limiting the policy change at each iteration. The optimisation problem in PPO is solved based on the following surrogate loss function :

$$L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\boldsymbol{\theta}) \hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \tag{5.4.4}$$

where:

$r_t(\boldsymbol{\theta}) = \frac{\pi_\theta(a_t|s_t;\boldsymbol{\theta}))}{\pi_{\theta_{\text{old}}}(a_t|s_t;\boldsymbol{\theta}))}$ is a probability ratio showing how much the new policy π_θ differs from the old policy $\pi_{\theta_{\text{old}}}$ in the sense of the probability of taking action a_t in state s_t.

The $\hat{A}_t$ element is an estimate of the advantage at time t, which shows the benefit of taking action a_t in state s_t compared to the expected value.

To compute the advantage $\hat{A}_t$, we use the Generalised Advantage Estimation (GAE):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{5.4.5}$$

where $\delta_t = r_t + \gamma v(s_{t+1};\mathbf{w})) - v(s_t;\mathbf{w})$ is the temporal differences, $v(s_t;\mathbf{w})$ is the value function, and γ and λ are the discount factors for reward and benefit, respectively.

Using the objective function, the policy is updated by a gradient ascent along $L^{CLIP}(\boldsymbol{\theta})$. The gradient of the clipped objective is defined as follows:

$$\nabla_\theta L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t, \nabla_\theta \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{5.4.6}$$

Updating parameters for policy neural networks:

$$\boldsymbol{\theta_{new}} \leftarrow \boldsymbol{\theta_{old}} + \beta\nabla_\theta L^{CLIP}(\boldsymbol{\theta}) \tag{5.4.7}$$

For a value neural network, the loss function has the form

$$L^{VF}(w) = \hat{\mathbb{E}}_t \left[(v(s_t;\mathbf{w}) - v_t^{target})^2\right] \tag{5.4.8}$$

where $(v(s_t;;\mathbf{w})$ is the network's estimate of the present value of the state (s_t), and $v_t^{target})$ is the target value of the state, which is estimated by accumulating discounted rewards.

The gradient of the loss function with respect to the parameter $\mathbf{w}$:

$$\nabla_w L^{VF}(\mathbf{w}) = \hat{\mathbb{E}}_t \left[2(v(s_t;\mathbf{w}) - v_t^{target})\nabla_w v(s_t;\mathbf{w})\right] \tag{5.4.9}$$

Parameter updates for value neural networks:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \alpha\nabla_w L^{VF}(\mathbf{w}) \tag{5.4.10}$$

where α and β are the learning rates of the policy neural network and the value neural network, respectively.

This update rule ensures that the policy does not change too abruptly, which helps to maintain a stable learning progress, especially in continuous action spaces.

For continuous action spaces, the policy $\pi_\theta(a|s)$ is usually parameterised as a Gaussian distribution, where the neural network outputs the mean and covariance of the distribution. This parameterisation allows the algorithm to efficiently handle the complexities of continuous action spaces.

By carefully tuning parameters such as the ϵ cut-off threshold, the γ coefficient and the λ coefficient for GAE, PPO can be adapted to different environments and reinforcement learning tasks.

5.4.3 Reinforcement learning for the CPG controller of a snake robot

From the analysis of Section 5.1, we can perform the task of moving the snake robot using the CPG oscillator model, but we cannot control the direction of the robot, so we need to find a way to control the direction of the robot by adjusting the parameters of the oscillator.

One simple implementation is similar to the one described in Section 5.2, i.e., tracking the target point. However, unlike the method described in Section 5.2, the period of change of the oscillator parameter should not be too short, as in this case the signal would change too quickly and would not be able to effectively guide the robot's motion. In addition, the method described in Section 5.2 only changes the output of the first link, which does not fully utilise the potential of the multidimensional motion space.

Therefore, we will change the signal amplitude of all oscillator neurons every 1 second, which is consistent with the idea of taking discrete states to form a Markov decision-making process in reinforcement learning. Thus, the multivariate output neural network is trained by reinforcement learning method to enable the snake robot to make the optimal control decision in different states, and this method achieves the task of moving the robot to the target point.

In the following, we describe the idea and implementation of this method separately.

State space

The input data of the neural network - the state space of the robot - should contain, if possible, all state values that can influence the choice of action.

Since our task is to train the snake robot to move towards a goal, all values in the state space should be relative to the goal point. The advantage of this approach is that the robot does not need to consider the absolute values of the states in the environment when learning successfully, which greatly reduces the complexity of learning. For example, if the robot's velocity in the environment is known and its relative position relative to the target is known, it will take some training for the robot to learn whether the velocity in the environment relative to the target is good or bad... In addition, using only relative state values also simplifies the experimental design of the snake robot; we can let the initial state of the robot be fixed, and we do not need to consider whether the state of the robot in the environment (e.g., the initial angle of each link in the environment and the initial direction of the robot in the environment) affects the choice of action[17].

