Spring 4-28-2023

MOTION CONTROL SIMULATION OF A HEXAPOD ROBOT

Weishu Zhan
weishu.zhan.th@dartmouth.edu

MOTION CONTROL SIMULATION OF A HEXAPOD ROBOT

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of

Master of Science

in

Engineering Sciences

by Weishu Zhan

Thayer School of Engineering
Dartmouth College
Hanover, New Hampshire

April 2023

Examining Committee:

Laura Ray Chair

Devin J. Balkcom

Minh Q. Phan

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies

Abstract

This thesis addresses hexapod robot motion control. Insect morphology and locomotion patterns inform the design of a robotic model, and motion control is achieved via trajectory planning and bio-inspired principles. Additionally, deep learning and multi-agent reinforcement learning are employed to train the robot motion control strategy with leg coordination achieves using a multi-agent deep reinforcement learning framework. The thesis makes the following contributions:

First, research on legged robots is synthesized, with a focus on hexapod robot motion control. Insect anatomy analysis informs the hexagonal robot body and three-joint single robotic leg design, which is assembled using SolidWorks. Different gaits are studied and compared, and robot leg kinematics are derived and experimentally verified, culminating in a three-legged gait for motion control.

Second, an animal-inspired approach employs a central pattern generator (CPG) control unit based on the Hopf oscillator, facilitating robot motion control in complex environments such as stable walking and climbing. The robot's motion process is quantitatively evaluated in terms of displacement change and body pitch angle.

Third, a value function decomposition algorithm, QPLEX, is applied to hexapod robot motion control. The QPLEX architecture treats each leg as a separate agent with local control modules,that are trained using reinforcement learning. QPLEX outperforms decentralized approaches, achieving coordinated rhythmic gaits and increased robustness on uneven terrain. The significant of terrain curriculum learning

is assessed, with QPLEX demonstrating superior stability and faster consequence.

The foot-end trajectory planning method enables robot motion control through inverse kinematic solutions but has limited generalization capabilities for diverse terrains. The animal-inspired CPG-based method offers a versatile control strategy but is constrained to core aspects. In contrast, the multi-agent deep reinforcement learning-based approach affords adaptable motion strategy adjustments, rendering it a superior control policy. These methods can be combined to develop a customized robot motion control policy for specific scenarios.

Preface

The author, Weishu Zhan, originally developed this thesis. The research was carried out at the Dartmouth Dynamics & Controls Lab.

Acknowledgements

During my master's journey and the initiation of my research, I am deeply grateful for the invaluable guidance and support provided by my advisor, colleagues, and friends.

I want to express my heartfelt gratitude to my advisor, Professor Laura Ray, for her exceptional guidance and unwavering support during my two years at Dartmouth. I vividly recall the excitement when Professor Ray welcomed me into her group. As I embarked on my first project, I gained invaluable knowledge. Subsequently, I collaborated with other talented individuals on various projects, with Professor Ray consistently supporting and assisting. I am grateful for her introduction to robotics and practical advice throughout my research. Her meticulous approach to research has profoundly influenced and inspired me to pursue excellence. And I would like to express my deepest gratitude to Professor Balkcom for his invaluable mentorship throughout my involvement in the soft robotics project. His expertise, guidance, and unwavering support have shaped my research and fostered my growth as a scholar in this fascinating field. Furthermore, I am incredibly grateful to Professor Phan for his exceptional, enriching and rewarding course. His profound knowledge, engaging

teaching style, and insightful advice have significantly contributed to my academic development and fueled my passion for the subject. I am truly honoured to have had the opportunity to learn from and work with these distinguished educators.

My time at Dartmouth College has been an unforgettable journey, and I am grateful for the friendships and connections I have made along the way. To my friends, thank you for your unwavering support, understanding, and shared memories.

I really enjoyed my life in Hanover, it's a beautiful little town. I can swim in the school gym in the hot summer, enjoy the beautiful forest in the fall, and ski in the winter.

Thanks so much for meeting Dartmouth!

Finally, I would like to express my profound appreciation to my family and my partner, whose love, support, and encouragement have been my foundation. Your belief in me has been my guiding light through the challenges and triumphs of this journey.

Thank you, once again, to all the amazing people who make my life better.

Contents

Chapter 1

Introduction

Section 1.1

Background and Significance

Throughout humanity's quest for knowledge and exploration of uncharted territories, numerous formidable and elusive domains have emerged, including disaster relief and interstellar exploration [1]. Supported by a confluence of academic institutions, medical and military departments, and interdisciplinary research, machine learning has made significant strides. Concomitant with the swift advancement of human technology, diverse robotic forms have been devised to address these challenges. Among these, legged robots have garnered considerable interest due to their robust weight-bearing capacity, exceptional stability, and adaptability to a variety of unstructured terrains, underscoring the paramount importance of researching legged locomotion. Legged robots' research primarily encompasses humanoid bipedal machines and multi-limbed robots, with the latter category further subdivided into quadrupeds, hexapods, and octopods. Hexapod robots, inspired by the innate attributes of six-legged insects, possess distinct advantages over their four- and eight-legged counterparts, including a more streamlined structure, enhanced stability through redundant legs, and com-

paratively straightforward motion control strategies [2].

In the realm of hexapod robot motion control, numerous scholars have achieved groundbreaking progress in areas such as structural design, gait planning, central pattern generation, and reinforcement learning. Reinforcement learning (RL) constitutes a crucial paradigm within machine learning, wherein the objective is to ascertain an optimal strategy that yields the maximum cumulative expected return through the training of an agent. The agent's responsibility is to optimize its policy for action-taking in order to maximize the cumulative expected return [3].

The deep reinforcement learning (DRL) model has been effectively translated to real-world applications, including the DRL-TRPO algorithm, among others. However, manual reward assignment remains extraordinarily arbitrary and convoluted, rendering replication virtually unattainable.when DRL is used to train a hexapod robot to walk on flat terrain, the outcomes on uneven terrain and varying inclines remain unpredictable [4]. Attaining a streamlined, reliable motion control strategy characterized by heightened flexibility, stability, and adaptability constitutes the crux of the hexapod robot motion control challenge.

This study commences with an examination of robot structural design, followed by gait planning, biomimetic motion control, and machine learning methodologies to facilitate the motion control of hexapod robots [5]. The efficacy of each motion control strategy is substantiated through simulation. By establishing diverse terrain configurations, the merits and drawbacks of each motion control strategy are meticulously analyzed and contrasted.

Section 1.2

Research Status

During the 1980s, the Massachusetts Institute of Technology pioneered the development of Genghis [6], an advanced hexapod robot designed for extraterrestrial exploration. As depicted in Fig 1.1, the apparatus employs a feedback motor to drive the servo, facilitating real-time adjustments in joint torque. The robot amalgamates components that gauge current values, thereby optimizing movement across intricate terrain. Owing to technological advancements and perpetual progress, a myriad of sophisticated walking robots have emerged, including NASA's hexapod robot ATHLETE [7], showcased in Fig 1.2. This formidable, 13-foot machine is capable of remote maneuvering, jumping, and even dancing. Designed to transport cumbersome objects and traverse slopes with substantial inclines, ATHLETE's six omnidirectional loads harmoniously collaborate to ensure seamless motion.

Fig 1.3 illustrates SILO-6, a hexapod robot devised by the Spanish Institute of Dynamics. Principally composed of sensors, a manipulator, and a positioner, the robot and its controller are partitioned into five distinct sections. The manipulator is specifically engineered to carry a mine detector and a positioning system, encompassing an electromagnetic compass, a GPS antenna, and a Wi-Fi antenna, all of which are remotely governed by a computer [8].

Lauron, a creation of the University of Karlsruhe in Germany, utilizes a behavior-based, flexible control system [9], as exhibited in Fig 1.4. Exceptionally adept at adapting to unfamiliar circumstances, Lauron is equipped with a panoramic camera on its rear, furnishing operators with a comprehensive overview of both the robot and its surroundings. Capable of autonomously gathering environmental data and devising a path towards a predetermined objective, Lauron can effortlessly circumnavigate

any obstacles deemed excessively tall.

Figure 1.1: Genghis robot

Figure 1.2: ATHLETE robot

Figure 1.3: SILO-6 robot

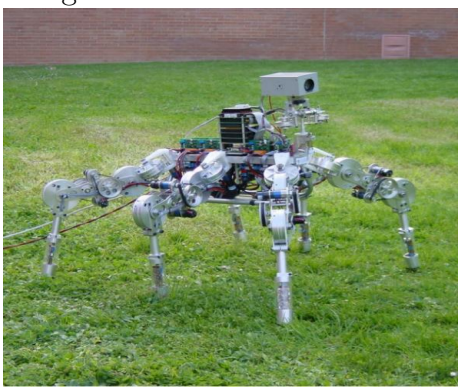


Figure 1.4: LAURON-III robot

In recent years, robotics research has developed rapidly. Spot/Spot mini, a quadruped robot dog developed by Boston Dynamics, as shown in Figure 1.5, weighs 30kg and can balance dynamically in uncertain surroundings with payloads of up to 14kg. Spot can achieve 3-axis attitude movement and 3-axis position complete control. Spot has a 3D vision system for simultaneous localization and mapping (SLAM), providing depth information, enabling the robot its surroundings, and avoiding obstacles.Spot's built-in computers are fully dedicated to robot to assess locomotion and navigation. The robot is remotely controlled by human operators while also navigating and performing some tasks autonomously.It can cruise over loose gravel, grass, curbs, and stairs, and reasonably coordinate the movements of the limbs [10].

Based on the review, it is clear that the last decade has moved the area of legged

robots to a new level, and at the same time, has opened a new research area for fundamental and topics with new opportunities. However, the traditional gait curve of the cycloid, which is employed to adjust Step Length Ratio(SLR) rate, cannot perfectly solve the slip problems associated with automatically adapt the angle of foot-end attack and leads to static friction forces within a large bound.

Figure 1.5: Spot

Section 1.3

Research Status of Hexapod

Since 1968, Mcghee and Frank [11] have pointed out the fluctuating gait of quadruped robots. An increasing number of people are focused on hexapod robots.

Research usually uses fixed gait methods, such as three-legged gait, quadrupedal gait, fluctuating gait, and so on.

Due to its high degree of freedom and high structural redundancy, the control process of hexapod robots is complicated. The control methods of hexapod robots can be roughly divided into the following types as described here.

1.3.1. Behavior-Based Motion Control Methods

Drawing from German scientist Cruse's investigations on stick insects, six fundamental principles of leg movement have been posited [12, 13]. A controller premised on

the "control-reflection" concept is designed to devise distinct motion actions for the robot initially. Each action comprises an array of sensor inputs and corresponding actuator outputs. Upon gathering sensor data, analysis and processing ensue to propel the robot toward executing the appropriate action. Nonetheless, robots adhering to this approach, exemplified by Brooks' "Genghis" struggle to develop a comprehensive action repertoire that accurately represents all terrain environments. The intricate and mutable nature of terrains renders the robot's generalization capabilities on unstructured ground rather inadequate.

1.3.2. A motion control method for a central pattern generator

This motion control method performs robot motion control by simulating the biological central nervous system, and the central pattern generator performs rhythmic motion control by generating periodic oscillating signals. CPG has three neurons: inhibition, excitation, and terminal inhibition. The neurons inhibit each other through the network structure, and phase interlocks are created to output signals that control the movement of the organism. The central mode generator can be coupled with external input, and various modal outputs can be achieved by adjusting the parameters of the oscillating signal [14, 15].

Venkataraman [16] first studied the role of the central pattern generator in control, and added a delay in the input and output of the control system to achieve a simple gait.

Barron-Zambrano et al. [17] proposed a CPG-based control strategy, which can adjust the motion speed according to visual information and manage the smooth gait transition in the hexapod robot.

Yu et al. [18] realized three gait switching of a hexapod robot by adjusting the model parameters of the central pattern generator (CPG). The rhythmic gait can realize the movement of regular terrain, but cannot adapt to rough terrain.

1.3.3. Motion control method based on deep reinforcement learning

This motion control method combines the perception ability of deep learning and the decision-making evaluation ability of reinforcement learning. Through the robot's perception of the environment, the training generates an adaptive motion control method to improve the generalization ability of the robot control strategy in different environments.

Vassilios et al. [19] use the accessibility evaluation criterion to replace the physical simulation to build the Markov decision process. They plan the trajectory of the robot in the high-dimensional continuous state-action space, control the robot shutdown to follow the trajectory, and finally, the robot realizes the stable motion in different environments.

Huaqiao Fu et al. [20] take the uneven plum-blossom piles used in the experimental environment, preset random start, and target fields. Using a deep reinforcement learning algorithm for training and learning that obtain the motion control strategy of the hexapod robot in the plum blossom pile environment. They realize the DDPG algorithm with a priority replay mechanism to speed up the training process.

Teymur Azayev [21] presents a scalable two-layer architecture for hexapod locomotion through complex terrain without the use of exteroceptive sensors. Their approach assumes that the target complex terrain can be modeled decomposed into N discrete terrain distributions that capture the individual difficulties of the target terrain. Expert strategies (physical motion controllers) modeled by Artificial Neural Networks were trained to use deep reinforcement learning independently in these individual scenarios. These strategies are then automatically multiplexed when reasoning with a recursive neural network terrain classifier. The architecture provide adaptive gaits appropriate to the current terrain based on state history, as well as evaluating the robustness and actuator properties of the strategies by varying various

parameters, such as contact, friction.

But there are still some shortcomings, such as having to combine trained experts through a multiplexer strategy. Another disadvantage is the need to manually program a terrain distribution generator.

In this thesis, we will improve on these two problems by exploring the appropriate looping architecture and training algorithms to train a looping strategy that is as effective as possible on all terrains.

Section 1.4

Main Topic of Research

This thesis commences with an examination and exposition of the current state of legged robotics, followed by the design of a robot body structure inspired by insects. The gait characteristics of the robot are then delineated. Approaching the subject from a gait planning perspective, the thesis analyzes and designs robot motion control methodologies. Subsequently, by emulating the rhythmic movement traits of animals, a robot motion control approach is devised through the lens of central pattern generation. Lastly, by delving into multi-agent reinforcement learning(MARL), an intelligent system premised on the QPLEX architecture is established. Within a simulation environment, the intelligent system undergoes training through continuous interaction with the robot, culminating in end-to-end robot motion control. The principal contents of this thesis are organized as follows:

Chapter two. The hexapod robot's body structure, employed in subsequent simulations, is devised by emulating the anatomical composition of insects. By investigating the nuances of insect locomotion, the gait classifications of the hexapod robot are segregated, and the motion trajectory of the robot's leg extremities is analyzed

and delineated. A toe motion trajectory planning method is designed, followed by an examination of the relationship between the robot leg displacement and joint rotation angle from the perspective of hexapod robot kinematics. The forward and inverse kinematic solutions for the robot legs are derived. Ultimately, robot motion control is achieved by integrating the trajectory planning of the robot leg extremities and the inverse kinematic solution for the robot legs.

Chapter three. Initially, we outline the animal motion control methodology and emulate its core module. Subsequently, a Hopf oscillator-based robotic motion control unit is designed and refined for hexapod robots. This enhanced oscillator facilitates diverse gait movements using a CPG oscillator unit. By constructing a CPG network and devising a joint mapping function, we actualize motion control for the hexapod robot.

Chapter four. We investigate a sophisticated robotic motion control technique utilizing deep reinforcement learning (DRL). We initially present the theoretical underpinnings of MARL, encompassing mathematical models of reinforcement learning, prevalent algorithms, and deep learning architectures inspired by human neural networks. Subsequently, we establish pertinent state values, action values, and reward functions for hexapod robot control via the DRL system. A module is devised in a simulated environment to ascertain these values and functions, followed by the construction of an intelligent system based on QPLEX. Robotic motion control is actualized through training and learning, incorporating an early termination scheme to expedite learning by preemptively concluding the current round when the robot encounters precarious learning states, thereby enhancing efficiency. We subsequently compare QPLEX to a fully-decentralized structure to assess the controllers' adaptability in uneven terrain conditions. Lastly, the significance of terrain path learning

is evaluated.

Chapter five. The robot motion planning methods are analyzed and compared regarding robot foot trajectory planning, CPG rhythm control, and deep reinforcement learning.

Chapter 2

Kinematics Analysis

In this chapter, we construct a hexapod robotic model, emulating the intricate architecture of insects and the framework of a standard hexapod automaton. This model serves as a foundation for ensuing simulation explorations. A comprehensive examination of the robot's ambulatory patterns and appendage dynamics is conducted. Moreover, the overarching locomotive control mechanisms are considered and substantiated through the lens of robotic extremity trajectory following.

Section 2.1

Structure Introduction

The hexapod's mechanical configuration is inspired by bionics. The primary focus of the body design is centered on selecting the six legs and determining their relative positions while fully considering the robot's leg stability and coordination during motion.The structure design of the hexapod robot body geometry is presented in Fig 2.1. The robot's body is rectangular in shape, with a length of 2 meters and a width of 0.5 meters. The legs are attached to the body at 135-degree angles from each of the four corners, with the leg connection measuring 0.5 meters in length. The body is bifurcated into two layers at the top and bottom, with a height of 0.5 meters. The

hexapod robot's 3D model was constructed using 3D modeling software, as illustrated in Fig 2.2.

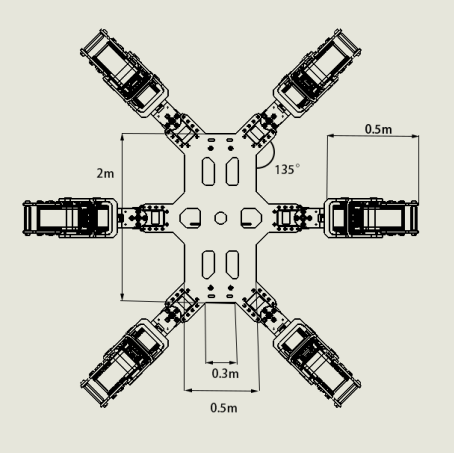


Figure 2.1: Fuselage sketch

Figure 2.2: Fuselage structure

The bionic-inspired mechanical framework of this hexapod emphasizes the strategic selection of six appendages and the determination of their relative positioning. This consideration ensures the robot's legged stability and coordination during locomotion. As depicted in Fig 2.1, the hexapod robot's geometric configuration consists of a rectangular top-down view of the chassis, from which the sextet of limbs extends. With dimensions of 2 meters in length and 0.5 meters in width, the robot's body exhibits 135-degree angles between its four corner-mounted legs and the chassis, each leg featuring a 0.5-meter connecting segment. Comprising dual layers—upper and lower—the body's height measures 0.5 meters. Utilizing 3D modeling software, the hexapod robot's three-dimensional body structure is illustrated in Fig 2.2.

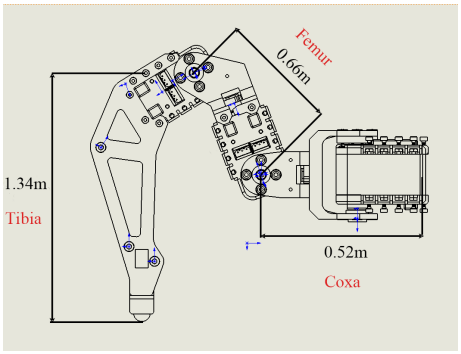


Figure 2.3: Single leg sketch

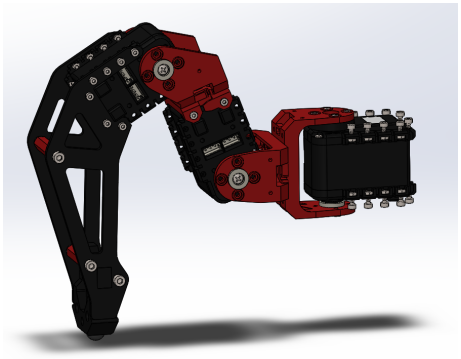


Figure 2.4: Single leg

Upon completing the design of the hexapod robot's body and individual leg structure, six iterations of the leg assembly are affixed to the chassis', culminating in the robot's comprehensive assembly. The assembled robot is depicted in Fig 2.5 and Fig 2.6.

Figure 2.5: Hexapod robot structure

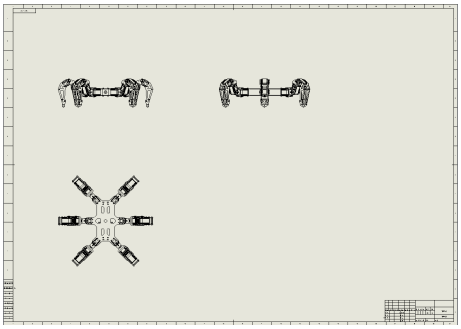


Figure 2.6: Hexapod robot three view

This study accomplishes the comprehensive design of the hexapod robot's structural model using SolidWorks. A SolidWorks and Matlab/Simulink plugin achieve integration with the Simulink simulation environment. This simulated environment facilitates subsequent motion control simulations for the robot.

Section 2.2

Swing Leg Structure Position Analysis

During the walking process, the leg belongs to a tandem structure. We need to identify the relationship between the steering motor's angle and each leg end's position.

2.2.1. Single Leg Analysis

As previously stated, the hexapod robot's body structure design is inspired by the animal form, with the motion process derived from the animal locomotion process. Consequently, adopting a motion control design that emulates the animal locomotion process is a viable approach. Initially, a locomotion trajectory is planned for the robot, followed by the identification of the robot's six footfall points using gait planning techniques. Subsequently, the relationship between the robot's toe displacement and leg joint angles is determined through kinematic analysis, ultimately enabling robotic motion control. The leg kinematic analysis can be conducted based on the robot's leg articulations, as the position of the proximal joint near the robot's body is predetermined. Thus, the leg kinematic analysis can reference the robot's leg joint coordinates.

The kinematic analysis of the robot leg can be interpreted in two directions: The first is to solve the leg toe coordinates in the body coordinate system with known leg joint angles and body position information. This is the inverse kinematic solution of the robot. The robot kinematic solution is modeled by the D-H method, and the steps are as follows:

(1) Establish the coordinate system of each joint with the robot geometry.

(2) Determine the transformation parameters between the coordinate systems, including link length, angle, etc.

(3) Determine the transformation matrix between the coordinate systems.

(4) Multiply the transformation matrices of all coordinate systems in (3) to obtain the transformation matrix from the toe of the leg.

(5) Finally, find the robot leg kinematic solution.

For the single-legged kinematic forward and inverse solutions of the robot to be analyzed, the coordinate system of each joint is first established according to the above steps as shown in Figure 2.7. The transformation relationships between the coordinates are shown in Table 2.1 below.

	θ_i	d_i	a_i	α_i
L1	θ_1	d_1=0	a_1=0.52	α_1=90
L2	θ_2	d_2=0	a_2=0.66	α_2=0
L3	θ_3	d_3=0	a_3=1.34	α_3=0

Table 2.1: Single leg D-H Parameter table

In the table, θ_i is the angle between the i - 1 coordinate system and the x-axis of the i coordinate system. d_i is the distance between the i - 1 coordinate system and the x-axis of the i coordinate system. a_i is the distance between the i - 1 coordinate system and the z-axis of the i coordinate system. α_1 is the angle between the i - 1 coordinate system and the z-axis of the i-coordinate system.

According to step 3 of the D-H method, the transformation matrices between coordinate systems are as follows: first, four basic transformation matrices are determined, including translation matrix 2.1, z-axis rotation matrix 2.2 , y-axis rotation matrix 2.3, x-axis rotation matrix 2.4

$$Trans(a,b,c) = \begin{pmatrix} 0 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

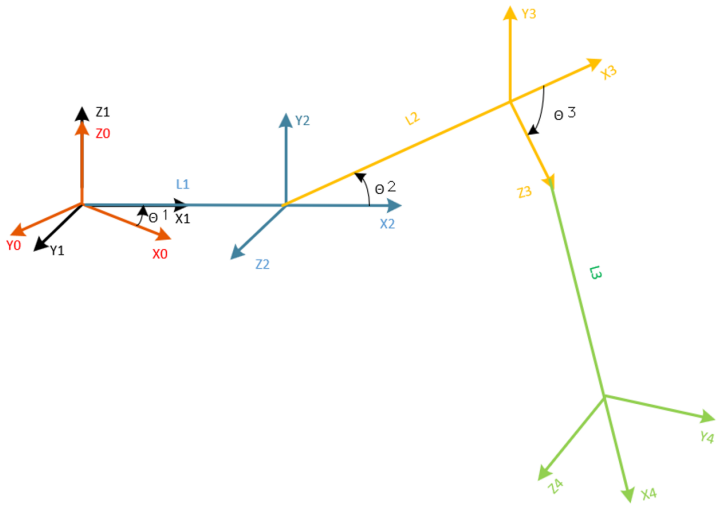


Figure 2.7: Robot single leg coordinate system

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

According to the D-H method, the transformation matrix between the coordinates are as follows:

$$ {}^{i-1}_{i}T = R_z(\theta)Trans(0,0,d_i)Trans(a_i,0,0)R_x(\alpha_i) \tag{2.5} $$

$$ {}^{0}_{1}T = R_z(\theta)Trans(0,0,d_1)Trans(a_1,0,0)R_x(\alpha_1) \tag{2.6} $$

$$ {}^{1}_{2}T = R_z(\theta)Trans(0,0,d_2)Trans(a_2,0,0)R_x(\alpha_2) \tag{2.7} $$

$$ {}^{2}_{3}T = R_z(\theta)Trans(0,0,d_3)Trans(a_3,0,0)R_x(\alpha_3) \tag{2.8} $$

Finally, the position transformation matrix of the single-legged foot end of the robot is obtained as:

$$ {}^{0}_{3}T = {}^{0}_{1}T^{1}_{2}T^{2}_{3}T \tag{2.9} $$

Forward kinematics Solution: If P=(x,y,z,1) is the coordinate of the foot end in some joint coordinate system, then according to the previous transformation relation:$P = {}^{0}_{1}T^{1}_{2}T^{2}_{3}T(0,0,0,1)^T$ brought into the previous equation then:

$x = a_1\cos\theta_1 + a_2\cos\theta_1\cos\theta_2 + a_3\cos\theta_1\cos\theta_2\cos\theta_3 - a_3\cos\theta_1\sin\theta_2\sin\theta_3$

$y = a_1\sin\theta_1 + a_2\sin\theta_1\cos\theta_2 + a_3\sin\theta_1\cos\theta_2\cos\theta_3 - a_3\sin\theta_1\sin\theta_2\sin\theta_3$

$z = a_1\sin\theta_2 + a_2\sin\theta_2\cos\theta_3 + a_3\cos\theta_2\sin\theta 1_3$

2.2.2. Single leg forward kinematics MATLAB simulation

The Robotics Toolbox in MATLAB provides the forward kinematics simulation function Fkine () to establish the single-leg model of the hexapod robot, and the joint angle can be given to obtain the end pose. By presetting the value of joint Angle($\theta_1\theta_2\theta_3$) is :$[\frac{\pi}{6}\ \frac{\pi}{6}\ -\frac{\pi}{4}]$,$[-\frac{\pi}{4}\ \frac{\pi}{4}\ \frac{-5\pi}{12}]$,$[-\frac{\pi}{4}\ \frac{\pi}{12}\ -\frac{2\pi}{3}]$,[-$\frac{\pi}{12}\ \frac{\pi}{4}\ -\frac{\pi}{2}$].According to the preceding, the If we set the coordinates of the head of each standard coordinate system according to $P_0P_1P_2P_3$, we can find

$$P_0 = \{0, 0, 0, 1\}$$

$$P_1 = {}^0_1T\{0, 0, 0, 1\}^T$$

$$P_2 = {}^0_1T^1_2T\{0, 0, 0, 1\}^T$$

$$P_3 = {}^0_1T^1_2T^2_3T\{0, 0, 0, 1\}^T$$

Then we can calculate the coordinates of the origin of those mentioned above in the joint coordinate system, using P_0P_1, P_1P_2 and P_2P_3 as the coordinates of the two ends of the connecting rod, and build a simple model by Matlab, as shown in Figure 2.8(a)(b)(c)(d), respectively.

