Multi-agent Footstep Steering with Deep Reinforcement Learning

by

Kun Peng
B.Sc., The Ohio State University, 2021

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Multi-agent Footstep Steering with Deep Reinforcement Learning

by

Kun Peng
B.Sc., The Ohio State University, 2021

Supervisory Committee

Dr. Brandon Haworth, Supervisor
(Department of Computer Science)

Dr. Teseo Schneider, Supervisory Committee Member
(Department of Computer Science)

Supervisory Committee

Dr. Brandon Haworth, Supervisor
(Department of Computer Science)

Dr. Teseo Schneider, Supervisory Committee Member
(Department of Computer Science)

ABSTRACT

Crowd simulation plays a crucial role in a wide range of fields, from digital media to urban planning. However, traditional particle-based algorithms often lack essential information to present realistic human bipedal locomotion. This research aims to propose a more realistic and efficient steering model for crowd simulation by combining Multi-Agent Reinforcement Learning (MARL) with bipedal locomotion modelling. This study explores the advantages of MARL and analyzes a mathematical approach to simplifying complex bipedal locomotion. The approach utilizes the Proximal Policy Optimization algorithm and trains the model in adjustable randomized maze-like environments. Assessment results of the model indicate that the model learns goal-reaching behaviours and learns to avoid static and dynamic obstacles. Furthermore, the agents can simulate complex steering behaviours such as side-stepping and turning-like behaviours with two feet. This research contributes to the advancement of the field of crowd simulation through a flexible and realistic approach to modelling human steering behaviours in complex and dynamic environments.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to everyone who helped my during my thesis work.

I would first like to sincerely thank Dr. Brandon Haworth, my supervisor, whose advice has been crucial to my academic career. His outstanding reactivity and ability to give clear direction when needed are the only things that can equal his extraordinary experience and comprehensive understanding in the subject. I have been able to overcome several technical obstacles and research hurdles thanks to his wise counsel.

In addition to his academic guidance, I am very appreciative of his comprehension, patience, and steadfast assistance. His supportive demeanour has greatly aided in my professional and personal development, transforming me from a timid first-year student into a more certain researcher. His mentoring approach combines professional competence with sincere concern for the growth of students, demonstrating the best aspects of academic supervision. One of the most significant benefits of my academic career will continue to be the confidence I have developed under his guidance.

I would also like to thank Dr. Teseo Schneider for serving on my supervisory committee. I appreciate his willingness to review my thesis and provide valuable feedback during my work.

I would like to thank lab members in Graphics, Artificial Intelligence, Design, and Games Lab (GAIDG Lab), who help me overcome various technical challenges with model training on high-performance computing resources.

I gratefully acknowledge the support from my friend, Minamoto Teiji. Their insightful feedback on the organization and presentation of my ideas has helped shape this thesis, and their emotional support helps me to overcome difficult moments.

Above all, I would like to express my sincere thank to my partner Nan, whose unwavering support has been my foundation throughout this journey. She has been my most devoted rubber duck debugger, listening to my technical explanations with patience and assisting me in resolving several issues, even when the topic was new to her. Her patience and emotional support is a priceless treasure to me.

DEDICATION

To everyone who helped make this journey possible.

Chapter 1

Introduction

Crowd simulation is a widely used technique across a variety of fields, designed to simulate complex dynamic crowds through computer animation as an alternative to the time-consuming and expensive manual labour. We can observe the scenes with large synthetic human crowds in digital media creations such as video games and films. On the other hand, in urban road and architectural designs, this technique is also introduced to guarantee safety and identify potential risks.