At time t, we choose the state $\mathbf{s} = [\phi, v_1, v_2, d, \boldsymbol{\theta}]$ as:

- $\phi(t) \in \mathbb{R}$ - Angle between the displacement vector from the robot's centre of mass to the target point $\vec{d}(t)$ and the robot's velocity vector relative to the environment $\vec{v}(t)$.
- $v_1(t) = |\vec{v}(t)| \cos\phi(t) \in \mathbb{R}$ - The velocity of the robot in the desired direction of motion.
- $v_2(t) = |\vec{v}(t)| \sin\phi(t) \in \mathbb{R}$ - Robot velocity perpendicular to $v_1(t)$, indicating the difference between the robot's direction of motion and the desired direction.
- $d(t) = ||\vec{d}(t)| \in \mathbb{R}$ - Distance between the robot's centre of mass and the target point.
- $\boldsymbol{\theta}(t) \in \mathbb{R}^6$ - A vector consisting of the angle between the first link and $\vec{d}$ and the angle between each link.

where:

$\vec{d}(t) = (x_2 - x_1, y_2 - y_1)$ - Difference between the target point (x_2, y_2) and the robot's centre of mass (x_1, y_1).

$\vec{v}(t) = (v_x, v_y)$ - The velocity of the robot in the environment (v_x, v_y in the Ox and Oy directions, respectively) determined by the velocity of the robot's centre of mass.

Figure 13 shows the relationship between the states.

During training, sometimes the robot's velocity sensor sometimes produces a very large "glitch"value because the oscillator is used to change the amplitude of the signal at regular intervals rather than controlling the robot at each time step as in Section 5.2, so

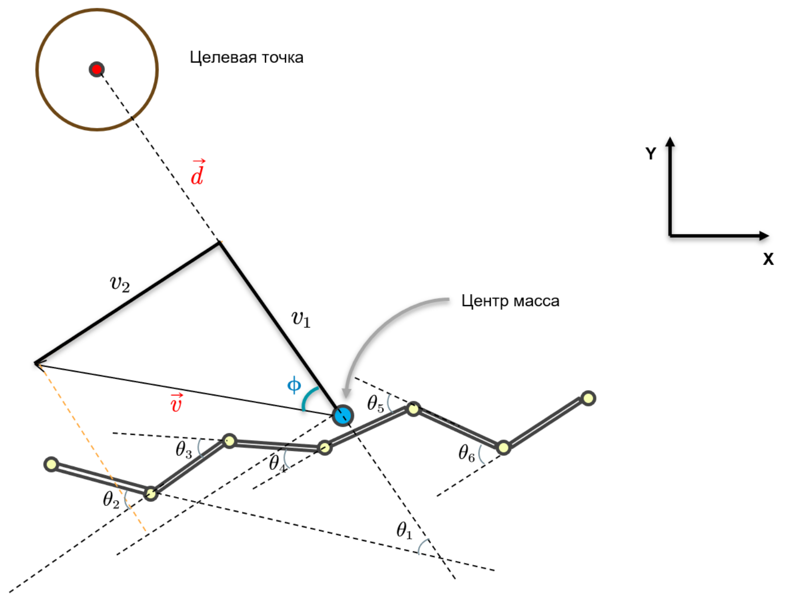


Рис. 13: Пространство состояний

the velocity fluctuates a lot. Therefore, we choose a sliding window and take the average of the velocity over the last 0.1 seconds, $\hat{\vec{v}}(t)$ as the velocity at time t. This method can effectively guarantee the stability of the data without loss of accuracy, which is favourable for the subsequent learning process.

Since we consider the angular values between links, the angular velocities can be very large if the frequency of the oscillator is higher during a cycle of 1 second, which means that the angular value $\boldsymbol{\theta}(t)$ at a certain time point does not provide a complete description of the robot's state during that cycle.And it may be difficult for the robot to learn the optimal state value function. The solution is to record the state every 0.1 seconds during the cycle. At the end of the cycle, all the state value data is used as a description of the state in that cycle. Thus, the input layer of the neural network is a 100-dimensional vector containing 10 data recorded during this cycle[18].

Action space

In Section 5.1, we analysed the parameters that can affect the output of the oscillator. We focused on two sets of parameters, u_i^e and u_i^f, that can directly change the direction of the robot's motion.

To reduce the dimensionality of the action space, we can decompose these two parameters using the following equation:

$$\begin{aligned} u_i^e &= u_m(\hat{u}_i + \tilde{u}_i) \\ u_i^f &= u_m(\hat{u}_i - \tilde{u}_i) \end{aligned} \tag{5.4.11}$$

where:

u_m controls the maximum allowable value of the oscillator output.

$\hat{u}_i = 1$ is a constant value that provides the basic kinematic capabilities of the snake robot.

$\tilde{u}_i$ is bounded by a certain interval, which represents the difference between the positive and negative amplitudes of the ith oscillator neuron in a given cycle, and indicates the directional choice of the ith robot controller.

The advantage of this design is that it allows the robot to not lose the ability to move forward by rubbing against the ground, due to sudden changes in the amplitude of the oscillator output, while at the same time gaining the ability to change direction and reducing the dimensionality of the action space. Therefore, we will use this design to convert the regulation of parameters u_i^e and u_i^f into the regulation of parameters $\tilde{u}_i$.