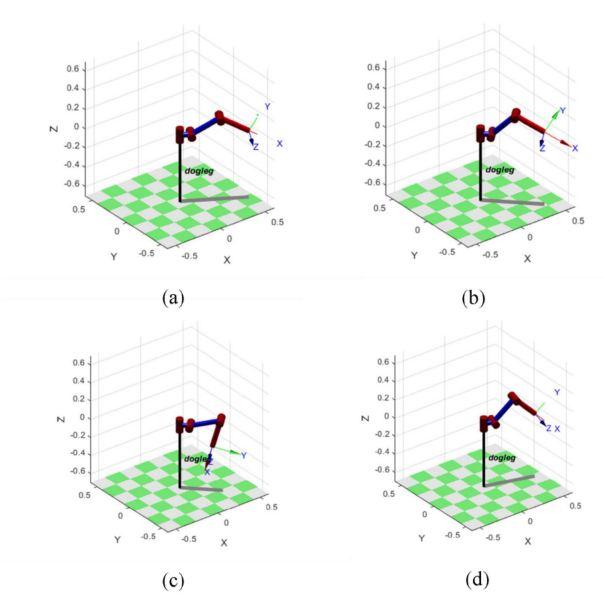


Figure 2.8: Single-leg forward kinematics simulation results

From Figure 2.8, it can be seen that the angle is the same as the preset angle, which proves that the robot positive kinematics solution is correct.

2.2.3. Inverse Kinematics Analysis

A transformation

$$ {}^{0}_{1}T^{-1}P = {}^{1}_{2}T^{2}_{3}T(0,0,0,1)^{T} \tag{2.10} $$

$$ {}^{0}_{1}T^{-1}{}^{1}_{2}T^{-1}P = {}^{2}_{3}T(0,0,0,1)^{T} \tag{2.11} $$

The kinematic inverse solution equation is obtained as:

$$ \theta_1 = \arctan(\frac{y}{x}) \tag{2.12} $$

$$ \theta_3 = \arccos\frac{M^2 + z^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{2.13} $$

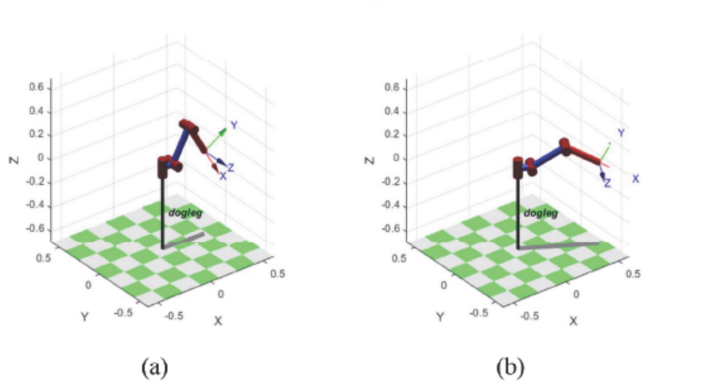
$$ \theta_2 = (\frac{za_3\cos\theta_3 + za_2 - Ma_3\sin\theta_3}{M^2 + z^2}) \tag{2.14} $$

$M = -a_1 + x\cos\theta_1 + y\sin\theta_1$

2.2.4. Single leg inverse kinematics MATLAB solution

The aforementioned derivation demonstrates that determining the inverse solution for the robot's kinematic equations constitutes a nonlinear problem. Two approaches can address this issue: the closed-form solution method and the numerical solution method. The MATLAB Robotics Toolbox offers the function ikunc() to solve the inverse kinematics problem using the numerical solution method. Employing the optimization method, this function conducts the inverse kinematics computation by providing the position transformation matrix, and subsequently deriving the angle of each joint in the individual leg.

Given the coordinates of the single-leg foot end (0.4, 0, 0),(0.566, -0.3098, 0) $\theta_1\theta_2\theta_3$ is obtained according to the equation above, and then the coordinate origin of each joint is determined, and then the model is built by Matlab Figure 2.9, from which it can be seen that the coordinates of the single-leg end are the same as the preset coordinates, indicating that the robot above inverse kinematics solution is correct.



(a) (b)

Figure 2.9: Single-leg inverse kinematics simulation results

Section 2.3

Gait Planning and Simulation

Drawing from the research presented in the article [22], it is evident that the interaction among the hexapod insect's six legs exhibits a specific pattern during the locomotion process. This systematic walking process is referred to as a robotic motion gait.

2.3.1. Gait parameter determination

The robot's leg moves in regular periodic motions when it is walking. This regularity of robot motion gait is quantitatively analyzed through the determination of gait parameters to distinguish different gait patterns. For the robot legs, its motion

process is mainly divided into swing state and support phase, and the two processes alternately complete the whole motion process. One condition is the swing phase. This phase includes leg lift up, forward swing and fall down on the ground [23]. The other state is the support phase, which means the leg contacts the ground, the leg loads the robot capacity and swings back, the legs support the robot until the swing phase and lift up off of the ground. Defining β was a parameter to describe the ratio of time spent in one motion cycle by the single-leg support phase of the robot give:

$$\beta = \frac{t_{support}}{T} = 1 - \frac{t_{swing}}{T} \tag{2.15}$$

where $t_{support}$ is the robot single-leg support phase duration, $t_{support}$ is the robot single-leg swing phase duration, T is the robot is the duration of one motion cycle of the robot, $\mathrm{T} = t_{support} + t_{swing}$.

The distance of translation of the centre of gravity of the multi-legged robot during a complete gait cycle T as Stride length, is denoted by λ.

The distance the foot end of the walking leg moves relative to the robot body during the support phase as the stroke, is denoted by R. For regular gait, the stroke R is related to the step length λ as follows:

$$R = \beta\lambda \tag{2.16}$$

The robot occupancy factor is defined as δ. It is intuitively clear that the more legs the robot has simultaneously in the support phase then the more stable the robot is, thus:

$$\delta = n * \beta \tag{2.17}$$

where n is the number of robot legs, which is 6 for a hexapod robot.

The robot forward velocity is defined as v, and let the robot motion step length

is s. Then

$$v = \frac{s}{t_{support}} \tag{2.18}$$

Combing 2.15 and 2.18, then

$$v = \frac{s}{t_{swing}}(\frac{1}{\beta} - 1) \tag{2.19}$$

Given that the robot's motion stride length s remains constant, a smaller β results in a faster yet less stable hexapod robot, while a larger β corresponds to longer standing legs and increased stability, which aligns with intuition. Consequently, robotic motion control necessitates a decision regarding the balance between stability and speed, highlighting the presence of a trade-off between these two factors.

2.3.2. Trajectory Planning Base on the Typical Gait

The investigation revealed that hexapod insects exhibit three prevalent gait patterns, specifically tripod, quadruped, and fluctuating gaits. Commencing from the robot's top-left corner, the six legs are sequentially arranged as (L1, L2, L3, L4, L5, L6).

Tripod gait.

The tripod gait divides the robot's six legs into two groups: Group A (L1, L4, L5) and Group B (L2, L3, L6). Each leg within the same group shares an identical motion process, and the two groups of legs exhibit interlocking phases. While one group is in the swing phase, the other is in the support phase. As one group transitions from the support phase to the swing phase, the other group simultaneously moves from the swing phase to the support phase, and the two groups alternate their movement. The robot's motion is achieved through the continuous and repeated alternating movement of these two leg sets. According to the previously defined parameters, the robot employs a tripod gait with β set at 0.5.

Assume the hexapod robot advances with a step of λ, Figure 2.10 illustrates the process of executing the hexapod robot's tripod gait. It also presents the positional changes of each joint in the robot's top view, following the subsequent sequence of actions:

a) The hexapod robot is in its initial position, with both Group A and Group B's walking legs in the support phase.

b) Group A's walking legs remain in the support phase, facilitating the robot body's parallel translation by $\frac{\lambda}{2}$ relative to the support surface, while Group B's walking legs enter the swing phase.

c) Group B's walking legs transition to the support phase upon making contact with the ground.

d) Group B's walking legs, now in the support phase, enable the robot body to translate $\frac{\lambda}{2}$ parallel to the support surface, while Group A's walking legs shift to the swing phase.

In Fig 2.10d), the first set of walking legs transitions to the support phase, returning the hexapod robot to its initial position. The hexapod robot's first and second walking leg groups repeatedly cycle through the sequence of actions from Figure 2.10a) to Figure 2.10d), ultimately achieving a tripod gait with a stride length (s) and advancing in a straight line.

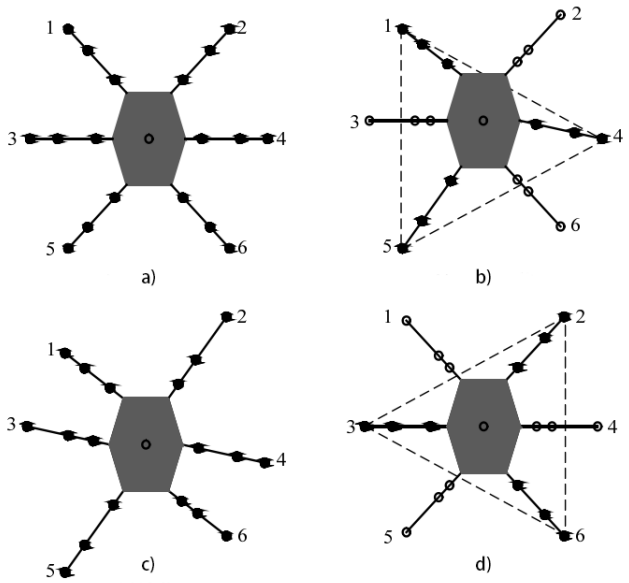


Figure 2.10: Diagram of tripod gait(three-legged gait)

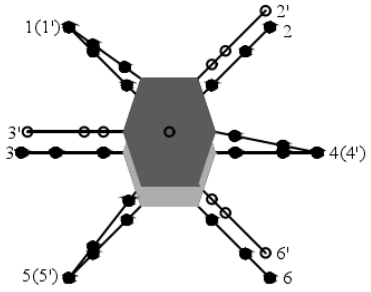


Figure 2.11: Schematic diagram of tripod gait robot translation

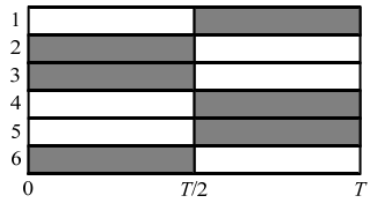


Figure 2.12: Tripod gait phase diagram

Quadrupedal gait.

The quadrupedal gait organizes the robot's six legs into three groups: Group A (L1, R2), Group B (L2, R1), and Group C (L3, R1). When one group's legs are in the swing phase, the remaining two groups' legs occupy the support phase, with all three groups interlocking. The three leg sets alternate movement to complete the robot's motion, with a β value of 0.67 for the quadrupedal gait. The planned and executed

quadrupedal gait is depicted in Figure 2.13, following the specific action sequence outlined below:

a) The hexapod robot is in its initial position, with the walking legs of Groups A, B, and C all in the support phase.

b) As Group A's walking leg is raised, the walking legs of Groups B and C serve as support phases, allowing the robot's body to move forward, parallel to the support surface by $\frac{\lambda}{3}$.

c) Group A's walking leg transitions to the support phase after traversing a distance R forward.

d) With Group B's walking leg raised, the robot's body is supported by the walking legs of Groups A and C in the support phase, advancing parallel to the support surface by $\frac{\lambda}{3}$.

e) Group B's walking leg, having spanned a distance R forward, shifts to the support phase.

f) As Group C's walking leg is raised, the robot's body is supported by the walking legs of Groups A and B in the support phase, progressing parallel to the support surface by $\frac{\lambda}{3}$.

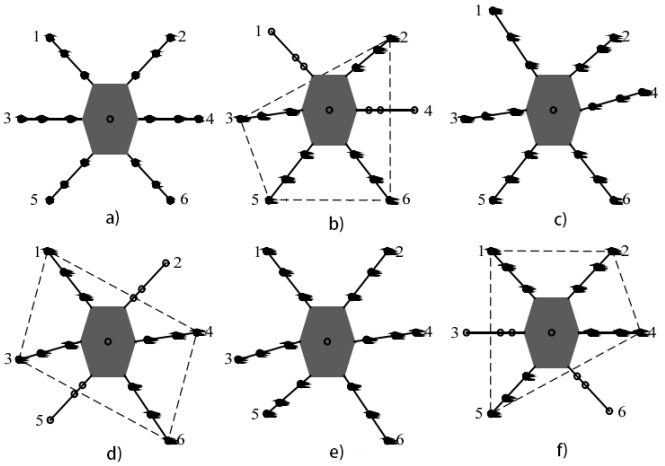


Figure 2.13: Quadrupedal gait diagram

The states illustrated in Figure 2.13c) and 2.13e) persist for only an instant, which can be disregarded. Consequently, during the gait cycle T, only four walking legs are in the support phase at any given moment. Figure 2.15 displays the phase diagram for a single gait cycle T of the quadrupedal gait. As per Figure 2.15, the time spent in the support phase and swing phase for any walking leg during the gait cycle T is $\frac{2T}{3}$ and $\frac{T}{3}$, respectively.

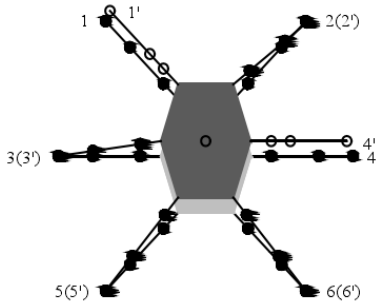


Figure 2.14: Quadrupedal gait robot translation diagram

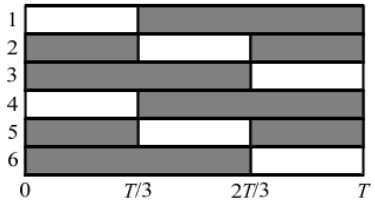


Figure 2.15: Quadrupedal gait phase diagram

Fluctuating gait.

In the fluctuating gait, the six legs of the robot are grouped individually. As the first leg transitions from the swing phase to the support phase, the subsequent leg shifts from the support phase to the swing phase. The six legs move independently and sequentially, repeatedly completing the robot's motion process. The fluctuating gait motion has a β value of 0.83. Figure 2.17 portrays the hexapod robot's implementation of the fluctuating gait, following the sequence of actions below:

a) The hexapod robot is in its initial position, with all walking legs in the support phase.

b) Leg 1 transitions from the support phase to the swing phase, while the remaining five legs support the body as it moves forward, allowing the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

c) Leg 1 steps forward by $\frac{5\lambda}{6}$ and lands in the support phase.

d) Leg 6 shifts from the support phase to the swing phase, while the remaining five legs support the body as it moves forward, enabling the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

e) Leg 6 steps forward by $\frac{2\lambda}{3}$ and lands in the support phase.

f) Leg 4 moves from the support phase to the swing phase, while the remaining five legs support the body as it advances, allowing the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

g) Leg 4 steps forward by $\frac{\lambda}{2}$ and lands in the support phase.

h) Leg 2 transitions from the support phase to the swing phase, while the remaining five legs support the body as it moves forward, enabling the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

i) Leg 2 steps forward by $\frac{\lambda}{3}$ and lands in the support phase.

j) Leg 5 moves from the support phase to the swing phase, while the remaining five legs support the body as it advances, allowing the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

k) Leg 5 steps forward by $\frac{\lambda}{6}$ and lands in the support phase.

m) Leg 3 shifts from the support phase to the swing phase, while the remaining five legs support the body as it moves forward, allowing the body's center of gravity to translate $\frac{\lambda}{6}$ parallel to the support surface.

Figure 2.18 displays the phase diagram of a single gait cycle T for the fluctuating gait. As shown in Figure 2.18, during gait cycle T, any walking leg's support and swing phases occupy $\frac{5T}{6}$ and $\frac{T}{6}$, respectively.

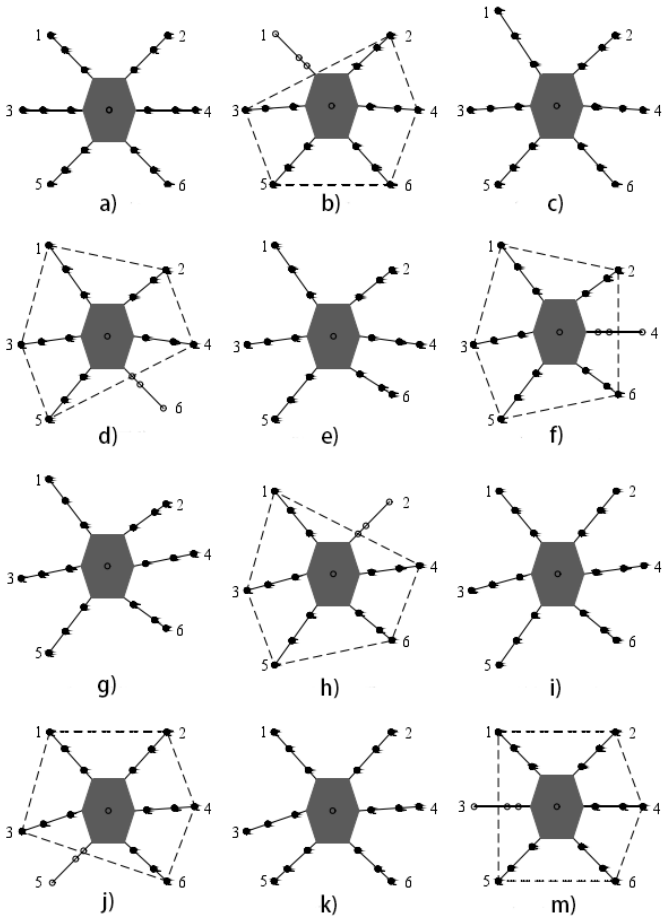


Figure 2.16: Diagram of fluctuating gait

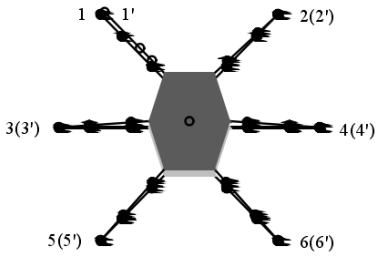


Figure 2.17: Fluctuating gait robot translation diagram

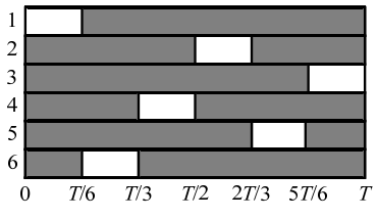


Figure 2.18: Fluctuating gait phase diagram

During the robot's motion, the phase difference between the legs determines the gait in which the robot moves and the robot's legs are lifted and dropped in sequence

according to the established phase to achieve motion.The three gait patterns are compared visually in the following table 2.2.Different gait patterns are adjusted in the different terrain environment. Three-legged gait suitable for flat ground, and when the ground is more rugged, the quadrupedal gait or the fluctuating gait is adjusted.

Type	β	δ	v
Tripod gait	0.5	3	Fast
Quadrupedal gait	0.67	4	Median
Fluctuating gait	0.83	5	Slow

Table 2.2: Gait comparison table

Section 2.4

Trajectory planning for the hexapod robot

In the aforementioned work, the mechanical structure of the robot has been designed, and the prevalent gait patterns of the hexapod robot have been analyzed. The support and swing phases of the robot's legs form the foundation of its motion. The swing phase dictates the stride length when the robot walks forward and the elevation of the lifted leg, ensuring the foot end can surmount obstacles and swing towards the target point. Numerous researchers have proposed various trajectory planning methods, including the circular arc model, elliptical trajectory, cubic curve fitting, polynomial interpolation, and curve fitting. This section accomplishes the foot-end trajectory planning of the robot when walking in a straight line on structured ground, which will be utilized in subsequent robot motion control analysis.

2.4.1. Single leg foot trajectory analysis

To ensure efficient and stable movement for the single-leg robot, the foot-end trajectory should be carefully planned, taking into consideration the support and swing phases of the leg. As illustrated in Figure 2.19, during the support phase, the body moves forward, and the foot-end trajectory is a straight line. The foot-end moves backward relative to the body. In the swing phase, the foot-end trajectory forms a semi-circular curve as the single leg moves forward.

When the single leg is in the swing phase, it is slowly raised until it reaches its highest point, with the angle between the robot's coxa and body being zero. The single leg then continues to move forward, with leg extension occurring. Throughout the robot's walking process, the foot-end repeatedly follows this trajectory to ensure the robot's forward motion.

To accurately achieve foot placement while clearing obstacles, the foot-end trajectory should meet the following criteria:

a) The robot should have minimal shaking and fluctuation during the leg-lifting process, ensuring smooth motion.

b) The hexapod robot's joints should not experience significant impact when the leg is lifted during the swing phase or when it lands.

c) The hexapod robot's swing leg should be capable of rapid lifting and landing, with continuous speed and acceleration at each joint, avoiding abrupt changes or sharp points.

By adhering to these criteria, the robot's foot-end trajectory can be optimized for efficient and stable movement, making it well-suited for various environments and terrains.

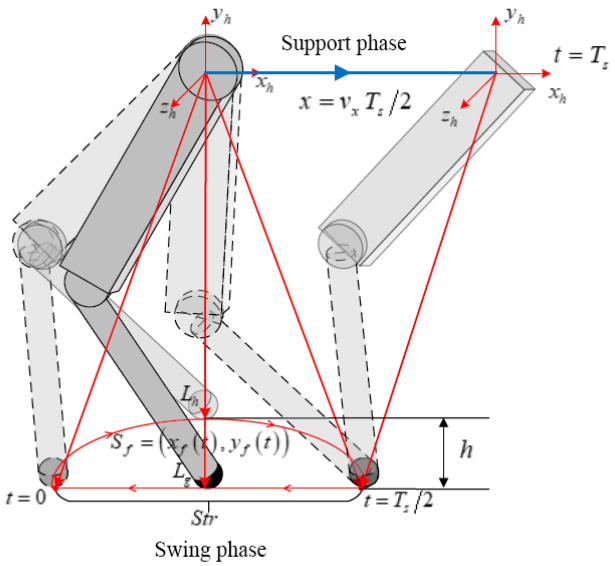


Figure 2.19: Single-leg foot end trajectory

2.4.2. Single-leg foot-end trajectory planning

Circular cycloid trajectory planning.

Figure 2.19 illustrates the devised trajectory, as delineated by the robotic hip coordinate system H, encompassing three salient components: initial point coordinates, targeted landing point coordinates, and leg elevation height. Notably, the leg elevation height can be modulated in various scenarios, exhibiting adaptability in accordance with terrain characteristics. The targeted landing point coordinates of the swing trajectory significantly influence the stride length, consequently affecting the robot's locomotion velocity and terrain adaptability proficiency. To ensure the formulated swing trajectory remains responsive to the robot's traveling speed, Raibert's methodology is employed in computing the horizontal coordinates of the desired landing point.

$$\begin{cases} x_f = \dfrac{1}{2}\dot{x}T_s t - k_{vx}(\dot{x}_d - \dot{x}) \\ z_f = \dfrac{1}{2}\dot{z}T_s t - k_{vx}(\dot{z}_d - \dot{z}) \end{cases} \tag{2.20}$$

$\dot{x}$ and $\dot{z}$ are the lateral and forward velocities of the robot body during the leg lift transient,T_{st} is the duration of the support phase, k_{vx} and k_{vz} are the velocity difference gain coefficients, and $\dot{x}_d$ and z_d are the desired motion velocities of the robot during the leg lift transient. Ts can be set as a constant (fixed step motion). When Ts becomes stable, the expected landing point coordinates calculated by using Equation 2.20 will change with the change of the robot's real-time speed, i.e., the larger the speed, the larger the step, the smaller the speed, the smaller the step. The speed is zero when the robot is stepping in place.

After determining the three characteristic points of the trajectory, the coordinates of all points of the swing trajectory can be planned based on the interpolation method. More intermediate points can be flexibly added to the swing trajectory so that the swing trajectory can be planned for special terrain.

The quantitative planning of the robot's single-legged foot-end trajectory is executed by establishing a mathematical model representing the single-legged foot-end trajectory, defined as:

$$\begin{cases} x_{sw} = (x_f - x_{10})\,\frac{(\varphi - \sin\varphi)}{2\pi} + x_{10} \\ z_{sw} = (z_f - z_{10})\,\frac{(\varphi - \sin\varphi)}{2\pi} + z_{10} \\ y_{sw} = \Delta h_d \frac{(1-\cos\varphi)}{2} + y_{10} \end{cases} \tag{2.21}$$

Here, Δh_d denotes the desired leg elevation height (peak height of the curve arc), and:

$$\varphi = \frac{2\pi t}{T_{\mathrm{SW}}}$$

Where T_{sw} signifies the swing phase duration (for a trot gait with $\rho = 0.5$, $T_{sw} =$

$T_{st} = 0.5T$, representing the gait period), and $t \in [0, T_{sw}]$ corresponds to the real-time duration of the swing motion, resetting to 0 with each leg elevation.

Assume a robot motion step length of 20 and leg elevation height of 10. Taking a three-legged gait as an example, the robot's single leg alternates between support and swing phases for 1 second each. As illustrated in Figure 2.20, the planned robot single-leg foot-end trajectory aligns with the trajectory established by qualitative analysis, confirming the applicability of this planning method for subsequent robot motion control on structured ground.

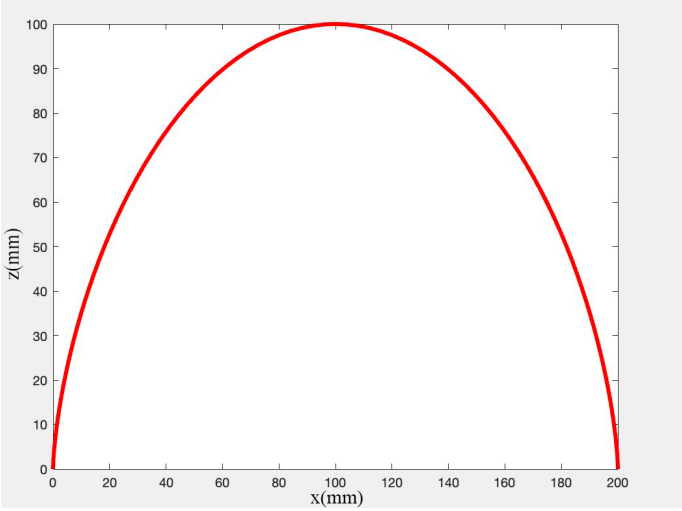


Figure 2.20: Cycloidal trajectory

Rectangular cycloid trajectory planning.

To maintain a consistent distance between the robot's swinging leg and the ground during the swinging motion, a rectangular cycloidal trajectory planning method is employed, enabling the hexapod robot to accomplish leg elevation and stride without foot-end visual sensors or precise terrain information. This approach facilitates the avoidance of collisions with ground-based obstacles during the swinging foot's motion, which could otherwise compromise the robot's balance and result in a fall.

As depicted in Figure 2.21, the swinging foot's trajectory can be segmented into

three distinct phases: initial vertical elevation, subsequent forward horizontal swing, and concluding vertical descent. Assume the time required for the robot leg to complete the elevation, swing, and descent is denoted by T_S, encompassing one full cycle of trajectory planning. To ensure a smooth leg elevation without destabilizing the robot due to excessively rapid or sluggish motion, the duration allocated for the vertical elevation, forward horizontal motion, and vertical descent of the robot's foot endpoint is set to $\frac{T_s}{3}$. Implementing this cycloidal trajectory planning approach affords sufficient time for each process to be executed, mitigating factors contributing to robot instability and enhancing practical applicability.