To emulate the complex crowd behaviours, researchers have developed agent-based steering algorithms, which model each individual in the crowd as an autonomous agent who behaves according to its perceived surroundings and predefined principles of action. However, the increasing demand for realistic human crowd synthesis reveals the limitations of major traditional steering algorithms. Most of the agent-based algorithms compute steering behaviours based on the premise that agents are particles, relying on limited information such as velocity, acceleration, and position of individual particles. Consequently, they fail to capture essential details such as the orientation, position, and velocity of their feet, thereby lacking the ability to accurately indicate bipedal actions like side-stepping and turning around with the movement and coordination of the feet. In media like movies, this problem may reduce the immersion of the audience as they enjoy it. For example, two human models that are actually particles may not be able to get too close together, and may appear as though they are hitting a wall of air when they are still some distance away. For urban designers, particle-based agents that are not sufficiently close to the actual reality may lead to simulated crowds walking along the wrong paths. Even minor deviations may affect the design and make it difficult to thoroughly investigate safety hazards. There have also been steering algorithms in recent years that are specifically applicable to

bipedal agents, but the actions and trajectories of agents with two feet or even a torso are much more complex than those of a single particle. For such algorithms, computational efficiency and model accuracy remain very challenging.

The main objective of this research is to develop a more realistic and efficient steering model for crowd simulation by integrating Multi-Agent Reinforcement Learning (MARL) and bipedal locomotion modelling. To achieve this goal, this study will explore the following research questions: What are the advantages of introducing MARL over other types of algorithms and how can it be integrated with bipedal locomotion modelling? In what ways can human bipedal locomotion, which is so complex, be simplified and computed to approximate real human trajectories? In what ways does this model enhance the realism of agent behaviour compared to traditional particle-based algorithms?

As a means of addressing these research questions and advancing MARL applications in crowd simulation, this study proposes a novel framework that utilizes bipedal locomotion modelling with MARL to generate steering algorithms. This approach offers several key advantages. First, MARL opens up new possibilities for crowd simulation because with the behavioural policy adjustments based on rewards and penalties, agents optimize their policy through interactions with the environment. This adaptability extends to scenarios that are not anticipated by the predefined formulas or existing datasets. Second, to maintain stable training process, the research incorporated the Proximal Policy Optimization (PPO) algorithm, a state-of-the-art reinforcement learning technique. Third, domain randomization is employed in this study to enhance policy generalization to diverse, unseen scenarios by adjusting the density and complexity of the generated environments. Last but not least, by integrating bipedal locomotion modelling, especially the 3D inverted pendulum theory, this framework aims to more accurately simulate human steering behaviours in crowds that are subject to biomechanical constraints, including complex actions like side-stepping and turning.

This research seeks to expand the field of crowd simulation by offering a more realistic and effective approach to synthesize human behaviour in complex, dynamic environments. To detail the approach proposed in this research from the elementary to the profound, this thesis is organized in the following manner:

Chapter 2 gives a comprehensive review of related works in the field of agent-based steering algorithms and analyzes the research gaps between previous works and this research.

Chapter 3 explains the theoretical foundation of MARL, introduces the main categories of different MARL algorithms as well as their advantages and disadvantages, and paves the way for the later introduction of the PPO algorithm used in this study.

Chapter 4 presents theoretical knowledge about biomechanics, starting from the concept of the human gait cycle to the mathematical derivation of 3D inverted pendulum theory.

Chapter 5 details how the theoretical knowledge mentioned in the previous two chapters was applied in this study. This chapter describes the construction of the agent model, the design of the state space, action space, reward function, as well as the selection of the RL algorithm (PPO) and the design of the training environment, the training hyperparameter settings and finally, the results of the training.

Chapter 6 provides an assessment of the trained model with quantitative and qualitative metrics, comparing it with two traditional steering algorithms. The chapter discusses the advantages of this model over the traditional models, and summarizes the limitations and future work of this study.

Chapter 2

Literature Review

This chapter provides the background on crowd simulation and offers a comprehensive overview of the existing studies and knowledge in the areas of agent-based steering algorithms, aiming to outline the gaps in the field and emphasize the prospective impact and value of this research.

In the following sections, this chapter presents some representative agent-based steering algorithms in the sequence of the development of crowd simulation algorithms. Over time and as technology evolves, researchers have been working to improve the accuracy and realism of simulations: from early, crude rule-based methods to more advanced models that introduce machine learning and include actual crowd data, and finally, the use of Multi-Agent Reinforcement Learning (MARL). Building upon this trajectory of development, this research integrates bipedal motion modelling with MARL with the goal of addressing existing gaps in the field and further enhancing the quality of crowd simulations.