The advantage of this design is that the robot can maintain a relatively stable motion speed due to continuous sinusoidal motion and does not lose the ability to move forward by rubbing against the ground due to abrupt changes in the amplitude of the oscillator output signal, resulting in loss of motion speed. In addition, the robot gains the ability to change the direction of motion and reduces the dimensionality of the motion space. Therefore, we will use this design to convert the u_i^e and u_i^f parameter control into $\tilde{u}_i$ parameter control.

Structure of neural networks

We present the structure of the policy neural network and the value neural network separately

If the value of the robot's state at $(0.1 \cdot i)$-th second of a 1-second cycle is $\mathbf{s}^i = [\phi^i, v_1^i, v_2^i, d^i, \boldsymbol{\theta}^i] \in \mathbb{R}^{10}$, then the input layers of the policy neural network and the value neural network receive state values $\mathbf{s} = [\mathbf{s}^1, \mathbf{s}^2, ..., \mathbf{s}^{10}] \in \mathbb{R}^{100}$

The output layer of the policy neural network is a 5-dimensional signal vector, which is bounded by $\tilde{u}_i aftervaluesareobtained \in [-0.1, 0.1]$.The output of the value neural network is a real number, which is a state estimate.

The hidden layer of the policy neural network consists of two layers, each containing 256 neurons, and is structured as a fully connected network. The activation function of each layer is a function $tanh$.

The structure of the hidden layer of the value neural network is the same as the policy neural network, but the activation function of each layer is the ReLu function.

Figure 14 shows the structure of the policy neural network $\pi(\mathbf{s};\boldsymbol{\theta})$ with the value neural network $v(\mathbf{s};\mathbf{w})$.

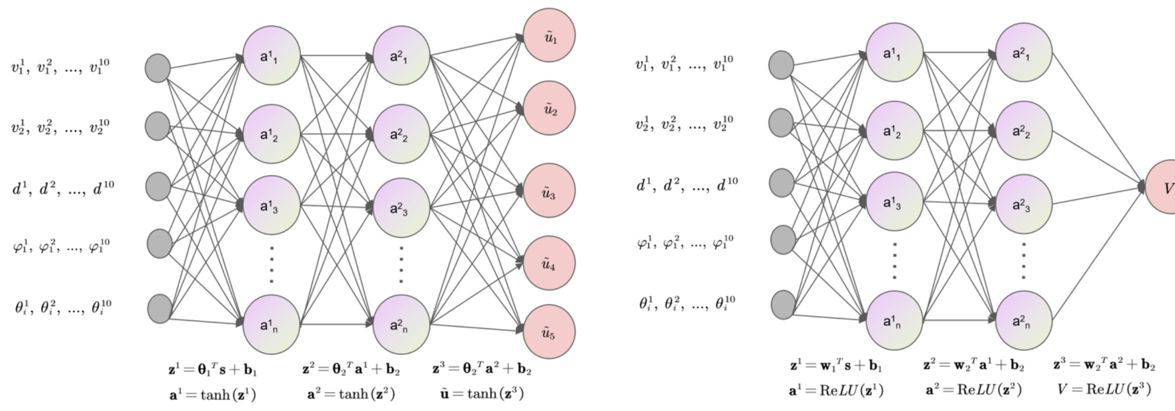

$v_1^1,\ v_1^2,\ ...,\ v_1^{10}$
$v_2^1,\ v_2^2,\ ...,\ v_2^{10}$
$d^1,\ d^2,\ ...,\ d^{10}$
$\varphi_1^1,\ \varphi_1^2,\ ...,\ \varphi_1^{10}$
$\theta_i^1,\ \theta_i^2,\ ...,\ \theta_i^{10}$
a^1_1
a^1_2
a^1_3
a^1_n
a^2_1
a^2_2
a^2_3
a^2_n
$\tilde{u}_1$
$\tilde{u}_2$
$\tilde{u}_3$
$\tilde{u}_4$
$\tilde{u}_5$
$\mathbf{z}^1 = \boldsymbol{\theta}_1{}^T\mathbf{s} + \mathbf{b}_1$
$\mathbf{a}^1 = \tanh(\mathbf{z}^1)$
$\mathbf{z}^2 = \boldsymbol{\theta}_2{}^T\mathbf{a}^1 + \mathbf{b}_2$
$\mathbf{a}^2 = \tanh(\mathbf{z}^2)$
$\mathbf{z}^3 = \boldsymbol{\theta}_2{}^T\mathbf{a}^2 + \mathbf{b}_2$
$\tilde{\mathbf{u}} = \tanh(\mathbf{z}^3)$
$v_1^1,\ v_1^2,\ ...,\ v_1^{10}$
$v_2^1,\ v_2^2,\ ...,\ v_2^{10}$
$d^1,\ d^2,\ ...,\ d^{10}$
$\varphi_1^1,\ \varphi_1^2,\ ...,\ \varphi_1^{10}$
$\theta_i^1,\ \theta_i^2,\ ...,\ \theta_i^{10}$
a^1_1
a^1_2
a^1_3
a^1_n
a^2_1
a^2_2
a^2_3
a^2_n
V
$\mathbf{z}^1 = \mathbf{w}_1{}^T\mathbf{s} + \mathbf{b}_1$
$\mathbf{a}^1 = \mathrm{Re}LU(\mathbf{z}^1)$
$\mathbf{z}^2 = \mathbf{w}_2{}^T\mathbf{a}^1 + \mathbf{b}_2$
$\mathbf{a}^2 = \mathrm{Re}LU(\mathbf{z}^2)$
$\mathbf{z}^3 = \mathbf{w}_2{}^T\mathbf{a}^2 + \mathbf{b}_2$
$V = \mathrm{Re}LU(\mathbf{z}^3)$

Рис. 14: Структура нейронной сети политики и ценности

Reward function

A good reward function effectively guides the learning algorithm towards the target behaviour. It should accurately reflect the goals of the task, ensuring that each reward or punishment given is designed to push the algorithm towards the desired outcome.