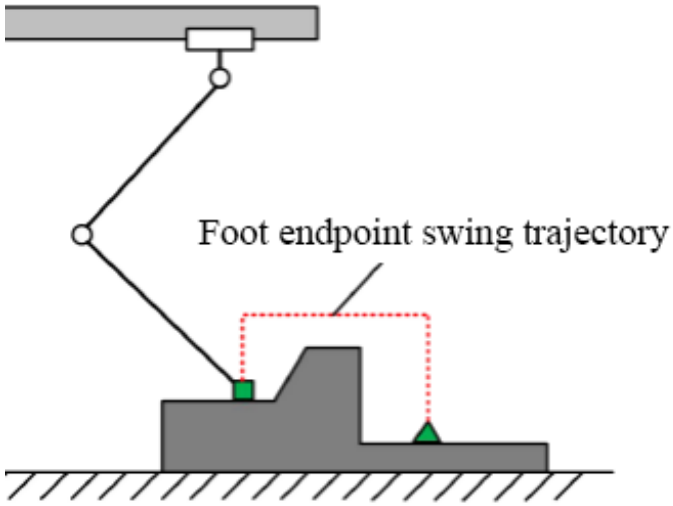


Figure 2.21: Foot endpoint swing trajectory diagram

a) Vertical elevation process of foot endpoints

During this phase of foot endpoint vertical elevation, the endpoint ascends uniformly along the Z-direction to a height of h. This strategy ensures the foot endpoint maintains a specific distance from the ground while swinging forward horizontally and clears obstacles on the ground without collisions. Throughout this vertical ascent process, the robot's foot endpoint exclusively moves in the Z-direction. The first

stage of the foot endpoint trajectory can be mathematically expressed as Equation 2.22.

$$\begin{cases} x_1(t) = 0 \\ y_1(t) = 0, t \in \left[0, \frac{T_s}{3}\right] \\ z_1(t) = \frac{3h}{T_s} \cdot t \end{cases} \tag{2.22}$$

b) Forward swing of the foot endpoint

In this stage of the forward swing of the foot endpoint, the distance between the target point and the starting point of the robot trajectory planning process is assumed to be L in the X-direction, and the distance in the Y-direction is also L. The foot end point swings forward to the end point of the trajectory planning (target landing point) along the direction directly above. The mathematical expression of the trajectory of the foot-end point at this stage can be obtained, as shown in Equation 2.23.

$$\begin{cases} x_2(t) = \frac{3L_x}{T_s} \cdot \left(t - \frac{T_s}{3}\right) \\ y_2(t) = \frac{3L_y}{T_s} \cdot \left(t - \frac{T_s}{3}\right), t \in \left[\frac{T_s}{3}, \frac{2T_s}{3}\right] \\ z_2(t) = 0 \end{cases} \tag{2.23}$$

c) Vertical drop phase

In the vertical drop phase, the robot's foot endpoint only moves downward at a uniform speed in the Z direction. When the desired drop point is lower than the starting height in the Z direction, the foot endpoint can continue to fall until it touches the ground.

$$\begin{cases} x_3(t) = L_x \\ y_3(t) = L_y \quad , t \in \left[\frac{2T_s}{3}, T_s\right] \\ z_3(t) = \frac{3h}{T_s} \cdot \left(t - \frac{2T_s}{3}\right) \end{cases} \tag{2.24}$$

Combining 2.22, 2.23, and 2.24, the equation for planning the trajectory of a rectangular pendulum with a foot endpoint is

$$x(t) = \begin{cases} x_1, \quad \mathrm{t} \in \left[0, \frac{T_s}{3}\right] \\ x_2, \mathrm{t} \in \left[\frac{T_s}{3}, \frac{2T_s}{3}\right] \\ x_3, \mathrm{t} \in \left[\frac{2T_s}{3}, T_s\right] \end{cases}$$

$$y(t) = \begin{cases} y_1, \quad \mathrm{t} \in \left[0, \frac{T_s}{3}\right] \\ y_2, \mathrm{t} \in \left[\frac{T_s}{3}, \frac{2T_s}{3}\right] \\ y_3, \mathrm{t} \in \left[\frac{2T_s}{3}, T_s\right] \end{cases}$$

$$z(t) = \begin{cases} z_1, \quad \mathrm{t} \in \left[0, \frac{T_s}{3}\right] \\ z_2, \quad \mathrm{t} \in \left[\frac{T_s}{3}, \frac{2T_s}{3}\right] \\ z_3, \mathrm{t} \in \left[\frac{2T_s}{3}, T_s\right] \end{cases}$$

Using the kinematic functions in MATLAB, the foot endpoint coordinate system is first translated upward by 400 mm in the direction of Z. Then, the foot endpoint coordinate system is translated forward by 200 mm in the direction of X and finally falls again in the direction of Z. According to the rectangular cycloid of the foot endpoint relative to the foot base coordinate system shown in Equation 2.22, 2.23, 2.24, and the inverse kinematic transformation equation, the desired motion trajectory and the actual tracking trajectory of the foot endpoint relative to the single-leg base coordinate system can be obtained as shown in Figure 2.23, Figure 2.24.

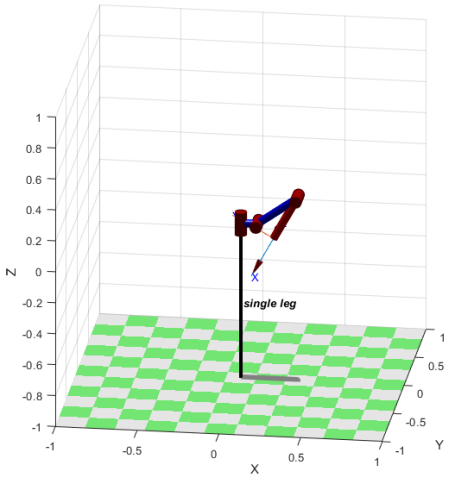


Figure 2.22: Single-leg initial position

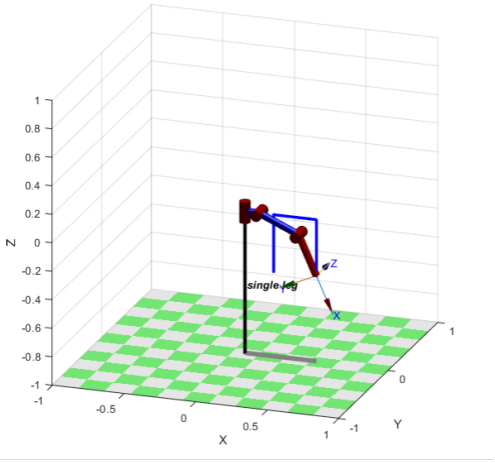


Figure 2.23: Single-leg Expected trajectory

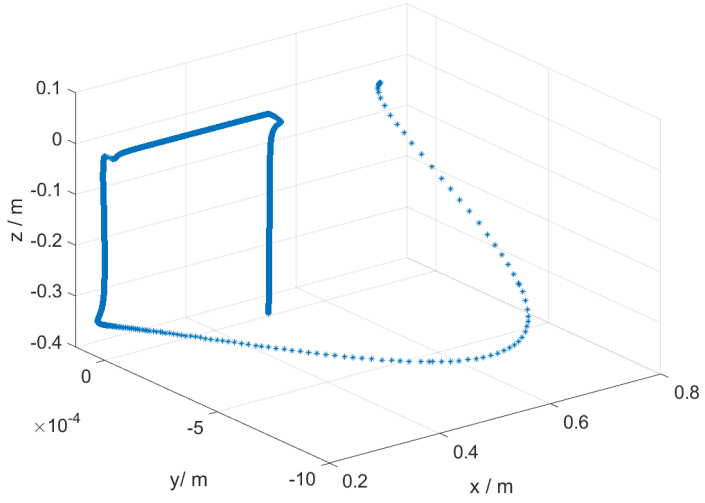


Figure 2.24: Foot endpoint trajectory tracking

From simulation results, it can be seen that, firstly, the end of the swing foot quickly moves from the initial position to the desired position. Then, the swing foot moves strictly vertically upward along the direction of Z and horizontally along the direction of X to achieve the desired effect.

Section 2.5

Verification of gait planning

In this section, the sweeping motion of the robot is experimentally verified. First, we take a three-legged gait as an example. The robot's six legs are divided into two groups, and the robot's single-leg motion differs by half a cycle between the two groups, meaning that when one group of legs is in the support phase, the other group of legs is in the swing phase. The kinematic inverse solution is then employed to solve the robot's single-leg motion trajectory inverse kinematics. The foot-end trajectory planning module and the inverse robot kinematics solution module are constructed in MATLAB using Simulink. The final calculated three-joint angles of the robot are depicted in Figure 2.25.

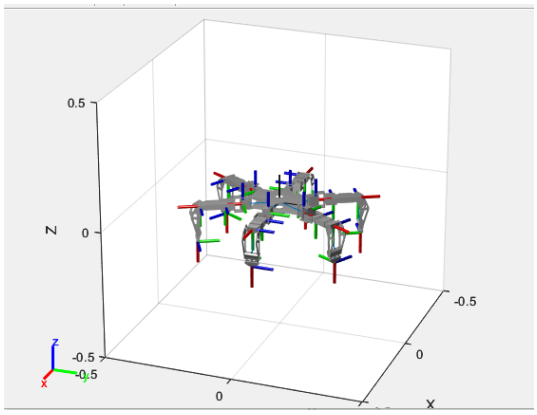


Figure 2.25: Robot foot joint coordinate system

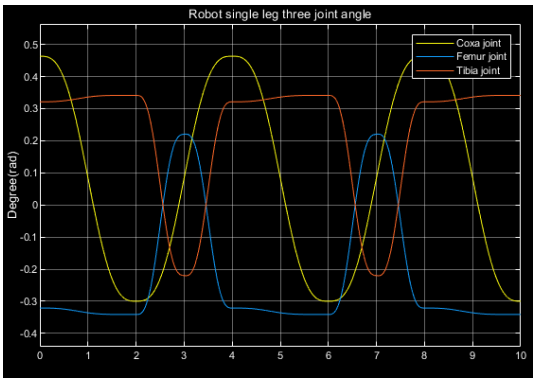


Figure 2.26: Robot single leg three joint angle

From the Figure 2.26, it can be seen that from 0 to 2s, the Coxa joint angle of the robot's single leg changes from positive to negative, while the femoral and tibial joint angles remain unchanged, and the robot's single leg is in the support phase at this time; from 2 to 4s, the coxa joint angle changes from negative to positive, and

the tibial joint angle undergoes a change from positive to negative to positive, which is precisely in line with the change in the angle of the three joints of the single leg when the robot is in the swing phase. In general, the robot's single-leg three-joint angle changes smoothly, and the transition between the support and swing phases is smooth. The period of angle change of the robot's single leg is 4s, and the support phase and swing phase each take 2s, so the angle change described in the Figure 2.26 can be directly used in the control model of the robot moving with a three-legged gait.

In the simulation environment, the gait planning module of the hexapod robot, the single-leg kinematics solution module of the robot, and the angle magnitude gain module are built in turn. The final solved angular values are input to the 18 joints of the hexapod robot to implement the robot's walking on the structured ground. Figure 2.27 and 2.28 show the simulation model built in Matlab/Simulink.

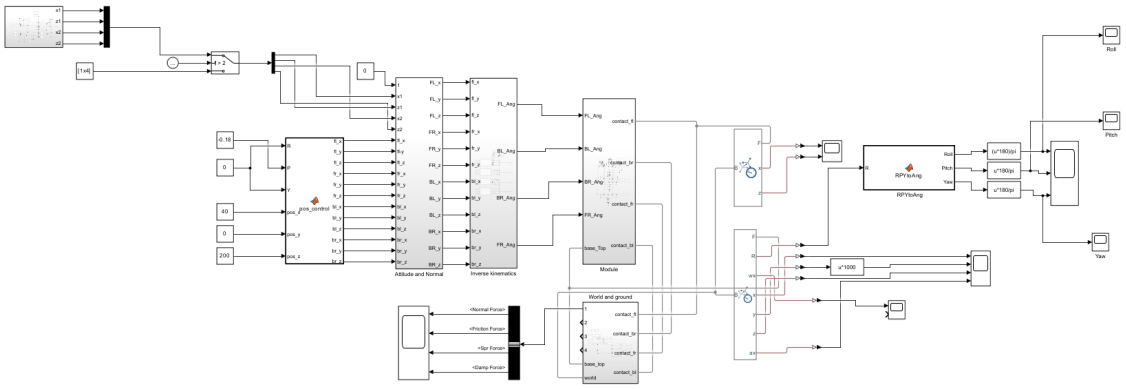


Figure 2.27: Robot motion control model file

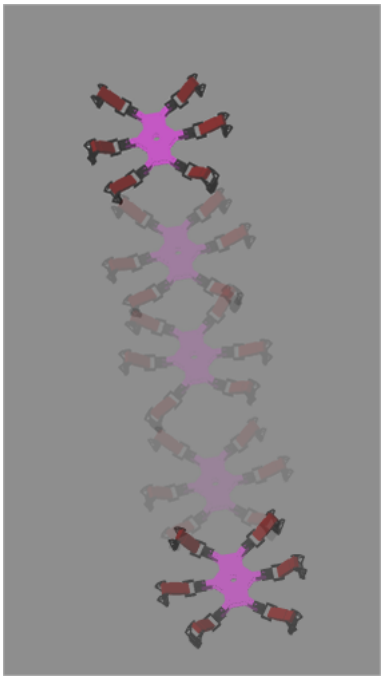


Figure 2.28: Simulation of movement

Utilizing the robot motion control model defined above, the robot's advancement from point A to point B can be visualized in the simulation environment. A quantitative analysis of the moving process is provided below, with the x-direction representing the robot's forward direction and the z-direction representing the robot's vertical direction. If the robot moves along a straight line as pre-planned, the displacement in the x-direction continuously increases, and if the robot motion is stable, the displacement in the z-direction remains constant. Figure 2.29 quantitatively depicts the displacement variation of the hexapod robot in the simulation environment.

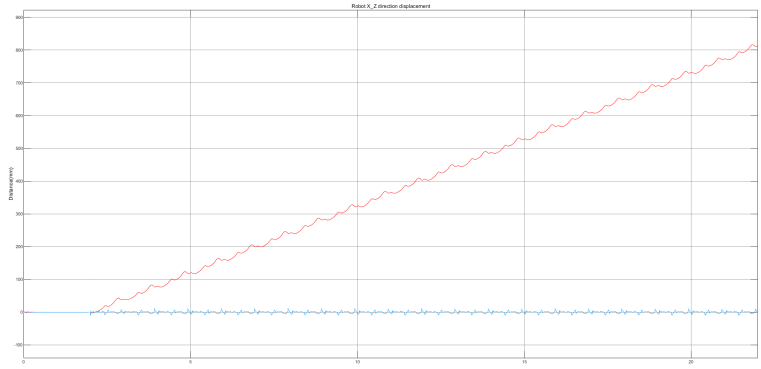


Figure 2.29: Robot X and Z motion displacement variation

From the graph, it can be seen that the displacement of the robot in the x-axis direction increases continuously from 0, and the curve is smooth without abrupt changes, indicating that the robot advances steadily along the positive direction of the x-axis.The displacement of the robot in the z-axis direction remains constant at 0, and the curve also remains smooth without sudden changes, indicating that the robot's body height remains stable and does not jump or crouch.

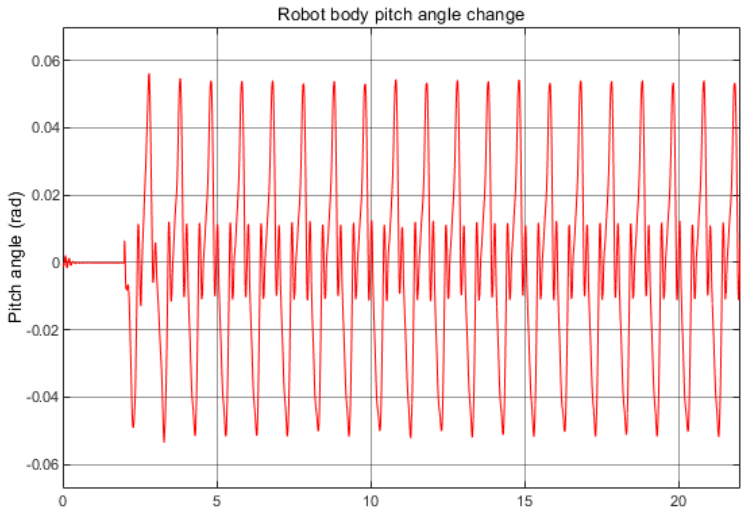


Figure 2.30: Robot body pitch angle variation

Figure 2.30 shows the change of the pitch angle of the robot body: from the figure;

it can be seen that the pitch angle of the robot body varies from -0.5rad to 0.2rad, indicating that the change of the pitch angle of the robot body is small during the motion of the robot. The robot body is stable without overturning.Combining the robot displacement variation and the body pitch angle variation shows that the gait above planning of the hexapod robot is valid, and the robot's kinematic solution is correct.

Section 2.6

Chapter Summary

In this chapter, the robot body and leg structures are designed for the simulation study, drawing inspiration from insect observations, and the overall hexapod robot body model is ultimately constructed using SolidWorks. Kinematic equations are solved to obtain the robot's kinematic equations, and kinematic solutions are verified. Inverse kinematic solutions are combined with single leg trajectory planning and inverse kinematic solutions to determine the angle changes of individual legs. By examining insect motion, hexapod robot movement is quantitatively categorized into three-legged gait, quadrupedal gait, and fluctuating gait, with various gait patterns compared in terms of speed and stability. Subsequently, single-legged swing phase planning for the hexapod robot is analyzed, and critical points for foot-end trajectory planning are identified. Based on this, a rectangular cycloid trajectory planning method is employed to examine motion characteristics and provide a mathematical description of the entire process. The foot endpoint swing process is simulated in MATLAB using the robot toolbox. Lastly, the comprehensive motion control module for the hexapod robot body is established by incorporating the three-legged gait characteristics, demonstrating the effectiveness of gait planning in controlling hexapod robot movement from the perspective of robot motion displacement and body

pitch angle.

Chapter 3

CPG-based Motion Gait Control

The problem of controlling motion is where neuroscience and robotics can be well integrated. How to generate high-dimensional rhythmic output signals while accepting only simple, low-dimensional input signals is essential in the motion control of robots. The Central pattern generators (CPGs) model was introduced to solve this problem.

Among the motion control methods for hexapod robots, bionic motion control is a very common motion control category, in which the motion control of CPG can generate its own oscillation signal, its motion control modes are numerous, it is easy to be regulated by high-level control signals due to its good coupling, and its structure is also relatively simple, which is in general very suitable for the motion control of hexapod robots.Among the motion control methods for hexapod robots, bionic motion control is a general motion control category in which the motion control of CPG can generate its oscillation signal. Its motion control modes are numerous; high-level control signals easily regulate it due to its good coupling. Its structure is also relatively simple, generally very suitable for the motion control of hexapod robots.This chapter illustrates the feasibility of CPG for motion control of hexapod robots from the perspective of imitating animal rhythmic motion control.Starting from a standard oscillator, we construct a CPG control unit, analyze the oscillator

parameters, and finally realize the pivot mode generator to control the motion of the hexapod robot.

Section 3.1

Principle of CPG control model

The central pattern generator (CPG) is a neural network capable of generating coordinated patterns of rhythmic activity without any rhythmic input from sensory feedback or higher control centres. Delcomyn (1980) [24] describes them as the basis of many rhythmic behaviours in invertebrates and vertebrates. Research has confirmed that this output rhythm does not require sensory information, and CPGs can be found in many organisms. Although sensory feedback is not required to produce rhythm, it plays a crucial role in shaping rhythmic patterns, which are fundamental to maintaining the coordination of CPGs and body movements. For example, a person on a treadmill is guided by the treadmill to stroll or run fast [25]. Experiments have shown a tight coupling between CPG and sensory feedback, i.e., their effect depends on the time within the exercise cycle [26].

In some animal experiments, it was found that low levels of stimulation can cause high-frequency movements. It indicates that the CPG is a complex circuit that can generate complex motor behaviour and achieve significant movement changes while receiving simple input signals. So from the control point of view, the CPGs model implements some internal model, which only needs to accept the command to control the motion to achieve the change of motion [27]. In summary, the vertebrate motor system is organized in this way: the spinal CPG is responsible for generating basic rhythmic patterns, while the advanced centers (motor cortex, cerebellum and basal ganglia) are responsible for modulating these patterns according to environmental conditions.In contrast, the structure of the motor control system in higher animals is

very complex, and the different motor forms result from a combination of advanced center, low-level center and effectors acting together as shown in Figure 3.1.

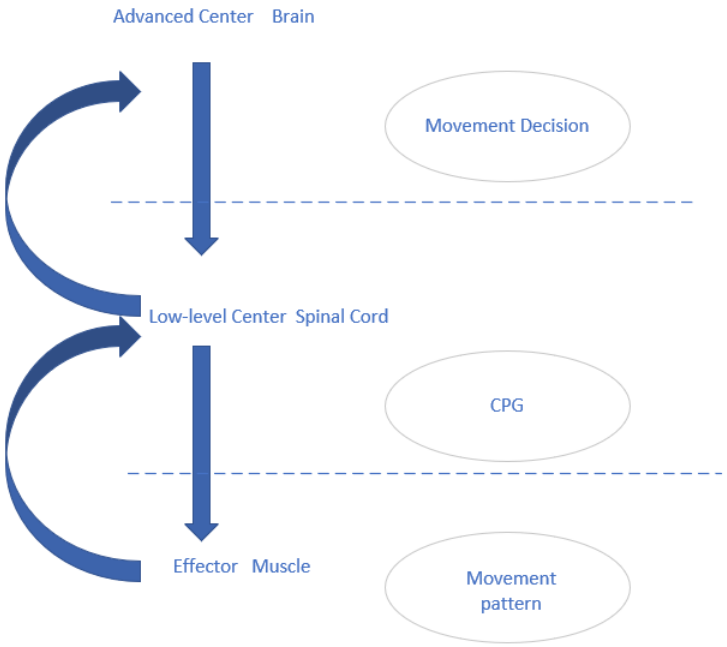


Figure 3.1: Animal motion control system

Section 3.2

Artificial CPG model

In the field of robot motion control, many kinds of CPG oscillating cell models exist, which are classified into two main categories: neuron-based models and nonlinear oscillator-based models. The most common of the first category is the Matsuoka neuronal oscillator model, where many parameters exist in the model. The equations have high dimensionality, strong coupling, nonlinearity, etc. The parameters have no apparent physical meaning, which has a compound influence on the oscillator's performance and increases the parameter adjustment and model. The complexity of parameter adjustment and model characterization is increased. The second model

type is Van der Pol (VDP) relaxation oscillator, Kuramoto phase oscillator, Hopf harmonic oscillator [28], etc. These models have relatively fewer parameters and relatively independent effects on the oscillator performance, and the parameters are easy to adjust, which is more suitable for controlling robot motion.

3.2.1. Hopf oscillator model

In several CPG oscillators models, the neuron oscillators such as Kimura have strong coupling and complicated forms. On the contrary, Hopf has a simple form,and its oscillator parameters correspond one-to-one with amplitude, phase, and frequency, and have no influence on each other. At the same time, it also has a stable limit cycle, which is suitable for the gait planning of multi-foot robots. Therefore, we adopt the Hopf oscillator as the basic oscillatory unit of the CPG network.

The Hopf oscillator is a nonlinear oscillator with fewer parameters, unlike the Matsuoka neuron oscillator, where each parameter individually affects the oscillator performance, and its mathematical model is

$$\begin{cases} \frac{dx}{dt} = \alpha\left(\mu - x^2 - y^2\right)x - \omega y \\ \frac{dy}{dt} = \alpha\left(\mu - x^2 - y^2\right)x + \omega x \end{cases} \tag{3.1}$$

where x, y are the output values of the oscillator, α is the convergence weight, μ is the squared oscillator amplitude, and ω is the oscillator frequency.$\sqrt{\mu}$ is the amplitude, and ω is the period. x and y are two state variables of the Hopf oscillator.

By setting $x = r\cos\theta$, $y = r\sin\theta$, and $\theta = \omega t$, equation 3.1 can be transformed into

$$\begin{cases} \dot{r} = \alpha\left(\mu - r^2\right)r \\ \dot{\theta} = \omega \end{cases} \tag{3.2}$$

From Figure 3.2, we can see that the Hopf oscillator model has a stable limit ring

of radius μ in the state space except for the unstable equilibrium point (0, 0). Except for the unstable equilibrium point (0, 0), the state variables x and y can converge to this limit ring by taking any initial values, which means that the Hopf oscillator can reach the limit ring from any state.

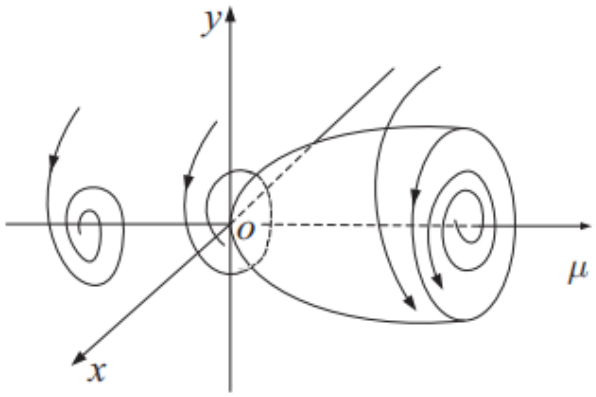


Figure 3.2: Schematic diagram of limit and equilibrium points as μ change

Figure 3.3, which shows the phase plane diagram of the Hopf oscillator output signal x and y at different initial values, shows that the Hopf limit loop is stable regardless of the initial value except for the singularity (0, 0).

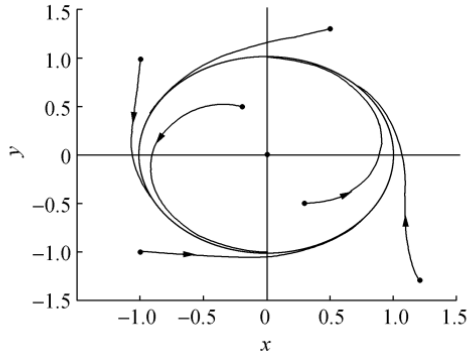


Figure 3.3: Output signal x vs. y phase plane diagram

In this thesis, the oscillator is used to control the motion of the hexapod robot. Its output signal controls its joint rotation angle. The rising part of the output

signal corresponds to the swing phase of the hexapod robot gait. The falling part corresponds to the support phase. As seen in Figure 3.4(x=0.5,y=0), the duration of the rising and falling part of the output curve is the same, i.e., the duration of the swing and support phases of the hexapod robot gait is the same. Furthermore, such an output curve can only be used as a control signal for the three-legged gait. In practical situations, hexapod robots usually use multiple motion gaits, and the swing time is not necessarily the same as the support time.

In order to adjust the oscillation time and support time, this thesis improves the Hopf oscillator model by adding the occupation factor β and the oscillator frequency ω to the mathematical expression of the oscillator,the equation is

$$\begin{cases} \omega = \frac{\omega_{\text{stance}}}{e^{-by}+1} + \frac{\omega_{\text{swing}}}{e^{by}+1} \\ \omega_{\text{stance}} = \frac{1-\beta}{\beta}\omega_{\text{swing}} \end{cases} \tag{3.3}$$

where: ω_{stance} is the support phase frequency; ω_{swing} is the swing phase frequency; b is the larger constant, which determines the conversion speed of ω between ω_{stance} and ω_{swing} ; β is the occupation factor, when $\beta = 1/2$, the swing time is the same as the support time, and changing the value of β can adjust the swing time and support time. With $\beta = 2/3$, b = 100, and $\omega_{\text{swing}} = \pi$, the output curve of the oscillator state variable x is obtained, as shown in Figure 3.5. The swing time is different from the support time with a ratio of 1:2, which indicates that the above improvement has achieved its purpose.