2.1 History of Simulation

As its literal meaning implies, a simulation is a reproduction of how a system or process might function in the real world. From the perspective of entertainment demand, simulations are widely applied in movies, games and other recreational activities including virtual elements, allowing people to achieve immersive experiences. On the other hand, from the perspective of analytical and decision-making needs, simulations are employed to replicate scenarios for studying complex issues that would be expensive, unsafe or impracticable to execute in the real world, such as emergency

evacuations and urban planning in the field of crowd simulation. It is necessary to make the behaviours of humans in the crowd as close to reality as possible, so there has been extensive research on steering algorithms over the years.

The research that laid the groundwork for the development of today's crowd simulation algorithms can be traced back to von Neumann's Self-Reproducing Automata[59], which was the foundation of cellular automata theory. This theoretical model was made up of an array of cells that resembled a 2D grid. Each cell followed a set of rules that determined the state of the cell based on the state of neighbouring cells. Based on the theory of cellular automata, John Conway developed Game of Life in 1970[10], which used simple rules to simulate the evolution of cells on a 2D grid. In this model, each cell had the potential to be either alive or dead, and the conditions of its eight adjacent cells determined its state in the next generation. This is similar to movements in a crowd that each person is an independent thinker who decides where to move based on their own set of rules of behaviour, which include how they respond to the actions of those closest to them. The concept of cellular automata subsequently gave rise to agent-based modelling, such as in the analysis of the dynamics of segregation[46], where small individual preferences resulted in significant group division. In agent-based modelling, each individual in a simulated system is represented as an agent, encapsulating its rules of actions, decisions and reactions, along with its current state and knowledge of the environment. With this approach, we can clearly simulate, observe and analyze how different or even emergent behaviours in massive and complex systems are generated by the combination of local interactions and individual actions, which is suitable for the research of crowd simulation.

2.2 Rule-based Algorithms

Early studies on crowd simulation mainly focused on studying the animation of animal crowds. One of the earliest and most famous models, a distributed behavioural model (usually referred to as the "Boids" model) developed by Reynolds in 1987[44], simplified each bird in a flock into particles and their behaviours were based on three rules: avoiding collisions with, matching velocity with and staying close to other nearby particles. Although it was a model for bird flocks, it laid the basis for further research in the field of agent-based crowd simulation for human agents.

In 1997, Musse and Thalmann applied this rule-based approach to human crowds, focusing on inter-group relationships and collision detection [38]. Their model de-

fined rules based on sociological principles, with agents responding to others according to individual parameters that represented their emotional status, dominance, group relationships, etc. For collision detection, the model used simple mathematical equations that predicted collisions with line-line intersections. Additionally, a multi-resolution collision avoidance system based on the observer's distance from the agents was presented in the research, striking a balance between computing efficiency and behavioural realism. Specifically, if the agents were close enough to each other, they used the most computationally expensive and detailed way, i.e., changing the angular velocity, to avoid collisions. If the distance was moderate, the agents chose to stop and let others pass first, and if they were far enough away then they did not consider taking collision avoidance actions at all.

2.3 Force-based Algorithms

In force-based models, agents' steering is under the influence of various repulsive and attractive forces. This technique was initially proposed in 1995 to describe the forces of social attraction and repulsion during locomotion, referred to as Social Forces [19]. This model was further developed in 2000 to simulate panic behaviour during emergency evacuations, helping people develop more effective evacuation strategies [18].

The Predictive Avoidance Model (PAM) proposed in 2009 [26] was also a kind of force-based approach. In this model, agents continuously calculated their future trajectories based on their current velocity. If their paths intersected with others', then an evasive force would be computed and applied to the agents to change their velocity to avoid the predicted collision. This model considered and planned in both temporal and spatial dimensions. It predicted where the agent would arrive and how long it would take to get there, and could plan in advance for the agent to either change direction to reach a different area, or to change its speed to avoid arriving in the area at the time when a collision might occur. With space-time planning, PAM allowed the agents to avoid sudden changes of direction, making the simulation of a dynamic crowd more realistic.