In our solution, we specify a reward function at time t:

$$r_t = (c_1 v_1 - c_2|v_2| + c_3|v|) \cdot e^{-c_4 \cdot d} + c_5 \cos\phi \cdot I(d < d_0) \tag{5.4.12}$$

where:

v_1- velocity of the robot in the desired direction of motion, which belongs to the vector of the current state of the robot.

v_2- the robot velocity directed perpendicular to v_1.

$|v|$ is the scalar of the robot's velocity relative to the environment.

d is the distance between the robot's centre of mass and the target point.

ϕ is the angle between the displacement vector from the robot's centre of mass to the target point and the robot's velocity vector relative to the environment.

$I(d < d_0)$ is an indicator function indicating whether the robot's centre of mass is within a radius d_0 of the target point.

The reward function designed in this way is reasonable and efficient.

First, the environment analyses the current state of the robot and rewards actions that move the robot towards the goal and penalises actions that move the robot away from the goal, which is achieved by setting v_1 and v_2. Second, an absolute speed reward $|v|$ is added to incentivise the robot to move as fast as possible rather than stay in a particular position. Before reaching the goal, the reward depends on the distance d to the goal, and the closer to the goal, the higher the reward. We can find the optimal reward function by adjusting four parameters c_1, c_2, c_3 and c_4.

When the robot reaches the target point, it receives a large reward, which is given by the indicator function $I(d < d_0)$ and the parameter c_5. This reward is related to the direction of the robot's movement when it enters the target zone. If the robot enters the target zone with the direction of motion towards the target point, it receives a large reward and vice versa it receives a small reward. This idea is realised using $\cos\phi$.

Training process

We describe step-by-step the process of training a snake robot on the MuJoCo simulator.

1. initialisation

We first complete the parameterisation of the snake robot model, the oscillator model, the neural networks and the learning process.

For the snake robot model, we have already defined this in chapter 3.2

For the oscillator model (5.4.1), we set the parameters as shown in Table 7.

For the reward function, we set the parameters as shown in Table 8.

The corresponding parameters for the reinforcement learning algorithm are shown in Table 9.

We wrote the programme in python and used mujoco_py, a python interface to the MuJoCo physics engine. We used the machine learning library pytorch to build and train the neural networks.

To initialise the two neural networks, we use the kaiming initialisation from the pytorch library.

2. Выполнить сброс среды

Таблица 7

a	2
b	10
τ_r	0.3125
τ_a	0.3125
$w_{ji\ (j=i+1)}$	$\pi/2$
$w_{ji\ (j\neq i+1)}$	0

Таблица 8

c_1	10
c_2	5
c_3	2
c_4	0.5
c_5	20

Таблица 9

β	2e-4
α	2e-4
γ	0.96
λ	0.95
ϵ	0.2
batch size	256

At the beginning of training, we set the initial state of the snake robot as follows: all links are located on the Ox-axis, the initial velocity is 0, and the center of mass is at $(-1.2, 0)$, as shown in Figure 15.

The center of mass is not placed at the origin because we want the snake robot to achieve a good initial velocity and initial gait after a launching period during which the oscillator signal will not change. At the end of the launch period, the center of mass of the robot will be able to move close to the origin of the coordinates and will have an initial velocity in the x-axis direction.

The reason for setting the start time is that the initial velocity of the snake robot is 0 and there is no angle between the links, so it has not yet formed the snake pose, and such a special state value can easily lead to the instability of the neural network parameters.

We set the training start time equal to 5 seconds. From the 5th second, the learning process starts, and the next series of actions and states are treated as an episode until the environment is reset.

Before starting each episode, we need to set a target point. In this task, we set the target point to appear randomly in an area centered at the origin, with a radius between 1.5 and 2.5 m, a center line along the x-axis, and a sector angle of 120 degrees, as shown in Figure 16.

3. Choice of actions

For a multidimensional continuous action space problem, a strategic neural network typically outputs a vector of mean $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\sigma}$ corresponding to the dimensions of the actions. These output parameters define a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$, where $\text{diag}(\boldsymbol{\sigma}^2)$ denotes the vector of values, starting with $\boldsymbol{\sigma}^2$ as diagonal elements, indicating that each dimension of the action is independent and has different variance.