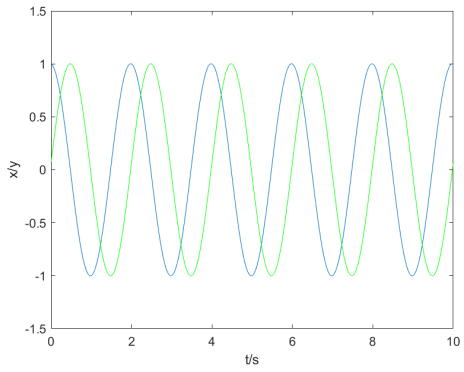


Figure 3.4: Output curves of the Hopf oscillator

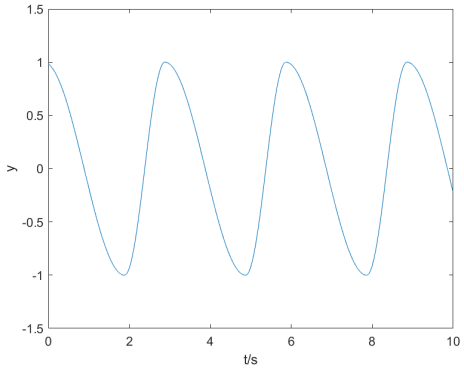


Figure 3.5: Output curve of x after β changed

In this paper, the effect of changing the parameters of the Hopf oscillator model on its output is analyzed by single-parameter analysis. The results of the parameter tuning are shown in Table 3.1.

Table 3.1: Parameter tuning results of the Hopf oscillator

Parameters	Value
α	100
μ	1
β	$\in (0, 1)$
ω_{swing}	π
b	100

Section 3.3

CPG network model improvement

Six coxa joints are controlled using a CPG network for a hexapod robot with 18 degrees of freedom of motion. The femur and tibia joints are controlled by transforming the control signals using mapping functions. The ring-type CPG network

is constructed using a graph theory approach with a directed weighted graph. One oscillator is used to control one coxa joint, and each oscillator is the vertex of the directed weighted graph, and they are connected in a fully symmetric way in Figure 3.6.

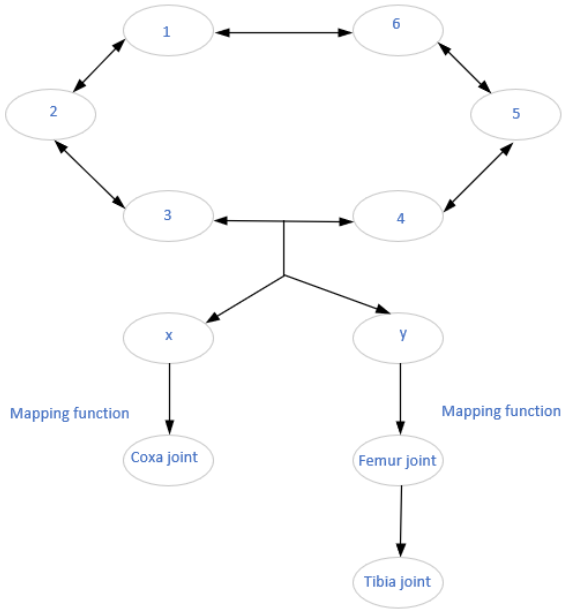


Figure 3.6: CPG control network topology of the hexapod robot

3.3.1. Single-leg joint mapping function

Since the output signal of the oscillator cannot be used as the joint control signal directly, this paper adopts the method of the mapping function to transform the output of the model so that the output is transformed into the joint rotation angle control quantity. Let the angles of the coxa, femur and tibia joints be θ_1, θ_2, and θ_3, respectively, then the mapping function between them and the output curve of the oscillator is:

$$\begin{cases} \theta_1 = k_0 x \\ \theta_2 = \begin{cases} k_1 y & y \geqslant 0 \\ k_2 y & y < 0 \end{cases} \\ \theta_3 = k_3 \theta_2 \end{cases} \tag{3.4}$$

k_0 is the mapping coefficient of the coxa joint; k_1 and k_2 are the mapping coefficients of the femur joint; k_3 is the mapping coefficient of the tibia joint, which is used to adjust the amplitude of the joint control signal.We set $k_0 = 0.3, k_1 = 0.18, k_2 = 0, k_3 = -0.12$, and the control signals of each joint are shown in Figure 3.7.

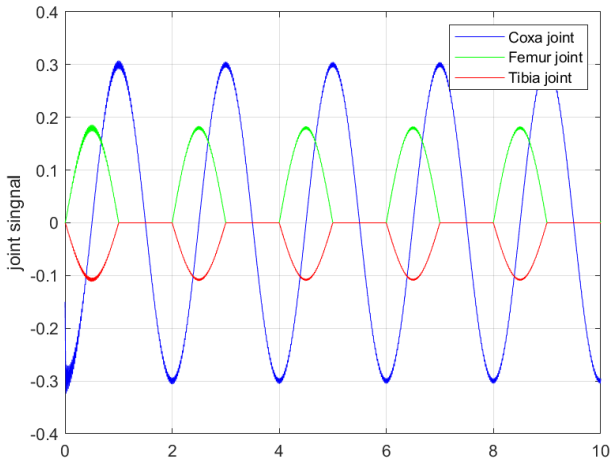


Figure 3.7: Control signals for each joint

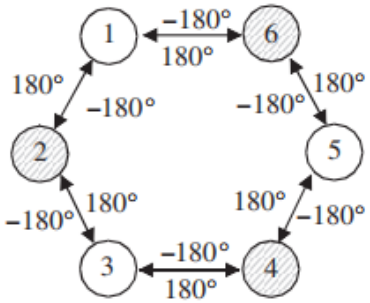
3.3.2. CPG Ring Coupling Network

The oscillators that control the robot's single leg are coupled and continuously output joint angle control signals, which enables the hexapod robot to use various gait patterns for movement. In this section, the ring coupling network topology is used to describe the phase coupling relationship between the output signals of each oscillator model, and its mathematical model is

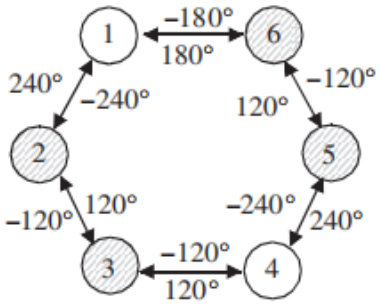
$$\begin{cases} \dot{x}_i = \alpha\left(\mu - x_i^2 - y_i^2\right) x_i - \omega_i y_i \\ \dot{y}_i = \alpha\left(\mu - x_i^2 - y_i^2\right) y_i + \omega_i x_i + \lambda\left(y_j \cos\theta_{ji} - x_j \sin\theta_{ji}\right) \\ \omega_i = \omega_{\text{stance}} \,/\, \left(\mathrm{e}^{-\mathrm{by}_i} + 1\right) + \omega_{\text{swing}} \,/\, \left(\mathrm{e}^{\mathrm{by}_i} + 1\right) \end{cases} \tag{3.5}$$

λ is the coupling strength parameter between two oscillators, and λ affects the articulation between the waveform in the rising and falling parts of the output curve. λ should not be too large to cause the system to produce dither. We take $\lambda = 0.6, \theta ji$ is the phase difference between oscillators i and $j, \theta ji = \theta i - \theta j$; the other parameters are defined in the same way as equation 3.2 and 3.3.

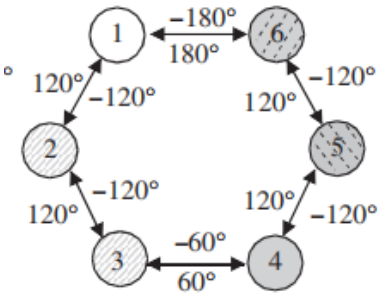
The following is an example of a three-legged, four-legged, and five-legged gait with a ring coupling network to illustrate each leg's phase difference when the hexapod robot walks with different gaits. When a hexapod robot walks with a three-legged gait, its legs are divided into legs 1, 3, 5 and legs 2, 4, 6, the three legs of the same group are in the same phase, and the two legs that are not in the same group are in 180° phase. The ring-shaped coupling network is shown in Figure 3.8(a); When walking with a quadrupedal gait, the legs are divided into three groups: legs 1 and 4, legs 3 and 6, and legs 2 and 5. The two legs of the same group have the same phase, and the two legs of adjacent groups differ in phase by 120°, and the circular coupling network is shown in Figure 3.8(b); When walking with a fluctuating gait, each leg enters the swing phase in the order of leg 1→leg 6→leg 2→leg 5→leg 3→leg 4, with a phase difference of 60° between adjacent legs, and its circular coupling network is shown in Figure 3.8(c).



(a) Three-legged gait



(b) Quadrupedal gait

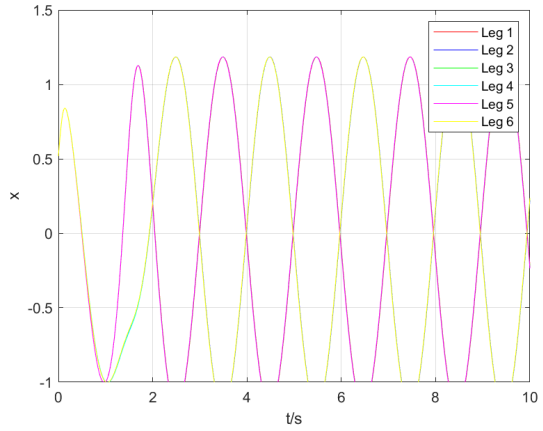


(c) Fluctuating gait

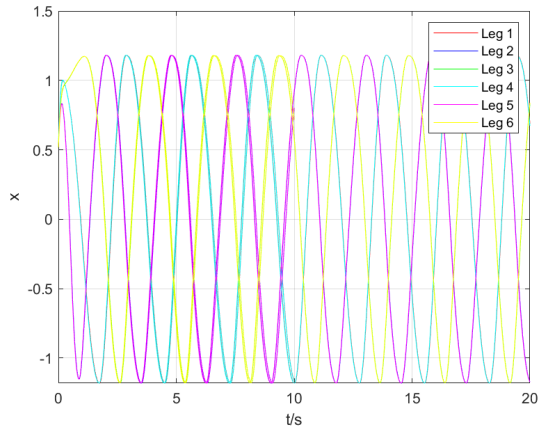
Figure 3.8: CPG ring-coupling network of typical gaits of the hexapod robot

The occupation coefficients are $\beta = 1/2, 2/3, 5/6$ for the three-legged gait, quadrupedal gait, and fluctuating gait. The remaining parameters are set according to the parameter calibration results. The output curve of each leg oscillator can be obtained when the hexapod robot walks with three gaits, respectively, as shown in Figure 3.9. As seen in Figure 9, using a ring coupling network ultimately results in a stable phase

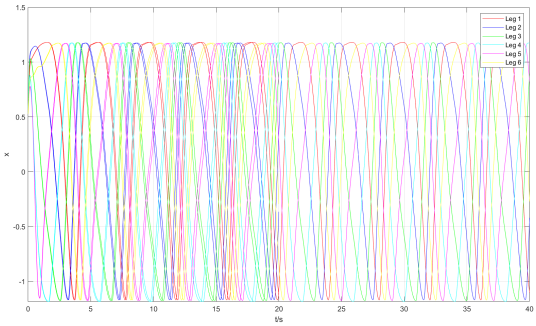
difference in the output curve of each oscillator. However, the coupling time is longer for both quadrupedal and fluctuating gaits, with the quadrupedal gait taking about 6 s before the phase difference stabilizes and the quintupled gait taking even longer, with the phase difference stabilizing after 28 s. This requires a specific delay time to be set when performing control to ensure that the hexapod robot can walk with a stable gait. This output signal is not suitable for the gait control of the hexapod robot. In addition, when the hexapod robot moves with a quadruped or fluctuating gait, the equilibrium position of the angle change curve is not necessarily on the zero line. However, it may be located above or below the zero line. Therefore, the previously designed scheme needs to be improved to meet the multiple gait control requirements of the hexapod robot.



(a) Three-legged gait



(b) Quadrupedal gait



(c) Fluctuating gait

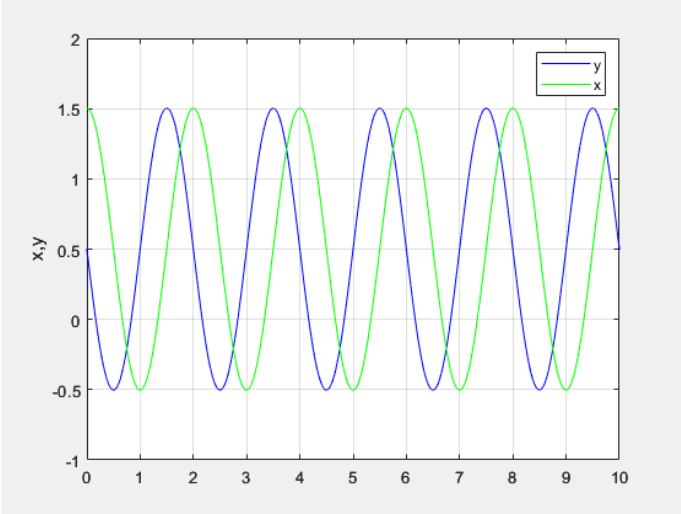
Figure 3.9: Output curves of each oscillator of typical gaits of the hexapod robot

3.3.3. CPG Control Program Improvement

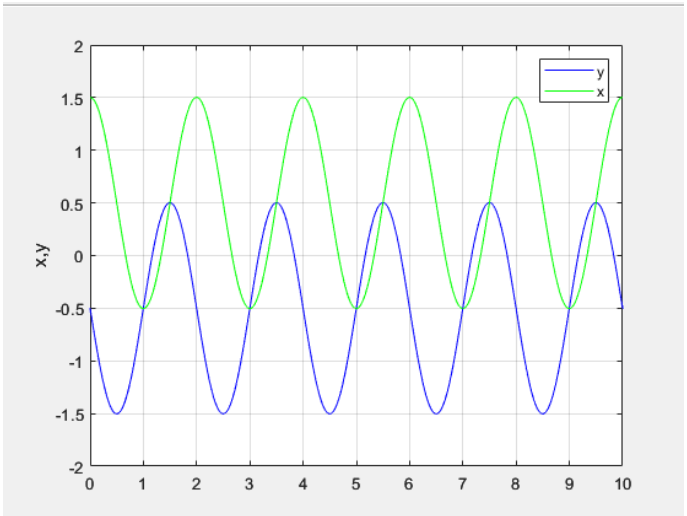
The CPG control scheme designed earlier is to adjust the balance position of the oscillator output curve so that it can be used as a coxa control signal when the hexapod robot walks with a quadrupedal gait or a fluctuating gait improved in this paper.Introduce the external feedback quantities feed $_1$ and feed $_2$ are introduced in equation 3.1 as

$$\begin{cases} \dot{x} = \alpha \left[\mu - (x - \text{ feed }_1)^2 - (y - \text{ feed }_2)^2\right](x - \text{ feed }_1) - \omega\,(y - \text{ feed }_2) \\ \dot{y} = \alpha \left[\mu - (x - \text{ feed }_1)^2 - (y - \text{ feed }_2)^2\right](y - \text{ feed }_2) + \omega\,(x - \text{ feed }_1) \end{cases} \tag{3.6}$$

Set the occupation factor is set at $\beta = 1/2$, the external feedback amount is set at feed $_1$ = 0.5, feed $_2$ = 0.5 and feed $_1$ = 0.5, feed $_2$ = -0.5 respectively. The output curves of the oscillator model can be obtained by setting the remaining parameters according to the parameter tuning results. From Figure 3.10, it can be seen that when the external feedback is positive, the output curve is located above the zero line; When the external feedback amount is negative, the output curve of the balance position is below the zero line, and the change of the balance position concerning the zero line is equal to the value of the external feedback. The other parameter characteristics of the oscillator are not affected by the change in external feedback. Therefore, the value of the external feedback can be adjusted to change the balance position of the oscillator output curve.



(a) feed $_1$=0.5,feed $_2$=0.5



(b) feed $_1$=0.5,feed $_2$=-0.5

Figure 3.10: Output curves of the oscillator when feed $_1$ and feed $_2$ take different values

In essence, we calibrate external feedback levels to yield an apt control signal for the hexapod robot's coxa joint rotation angle during ambulation with quadrupedal or undulating gaits. Concurrently, to generate a consistent and uninterrupted phase disparity among each oscillator's output curves initially, this study presents an approach that adjusts the oscillator model state variable's initial value, utilizing a circular cou-

pling network. By modifying the inaugural value of every oscillator state variable, the output curve can exhibit distinct phase variations at the outset, with the optimal initial value ascertainable through iterative experimentation. Discovering the ideal initial value via trial and error enables each oscillator's output curve to manifest the requisite phase discrepancy for the gait's inception. Consequently, the hexapod robot commences locomotion with a stable stride.

For the three-legged gait, the equilibrium position of the coxa joint turning angle change curve of each leg is on the zero line; For quadrupedal gait, the equilibrium position of the coxa angle change curve for legs 1 and 4 is above the zero line, the equilibrium position of the coxa angle change curve for legs 3 and 6 is on the zero line, and the equilibrium position of the coxa angle change curve for legs 2 and 5 is below the zero line;For the fluctuating gait, the equilibrium position of the hip angle change curve for legs 1, 2, and 6 is above the zero line, and the equilibrium position of the hip angle change curve for legs 3, 4, and 5 is below the zero line. The values of the external feedback and the initial values of the state variables of each leg oscillator for the hexapod robot with each walking gait are shown in Tables 3.2 and 3.3.

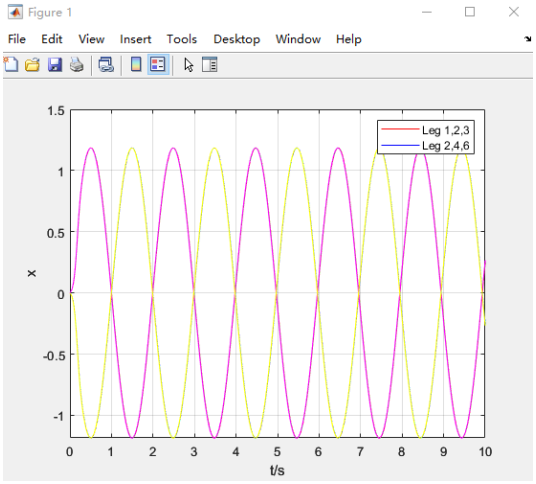
Table 3.2: External feedback of each oscillator and initial values of the state variables in the three-legged gait

Parameters	feed $_1$	feed $_2$	x_0	y_0	Parameters	feed $_1$	feed $_2$	x_0	y_0
leg_1	0	0	0	-0.1	leg_4	0	0	0	0.1
leg_2	0	0	0	0.1	leg_5	0	0	0	-0.1
leg_3	0	0	0	-0.1	leg_6	0	0	0	0.1

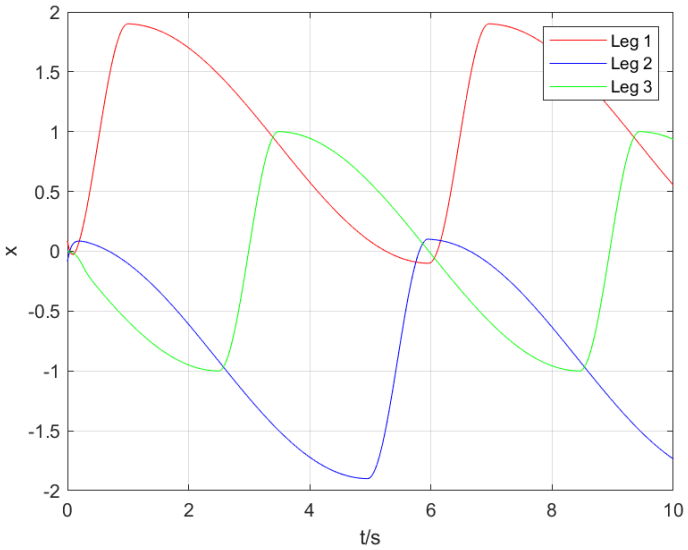
Table 3.3: External feedback of each oscillator and initial values of the state variables in the quadrupedal gait

Parameters	feed $_1$	feed $_2$	x_0	y_0	Parameters	feed $_1$	feed $_2$	x_0	y_0
leg_1	0.9	0	0.1	0	leg_4	0.9	0	0.1	0
leg_2	-0.9	0	-0.1	0	leg_5	-0.9	0	-0.1	0
leg_3	0	0	0	0.1	leg_6	0	0	0	0.1

The output curves of each leg oscillator for the hexapod robot walking with three-legged gait, quadruped gait and fluctuating gait are obtained according to the previous CPG ring coupling network settings, as shown in Figures. 3.11(a) to (b), respectively



(a) Three-legged gait



(b) Quadrupedal gait

Figure 3.11: Improved output curves of each oscillator of typical gaits of the hexapod robot

The improved CPG network model can output a signal with a stable phase difference from the beginning, and the balance position of the output curve can be adjusted so that the output curve can be ideally used as the coxa control signal for each walking gait of the hexapod robot. It can be shown that the improved CPG control scheme is practical and feasible, and the output curve of the improved CPG

network model can be used to control various motion gaits of the hexapod robot.

3.3.4. Simulation experiment verification

Take the example of a three-legged gait.The control curves for each joint of the hexapod robot with three-legged gait motion are shown in Figure 3.12. The output curves of the CPG network model all enter into stable oscillations quickly. The phase difference of different groups of legs is 180°, which is consistent with the tripod gait characteristics. The control curve of each joint of a single leg also satisfies the motion phase relationship of each joint.

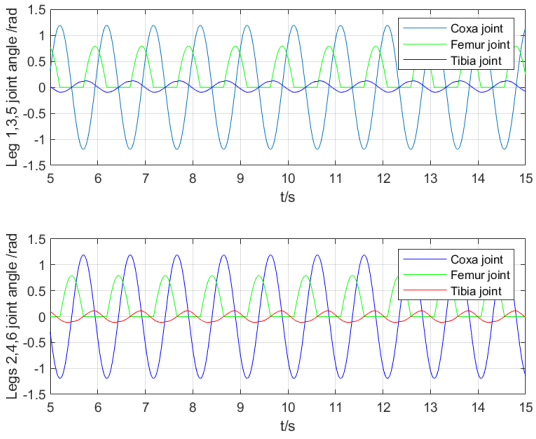


Figure 3.12: Joint's control curves of the three-legged gait

The control curves of each joint of the hexapod robot with quadrupedal gait motion are shown in Figure 3.13 shows. According to the swing order, the phase difference of different groups of legs is 120°, which is consistent with the quadrupedal gait characteristics, and the control curve of each joint of a single leg also satisfies the same The phase relationship of the motion of each joint of the same leg.

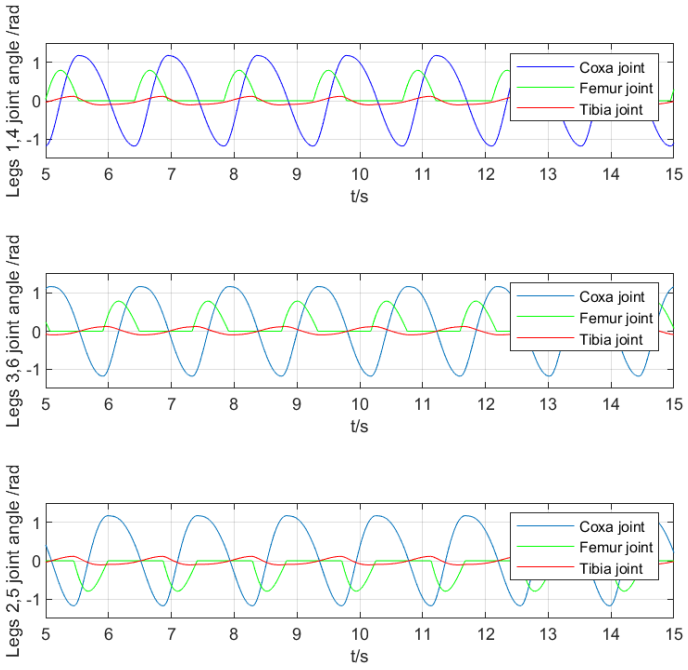


Figure 3.13: Joint's control curves of the quadrupedal gait

Section 3.4

Simulation and experiment results

3.4.1. Transition motion from flat to slope

Using the motion control block diagram shown in Figure 3.14 to simulate the transition motion from flat to 20° slope, information such as joint signal and foot force during simulation can be exported via Simulink Simscape Multibody. By measuring the rise and fall of the pitch angle of the body, a smooth gait transition can be achieved.

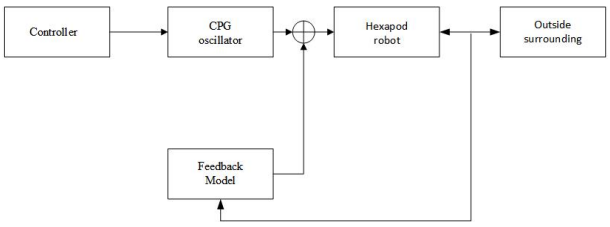


Figure 3.14: Transition motion control block diagram

With a tripod gait based on the Hopf and Hopf with feedback models separately, the slope motion of a hexapod robot is simulated. Figure 3.15 shows the static force of the hexapod robot on the slope, which must meet the mechanical balance.

$$\begin{cases} \sum_i^n F_{Xi} = 0 \\ \sum_i^n F_{Yi} - F_{\mathrm{G}} \sin\theta_{\mathrm{p}} = 0 \\ \sum_i^n F_{Zi} - F_{\mathrm{G}} \cos\theta_{\mathrm{p}} = 0 \end{cases} \tag{3.7}$$

where n is the number of foot support for the current movement of the hexapod robot, θ_{p} is the slope angle of the slope, F_{G} is the gravity at the center of mass of the body, F_{Xi}, F_{Yi}, and F_{Zi} are the forces in the three-axis direction of the i th foot support leg of the hexapod in the centroid coordinate system.

All of the supporting feet of the hexapod robot's legs must exert sufficient friction force.

$$F_f = \mu \left(\sum_i^2 F_{iZ} \right) \geq \mu \sqrt{\left(\sum_i^2 F_{iX} \right)^2 + \left(\sum_i^2 F_{iY} \right)^2} \tag{3.8}$$

where μ is the coefficient of friction determined by the slop and the legs of the hexapod robot.

Combining equation 3.7 into equation 3.8, we can obtain the following equation.

$$\theta_{\mathrm{p}} \leq \arctan(\mu) \tag{3.9}$$

The coefficient of static friction in our set simulation environment is 0.3. We

choose a slope of 20° for the simulation. The three-joint signal of the single leg can be obtained during the simulation of both states because the walking posture of the hexapod robot will show the front high and back low, which leads to uneven force on the foot end and affects the stability of the slope movement of the hexapod robot. In order to improve the stability of the side slope motion, it is necessary to change the pitch angle of the body. The body pitch angle can be adjusted by switching the gait to change the support leg's joint angle to improve the slope motion's stability. From the Table 2.2, we can conclude that the quadrupedal gait is more stable than the tripodal gait, so we decided to use the tripodal gait to move on flat ground and the quadrupedal gait to move on slopes. Screenshots of the simulation are shown in Figure 3.16.