2.4 Velocity-based Algorithms

The velocity-based approaches introduced the concept of velocity obstacles (VO) [12, 13]. A VO is an abstract obstacle consisting of a set of velocities, where velocities within the region will cause a collision to occur, and the agent needs to choose velocities outside the region to avoid the collision, gradually adjusting its current velocity and changing it to the new one.

This strategy has been expanded to incorporate reciprocal collision avoidance (RVO) [58] and optimal reciprocal avoidance for several agents (ORCA) [57]. In RVO, all agents are supposed to make collision avoidance decisions simultaneously and symmetrically. Rather than incrementally modifying their velocities, which may result in jagged trajectories, all agents move directly towards the predicted, collision-free path. ORCA is a further extension of RVO in which the optimal velocity was derived from solving an optimization problem based on the assumption that the agents cooperatively adjust their velocities.

These algorithms, which depend on behavioural guidelines that have been predefined, provide highly generalized depictions of agent behaviour, endow these types of models with simplicity and computational ease, thereby enabling the output of the models to be predictable and controllable. On the contrary, the behavioural guidelines may be too simplistic to account for all the complex real-world behaviours, and since the rules are fixed, the agents are unable to improve their movements when they are faced with new situations that they cannot handle using these guidelines.

2.5 Potential Field-based Algorithms

With potential field-based algorithms, agents' behaviours can be more adaptive by optimizing their movements in response to dynamically defined potential fields. By defining a potential field [61, 62] in the global environment, agents adjust their trajectories based on potential gradients, which can be related to the positions or velocities of obstacles and the distance to the targets. Traditional potential field approaches, however, have drawbacks, such as a high dependence on environmental complexity and problems with resolution and scale. To overcome these obstacles, the Egocentric Affordance Field framework has been proposed [25]. The framework, representing an egocentric approach and enabling dynamic scaling, not only ensures that computational cost is unaffected by the complexity of the environment, but also facilitates

efficient space-time planning. While agents are able to adapt to new conditions by changing their actions to maximize the potential utility, these algorithms are more computationally expensive, and to design appropriate potential functions can be challenging.

2.6 Data-driven Algorithms

In comparison with traditional algorithms, the data-driven models exploit Artificial Intelligence and enable the agents to learn to optimize their behavioural policy during the training process. Data-driven models take advantage of machine learning, allowing the agents to learn from datasets [30, 31] and capture temporal dependencies in the data [1]. Both real-world crowd data, such as [49, 11], and synthetic crowd data, such as [60], enable the agents to handle a wide range of different cases and perform more complex and realistic behaviours. However, ideal outputs rest on properly tuned hyper-parameters such as the structure of neural networks and the learning rates. Additionally, using large amounts of data is a double-edged sword: such methods are heavily dependent on the quality and quantity of the data, not to mention the high computational cost of generating datasets and training.

2.7 MARL Applications

As another option, Reinforcement Learning relies less on data because agents learn optimal policies through interactions with the environment. In the context of Multi-Agent Reinforcement Learning (MARL), a group of agents, given predefined reward functions, is in a constant state of trial and error to maximize their long-term rewards when they interact with not only the environment but also other agents [66]. As a result, the principle of MARL makes it possible for agents to learn complex behaviours and adapt to dynamic environments. Several studies have already demonstrated the potential of applying MARL in the field of crowd simulation. For instance, Godoy et al. [14] presented ALAN, an adaptive learning approach that modelled navigation problems as multi-armed bandit action selection problems, dynamically adjusting agent motion based on recent observations using offline Markov Chain Monte Carlo. The approach was proven by experiments to be more time-efficient than models such as ORCA. Additionally, there have also been studies introducing policy parameterization techniques indicating the application of MARL enables more generalized policies

[21, 41].

2.8 Bio-mechanical Steering in Crowd Simulation

Traditional crowd steering algorithms usually oversimplify characters as particles. While simplifying computations, problems also occur. For example, particle-like agents may slide while real humans don't perform the same behaviours. As a result, researchers incorporate biomechanics to ensure that synthesized human crowds are more realistic.