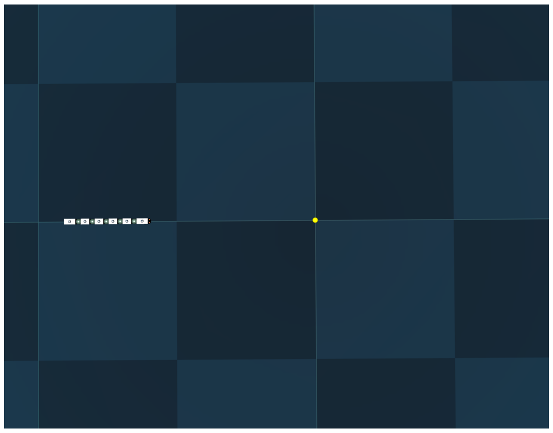


Рис. 15: Начальное состояние ($t = 0$)

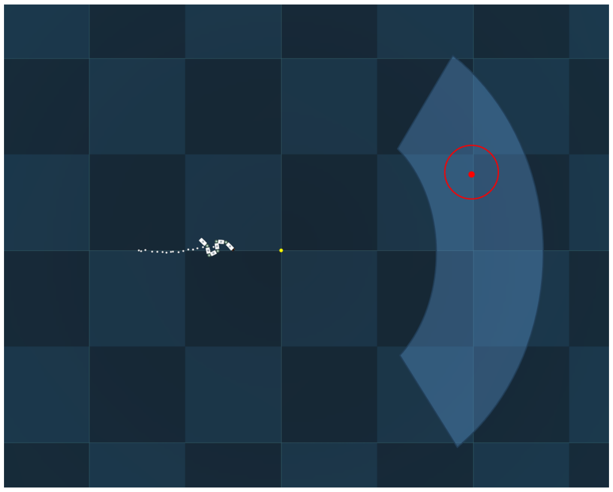


Рис. 16: Состояние после запуска ($t = 5$)

In the simulation process, we will collect and store the robot state several times according to a given time period. At the end of the cycle, the current time is set to t, and the state vector $\mathbf{s}_t$ at that time is input to the policy neural network. The neural network outputs a vector of mean $\boldsymbol{\mu}_t$ and a vector of standard deviation $\boldsymbol{\sigma}_t$ of the same dimensionality as the action space, describing the probability distribution of actions in this state.

To determine the choice of action at a given moment, the action vector $\mathbf{a}_t$ is chosen from the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_t, \text{diag}(\boldsymbol{\sigma}_t^2))$ of the network output. This selection process can be expressed as:

$$\tilde{\mathbf{u}}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \text{diag}(\boldsymbol{\sigma}_t^2)) \tag{5.4.13}$$

The selected action $\tilde{\mathbf{u}}_t$ will change the amplitude of the oscillator, and the robot will update the state of its own oscillator according to this action, change its own state, and interact with the environment, receiving feedback from it as described below.

If the value of the actions goes out of bounds, they are pruned.

Figure 17 shows a process where the agent (robot) makes a decision and takes actions based on the state of the previous period.

4. Performing actions and interacting with the environment

The actions $\tilde{\mathbf{u}}_t$ change the output amplitude of each oscillator neuron in the next cycle, i.e., the maximum and minimum output values. The robot obtains new control due to the new oscillator dynamics and implements these controls on the MuJoCo simulator.

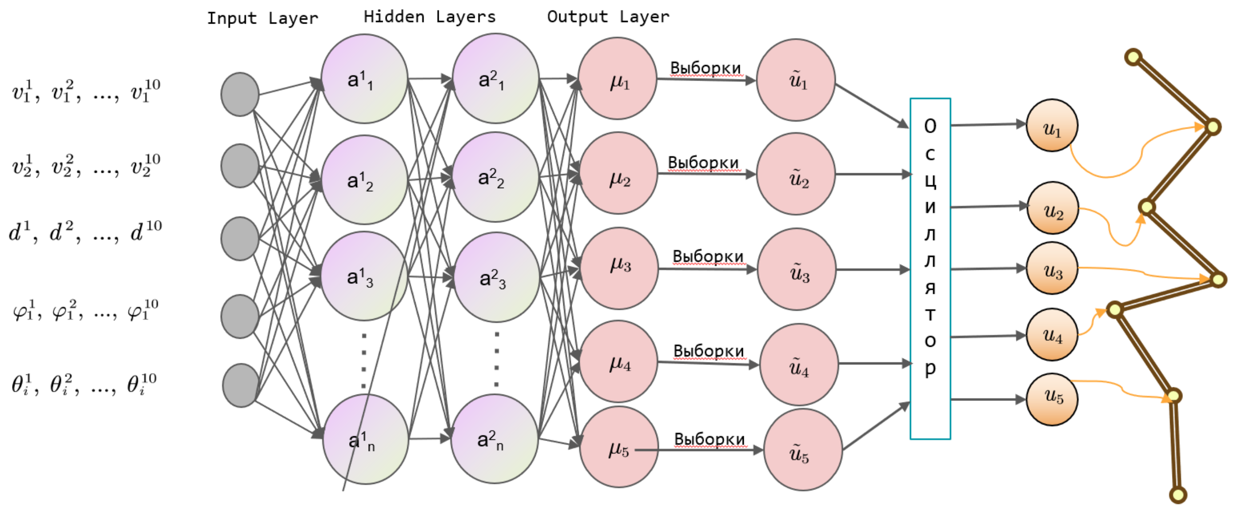

Input Layer
Hidden Layers
Output Layer
$v_1^1, v_1^2, ..., v_1^{10}$
$v_2^1, v_2^2, ..., v_2^{10}$
$d^1, d^2, ..., d^{10}$
$\varphi_1^1, \varphi_1^2, ..., \varphi_1^{10}$
$\theta_i^1, \theta_i^2, ..., \theta_i^{10}$
a^1_1 a^1_2 a^1_3 a^1_n
a^2_1 a^2_2 a^2_3 a^2_n
μ_1 μ_2 μ_3 μ_4 μ_5
Выборки
$\tilde{u}_1$ $\tilde{u}_2$ $\tilde{u}_3$ $\tilde{u}_4$ $\tilde{u}_5$
Осциллятор
u_1 u_2 u_3 u_4 u_5