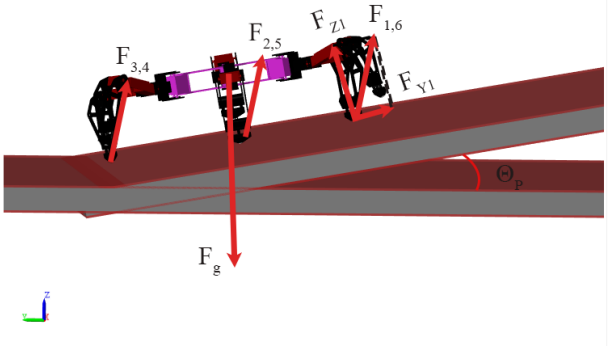
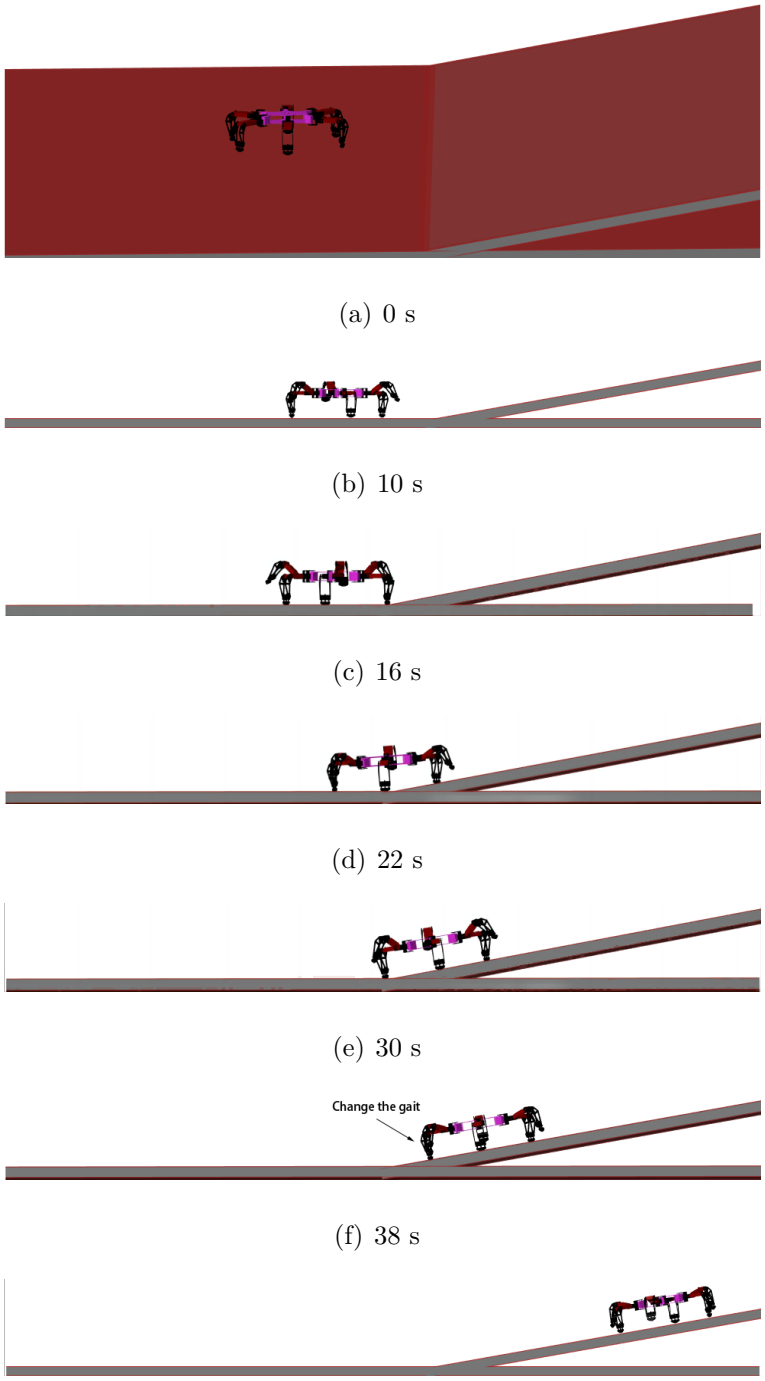


Figure 3.15: Static schematic diagram of a hexapod robot on a slope



(a) 0 s

(b) 10 s

(c) 16 s

(d) 22 s

(e) 30 s

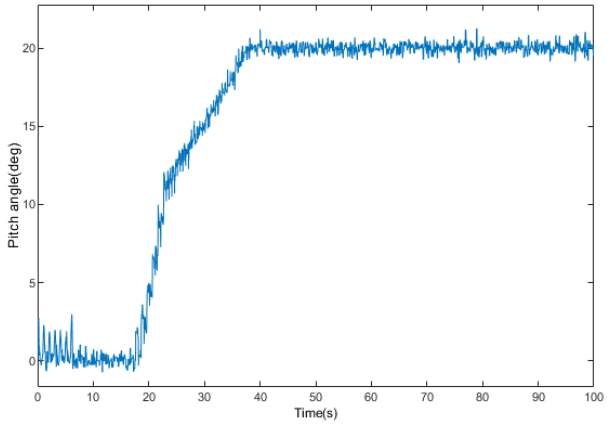
(f) 38 s

(g) 45 s

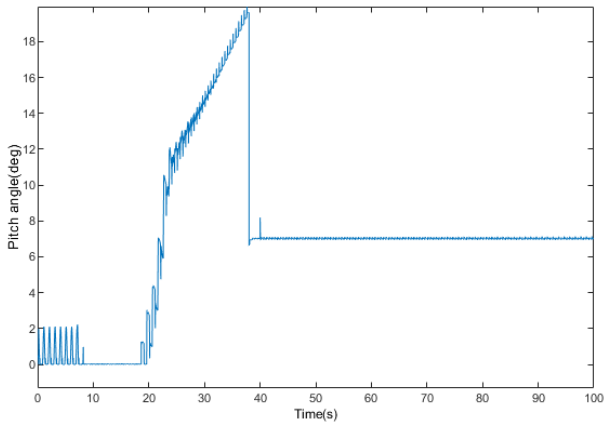
Figure 3.16: Simulation screenshot of hexapod robot's motion from flat to 20° slope: (a) 0 s, (b) 10 s, (c) 16 s, (d) 22 s, (e) 30 s, (f) 38 s, (g) 45 s.

3.4.2. Slope motion

Pitch angle of hexapod robot while climbing 20° slope is shown in Figure 3.17. Base on the Hopf model, the hexapod robot's pitch angle ranges from 18.5° to 22.5°, which is almost parallel to the slope.When the pitch angle of the hexapod robot is greater than or equal to 20°, the support leg's joint angle signal is adjusted to switch the gait and the pitch angle is reduced to between 7.1° and 7.5°.



(a) pitch angle based on Hopf model



(b) pitch angle based on Hopf model with the gait change

Figure 3.17: Pitch angle of hexapod robot's motion of climbing 20° slope: (a) pitch angle based on Hopf model and (b) pitch angle based on Hopf model with the gait change.

The following is an analysis of the CPG network built on a Hopf oscillator to control the robot's motion in Figure 3.18 .We can see that the robot displacement fluctuates from 0 to 7s in the z-axis direction, indicating that the robot is jumping and squatting during the motion; from 7 to 38s, the z-axis direction remains constant, indicating that the robot is moving on the flat ground; after 38s, the robot displacement changes from 0 to 1.5 in the z-axis direction, which is due to the robot switching from flat ground to slope motion.

The displacement of the robot in the y-axis direction changes from 0 to 0.1, and then the y-axis direction changes from 0.1 to 5.1. This is because, in the simulation environment. Initially the robot performed squatting, resulting in small displacement fluctuation changes at the beginning. However, the displacement change curve of the robot in each section of the y-axis is straight, indicating that the robot still moves in a straight line in the simulation environment.

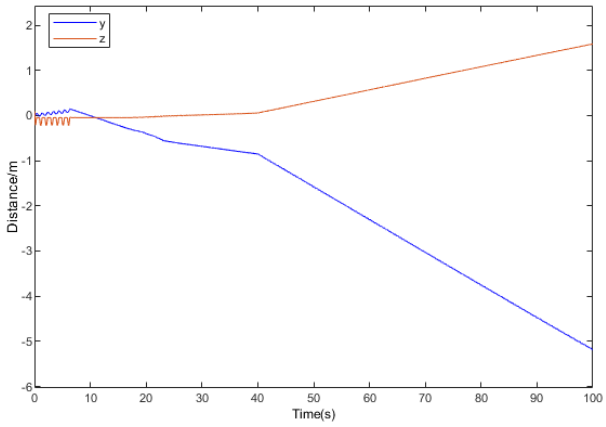
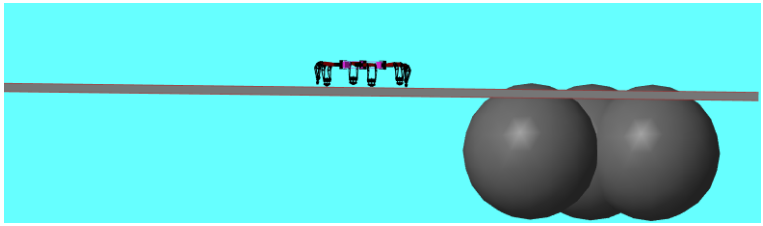


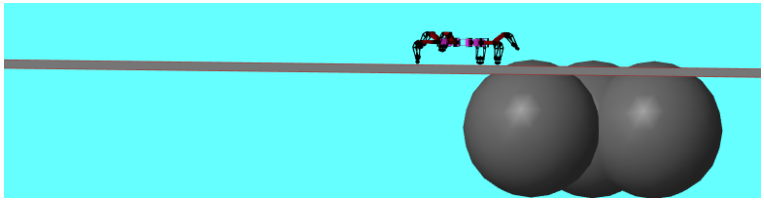
Figure 3.18: Hopf oscillator controls robot motion

3.4.3. Uneven terrain motion

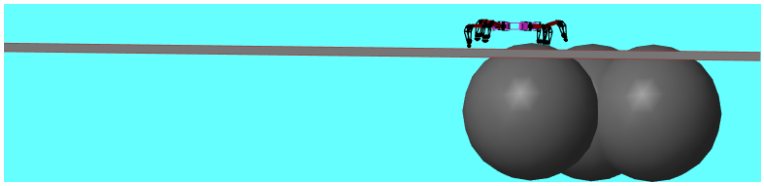
The simulation screenshots of hexapod robot's motion on uneven terrain are shown in Figure 3.19. Based on the Hopf model, the center-of-mass projection point deviates from the center of the supporting field, the foot–end force distribution is uneven, and the hexapod robot has severe skidding during its movement and becomes immobile on slopes. Based on Hopf with gait change, the hexapod robot's attitude is adjusted, which stabilizes the centroid projection point in the center of the supporting area and improves the foot–end force condition, allowing it to traverse uneven terrain. The robot is in the initial state when t = 0. Between t = 20s 30s, the robot traverses the uneven terrain, and when t > 30s, the robot enters the uneven terrain. In addition, at t = 45s, the robot completes traversing the uneven terrain in motion, indicating successful uneven terrain movement.



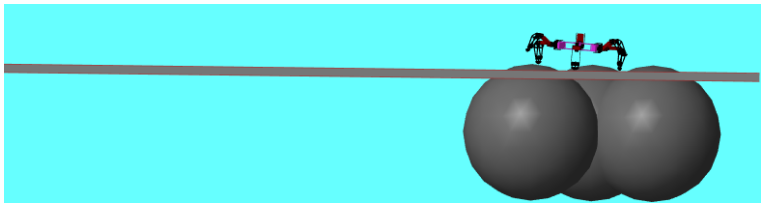
(a) 0 s



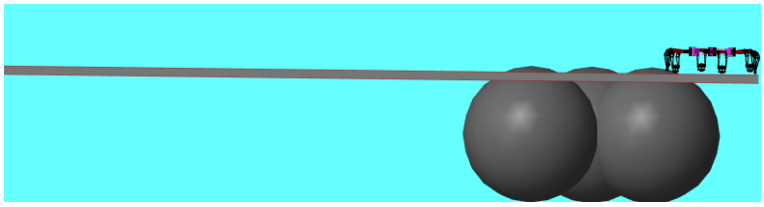
(b) 20 s



(c) 30 s



(d) 40 s



(e) 45 s

Figure 3.19: Uneven terrain motion

The following is a quantitative analysis of the robot's state during robot motion.

As shown in Figure 3.20, the pitch angle of the robot is constant from 0 to 20s, and after 28s, the robot starts to move in the uneven terrain. The pitch angle of the robot starts to change; although there was a sudden angle change in the middle, the robot did not roll over during the climbing process.In Figure 3.21, the displacement of the robot in the x-axis direction keeps increasing, and its displacement in the z-axis direction remains constant, indicating that the robot has not fallen and has successfully traversed the uneven terrain.

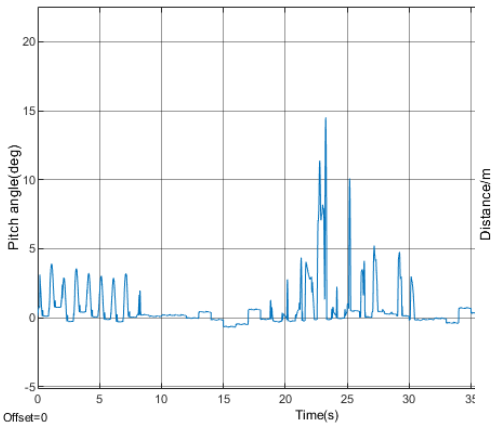


Figure 3.20: Pitch angle of uneven terrain

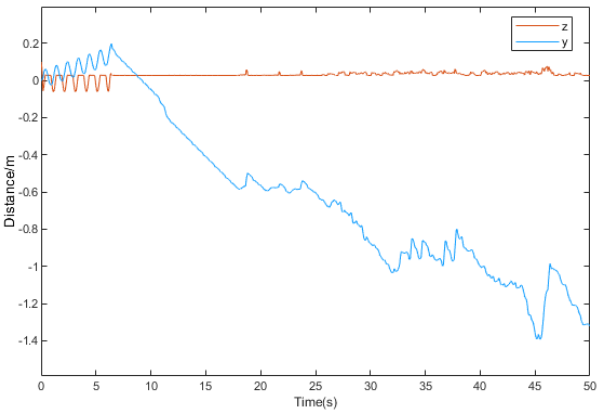


Figure 3.21: Robot displacement change in uneven terrain

Section 3.5

Chapter Summary

This chapter devises and refines a CPG-centric locomotion control strategy for a hexapod robot, achieving stable regulation of each movement pattern. Simulation findings indicate: a) The enhanced Hopf oscillator model can modulate swing duration and support intervals for the hexapod robot's motion patterns, facilitating adjustments to the output signal's duty cycle; b) The single-leg joint mapping function conceived within this chapter is rational and harmonious with each individual leg joint's move-

ment phase relations. The CPG annular coupling network generates control signals displaying stable phase discrepancies, congruent with respective hexapod robot movement patterns; c) Subsequent to CPG control scheme optimization, the CPG network yields control signals with consistent phase discrepancies from the outset, enabling balance position adjustments for the control signal. The control demands of various hexapod robot motion patterns are accommodated.

The hexapod robot's transitional motion from level terrain to an inclined surface utilizing a tripod gait was scrutinized, establishing the quantitative association between the joint angle variable introduced by the robot and its pitch angle. A transition gait predicated on CPG's base feedback was devised. A simulation platform was assembled, performing a transitional motion simulation from level ground to ascending a 20° incline. Simulation results validate the transition gait's practicability, with the slope gait anchored in the Hopf model accommodating gait modifications. Uneven terrain was subsequently assessed. Although the terrain was ultimately traversed successfully, there were instances during the simulation where the hexapod robot slipped, erroneously transitioning to an unsuitable gait while navigating obstacles, resulting in a rollover. To date, no robust CPG-based methodology has been discovered capable of tackling diverse, demanding terrains.

Our aim is to examine the interplay between yaw angle, roll angle, and the stability of a hexapod robot's transitional motion, while incorporating attitude feedback into the transitional gait planning to further bolster the robot's transitional motion stability.

Chapter 4

Multi-agent Deep Reinforcement Learning-based Motion Control for Hexapod Robots

The previous section designed a hexapod robot model imitating a hexapod insect and analyzed the implementation of the robot from the perspective of foot-end trajectory planning. The robot motion is interpreted and implemented from the perspective of CPG. This chapter explores the motion control of a hexapod robot from the perspective of deep reinforcement learning. The theoretical foundations related to deep reinforcement learning are described and followed by experimental verification of robot motion control based on typical deep reinforcement learning algorithms.

Section 4.1

Introduction to Reinforcement Learning

Reinforcement learning [3] is a type of machine learning which becomes the third machine learning paradigm, in addition to supervised and unsupervised learning. Re-

inforcement learning is a computational method for understanding and automating goal-directed decision-making and learning. It is distinguishable from previous computational approaches by emphasizing an agent's learning through direct interaction with its environment without relying on exemplary supervision or exhaustive models of the environment. It also evaluates the actions selected by the agent, making the selection of subsequent actions more in line with expectations.

The reinforcement learning problem is a direct framework for the learning problem of achieving goals from interactions. A Markov decision process can mathematically describe it. The learner and decision-maker are called the agent. The agent interacts with the environment, including everything outside the agent. The Markov decision process means that the future state of the agent is only related to the state and behavior at the present moment, and all the historical states have no influence. The Markov decision process consists of the parameters S A, P, and R. S is the set of possible states. A indicates the set of actions in state S_t. P denotes the agent state transition probability, and R is the reinforcement learning reward function designed for after a time step, as a partial result of its action, the agent receives a numerical reward. $P\left(s_t, a_t, s_{t+1}\right)$ is the probability that an intelligent system employing the action to bring the environment from the original state s_t to the probability of reaching the new state s_{t+1}. $R\left(s_t, a_t, s_{t+1}\right)$ denotes the reward value obtained by the intelligent system through the action the reward value obtained by the action at to bring the environment from the original state s_t to the new state s_{t+1}, where $s_t \in$ S denotes the external environment state value at time t. $a_t \in$ A means the action value provided by the agent to the external environment at time t. Figure4.1 diagrams the agent–environment interaction.

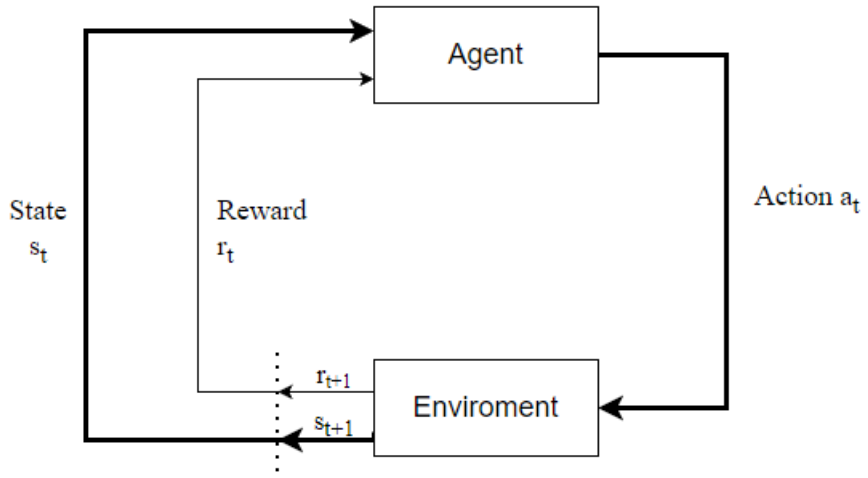


Figure 4.1: The agent–environment interaction

In reinforcement learning, Policy π:$S \rightarrow A$ represents the mapping from the state space to the action space. More specifically, The agent selects the action value when the state of the external environment is s_t and executes the action value so that the external environment arrives at the new state s_{t+1} with probability P. In contrast, the reward function calculates the current reward value r_t. Since the reward value, r_t at any moment, differs from the actual contribution, a discount factor γ_t is set for the reward value r_t at time t. Then the sum of the rewards of the agent from the beginning of any moment to the final destination is :

$$R_t = \sum_{t'=t}^{T} \gamma_{t'} r_{t'} \tag{4.1}$$

where $\gamma_{t'} \in [0, 1]$, indicating the effect of the subsequent reward value on the cumulative reward. The action-value function for policy π is:

$$Q_\pi(s, a) = E[R_t \mid s_t = s, a_t = a] \tag{4.2}$$

Return R_t depends on states s_t, s_{t+1}, s_{t+2}, ... and actions a_t, a_{t+1}, a_{t+1}, ... For

all state-action pairs, if the expected reward from a policy $\pi*$ is greater than or equal to any other policies, then the policy $\pi*$ is optimal. There may be many optimal policies, but they have the same state action function:

$$Q^*(s,a) = \max_{\pi} E[R_t \mid s_t = s, a_t = a] \tag{4.3}$$

For reinforcement learning, as a Markov decision process, its optimal state action function conforms to the Bellman optimal equation, i.e

$$Q^*(s,a) = E_{s'S}\left[r + \gamma \max_{a'} Q\left(s', a'\right) \mid s, a\right] \tag{4.4}$$

The Q-value function is often solved in reinforcement learning by iterating the Bellman equation:

$$Q_{i+1}(s,a) = E_{s'S}\left[r + \gamma \max_{a'} Q_i\left(s', a'\right) \mid s, a\right] \tag{4.5}$$

We find that where $Q_i \to Q^*$ when $i \to \infty$.After continuous training iterations, the state action function will converge so that the optimal policy $\pi^* = \arg\max_{a \in A} Q^*(s,a)$ can be obtained. However, the formula to solve the optimal policy often does not work relative to the situation. The state space is larger through the iterative Bellman equation to find the Q value function. This method is often too expensive. In order to solve this problem, the reinforcement learning algorithm often uses a deep neural network to represent state action value as $Q(s, a \mid \theta) \approx Q^*(s,a)$, combined with the reinforcement learning in deep neural networks for large state space has good application effect.

Reinforcement learning algorithms, as a general term for a class of algorithms, are rich in content and contain a wide variety of algorithms. Its main algorithms can be classified as model-based reinforcement learning(MBRL) and model-free reinforce-

ment learning(MFRL), depending on the level of comprehension of the environment. From the perspective of policy learning methods, they can be divided into value function-based reinforcement learning and policy-based reinforcement learning. According to the descriptiveness of the learning goal, it can be divided into forward reinforcement learning and inverse reinforcement learning.The following explains the common algorithms from the perspective of policy learning:

4.1.1. Q-Learning algorithm

The use of time-series difference to solve reinforcement learning decision problems does not require an environment state transformation model but rather updates the policy by updating the value function. As shown in Figure 4.2, the environment starts in state s_t, the action at is selected by the - greedy algorithm, and after acting a_t, it enters the state s_{t+1} and receives the reward R. After that, the next action a_{t+1} is selected by the greedy algorithm, i.e., the action that maximizes the value function is selected as a. Then the update can be expressed as

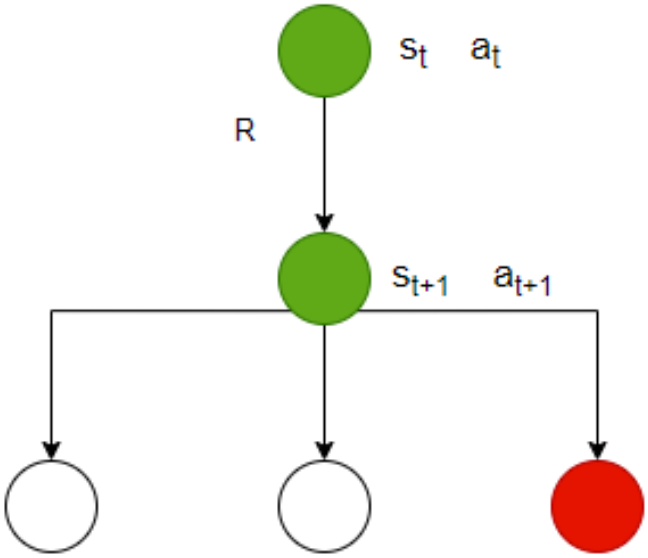


Figure 4.2: Q-Learning algorithm

$$Q'(s_t, a_t) = Q(s_t, a_t) + \mu\left(R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \tag{4.6}$$

where μ is the learning rate, the direction of iteration is determined by the learning rate in parentheses to the right, and the final iterative goal is

$$Q(s_t, a_t) = E[R_t \mid s_t, a_t] \tag{4.7}$$

In this case, the action that achieves the maximum reward at any given moment can be selected based on the Q value as shown in Figure 4.3 for the Q-Learning algorithm.

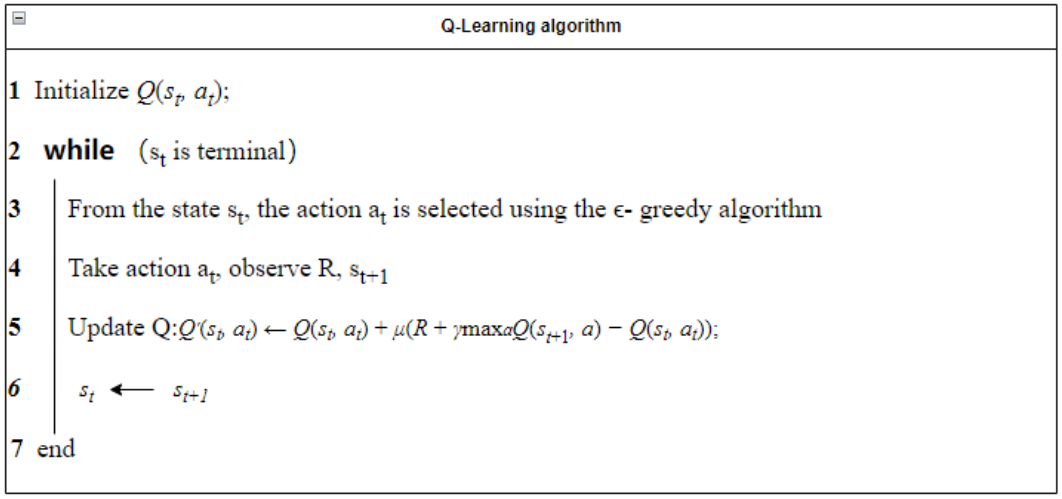


Figure 4.3: Q-Learning algorithm diagram

4.1.2. Policy-based reinforcement learning algorithms

Reinforcement learning based on the value function learns through updating the evaluation to obtain better execution strategies. However, it cannot handle the situation when the state space is too large or the action values are continuous. Policy-based reinforcement learning parameterizes the policy, uses optimization-related methods to construct an optimization problem with constraints, and then performs a global search to find the optimal policy. The policy selection can be shown as follows.

$$\pi\left(a_t \mid s_t, \theta\right) \tag{4.8}$$

where θ is the policy parameter, π is the action selection strategy of the agent under s_t. The corresponding probability is calculated for each action of the output, and the action is then selected by probability to maximize the reward. Define the objective function as

$$J(\theta) = \sum d^{\pi_\theta}(s) V^{\pi_\theta}(s) \tag{4.9}$$

where $d^{\pi_\theta}(s)$ is the distribution of the Markov chain for states generated by policy π_θ.Optimization of the policy parameters seek the maximum value of the objective function, using the policy gradient for optimization is a common method. Then

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta(\theta)}(s,a)\right] \tag{4.10}$$

after obtaining the determined optimization policy parameters, the policy function can be applied to determine the action value. Policy-based reinforcement learning algorithms are shown in Figure 4.4.

Policy-based reinforcement learning algorithm

1 Initialize θ;
2 **while**
3 Calculate the objective function and find the gradient $\nabla\theta J(\theta) = E_{\pi\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi\theta(\theta)}(s, a)]$;
4 Update the policy parameter θ according to the gradient;
5 Repeat until the optimal solution is obtained and the loop ends;
6 end

Figure 4.4: Policy-based reinforcement learning algorithm

4.1.3. Actor–Critic Methods

Actors will follow the critics' comments and constantly evolve their acting skills to make them progress better. Actor–critic methods are the natural extension of the idea of gradient-bandit methods to Temporal-Difference (TD) learning and to the full reinforcement learning problem. On the one hand, the value function is evaluated, and on the other hand, the policy itself is updated and verified. That is, the policy value is estimated through Critic as

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a). \tag{4.11}$$

Learning the Actor-Critic(AC) policy gradient can be divided into two parts: Actor and Critic. The estimate function is called the critic because it criticizes the actions done by the actor.The policy structure is called the actor because it is used to select actions. The policy gradient is

$$\begin{aligned} \nabla_\theta J(\theta) &\approx E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)\right] \\ \Delta\theta &= \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a) \end{aligned} \tag{4.12}$$

Actor-Critic, which combines value-based reinforcement learning and policy-based reinforcement learning, has been shown in related studies to learn better policies in a larger action space. The schematic diagram is shown in Figure 4.5.