Beacco et al. [2] proposed a dynamic footsteps planning algorithm in 2013. They generated a set of animation clips of human walking which were either extracted from the motion capture database or manually created. By analyzing the clips, individual step motions were extracted, annotated and formed into a basic library for the algorithm. The planner first used the A* algorithm for high-level path planning, and then sought the optimal sequence of actions of each agent through real-time planning algorithm based on minimal energy consumption. For dynamic obstacles with predictable trajectories, the planner took them into account in the planning phase. While for unpredictable dynamic obstacles, multi-resolution collision detection was applied so the trajectories were re-planned in real-time. Additionally, for more precise collision detection, the high-resolution model used five cylinders to enclose the agent's head and four limbs. Finally, the planned footstep trajectories were delivered to the animation engine to generate fluent walking animations with the pre-processed animation clips.

Singh et al. [52] also harmonized space-time planning and biomechanics to select the optimal route for the agent based on energy functions and using a heuristic algorithm. Furthermore, the model used a mathematical model of a 3D inverted pendulum to compute the body's trajectory, rather than applying predefined animation clips. The model also used five circles to track the positions of the torso and limbs for collision detection, which allowed for closer proximity between agents compared to a single sphere radius as was the case of particle-like agents. Experimental results showed that the model allowed the agents to produce predictive cooperation and make side-step-like movements to avoid collisions.

There have also been studies on full body simulation with the application of MARL such as [17], but these are more related to the future work of this research because this research focuses more on the steering layer.

2.9 Discussion

This chapter details the evolution of agent-based crowd simulation algorithms, from early simple rule-based methods to more sophisticated approaches that combine machine learning and real crowd data. Researchers have been working to improve upon earlier algorithms in order to simulate crowds that are more realistic and adaptable to changing environments. The use of MARL stands out among these developments as it overcomes the limitations of data-driven algorithms, which are influenced by database quality while maintaining the flexibility of machine learning. On the other hand, this review also reveals a significant research gap in bipedal locomotion simulation within crowd models. This gap highlights the potential value of this study, which aims to combine MARL with biomechanics-based bipedal locomotion modelling. These two key points will be covered in depth in the next chapters, along with how this innovative method may contribute to the field of crowd simulation.

Chapter 3

Theoretical Foundations of Multi-Agent Reinforcement Learning

This chapter explains the theoretical basis for MARL, laying the foundation for the introduction of the methodology in the following chapter. As mentioned in Chapter 2, with the high demand for realistic crowd simulation, it is difficult for traditional particle-based steering algorithms to meet the needs of synthesizing bipedal locomotion for human characters. On the other hand, bipedal locomotion involves the orientation of the two feet, the velocity and the duration of each footstep, which, while allowing more information to be expressed compared to particles, also makes some simple movements require more complex computations, making real-time rendering difficult when computing power is insufficient. Thus, the introduction of artificial intelligence has greatly improved the efficiency of real-time computing. Pre-trained models enable faster and more accurate processing of information under dynamic conditions, reducing barriers for users to implement complex algorithms effectively.

3.1 Introduction

This study uses MARL based on the characteristics of crowd simulation, which involves an algorithm where multiple agents improve the model through constant interactions with the environment and trial and error. Since in crowd simulation, each person is an agent, and their behaviours continuously alter the environment from

the perspective of other agents, influencing each other's decision-making, the MARL algorithm is suitable for use. Moreover, since it does not rely on a static, limited dataset, MARL eliminates the tedious step of collecting and filtering data to avoid noise affecting the training results, while also allowing the model to be more flexible and applicable to a wider range of scenarios by adapting to the training environment rather than restricting itself to the scenarios covered by the dataset.

In order to give the reader a basic understanding of the principles of the MARL algorithm and some of the basic concepts, this chapter will present them in a step-by-step manner. First, the concept of a Markov Decision Process (MDP) is introduced, which serves as the foundational framework of the RL model. Next, RL algorithms in single-agent scenarios are presented to explain the fundamental concepts involved, as well as common ways of categorizing the algorithms and the scenarios they are applicable to, such as model-based and model-free algorithms. These concepts also apply to the multi-agent scenario, although the framework of MARL is more complex because multiple agents interact with each other and affect each other's decisions and policy updates. Therefore, the following subsections first introduce the concept of stochastic games, which apply to MARL, and then formally introduce the concept of MARL, explaining how it differs from single-agent scenarios and listing algorithmic categorizations that would only be found in multi-agent scenarios, such as cooperative or competitive settings. Finally, the chapter discusses the challenges that often need to be faced when solving problems with MARL.