Рис. 17: Схема процесса решения и действия

At the end of the new period, a piece of data is generated including:

–old states of the previous period,

–new states of this period,

–selected actions,

–the reward obtained according to the new state using the reward function,

–the probability that such actions will be selected,

–indicator function whether the episode has ended or not.

This piece of data will be placed in a buffer.

5. Continuous data collection

As long as the episode is not interrupted, follow steps 3 and 4.

6. Conditions for terminating an episode

There are 3 possibilities for ending an episode.

1. The center of mass of the robot reaches the target region.

2. The robot deviates from the target direction. In this task, if the state value $v_1 < 0$ at the end of the last three periods, the robot is said to have deviated from the target direction.

3. The maximum learning step of the episode has been reached.

Once these situations occur, the episode ends, the environment is reset, a new episode is created, and steps 3 and 4 continue, and so on.

7. Network updates

If the data buffer is full, the policy neural networks and the value neural networks are updated according to the algorithm described in 5.4.2.

8. Graduation

We set the condition that learning is successful if the robot reaches the target point in 80 of the last 100 episodes.

5.4.4 Analyzing the results

After about 800 episodes and about 80 updates of the neural network, we obtained a good result using the PPO algorithm on the MuJoCo simulator according to the above training process and parameters.

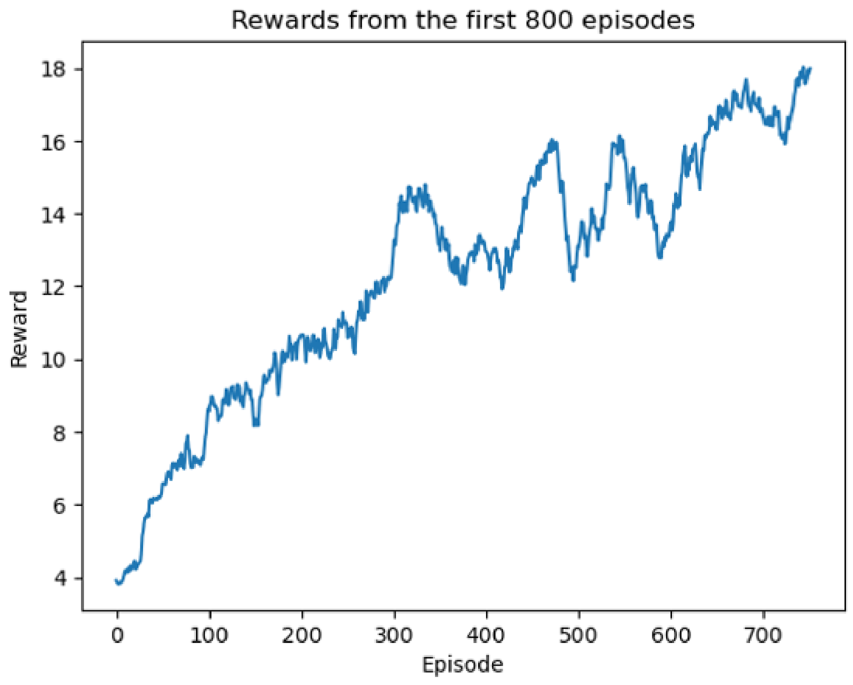


Рис. 18: Значения награды каждого эпизода

Figure 18 shows the reward values obtained for each episode during training (sliding window optimization). There is a clear trend of increasing reward values as training progresses. This demonstrates the effectiveness of combining an oscillator with a neural network with a policy trained using reinforcement learning to achieve robot movement towards a target point.