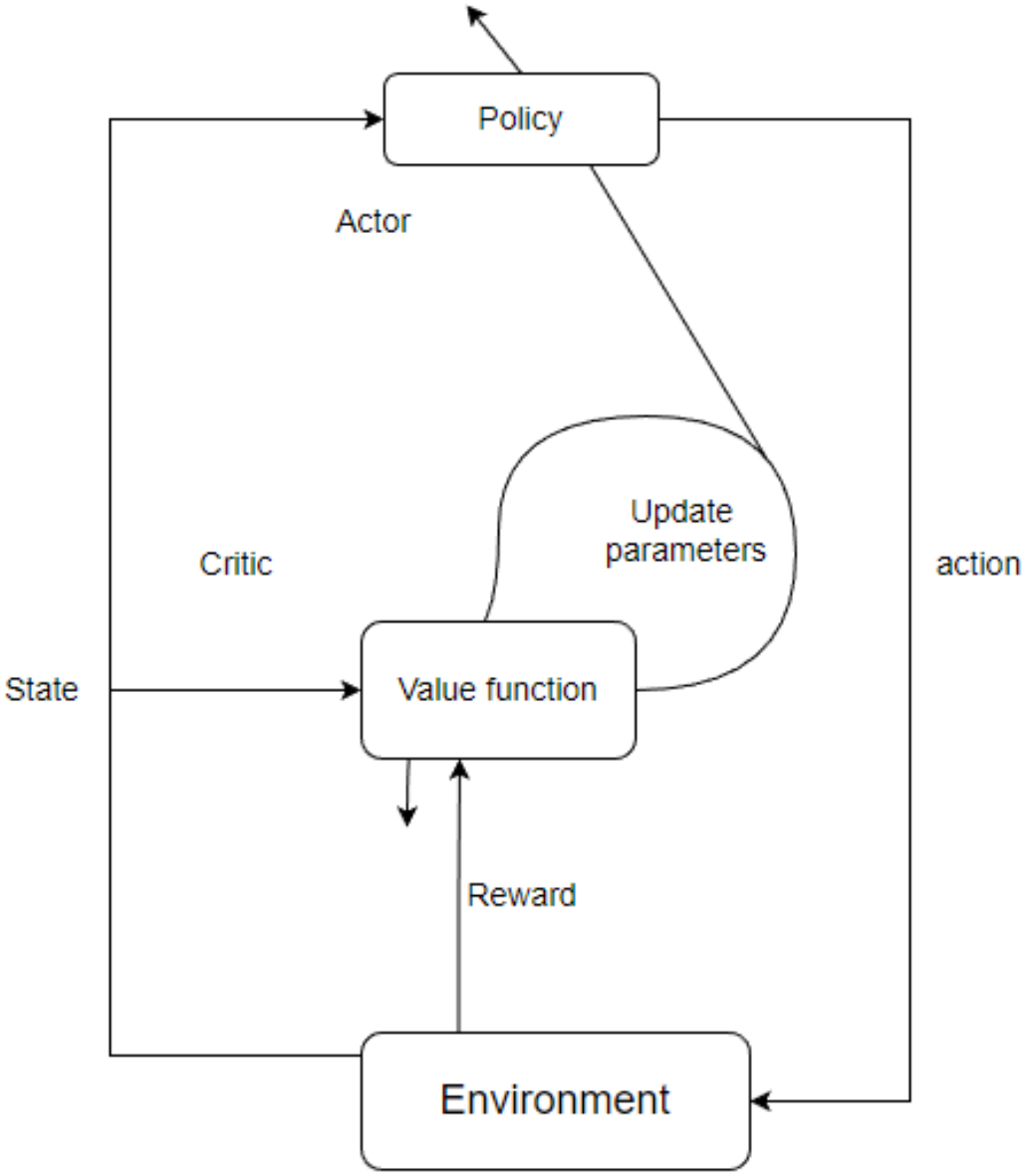


Figure 4.5: The actor–critic architecture

The Actor network inputs the environment state values, outputs the action values to the environment, and provides them to the Critic network as input. The Critic network inputs reward and state values evaluate the Actor-network output and correct and update reward. The Critic network inputs reward and state values, evaluate the

Actor network's output and corrects and updates the Actor network and its network parameters to enable the agent to achieve the preset goals.

Section 4.2

Introduction to deep learning

Deep learning is mainly about the representation learning of data [29]. Data features can be presented in various ways, such as pixel intensity values of an image or as some edges or different forms of domains [30]. The advantage of deep learning is that it can be stratified to extract features instead of manual acquisition. Representation learning mainly aims to find better methods, build better mathematical models, and learn the representation methods from a large amount of unlabeled data. The foundation of deep learning is a decentralized representation of machine learning. The decentralized representation assumes that the data results from the interaction of different sides. It is further assumed that this side interaction process can be divided into many layers, representing multiple data abstraction levels, with varying numbers of layers and sizes of layers. The number of layers and the size of the layers vary, and different levels of abstraction are used. Deep learning is based on hierarchical abstraction, where concepts are learned sequentially from higher to lower levels. Deep learning has been most successful in its application to artificial neural networks. Deep learning models are often constructed by combining multiple layers of nonlinear arithmetic units, which use lower-level outputs as higher-level inputs, and in this way, automatically learn from a large number of samples of training data. Deep conceptual learning means that it has multilayer perceptrons [31].

Neurons are the most basic structures in neural networks and the basic operational units of neural networks, which are based on the information propagation mechanism of neurons in biology. It is now a very common "M-P neuron model," as shown in

Figure 4.6:

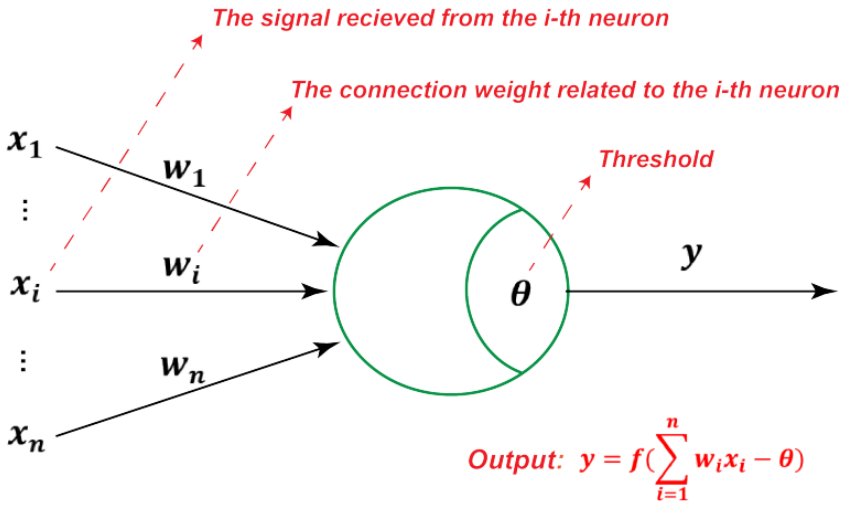


Figure 4.6: M-P neurons

The neuron output is:

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i - \theta\right) \tag{4.13}$$

The function f is the activation function, which can be expressed as a step equation. When greater than a threshold value, the neuron is activated; otherwise, the neuron is inhibited. Common activation functions are mainly

(1)The Sigmoid function, which mathematical form

$$f(z) = \frac{1}{1 + e^{-z}} \tag{4.14}$$

The output is between 0 and 1, regardless of the value of the input. If the output is less than 0, 0 is output; if the input is much greater than 0, 1 is output.

(2) The tanh function, which takes the mathematical form is:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.15}$$

Its value range is [-1,1], which is good when the features are significant and can be continuously expanded during the training-learning process.

(3) The Relu function, which has the mathematical form of

$$\mathrm{Re}\,lu(x) = \max(0, x) \tag{4.16}$$

The Relu function can effectively alleviate the gradient disappearance problem during training. However, it may cause some of the inputs to fall into the hard saturation zone, and the weights cannot be updated. There are many other activation functions, each with its advantages and disadvantages. The actual use of these activation functions can be chosen flexibly according to different scenarios.

As mentioned above, the neurons are weighted to the input and then processed by the activation function to solve the linear classification problem. However, in practice, many problems are not linearly separable problems. In order to solve the nonlinear problems, the neuron layers are stacked together to form a neural network structure, specifically by adding between the input and output layers. The activation function needs to process the hidden layer and the output layer. As shown in Figure 4.7, a shared neural network has a hierarchical structure in which all the neurons in two adjacent layers are connected, and the neurons in the same layer are not connected.

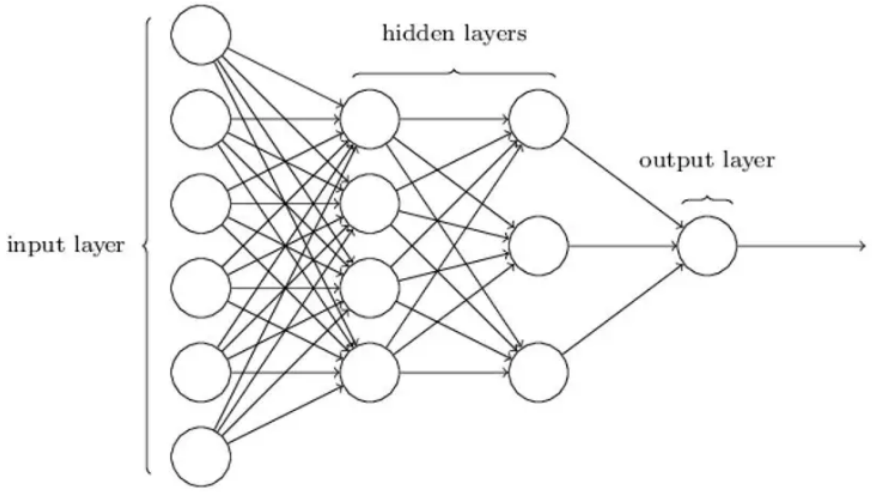


Figure 4.7: Artificial Neural Network model

To determine the weights and thresholds of the many connections in a neural network, researchers have proposed the error backpropagation (BP) algorithm, which connects multiple layers of neurons to form a neural network.The BP algorithm is an algorithm that updates the neural network connection weights and thresholds to reduce the output error of the neural network, assuming that the samples (x_k, y_k) are input to the neural network, the sample mean square error is

$$E_k = \frac{1}{2}\sum_{j=1}^{l}\left(\hat{y}_i^k - \hat{y}_j^k\right)^2. \tag{4.17}$$

then the neural network parameters are adjusted in the direction of

$$v = v - \eta\frac{\partial E_k}{\partial v} \tag{4.18}$$

where η is the learning rate between 0 and 1, which controls the update step in each iteration, and The parameters are updated using the gradient. In the threshold update, the current layer's threshold gradient depends on the next layer. In contrast, the current layer's connection weights depend on the threshold gradient of neurons

in the current layer and the output of neurons in the previous layer.

If a neural network has enough hidden layers, then it can fit any function, but because of its powerful fitting ability, overfitting often occurs, resulting in a low error in the training set, but the error is likely to be larger in the test set. There are currently two approaches to this problem. We divide the training samples into training and validation sets and use the training set to calculate the gradient, update the connection weights and thresholds, and use the validation set to estimate the error. Suppose the error in the training set decreases, but the error in the validation set increases. In this case, the training is terminated, and the connection weights and thresholds with the lowest error in the validation set are used. There is also the regularization method, where a part used to describe the complexity of the network is added to the error objective function, then the kth training sample's error can then be expressed as

$$E = \lambda \frac{1}{m} \sum_{k=1}^{m} E_k + (1 - \lambda) \sum_{i} v_i^2 \tag{4.19}$$

where here λ is called the regularization factor, which performs this on the empirical error and network complexity.

Deep reinforcement learning combines the decision-making capabilities of reinforcement learning described above with the representation capabilities of deep learning for feature extraction, allowing intelligent systems to input information about the environment and directly output action values that can act on the environment. In robot motion control, for example, the robot relies on an intelligent system that can directly input its positional and environmental information and output key action values for robot motion, such as robot motor drive torque and angle. There are many types of deep reinforcement learning algorithms. They can be divided into model-based deep reinforcement learning, model-free deep reinforcement learning, and model-free

deep reinforcement learning. Among them, model-free deep reinforcement learning can be divided into value function-based deep reinforcement learning, such as Deep Q-Network(DQN), and policy-based deep reinforcement learning, such as Proximal Policy Optimization(PPO) and Deep Deterministic Policy Gradient(DDPG).This thesis's robot motion control needs continuous motion output. Therefore, strategy-based deep reinforcement learning is selected for robot motion control strategy learning.

4.2.1. Deep reinforcement learning is used for robot motion control

Chapter 2 of this thesis investigates the implementation of motion control of a hexapod robot from the perspective of foot-end trajectory planning. Chapter 3 examines robot motion control from a bionic perspective. The two previous approaches require separate modeling of the environment or high-level processing of the environment and robot pose information to give external inputs. However, the external environment is complex and variable, the modeling of the environment cannot cover all situations, and the separation of the control strategy and robot sensors may lead to the high complexity of the robot control strategy. Deep reinforcement learning-based robot motion control gives important information, such as joint commands for robot motion control directly based on external sensor information and positional information to achieve end-to-end robot motion control.

In 2018, Peng et al. [32]used the reinforcement learning algorithm PPO in a simulation. Humanoid and cheetah structures were trained in the environment, and surprising results were achieved. Peng et al. performed the initialization operation on the reference action of the agent during the agent training so that the agent starts from the existing strategy and trains the strategy by imitating the position and speed of the agent joints in the existing strategy. At the same time, new tasks are added to guarantee that the agent can adapt to the complex external environment when training the existing strategies. Subsequently, the complex actions are decomposed

in different stages as the conditions for agent initialization. Objectively, the complex task is decomposed to reduce the learning task difficulty. In order to reduce the learning cost and improve learning efficiency, Peng et al. have made early termination of some states in the training process to prevent the robot from entering the wrong strategy during the training process. Including early termination can make the agent reduce the trial and error cost in the training process and avoid failure as much as possible.

Harrnoja et al. [33] proposed the Soft AC algorithm. The algorithm is dedicated to increasing the information entropy of the strategy itself and thus expanding the exploration space of the robot control algorithm. The information entropy is used to measure the information content of the reinforcement learning strategy and consciously increase the information entropy of the strategy. When the information entropy is high, it indicates that the action strategy is informative, i.e., the strategy exploration ability is strong.

Section 4.3

Introduction to MARL

Multi-agent reinforcement learning (MARL) [34] is a subfield of RL that focuses on developing algorithms and techniques for learning in environments where multiple agents interact with each other. In a multi-agent system, each agent makes decisions based on its own observations and objectives, as well as the behavior of other agents in the system. The goal of MARL is to design algorithms that enable agents to learn how to cooperate or compete with each other in order to achieve a common goal, or to optimize their own individual objectives while taking into account the behavior of other agents [35]. MARL is relevant in a wide range of domains, including robotics, game theory, economics, traffic control, and more. One of the key challenges in

MARL is that the environment in which agents operate is no longer stationary, as the actions of one agent can affect the environment and therefore the observations and decisions of other agents. This means that each agent must learn not only its own optimal policy, but also how to respond to the actions of other agents and how to adapt its behavior over time. MARL algorithms can be categorized into two main types: cooperative and competitive. In cooperative MARL, the agents work together to achieve a common goal, while in competitive MARL, the agents compete with each other to maximize their own objectives. Some popular algorithms for MARL include Q-learning, actor-critic, and deep reinforcement learning algorithms such as deep Q-networks (DQNs) and policy gradient methods. MARL has seen applications in various domains such as multi-robot coordination, autonomous driving, and game AI, among others.

Cooperative Multi-Agent Systems (MAS) refer to a group of agents that work together to achieve a common goal. In cooperative MAS, the agents coordinate their actions to achieve a shared objective that cannot be accomplished by any individual agent alone. Cooperative MAS has applications in various domains such as robotics, transportation, and healthcare. In cooperative MAS, the agents are interdependent, which means that the actions of each agent affect the state of the environment and the actions of other agents. This makes learning and coordination between the agents a challenging problem. Some of the key issues that arise in cooperative MAS are communication, coordination, and cooperation.

Communication: Agents in a cooperative MAS must communicate with each other to coordinate their actions. Communication can be explicit or implicit, depending on the design of the MAS. Explicit communication involves exchanging messages between agents, while implicit communication involves observing the actions of other agents.

Coordination: Coordination refers to the process of ensuring that the actions of the agents are consistent with the shared objective of the MAS. This requires agents to have a common understanding of the objective and to take actions that are complementary to each other.

Cooperation: Cooperation refers to the process of working together to achieve a shared objective. Cooperation requires agents to act in a way that benefits the group as a whole, even if it is not in their individual best interest.

Several techniques have been proposed for learning in cooperative MAS, including joint action learning, distributed learning, and centralized learning. Joint action learning involves training a single agent to take actions that are coordinated with other agents. Distributed learning involves training each agent independently, while centralized learning involves training a global model that controls the actions of all agents. In conclusion, cooperative MAS is an important area of research in multi-agent systems. Developing effective learning and coordination techniques for cooperative MAS has the potential to enable more efficient and effective collaboration between agents in a wide range of domains.

In the multi-agent domain, deep reinforcement learning faces significant challenges [36] [37]: First, the exponential growth of the action space due to the increase in the number of agents makes the computation of Q values very difficult; Second, the different goals and tasks of the agents and their interactions with each other lead to difficulties in determining the target reward, which has a serious impact on the convergence of the algorithm; Third, each agent's exploration will cause changes in the environment, and will also affect the policy choices of other agents, resulting in a slow learning rate of the algorithm.

Researchers have proposed multi-agent deep reinforcement learning algorithms to address the above mentioned problem. These algorithms can be classified into

association-free, rule-based communication, and value function decomposition-based. The association-free algorithm [38] extends the single-agent deep reinforcement learning algorithm to a multi-agent environment where each agent interacts independently with the environment during training without communication. The rule-based communication algorithm [39] introduces a communication mechanism between agents, where each agent learns based on the messages other agents communicated during training. The value function factorization algorithm aims to decompose joint rewards into a specific combination of individual agent rewards through the imposition of constraints to achieve global optimality when individual optimality is attained. During training, each agent makes decisions based on local observations and updates the network using the joint rewards obtained from the value function factorization algorithm.

The association-free algorithm [40] [41] is relatively simple to implement. However, due to the lack of communication between agents, the non-smoothness of the environment can adversely affect the algorithm's learning rate and efficiency. On the other hand, the rule-based communication algorithm [42] [43] can establish a better joint policy, but it requires a larger number of parameters to create a communication channel and a more complex structural design.

Compared to the first two types of algorithms, the value function factorization algorithm [44] [45] provides a solution to the exponential growth of action space by decomposing the joint value function into individual value functions. It also ensures algorithm convergence since each agent only selects actions based on its local observation and solves the difficulty of determining rewards caused by interaction between agents. Additionally, this algorithm has high learning efficiency, can learn the best joint policy faster, has a relatively simple structural design, and is highly scalable [46] [47]. For instance, QMIX [48] uses a monotonicity constraint to decom-

pose the value function, thereby improving the agent's performance on challenging StarCraft 2 micromanagement tasks. QPLEX [18] decomposes the value function using a duplex duel structure, which enhances the algorithm's performance in both online and offline data collection and achieves high sample efficiency.

4.3.1. MARL based on value factorization

Deep reinforcement learning using value decomposition for multiple agents typically involves two types of value functions: joint action value functions and individual action value functions. The joint action value function is shared among all the agents and is used to update the network, while the individual action value function is the Q-value obtained by each intelligence based on their local observations. This Q-value guides each intelligence in selecting actions. However, this approach is only suitable for multi-intelligence reinforcement learning tasks in cooperative environments, as the joint action value function is shared among all the agents.

Decentralized Partially Observable Markov Decision Process(Dec POMDP. Reinforcement learning (RL) solves the fundamental problem of ranking intelligent decisions in environments with discrete time steps to maximize long-term cumulative returns and is often formalized as a Markov Decision Process (MDP). The locomotion of a hexapod robot on an unstructured ground generalizes the single-agent MDP to a decentralized partially observable semi-Markov decision process (Dec-POMDP) [49] described by $(N, S, A, P, R, Z, O, \gamma)$ where

- N is the number of agents
- S is a set of global states and $s \in S$.
- A is the joint action space, defined as $A = A_1 \times A_2 \times \cdots \times A_N$.

- P is the transition model and $P(s_{t+1}|s_t, a_t)$ that that denotes the probability of next state s_{t+1} when all agents select joint action $a_t \in A$ along with state s_t.
- R is the reward function, which returns one total credit for all agents when they select $a_t \in A$ in state s_t.
- Z is the partially observable setting.
- O is the observation functions where each agent i receives an individual partial observation $o_i \in Z$ by means of the observation probability function $o(o_i|, a_i)$.
- $\gamma \in [0, 1]$ is the discount factor.

We use $\boldsymbol{\tau}$ to denote the history of joint action-observation. The action observation history τ_i is maintained for each agent i, and based on this history, the agent constructs its personal policy $\pi_i(a|\tau_i)$. The objective function is to find a joint policy $\pi = \langle \pi_1, \cdots, \pi_n \rangle$ that maximizes the joint value function $V_\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi]$ and joint action-value function $Q_\pi(s, a) = r(s, a) + \gamma E_{s_0}[V_\pi(s_0))]$. The goal is to find the optimal policies that maximize cumulative reward over time.

Centralized Training and Decentralized Execution (CTDE). Cooperative multi-agent deep reinforcement learning employs a single joint reward signal to train the network, which poses a significant challenge to the learning process. Additionally, the coordination problem between agents cannot be effectively addressed by independent Q-learning or fully centralized learning methods, owing to the partial observability of the environment.

Independent Q-learning optimizes the reward function by training an independent Q-learner. In this method, each agent treats the other agents as part of the environment, so each agent's policy and reward are influenced not only by the environment but also by the behavior of other agents, and there is the problem of false reward [50].

Fully centralized learning [51] merges the individual agents' action space and state space into a joint action space and joint state space, resulting in a single-agent problem. This approach partially addresses the consistency and convergence issues. However, as the number of agents increases, the action space undergoes an exponential explosion. Furthermore, in situations where one agent discovers an improved policy, the other agents may not learn from it unless their exploration positively affects it, leading to the occurrence of sluggish agents.

Therefore, researchers compromised between these two extreme approaches and proposed the concept of centralized training and decentralized execution to solve the problem in multi-agent tasks.

Centralized training: Unconstrained communication between agents during training, and a joint action value function $Q^{\approx}(s_t, A_t)$ is used to learn the algorithm. During training, the algorithm has access to the action observation history $\boldsymbol{\tau}$ and the global state s of all agents.

Decentralized execution: During execution, communication between agents is restricted to select actions only by computing individual action value functions $Q^a(z, a_t)$ from their own observation histories τ_i, without considering the actions of other agents.

4.3.2. MARL is used for robot motion control

MARL has been used for the control of legged robots in recent years, providing a powerful solution to adapt to new environments and learn from experience. One example of MARL for legged robots is the work of Hwangbo et al. [4], where a policy network for a quadruped robot was initially trained in simulation and then transferred to the real robot for further adaptation. To address the overfitting issue in reinforcement learning, a hierarchical structure was proposed in [52], which allowed the system to switch between different subtasks and behaviors, such as obstacle avoidance, wall

following, and straight walking. Results showed that the approach was effective in adjusting to severe interventions, such as the loss of a leg. However, this method falls short of exhibiting a level of adaptivity within a specific behavior as that observed in organisms with the ability to handle broad variations within a defined context, such as climbing through an object with sparse and unpredictable support. Guillaume et al. [53] proposed a MARL algorithm for decentralized control of a hexapod robot for dynamic walking tasks. The control of the robot was distributed among multiple agents, such as individual limbs or sensors, to improve adaptability and robustness in complex and dynamic environments, rather than being centralized in a single control unit. The task of stabilizing a hexapod robot's body is challenging due to its non-linear and highly dimensional nature. The multiple paths for stability make it difficult for multiple agents to reach agreement and collaborate effectively without communication, complicating the task further. In contrast, we focus on adaptability, including improved stability and robustness, enhanced coordination and cooperation between the legs, and the ability to adapt to changing environments.

Section 4.4

MARL using QPLEX

Our hexapod robot is inspired by the field of MARL to enhance its decision-making abilities. By incorporating principles from MARL, the hexapod can coordinate its movements and optimize its actions in real-time for more efficient and effective performance. The decentralized control architecture, which has already been successfully realized in previous work, shows robustness in dealing with unpredictability and minor disturbances. However, scaling up to more challenging tasks, such as walking on uneven terrain or climbing, is difficult as the controllers in previous work are handcrafted taking inspiration from biology [54].

We propose a learning-based approach using the MARL QPLEX method to overcome this challenge. The Q-Learning algorithm and the "DUPLEX" and "DUELING" components improve coordination and cooperation between the legs, enabling the hexapod to adapt and improve its behavior over time. Through terrain curriculum learning, the hexapod can traverse unstructured terrain with steady and fast walking patterns. Our approach provides a new perspective on hexapod robot control and has the potential to lead to more efficient solutions for controlling multi-legged robots in complex environments.An overview of the method, which is derived from the general QPLEX algorithm presented in [47] is given in Fig.4.8. This section describes each element of the system.

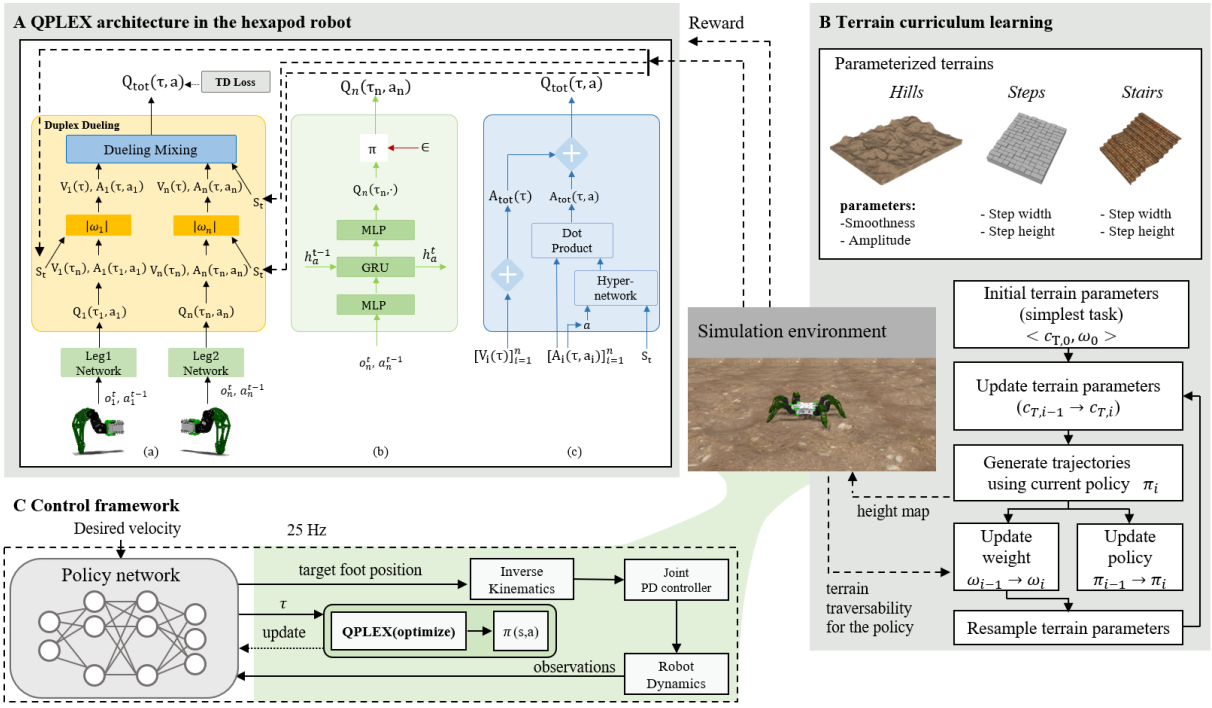


Figure 4.8: Overview of the presented approach. (A) The overall QPLEX architecture(a), the agent network(b), and the dueling mixing network(c). (B) Terrain curriculum learning synthesizes maps with a suitable level of difficulty. (C) Motion controller frame (after [47]).