The theoretical background presented in this chapter is crucial for understanding the rationale behind selecting the Proximal Policy Optimization (PPO) algorithm as the algorithm for this study in subsequent chapters. The different classifications of RL algorithms are explored in this chapter to pave the way for the detailed presentation of the PPO algorithm and its application to the training of steering algorithms for bipedal locomotion in subsequent chapters, so that the reader can more clearly comprehend the advantages of the PPO algorithm in later chapters, within the context of this research.

3.2 Markov Decision Process Framework

Suppose there is an empty plane as the ground, where a human-like agent can walk on it and is assigned a target area to reach. The agent can walk towards any direction and at different speeds, and can also stop moving. In other words, the agent can take

any possible action as long as the actions are limited by biomechanics. Additionally, the agent can sense any information about the whole environment such as the distance between itself and the target area, the direction from itself to the area, etc.. While there seem to be unlimited ways to reach the target area, optimal solutions exist if the agent is asked to finish the task with minimum energy and time. To learn how to act optimally through trial-and-error, the agent first tries to take an action based on what it has just sensed, and then according to a predefined criteria, a reward or penalty is given to the agent so that it understands whether such an action under that circumstance is good or bad. The larger the reward is, the more likely the agent will take the same action the next time it is in the same state. With this process repeated, the agent will eventually learn how to reach its target at an appropriate speed.

The illustrated case can be likened to a Markov Decision Process (MDP). In the MDP framework, the agent mirrors the player who tries to achieve its goal. Everything in the environment, including the status of the agent, constitutes a "state" in MDP. By taking different actions, the agent can transition from one state to another, and according to the criteria, the agent will be rewarded or penalized for this transition. An MDP can thus be defined as a 5-tuple: $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$ [66]. State space $\mathcal{S}$ includes all possible states s of the environment, and action space $\mathcal{A}$ contains any action a that an agent can take in any state. Both $\mathcal{S}$ and $\mathcal{A}$ can be discrete or continuous. With a discrete state space and action space such as a tic-tac-toe problem, there are countable possible states or actions in the spaces. By contrast, problems like robotic arm gripping objects have continuous ranges of actions and states. The transition probability function $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ indicates the likelihood of transitioning between states, where $P(s', r \mid s, a)$ implies the probability for the player to transition from one state s to another state s' and receive a reward r when taking an action a. The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ computes a specific reward for the player when a transition happens. γ is a discount factor in the range $[0, 1]$, which determines how important the future rewards are.

In some cases where the agent does not have full access to the current state of the environment. For example, a human player only knows information within its field of vision. We define such an MDP as a partially observable MDP (POMDP) with a 7-tuple: $< \mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, O, \gamma >$. The observation set $\mathcal{O}$ includes all the observations o that the player can get at a specific timestep, while $O : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{O})$ indicates the observation function, with $O(o \mid a, s')$ denoting the probability of having the

observation o when the agent takes the action a and transitions to the new state s'.

3.3 Core Concepts in Reinforcement Learning

As a branch of machine learning, reinforcement learning is an approach used to solve MDP problems, where the agent aims to learn a policy that can maximize the cumulative rewards it receives through interactions with the environment. Before delving into the major concepts of RL, it is necessary to clarify that an agent may either fully observe the state of the environment, where $O_t = S_t$, with O_t denoting the observation of an agent at time t, or it may only get a partial observation of the state. In the context of this research, to avoid repetitive explanations, unless otherwise specified, we will use *state* (s) to represent the information that an agent can get from the environment at a time step, which can either be the actual state or correspond to the partial observation in different scenarios.

3.3.1 Concepts and Terms

A *policy* π is a rule that determines how the agent acts [53]. Depending on the design of the problem, a policy π can be deterministic, represented as $\pi(s) = a$, so that for each state there will only be one available action, while a stochastic policy returns a probability distribution over some actions, with $\pi(a \mid s) = Pr(a \mid s)$ representing the probability of taking action a in the state s.