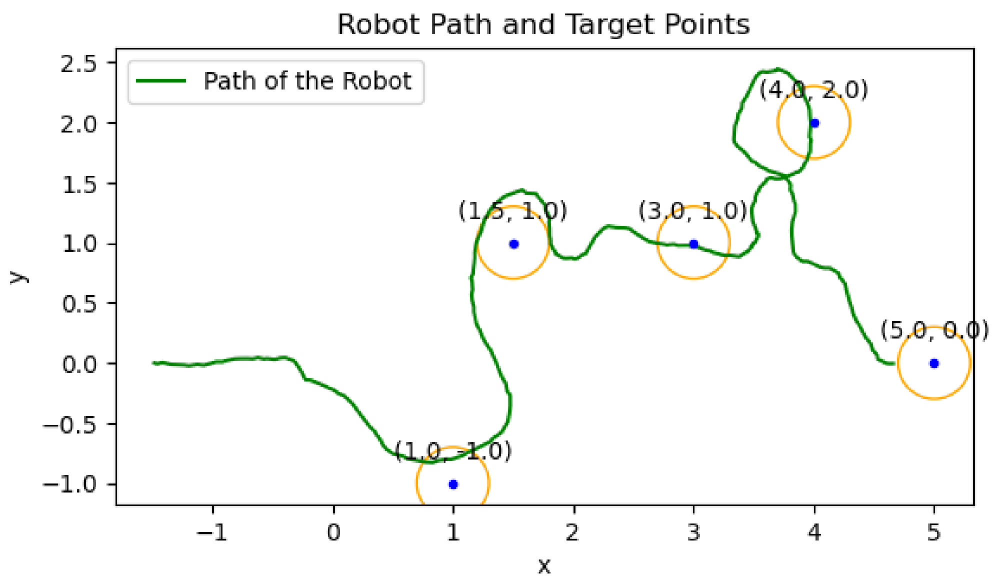


Рис. 19: Результат движения к целевой точке (1)

To demonstrate the results, we conduct an experiment. In this experiment, five target points are predefined:

$$(1.0, -1.0), (1.5, 1.0), (3.0, 1.0), (4.0, 2.0), (5.0, 0.0)$$

and the robot moves sequentially towards them. If the robot reaches the target zone (a circular zone centered at the target point with a radius of 0.3 m), the target point changes.

Figures 19, 20, and 21 show the movement of the snake robot toward the target point. Figure 22 shows the scalar value of the velocity of the robot during its motion. It can be seen that the robot does an excellent job of tracking the target points and can maintain a certain speed. However, there remains a problem that the robot cannot control the angle of motion in the environment when it reaches the target points. This requires further investigation.

6 Conclusion

In this paper, we build a physical model of a snake robot and demonstrate that the snake robot can perform snake-like sinuous motions under friction using a virtual constraint control method, and can control the robot's motion direction by controlling the head angle.

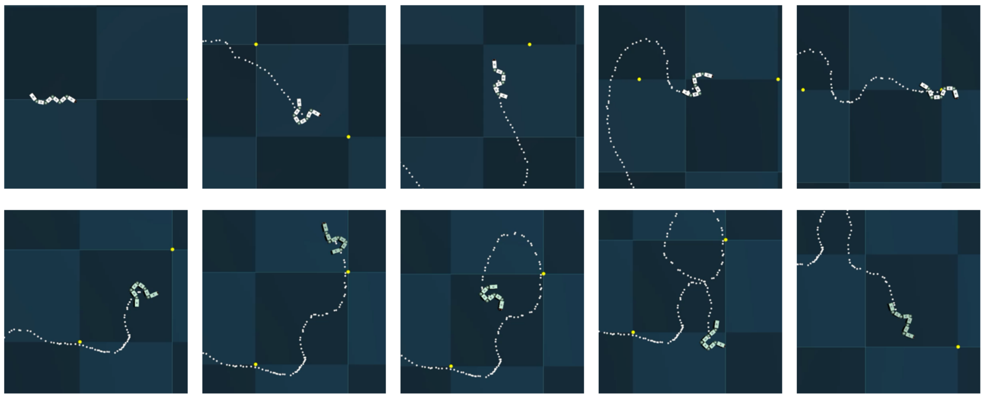


Рис. 20: Результат движения к целевой точке (2)

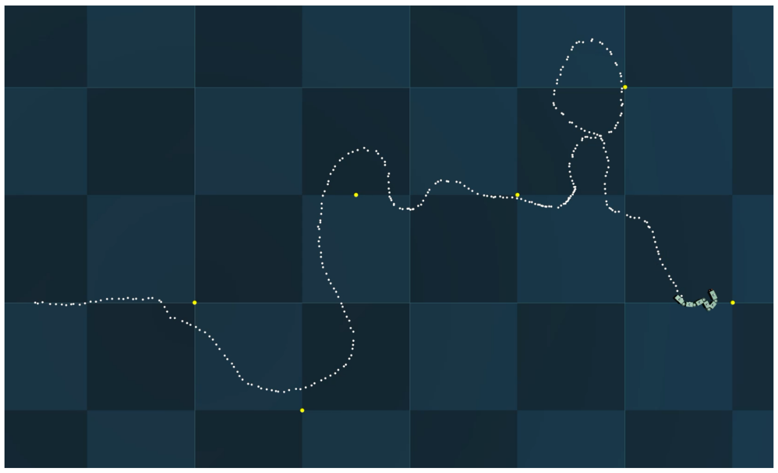


Рис. 21: Результат движения к целевой точке (3)

In addition, we investigate two control methods for unknown environments.

First, we use a genetic algorithm to find the optimal parameters for the snake to move in a straight line by alternating fast and slow control. This method is not based on a friction model and has good potential for application.

Second, the oscillator is used to model the periodic output signal of the snake robot, and the reinforcement learning algorithm is used to train the robot to select the oscillator signals in different states to change the direction of the robot's motion and accomplish the task of moving to the target point.

The above studies show that these methods are effective in solving the complex motion control problem of the snake robot.