4.4.1. The hexapod platform

The hexapod platform is chosen for this study due to its versatility and stability. A hexapod robot is statically stable and remains upright with little or no active control due to redundancy, i.e., six legs, making it ideal for exploring unstructured terrain. Each leg has three links: the hip, femur, and tibia, with each containing a motorized joint. These 18 joints are usually actuated using a dedicated servo system. Our platform model assumes position servos because they have an internal feedback control loop for low-level control. The joint actuator also needs information about the current joint motor, e.g., the instantaneous current or torque. On such a platform, it is likely that unrealistic or impractical leg positions, such as the tibia touching the hip, can occur. Therefore, limits are placed on joint ranges. We import the robot model into the Gazebo simulation environment [55] and use ROS to provide modularity, interoperability, simulation support, and scalability for developing and testing simulated robots [56]. ROS is run at a fixed rate of 25 Hz to ensure consistency of sensor data and actuator data.

The robot is programmed to walk in various conditions, including uneven terrain, slopes, and obstacles. The robot is assumed to be equipped with six inertial measurement units(IMU), encoders on each joint and contact force sensors on each foot to collect data on its movement and performance during learning trials (Fig.4.9).

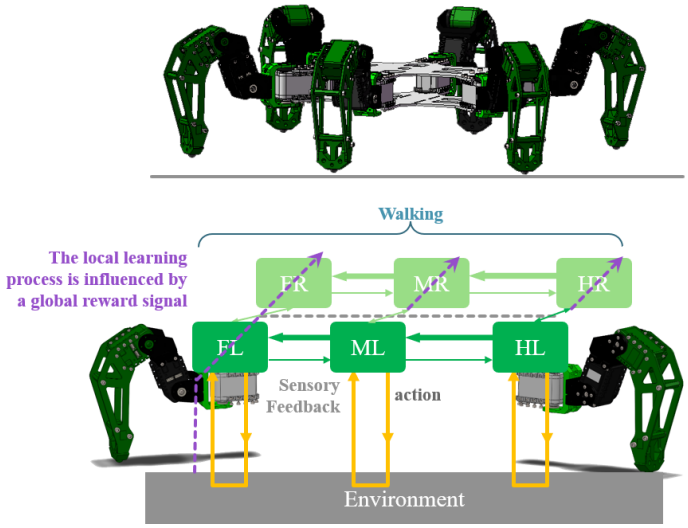


Figure 4.9: This decentralized architecture is used on a hexapod robot and learning of the six local control modules is driven by a reward signal as in MARL.

4.4.2. Reset Condition

In the context of reinforcement learning for the hexapod robot, it is crucial to reset the robot to prevent behaviors that violate its mechanical constraints and impact the sensor data. To achieve this, four conditions are defined for resetting the robot based on pose angle, body center height, total motion time, and body collision judgment and constraint. The pose angle constrains the roll angle to ± 0.4 rad, pitch angle to ± 0.3 rad, and yaw angle to ± 0.3 rad. The body center height is defined as the height range of the center of mass (COM), which is restricted to [0.15, 0.3]. The maximum learning time for each episode is set to 40 seconds, and the reward is reset to zero if the body (excluding legs) comes into contact with the ground.

4.4.3. Reward function definition

A hexapod robot moving on unstructured ground aims to produce a rhythmic gait while satisfying all kinematic and dynamic constraints. The reward function is as

follows:

$$r_t = r_{mf} + r_{fl} + r_{ft} + r_{cp} + r_{dx} \tag{4.20}$$

Each element of the reward function is provided in Table 4.1. $c_1, c_2, c_3, c_4, c_5, c_6$ represents foot contact of front left, middle left, hind left, front right, middle right, hind right legs respectively. t_{stance} denotes the cumulative length of time that a particular foot of the robot touches the ground. vel_d is the desired velocity. θ_{cp} represents the pitch angle of the body. α_{cp} represents the roll angle of the body. d_x is the vertical distance of body from x-axis. The role of r_{mf} is to maintain a desired footstep frequency and keep a consistent and steady pace. r_{fl} enables the hexapod to place its feet accurately and in a controlled manner, while r_{ft} is enables the hexapod to move its feet smoothly and without sudden changes in trajectory. r_{cp} is designed to maintain a balanced COM and stable posture, and r_{dx} is designed to encourage the hexapod to move in a manner that conserves energy and minimizes unnecessary movement. By combining these various rewards, the overall reward function aims to provide comprehensive control of the hexapod's gait and encourage the development of rhythmic and efficient movement.

In reinforcement learning, the reward function plays a crucial role in guiding the learning process by providing feedback to the agent. However, designing an effective reward function can be challenging, particularly in complex environments like hexapod robot locomotion. The approach described here provides a means to investigate reward shaping techniques that can be used to achieve alternate gaits, encourage exploration, or set alternate objectives. For example, if a leg motor fails, the reward function could be shaped to avoid use of the failed leg. Alternate gaits, such as jumping or trotting, could be achieved through shaping r_{mf} and r_{fl}. Shaping the reward function can encourage the robot to minimize energy use.

Table 4.1: REWARD COMPONENTS

r_{mf}	$\|3c_1 - c_2 - c_4 - c_6\| + \|2c_2 - c_3 - c_5\| + \|2c_3 - c_4 - c_6\|$ $+ \|c_4 - c_5\| + \|c_5 - c_6\| - \|2c_1 - c_3 - c_5\| - \|c_3 - c_5\|$ $- \|2c_2 - c_4 - c_6\| - \|c_4 - c_6\|$
r_{fl}	$\sum_{i \in N} \|t_{stance}(i) - 0.4\|$
r_{ft}	$\frac{vel_d + 1}{vel_d \|x_{vel} - vel_d\| + vel_d} - \frac{1}{vel_d}$
r_{cp}	$\sqrt{\theta_{cp}^2 + \alpha_{cp}^2}$
r_{dx}	$\|d_x\|$

4.4.4. QPLEX architecture in the hexapod robot

The main contribution of this thesis is that of applying a QPLEX architecture modified from [47] to hexapod robot MARL as shown in Fig 4.8.A. The robot has six legs, each with three degrees of freedom, and is designed to navigate complex environments. The goal is to coordinate the movements of the six legs to achieve stable and efficient locomotion.

To apply the QPLEX architecture, the hexapod robot is modelled as a multi-agent system where each leg is treated as a separate agent; each leg has a local control module. The state of each leg is represented by its joint angles, velocities, and ground contact, and the action of each leg is represented by the control signals applied to its drive system. Each element of the QPLEX architecture is described as follows:

Individual action-value function: the individual action-value function of each leg consists of an RNN network (Q network), as in Fig 4.8.A(b) that has three layers: the input layer (MLP multilayer neural network) → intermediate layer (GRU gated recurrent neural network) → output layer (MLP multilayer neural network)). The

inputs to each leg controller are local information from that leg and previous action signals. The Q network is trained to estimate the expected cumulative reward associated with each action.

Duplex Dueling component: The double-sided duel component is used to compose a single action-value function into a joint action-value function to satisfy the dominance-based individual-global-max(IGM) constraint, as shown in Fig 4.8.(c). The component consists of a conversion network module and a dual hybrid network module. The conversion network module converts the individual duel structures conditional on the joint action-observation history. The dual hybrid network module uses the output of the transformation to generate the value of the joint action-value function.

Control signal generation: The joint action-value function is used to determine the control signal for each leg. The control signal is computed by selecting the action that maximizes the joint action-value function at each time step.

Reinforcement learning: The QPLEX algorithm is trained in a simulated environment using reinforcement learning. The robot is rewarded for maintaining stability and moving efficiently. The training process is centralized, and the entire network is learned end-to-end to minimize temporal difference (TD) loss. QPLEX's policy network is based on a centralized training paradigm, where a central critic network estimates the joint action-value function, and each agent has its own local actor network that selects its individual action based on its local observation of the environment and communication with other agents. A neural network approximates the learned policy function. The policy networks consist of two hidden layers of 64 units each with tanh activation functions. After training, the duplex dueling component is removed, and each leg is controlled using its individual Q-function based on the local action-observation history. The hexapod robot is tested in various simulated

environments, and its performance is evaluated by comparing its movements with those generated by other algorithms.

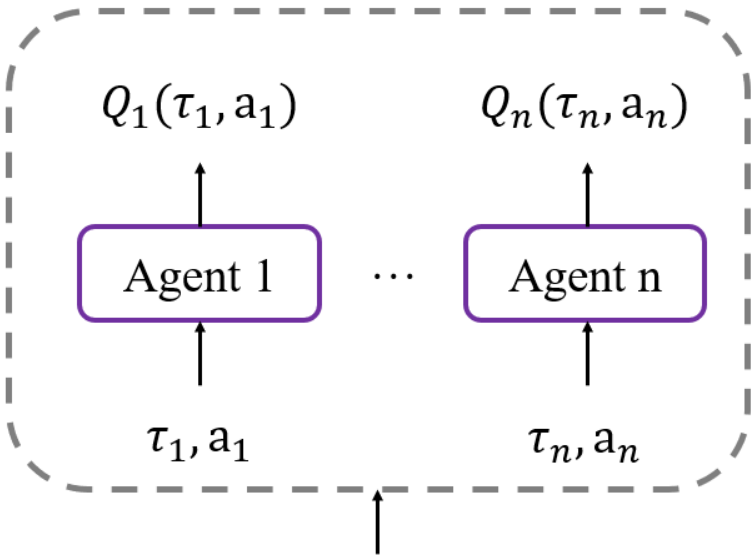


Figure 4.10: Fully-decentralized MARL method.

The fully-decentralized method is used as a baseline for comparison with the QPLEX algorithm: for this baseline, each leg of the hexapod robot selects its actions independently (Fig. 4.10), based on its local observation history, without considering the actions of other legs. Each leg is equipped with a Q-network which is optimized using the Proximal Policy Optimization (PPO) [57] algorithm to estimate the expected cumulative reward associated with each action. The PPO algorithm is selected due to its successful performance on continuous tasks without intensive hyperparameter tuning, as well as its relatively low sampling complexity for identifying usable policies. During the execution, the Q-network of each leg is used to determine the control signals for the corresponding leg based on the current state of the leg and its previous hidden state. The fully-decentralized method does not consider the interactions between the legs and thus may not result in coordinated movements. However, it is simple to implement and does not require communication between legs. Our experiments investigate the QPLEX algorithm's effectiveness in

improving the coordination and efficiency of hexapod locomotion compared to the fully-decentralized method. The experiments also examine the role of the reward function in promoting rhythmic and efficient movement in the hexapod robot.

4.4.5. Terrain curriculum learning

Our method is inspired by Automatic Course Learning (ACL) [58] [59]. In this approach, robots are trained to learn how to navigate various terrains, starting with simple and predictable surfaces and gradually increasing in difficulty as it improves. Our approach also implements a training course that progressively modifies the environment parameters, and the hexapod robot terrain course learning aims to enable the robot to navigate over unstructured surfaces successfully. Fig.4.8B and Fig. 4.11 shows the terrains used for our training environments, such as hills, steps, stairs, and terrain parameters generated by $c_T \in C$. We evaluate c_T by the traversal ability of the generated terrain and use importance weights ω to determine the relative priority given to different tasks in the course to improve learning efficiency.

The training process typically begins by presenting the robot with a series of simple and predictable terrain types, such as flat surfaces or gentle slopes. The robot is then trained to navigate these surfaces using its six legs, and the difficulty is gradually increased by introducing more complex terrain types, such as rough or uneven surfaces. As the robot encounters new terrain types, it learns through trial and error to adjust its movements and balance to successfully navigate the terrain. Feedback from the robot's sensors and its success or failure in completing a task is used to update its learning algorithms, allowing it to continuously improve. One important aspect of terrain curriculum learning for hexapod robots is that it allows the robot to generalize its experiences from one terrain type to another. For example, if the robot has learned how to navigate a particular type of rough terrain, it can apply this knowledge to navigate similar terrain types more efficiently.

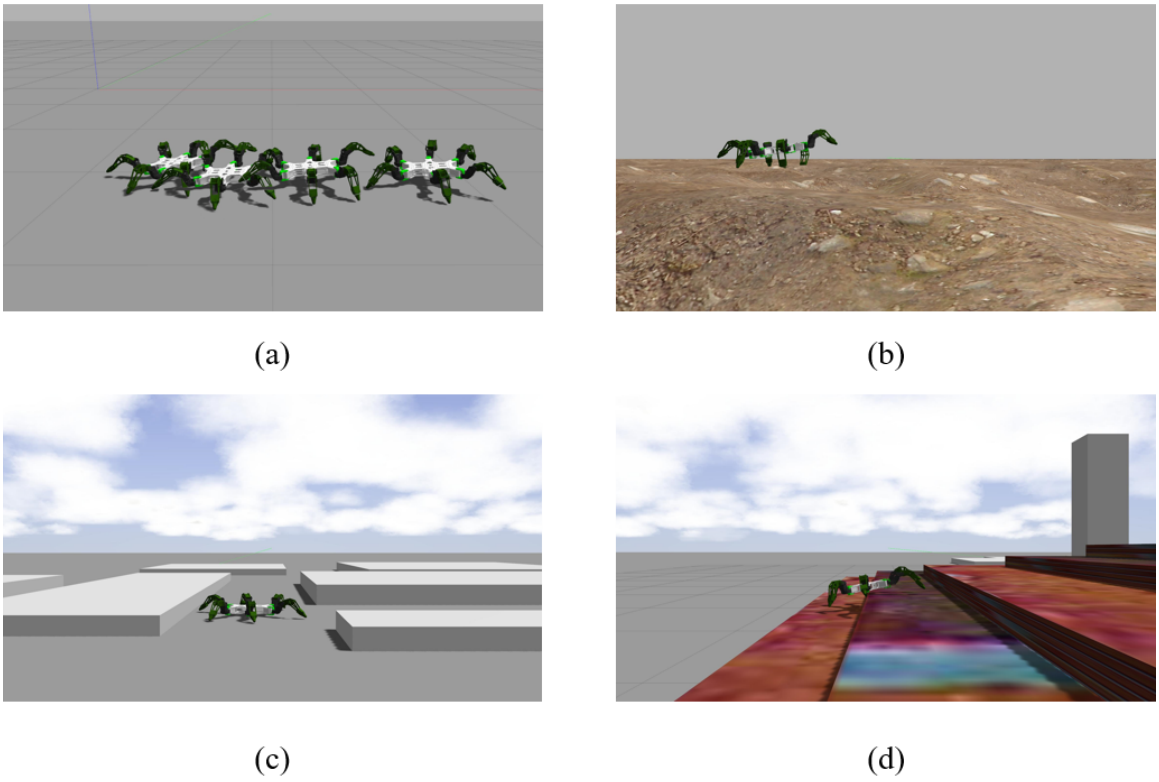


Figure 4.11: The simulated hexapod robot is visualized in different environments. Panel (a) depicts a walking sequence on flat terrain. Panel (b) illustrates the robot navigating uneven terrain conditions. Panels (c) and (d) show the robot traversing stochastic height steps and a stochastic staircase terrain, respectively.

Section 4.5

Results

The present study investigates the performance of hexapod robot motion policies trained via the proposed value function decomposition algorithm, QPLEX. The trained policies are evaluated in various simulated environments, and their effectiveness is assessed based on Mean Reward and Standard Deviation measures. To demonstrate the efficacy of QPLEX in the hexapod robot application, we trained two distinct structures, one employing QPLEX and the other employing a fully-decentralized approach. Subsequently, we conducted a comparative analysis of the learning experiments be-

tween the QPLEX and fully-decentralized structures. Our findings provide insights into the relative merits of these two structures for the hexapod robot motion control.

4.5.1. Performance of Locomotion Architectures and Reward Function Impact on Rhythmic Gaits on Flat Terrain

In the first experiment, we study learning to walk on flat terrain and producing a rhythmic gait. We compare the two architectures. Each is trained 15 times using random seeds and trained for 5000 epochs of environmental simulation.

To confirm the coordination and symmetry of the gait produced by our approach, we utilized the gathered foot contact data to derive the foot contact map. Fig.4.12 shows the foot contact pattern for robot speeds greater than 0.7m/s. Both learning methods can achieve performance in coordination and high-speed walking at a high rate. Both methods use a trained controller to implement the behavior and evaluate performance using the average reward of 100 individual episodes. Results reveal that the proposed QPLEX architecture outperforms the fully-decentralized baseline approach in achieving walking behavior shown in Fig. 4.11a and the supplemental video, with a mean reward of 717.4 and the standard deviation is 42.4 for QPLEX compared to 592.4 and a standard deviation of 86.9 for the baseline approach. The QPLEX method performs significantly better than the fully-decentralized method. The average reward for the QPLEX method is significantly higher with Cohen's D effect size (i.e., Standard mean difference) of 1.84.

Since the results show some variation, we focus on the best solution generated by running multiple seeds during the learning process. We compare the best-performing controllers in both structures and find that the standard deviations are similar in both cases. However, the distribution seemed larger for the fully- decentralized approach. Table 4.2 presents the ten best-ranked seeds, with the best-performing approach being the QPLEX architecture and QPLEX accounting for 80% of the rankings.

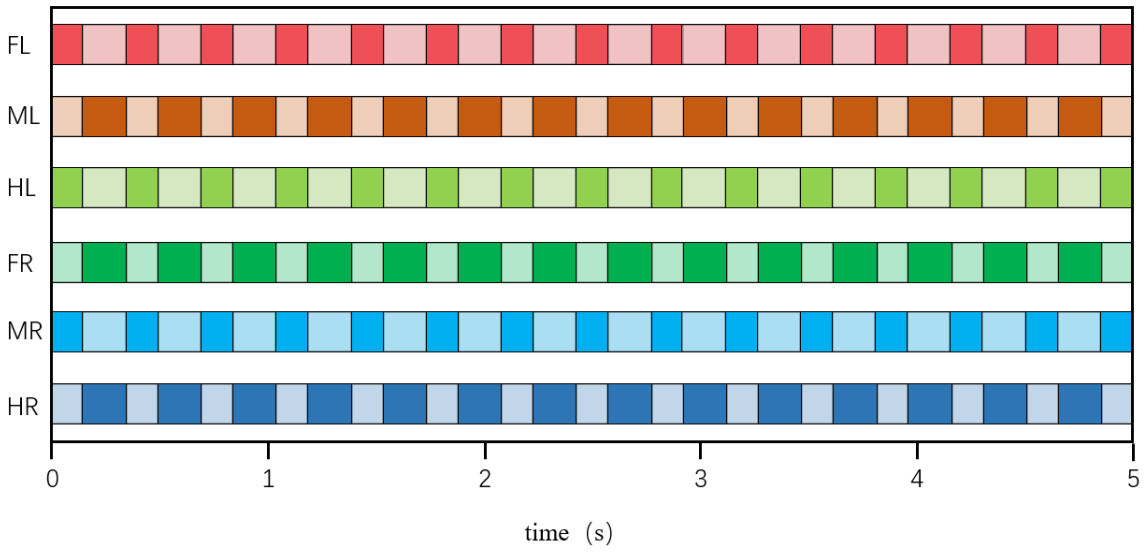


Figure 4.12: Foot contact plots, A dark color indicates that the foot is touching the ground and a light color indicates that the foot is being lifted.

In summary, as a first result, we found that a QPLEX control architecture produces high performance and well-coordinated rhythmic gaits. The QPLEX controller also shows significantly better performance at larger effect sizes.

4.5.2. Comparison of Learning

In this study, the trained architectures underwent 5000 epochs of training. Preliminary tests revealed that, at this stage, controller performance had reached convergence for both architectures. This observation supports the notion of an achievable desired velocity. Herein, we present a analysis of the training performance evolution over time, illustrated in Fig. 4.13. The average performance of both architectures was computed during the training process using 15 seeds. Despite the reward signal being smoothed through averaging, the learning process still exhibited a high degree of variability as observed in individual runs. A noticeable difference in performance between the two architectures is evident in Fig. 4.13; the QPLEX architecture achieves a reward level at after ∼2000 training epochs that is comparable to the steady-state

Table 4.2: Results of Selecting the 10 Seeds with the Best Average Reward in the QPLEX and Fully-Decentalized Architectures.

Rank	Architecture	Mean Reward	Standard Deviation
1.	QPLEX	814.2	17.7
2.	QPLEX	761.5	27.0
3.	QPLEX	754.8	14.1
4.	QPLEX	753.0	45.9
5.	Fully-decentralized	746.8	23.6
6.	QPLEX	739.4	16.3
7.	QPLEX	739.7	37.9
8.	QPLEX	729.7	28.7
9.	QPLEX	724.1	15.1
10.	Fully-decentralized	699.0	18.3

reward for the fully-decentralized architecture, and the mean reward function is approximately 21% larger.

4.5.3. Generalization to Uneven Terrain

This study examines the efficacy of diverse trained control architectures when subjected to uneven terrain conditions (see Fig. 4.11(b)). This new problem poses a challenge to the trained controller and thus provides an opportunity to test its generalization and robustness. We conducted 100 simulations in which all controllers trained on flat terrain were tested on the uneven terrain generated by the DeepMind control suite's uneven terrain generator [60]. The terrain was implemented as a height field in the simulations.

The surface smoothness was varied systematically in this study, from a smooth surface (set at 1.0, the same as in the flat terrain training) to a rockier surface (with smoothness factors of 0.9, 0.8, and 0.7), until the controller experienced difficulty generating any further motion. Both architectures displayed a decline in performance on uneven terrain, as expected (see Fig. 4.14(a)). To further examine this, 15 additional controllers were trained on uneven terrain with a smoothness of 0.8 for both architectures (see Fig. 4.14(b)). Comparing the results on uneven terrain, no significant

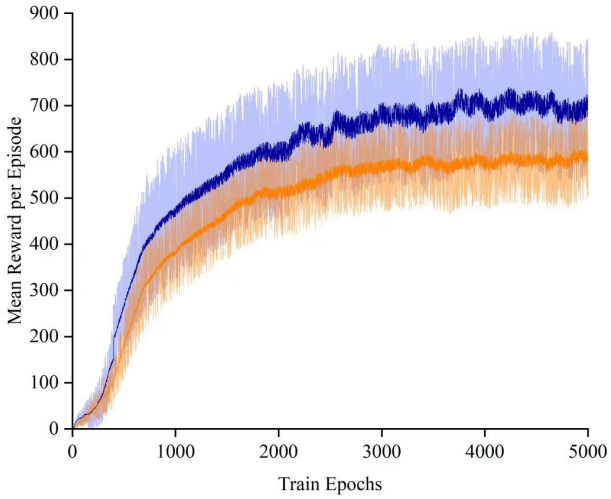


Figure 4.13: Comparison of the mean reward during training: The mean performance of the fifteen QPLEX controllers is depicted in blue, while the average reward for the baseline fully-decentralized approach is displayed in orange. The standard deviation is denoted by the shaded areas. The measure of performance is the reward obtained per episode.

difference is observed between the controllers trained on flat terrain and the specialized controllers trained on uneven terrain with smoothness of 0.8. This is observed by comparing Fig. 4.14a and b at 0.9, 0.8, and 0.7 smoothness factors, respectively.

Finally, the controllers trained on uneven terrain are evaluated on flat terrain, but they do not reach the level of the performance of the controllers trained on flat terrain, and there is a significant performance difference. The QPLEX and fully-decentralized methods showed the same result in this regard. However, the QPLEX architecture shows lower performance reduction compared to the fully-decentralized architecture in Fig. 4.14(a)(b). These results show that the scope of information used by the control architecture is important for generalization of the controller. A global, decentralized architecture is advantageous over the fully-decentralized approach, especially on challenging terrain, where more sensory information becomes beneficial and important for behavior. We evaluate performance robustness on uneven terrain. The

conclusion is further supported by the fact that the QPLEX approach significantly outperforms the fully-decentralized approach on uneven terrain.

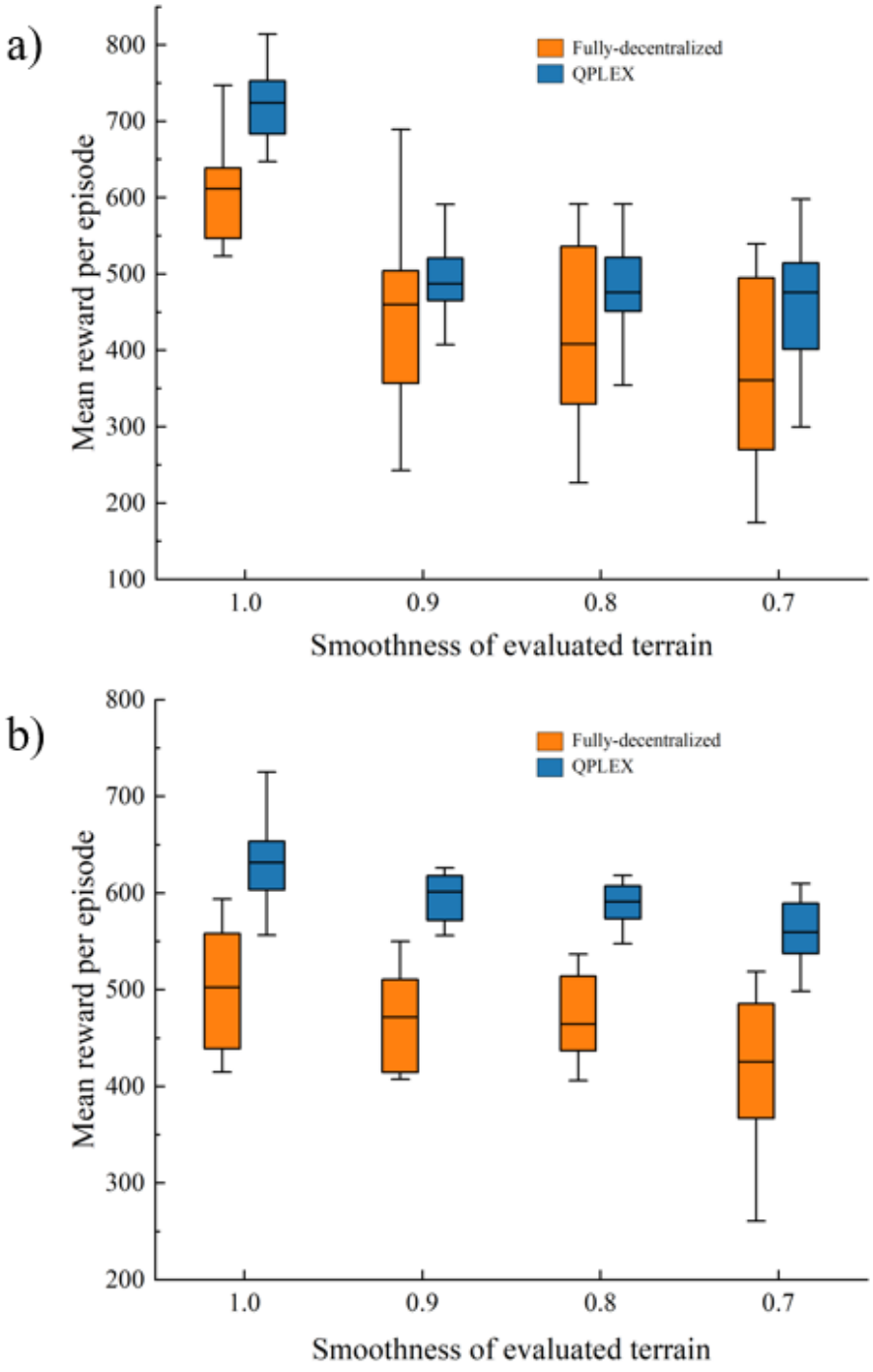


Figure 4.14: Box plots representing the evaluation of diverse terrain types across two distinct training conditions: (a) training on flat terrain and evaluation on progressively rough terrain (where smoothness of 1.0 denotes flat terrain, and 0.0 indicates very rugged terrain), target velocity is set to 2.0 and (b) training on uneven terrain with a smoothness value of 0.8, followed by evaluation on increasingly uneven terrain.

4.5.4. Importance of the terrain curriculum learning

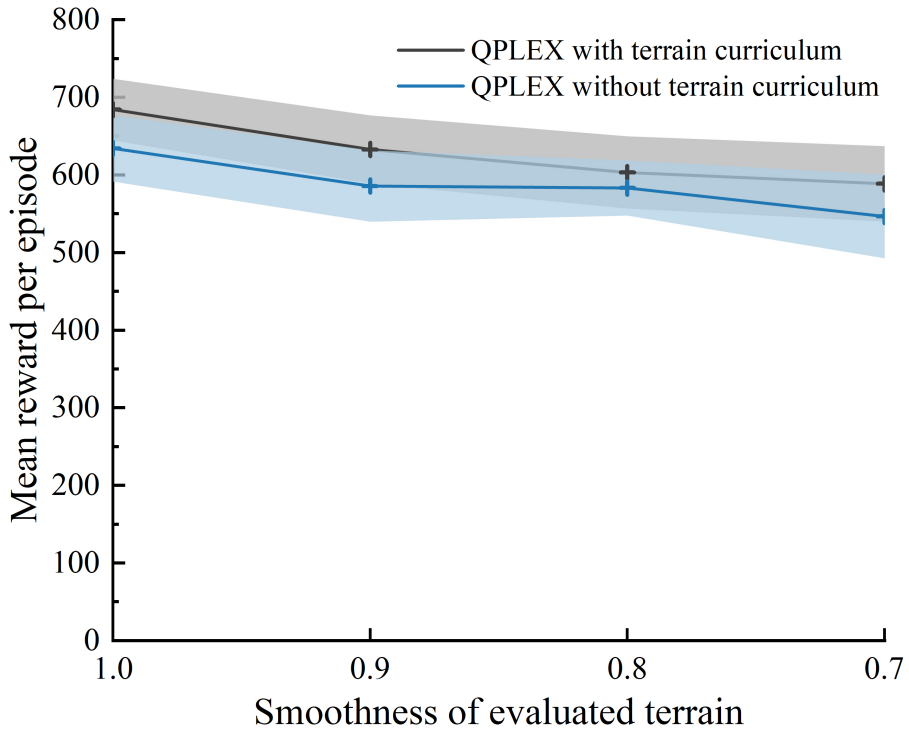


Figure 4.15: Importance of the terrain curriculum.

In the process of training a hexapod robot, it is not feasible to train it directly on challenging terrain. Instead, similar to the human learning process, the robot must commence with basic tasks and gradually progress to more complex ones. Specifically, the hexapod robot undergoes initial training on a flat surface to establish a stable walking gait, followed by an incremental increase in surface feature size. The initial step height is set at 1 cm, after which it is raised by increments of 1 cm until reaching a maximum height of 10 cm. Through the application of the QPLEX technique with terrain curriculum learning, the hexapod robot can successfully overcome 10 cm stairs (Fig. 4.11(c)(d)). Likewise, when preparing for movement on unstructured, uneven terrain, the training process initiates at a minimal height of 1 cm and gradually advances to higher elevations and greater ground roughness, leading to the creation of effective policies capable of navigating through various complex terrains.

Fig. 4.15 evaluates the impact of terrain curriculum learning on QPLEX training. The actual terrain used for training is shown in Fig. 4.11. As a baseline, we use QPLEX previously trained on uneven terrain with smoothness = 0.8. QPLEX with terrain curriculum consistently outperforms QPLEX without terrain curriculum in terrain testing and is more stable.

Section 4.6

Chapter Summary

In summary, this chapter introduces a sophisticated value function decomposition algorithm, QPLEX, tailored for hexapod robot motion control. The QPLEX framework is conceived as a multi-agent system, wherein each leg operates as an individual agent possessing its dedicated control module. Employing reinforcement learning within a simulated environment, the QPLEX algorithm is trained to attain stable and efficient locomotion. The research demonstrates that QPLEX surpasses the fully-decentralized baseline methodology in accomplishing ambulatory behavior, yielding exceptional performance and harmonious rhythmic gaits. Moreover, the investigation assesses the adaptability of trained controllers on irregular terrain and reveals the superior resilience and efficacy of QPLEX compared to the fully-decentralized approach. Lastly, the significance of terrain curriculum learning is appraised, revealing that QPLEX, when integrated with terrain curriculum, consistently outshines its counterpart without terrain curriculum, exhibiting enhanced stability.

Section 4.7

Appendix

Table 4.3: Detailed results after 5000 epochs of learning. The average reward of each of the 15 learned controllers of the QPLEX architecture was evaluated in 100 simulation runs after training.

Seed (Architecture)	Avg. Reward	Std. Dev. (btw. runs)
Seed 1, QPLEX	724	15
Seed 2, QPLEX	647	23
Seed 3, QPLEX	730	29
Seed 4, QPLEX	739	16
Seed 5, QPLEX	814	17
Seed 6, QPLEX	754	14
Seed 7, QPLEX	753	46
Seed 8, QPLEX	739	38
Seed 9, QPLEX	658	19
Seed 10, QPLEX	676	13
Seed 11, QPLEX	697	28
Seed 12, QPLEX	762	27
Seed 13, QPLEX	684	47
Seed 14, QPLEX	700	19
Seed 15, QPLEX	683	18
Mean performance	717	
Std. dev.	43	

Table 4.4: Detailed results after 5000 epochs of learning. The average reward of each of the 15 learned controllers of the Full-decentralized architecture was evaluated in 100 simulation runs after training.

Seed (Architecture)	Avg. Reward	Std. Dev. (btw. runs)
Seed 1, Full-decentralized/baseline	523	27
Seed 2, Full-decentralized/baseline	616	22
Seed 3, Full-decentralized/baseline	371	86
Seed 4, Full-decentralized/baseline	613	13
Seed 5, Full-decentralized/baseline	699	18
Seed 6, Full-decentralized/baseline	549	15
Seed 7, Full-decentralized/baseline	612	26
Seed 8, Full-decentralized/baseline	747	24
Seed 9, Full-decentralized/baseline	659	21
Seed 10, Full-decentralized/baseline	603	27
Seed 11, Full-decentralized/baseline	547	17
Seed 12, Full-decentralized/baseline	618	27
Seed 13, Full-decentralized/baseline	526	16
Seed 14, Full-decentralized/baseline	677	15
Seed 15, Full-decentralized/baseline	547	14
Mean performance	592	
Std. dev.	86	

Table 4.5: Comparison of rewards between the different control architectures during evaluation. Data was collected during evaluation for 100 episodes for each controller. Given are average rewards (and standard deviation in brackets) for each group of controllers (each group consisted of fifteen individually trained controllers). There were two different controller architectures, QPLEX and Fully-decentralized approaches, and two different training conditions,trained on flat terrain or on uneven terrain ((Smoothness=0.8).followed by evaluation on increasingly uneven terrain.

Condition	QPLEX Arch.	
Trained on ...	Flat Terrain	Uneven Terrain
Evaluation on flat terrain (Smoothness=1.0)	717(43)	634(43)
Evaluation on uneven terrain (Smoothness=0.9)	495(54)	586(46)
Evaluation on uneven terrain (Smoothness=0.8)	482(57)	583(36)
Evaluation on uneven terrain (Smoothness=0.7)	458(75)	546(54)
Condition	Fully-decentralized Arch.	
Trained on ...	Flat Terrain	Uneven Terrain
Evaluation on flat terrain (Smoothness=1.0)	592(86)	499(54)
Evaluation on uneven terrain (Smoothness=0.9)	441(106)	472(45)
Evaluation on uneven terrain (Smoothness=0.8)	431(110)	469(40)
Evaluation on uneven terrain (Smoothness=0.7)	372(112)	428(67)

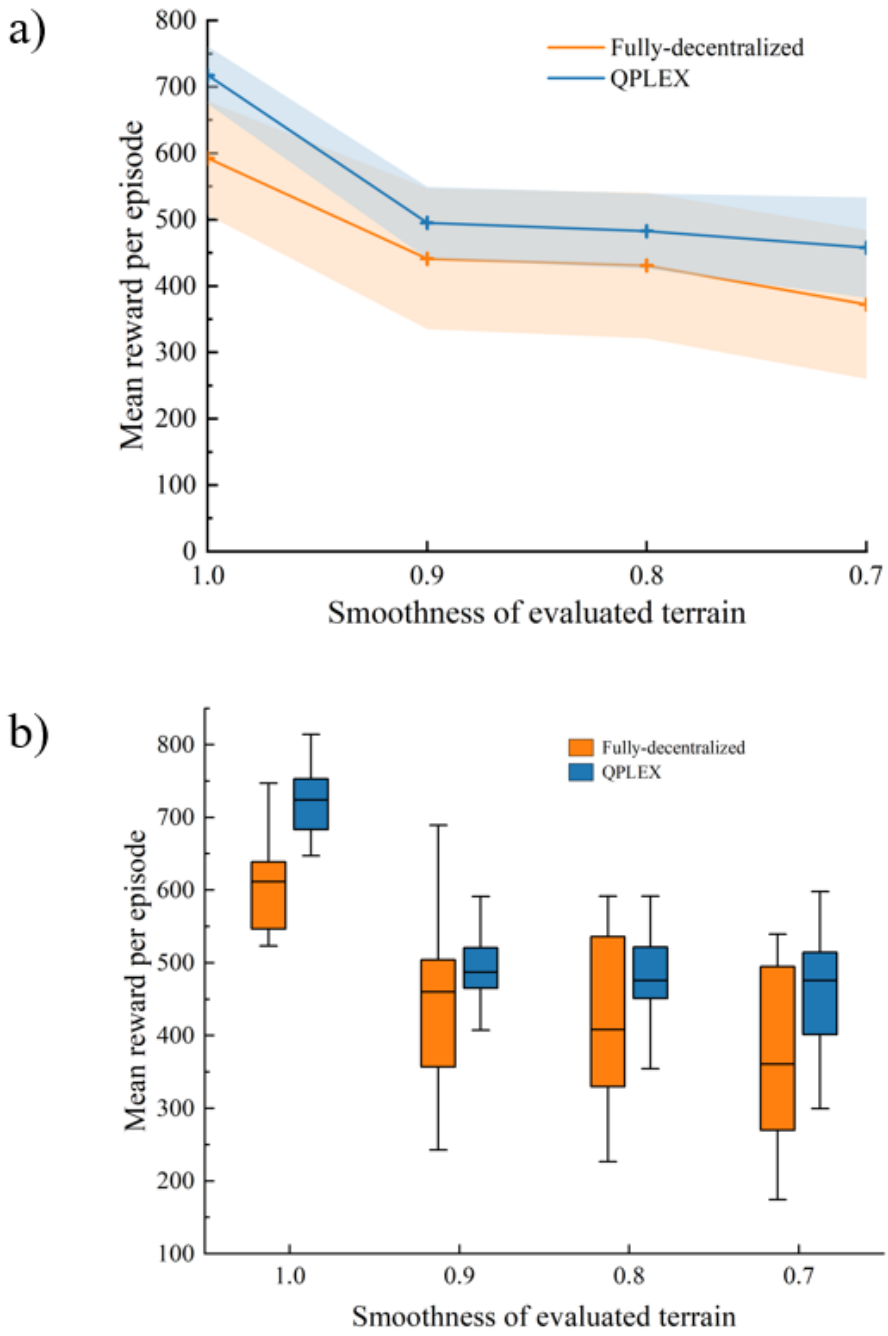


Figure 4.16: Evaluation on different types of terrain:training on flat terrain and evaluation on progressively rough terrain (where smoothness of 1.0 denotes flat terrain,and 0.0 indicates very rugged terrain)In (a) mean return for the two architectures is shown and how performance decreases for more difficult terrain. Target velocity is set to 2.0 (fast walking). Panel (b) compares mean returns for increasingly uneven terrain for the different architectures. QPLEX architecture performs highly significant better compared to fully-decentralized architecture.

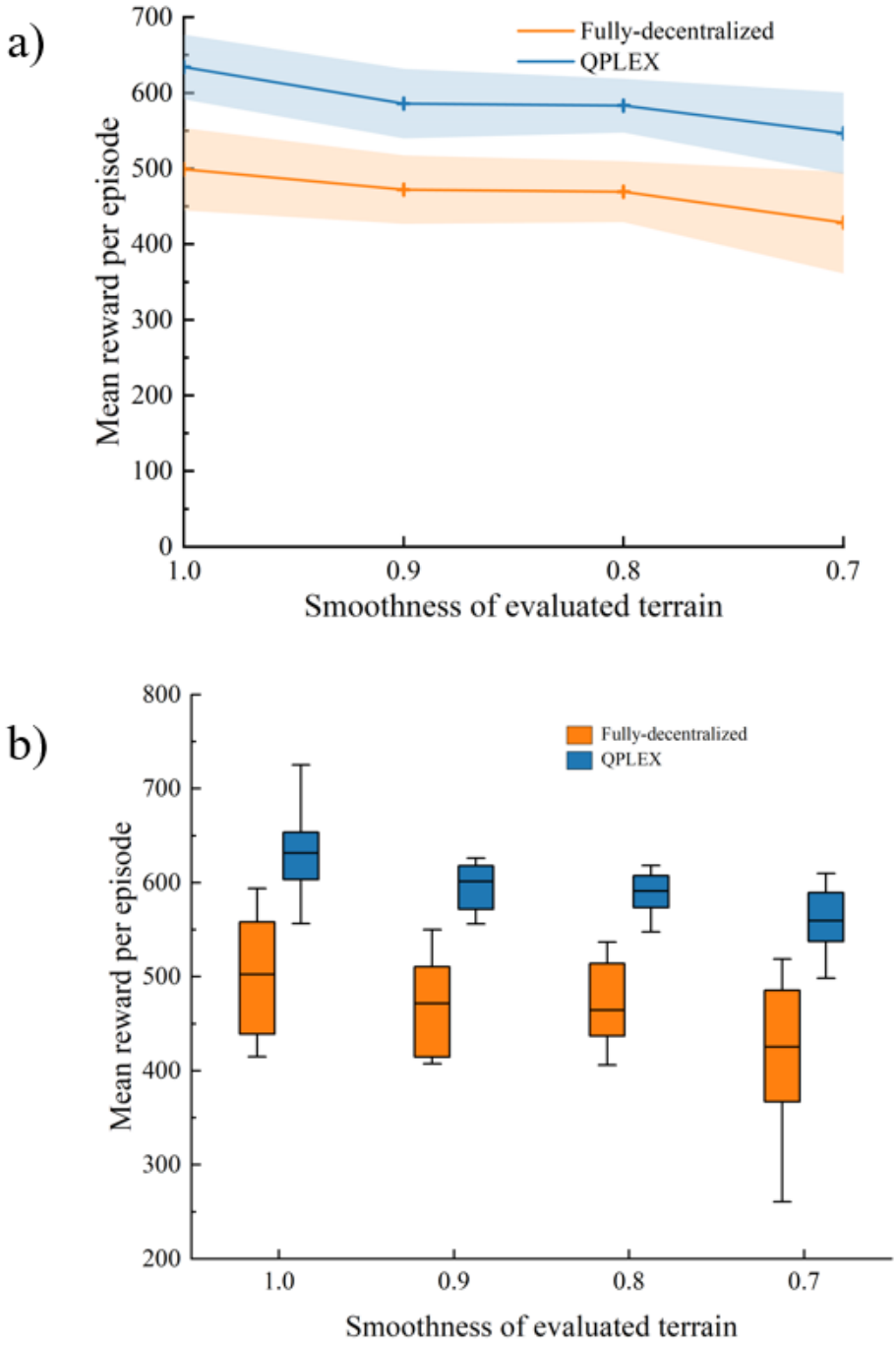


Figure 4.17: Evaluation on different types of terrain:training on uneven terrain with a smoothness value of 0.8, followed by evaluation on increasingly uneven terrain.In (a) mean return for the two architectures is shown and how performance decreases for more difficult terrain. Target velocity is set to 2.0 (fast walking). Panel (b) compares mean returns for increasingly uneven terrain for the different architectures. QPLEX architecture performs highly significant better compared to fully-decentralized architecture.

Chapter 5

Conclusion and Future Work

Section 5.1 Conclusion

This paper aims to address the issue of motion control for footed robots, specifically focusing on hexapod robots. These robots possess a high degree of redundancy and relatively low control policies, making them an appropriate research subject. By studying insect morphology and movement patterns, a robot model is designed to verify and study motion control algorithms. Robot motion control is achieved through trajectory planning, utilizing the study of positive and negative kinematics of hexapod robots. Bionic principles derived from the study of animal rhythmic motion are applied to implement robot motion control. Furthermore, the theoretical basis of deep learning and multi-agent reinforcement learning is explored to train the robot motion control strategy with a multi-agent deep reinforcement learning design. The key contribution of this paper lies in the successful implementation of robot motion control.The main work accomplished in this paper is: (1)We summarize the research on footed robots conducted by multiple universities and enterprises, with a specific focus on the motion control of hexapod robots. Through an analysis of insect form,

a hexagonal robot body and a three-joint robot single leg were designed, and a structured model of the robot was assembled using SolidWorks. The walking gait of the hexapod robot was classified into three-legged, four-legged, and fluctuating gait, and various gait patterns were analyzed and compared based on the study of insect motion. Regardless of the gait, it was observed that the single leg moves in the same manner on structured ground. By analyzing the single leg foot end trajectory, the leg tip trajectory of the hexapod robot was planned, and the robot leg kinematic solution was derived from the perspective of robot kinematics. The robot leg kinematic solution was experimentally verified, and the robot motion control was completed using a three-legged gait, including foot end trajectory planning and kinematic solution. The robot effect was judged and explained from the perspective of displacement change and body pitch angle change during the robot walking process.

(2)We presents an animal-inspired approach to motion control, utilizing the construction of a CPG control unit based on the Hopf oscillator. The oscillator's mathematical model parameters are analyzed to determine their influence on the output waveform, and the values of these parameters are optimized for robot motion control. To simplify the network, the control signals of the tibial and femoral segments of the robot's legs are reduced to the output signals after half-wave processing of the base segment signals. Ultimately, the CPG network is constructed, enabling the robot to achieve motion control in complex environments such as stable walking and climbing. The robot's motion process is further analyzed and quantitatively evaluated in terms of displacement change and body pitch angle.

(3)We proposes a value function decomposition algorithm, QPLEX, applied to hexapod robot motion control. The QPLEX architecture is designed as a multi-agent system, where each leg is treated as a separate agent with its local control module. The QPLEX algorithm is trained using reinforcement learning in a simulated

environment to achieve stable and efficient locomotion. The study shows that the QPLEX architecture outperforms the fully-decentralized baseline approach in achieving walking behavior and produces high performance and well-coordinated rhythmic gaits. Furthermore, the study examines the generalization of the trained controllers on uneven terrain conditions and shows that the QPLEX architecture is more robust and performs significantly better than the fully-decentralized approach on uneven terrain. The importance of terrain curriculum learning is also evaluated, and the results show that QPLEX with terrain curriculum consistently outperforms QPLEX without terrain curriculum in terrain testing and is more stable.

The robot motion control approach of foot-end trajectory planning enables robot motion control through inverse kinematic solution, provided that a determined foot-end trajectory is available. However, the real-world environment is complex and ever-changing, making it difficult to plan different trajectories for various types of ground. Therefore, this motion control method has limited generalization capability for diverse terrain. In contrast, the animal-inspired central pattern generator-based robot motion control method relies on rhythmic motion and has a simple and user-friendly control strategy that can be coupled with high-level control information for more complex robot motion. However, this method's control policy is limited to the core aspects of the approach. In this paper, a robot motion control policy based on multi-agent deep reinforcement learning input is proposed, using robot pose information to adjust the policy output robot motion control signal via continuous training. Compared to the previous two approaches, this method offers more flexible motion strategy adjustments, making it a more advantageous control policy. The combination of these three approaches can be used to develop an easy-to-use robot motion control policy tailored to specific robot motion control scenarios.

Section 5.2

Future Work

In this paper, all experiments were conducted in a simulation environment, which allowed us to verify the effectiveness of our proposed methods. However, it should be noted that the mechanical structure of the robot, including factors such as motor selection and manufacturing process, can significantly impact the actual motion control performance of the robot. The intelligent system used for training in the simulation environment employed ideal motors, whereas actual motors can experience power and delay issues. As a result, the robot motion control policies developed in the simulation environment may not translate directly to the actual robot. Therefore, further work is needed to bridge the gap between simulation and practical implementation, including the development of more realistic simulation models and the exploration of methods for transferring trained policies to real-world robots.

Reference

[1] Xiaohui Yuan, Longbo Kong, Dengchao Feng, and Zhenchun Wei. Automatic feature point detection and tracking of human actions in time-of-flight videos. *IEEE/CAA Journal of Automatica Sinica*, 4(4):677–685, 2017.

[2] Baofu Fang, Xiaoping Guo, Zaijun Wang, Yong Li, Mohamed Elhoseny, and Xiaohui Yuan. Collaborative task assignment of interconnected, affective robots towards autonomous healthcare assistant. *Future Generation Computer Systems*, 92:241–251, 2019.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[4] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[5] Sangok Seok, Albert Wang, Meng Yee Chuah, David Otten, Jeffrey Lang, and Sangbae Kim. Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 3307–3312. IEEE, 2013.

[6] Rodney A Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2):253–262, 1989.

[7] Julie Townsend, Jeffrey Biesiadecki, and Curtis Collins. Athlete mobility performance with active terrain compliance. In *2010 IEEE Aerospace conference*, pages 1–7. IEEE, 2010.

[8] Joaquín Estremera, Jose A Cobano, and P Gonzalez De Santos. Continuous free-crab gaits for hexapod robots on a natural terrain with forbidden zones: An application to humanitarian demining. *Robotics and Autonomous Systems*, 58(5):700–711, 2010.

[9] José Luis Albarral and Enric Celaya. Implementation of a driver level with odometry for the lauron iii hexapod robot. In *Climbing and Walking Robots*, pages 135–141. Springer, 2005.

[10] Priyaranjan Biswal and Prases K Mohanty. Development of quadruped walking robots: A review. *Ain Shams Engineering Journal*, 12(2):2017–2031, 2021.

[11] Robert B McGhee and Andrew A Frank. On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, 3:331–351, 1968.

[12] Holk Cruse. What mechanisms coordinate leg movement in walking arthropods? *Trends in neurosciences*, 13(1):15–21, 1990.

[13] Holk Cruse and Rüdiger Wehner. No need for a cognitive map: decentralized memory for insect navigation. *PLoS computational biology*, 7(3):e1002009, 2011.

[14] Joseph Ayers and Jan Witting. Biomimetic approaches to the control of underwater walking machines. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1850):273–295, 2007.

[15] Jan H Witting, Joseph Ayers, and Koray Safak. Development of a biomimetic underwater ambulatory robot: advantages of matching biomimetic control archi-

tecture with biomimetic actuators. In *Sensor Fusion and Decentralized Control in Robotic Systems III*, volume 4196, pages 54–61. SPIE, 2000.

[16] ST Venkataraman. A simple legged locomotion gait model. *Robotics and Autonomous Systems*, 22(1):75–85, 1997.

[17] Jose Hugo Barron-Zambrano, Cesar Torres-Huitzil, and Bernard Girau. Perception-driven adaptive cpg-based locomotion for hexapod robots. *Neurocomputing*, 170:63–78, 2015.

[18] Haitao Yu, Haibo Gao, Liang Ding, Mantian Li, Zongquan Deng, and Guangjun Liu. Gait generation with smooth transition using cpg-based locomotion control for hexapod walking robot. *IEEE Transactions on industrial electronics*, 63(9):5488–5500, 2016.

[19] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706, 2020.

[20] Huiqiao Fu, Kaiqiang Tang, Peng Li, Wenqi Zhang, Xinpeng Wang, Guizhou Deng, Tao Wang, and Chunlin Chen. Deep reinforcement learning for multi-contact motion planning of hexapod robots. In *IJCAI*, pages 2381–2388, 2021.

[21] Teymur Azayev and Karel Zimmerman. Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification. *Journal of Intelligent & Robotic Systems*, 99(3):659–671, 2020.

[22] Ruiqin Li, Hongwei Meng, Shaoping Bai, Yinyin Yao, and Jianwei Zhang. Stability and gait planning of 3-upu hexapod walking robot. *Robotics*, 7(3):48, 2018.

[23] Li Deng and Dong Yu. Foundations and trends in signal processing: Deep learning–methods and applications. 2014.

[24] Fred Delcomyn. Neural basis of rhythmic behavior in animals. *Science*, 210(4469):492–498, 1980.

[25] Serge Rossignol. Locomotion and its recovery after spinal injury. *Current opinion in neurobiology*, 10(6):708–716, 2000.

[26] Ilya A Rybak, Natalia A Shevtsova, Myriam Lafreniere-Roula, and David A McCrea. Modelling spinal circuitry involved in locomotor pattern generation: insights from deletions during fictive locomotion. *The Journal of physiology*, 577(2):617–639, 2006.

[27] Ludovic Righetti and Auke Jan Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *2008 IEEE International Conference on Robotics and Automation*, pages 819–824. IEEE, 2008.

[28] Juan A Acebrón, Luis L Bonilla, Conrad J Pérez Vicente, Félix Ritort, and Renato Spigler. The kuramoto model: A simple paradigm for synchronization phenomena. *Reviews of modern physics*, 77(1):137, 2005.

[29] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[30] Hyun Ah Song and Soo-Young Lee. Hierarchical representation using nmf. In *International conference on neural information processing*, pages 466–473. Springer, 2013.

[31] Sven Behnke. *Hierarchical neural networks for image interpretation*, volume 2766. Springer, 2003.

[32] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

[33] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[34] James E Kittock. Emergent conventions and the structure of multi-agent systems. In *Proceedings of the 1993 Santa Fe Institute Complex Systems Summer School*, volume 6, pages 1–14. Citeseer, 1993.

[35] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[36] Sun Chang-Yin and Mu Chao-Xu. Important scientific problems of multi-agent deep reinforcement learning. *Acta Automatica Sinica*, 46(7):1301–1312, 2020.

[37] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[38] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[39] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[40] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and*

Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16, pages 66–83. Springer, 2017.

[41] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

[42] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Decentralized multi-agent reinforcement learning with networked agents: Recent advances. *Frontiers of Information Technology & Electronic Engineering*, 22(6):802–814, 2021.

[43] Miloš S Stanković, Marko Beko, and Srdjan S Stanković. Distributed value function approximation for collaborative multiagent reinforcement learning. *IEEE Transactions on Control of Network Systems*, 8(3):1270–1280, 2021.

[44] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[45] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[46] Jianhao Wang, Zhizhou Ren, Beining Han, Jianing Ye, and Chongjie Zhang. Towards understanding cooperative multi-agent q-learning with value factorization. *Advances in Neural Information Processing Systems*, 34:29142–29155, 2021.

[47] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020.

[48] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*, 2020.

[49] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[50] Qingling Wang, Haris E Psillakis, and Changyin Sun. Cooperative control of multiple agents with unknown high-frequency gain signs under unbalanced and switching topologies. *IEEE Transactions on Automatic Control*, 64(6):2495–2501, 2018.

[51] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[52] Malte Schilling, Thierry Hoinville, Josef Schmitz, and Holk Cruse. Walknet, a bio-inspired controller for hexapod walking. *Biological cybernetics*, 107:397–419, 2013.

[53] Guillaume Sartoretti, William Paivine, Yunfei Shi, Yue Wu, and Howie Choset. Distributed learning of decentralized control policies for articulated mobile robots. *IEEE Transactions on Robotics*, 35(5):1109–1122, 2019.

[54] Malte Schilling, Kai Konen, Frank W Ohl, and Timo Korthals. Decentralized deep reinforcement learning for a distributed and adaptive locomotion controller of a hexapod robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5335–5342. IEEE, 2020.

[55] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Con-*

ference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), volume 3, pages 2149–2154. IEEE, 2004.

[56] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 1. 2015.

[57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[58] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[59] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.

[60] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.