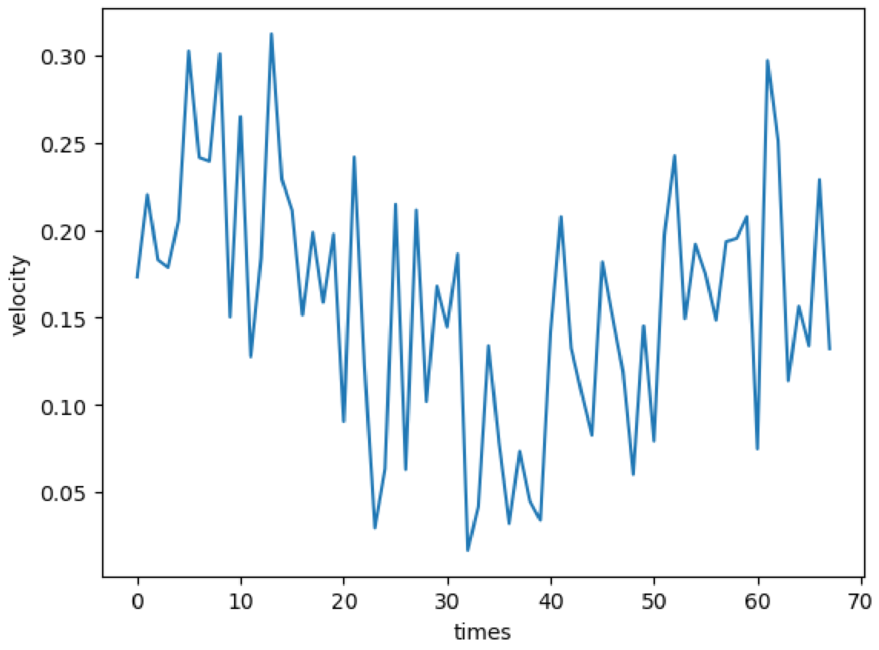


Рис. 22: Скалярное значение скорости

Список литературы

[1] *Owen T. "Biologically Inspired Robots: Snake-Like Locomotors and Manipulators"by Shigeo Hirose Oxford University Press, Oxford, 1993, 220pages, incl. index (£40). Robotica. 1994;12(3):282-282.*

[2] *Liu,J., Tong,Y., Liu,J. "Review of snake robots in constrained environments."Robotics and Autonomous Systems, 141, 103785 (2021).*

[3] *M. Tanaka and K. Tanaka, "Control of a Snake Robot for Ascending and Descending Steps,"in IEEE Transactions on Robotics, vol. 31, no. 2, pp. 511-520, April 2015*

[4] *P.Liljeback, K.Y.Pettersen, Ø.Stavdahl and J. T. Gravdahl, "Snake robots: modelling, mechatronics, and control", Springer-Verlag, London, 2013.*

[5] *Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347 (2017).*

[6] *E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in Proc. IEEE/RSJ Int. Conf. Int. Robots Syst., 2012, pp. 5026–5033*

[7] *P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl and J. T. Gravdahl, "A simplified model of planar snake robot locomotion,"2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 2010, pp. 2868-2875*

[8] *Z. Bing, C. Lemke. "Energy-efficient and damage-recovery slithering gait design for a snake-like robot based on reinforcement learning and inverse reinforcement learning" // Neural Netw, 2020 Sep:129:323-333.*

[9] *Rezapour, E., Pettersen, K.Y., Liljebäck, P. et al. Path following control of planar snake robots using virtual holonomic constraints: theory and experiments. Robot. Biomim. 1, 3 (2014).*

[10] *J. Mukherjee, I. N. Kar and S. Mukherjee, "Adaptive sliding mode control for head-angle and velocity tracking of planar snake robot,"2017 11th Asian Control Conference (ASCC), Gold Coast, QLD, Australia, 2017, pp. 537-542*

[11] *A. Mohammadi, E. Rezapour, M. Maggiore and K. Y. Pettersen, "Maneuvering Control of Planar Snake Robots Using Virtual Holonomic Constraints,"in IEEE Transactions on Control Systems Technology, vol. 24, no. 3, pp. 884-899, May 2016*

[12] *Черноусько Ф. Л., Болотник Н. Н. "Динамика мобильных систем с управляемой конфигурацией", 2022.*

[13] *Mitchell, M. (1996). "An introduction to genetic algorithms."The MIT Press.*

[14] *Xiaodong Wu, Shugen Ma, "CPG-based control of serpentine locomotion of a snake-like robot Mechatronics, Volume 20, Issue 2, 2010, Pages 326-334*

[15] *Matsuoka, K. Mechanisms of frequency and pattern control in the neural rhythm generators. Biol. Cybernetics 56, 345–353 (1987).*

[16] *Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99). MIT Press, Cambridge, MA, USA, 1057–1063.*

[17] *X. Liu, R. Gasoto, Z. Jiang, C. Onal and J. Fu, "Learning to Locomote with Artificial Neural-Network and CPG-based Control in a Soft Snake Robot,"2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 7758-7765*

[18] *Jiayu Wang, Chuxiong Hu, and Yu Zhu, "CPG-Based Hierarchical Locomotion Control for Modular Quadrupedal Robots Using Deep Reinforcement Learning"IEEE Robotics and Automation Letters (Volume: 6, Issue: 4, October 2021)*