As an MDP problem, the current state and the action taken by the agent include all information needed to provide the next state[32], so the agent transitions to a new state s_{t+1}. The state transition can also be deterministic or stochastic.

Then the agent receives a *reward signal* $r_t = R(s_t, a_t, s_{t+1})$, a real-number value determined by the reward function, which is an immediate feedback from the environment after a state transition, telling the agent whether the action is good or not.

The policy will be updated based on the gained rewards and the agent repeats this procedure to learn a policy that maximizes the cumulative reward over time. We define the sequence of states and actions as the *trajectory* of the agent:

$$\tau = (s_0, a_0, s_1, a_1, \dots) \tag{3.1}$$

and the cumulative reward is called *return*, denoted as G_t, indicating that the return is computed after the time step t. When the number of steps is finite, it is easy to compute a return by summing up the rewards received after every step:

$$G_t = r_t + r_{t+1} + \cdots + r_T \tag{3.2}$$

where T is the final step. This type of return can not only be applied to the case where finite steps are required to finish a task, but can also be used to analyze the short-term performance of a policy. While there is no certain endpoint in an MDP problem, simply adding reward signals together may lead to an infinite result. To make the return converge, the discount factor γ is introduced, such that:

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_{t+k} \tag{3.3}$$

Since γ^k decreases as k increases, the formula indicates that immediate rewards are more important than future rewards.

With the definition of trajectory and return, the goal of reinforcement learning becomes more specific: to find a policy that maximizes the expected return. To calculate the expected return during training, major RL algorithms take the value function into consideration, denoted as V^π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s], s \in \mathcal{S}, \tag{3.4}$$

where $\mathbb{E}_\pi$ refers to the expected value when the agent acts following policy π. This function indicates the value of state s, the expected return at time step t when the agent is in the state s and will always use π as its policy in the future. Similarly, an action-value function, also referred to as the Q-function, is defined to estimate the value of a state-action pair:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a], s \in \mathcal{S}, a \in \mathcal{A}. \tag{3.5}$$

The Q-function computes the expected return when an agent is in the state s, takes an action a (which may not be derived from the current policy π), and then always follows π in the future.

According to Equation 3.2 and Equation 3.3, it is evident that G_t can be written in

a recursive form. The recursive relationship between the current return and the next return is helpful when calculating V^π and Q^π, especially in infinite-horizon problems. Taking the infinite-horizon return 3.3 as an example, we can expand V^π as follows:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t|s_t] \\ &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^k r_{t+k} + \cdots |s_t] \\ &= \mathbb{E}_\pi[r_t + \gamma \left(r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \cdots\right) |s_t] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1}|s_t] \end{aligned} \tag{3.6}$$

To calculate the expected value, we need to take all possible cases into consideration. If the policy is deterministic, then there is only one possible set of a_t, s_{t+1} and r_t, while with a stochastic policy, the probability of taking action $a_t = a$ in state $s_t = s$ is $\pi(a \mid s)$, and with an arbitrary a, and the probability of transitioning to state $s_{t+1} = s'$ and receive a reward r is $P(s', r \mid s, a)$. So the expectation of r_t is:

$$\mathbb{E}_\pi(r_t) = \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a) r \tag{3.7}$$

Similarly, every possible G_{t+1} is determined by which a_t is taken, which s_{t+1} is reached and which r_t is received. So the expectation of G_{t+1} is also weighted by the probabilities of taking action $a_t = a$ and the transition probability $P(s', r|s, a)$:

$$\mathbb{E}_\pi(G_{t+1}) = \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a) \mathbb{E}_\pi[G_{t+1}|s_{t+1} = s'] \tag{3.8}$$

As a result, the value function with infinite-horizon can be expanded as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|s_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s', r} P(s', r|s, a)[r + \gamma V^\pi(s')] \end{aligned} \tag{3.9}$$

In the Q-function, given $s_t = s, a_t = a$, when the policy is stochastic, the probability to transit from state s to s' by taking action a and receive a reward r is $P(s', r \mid s, a)$, while the probability to take an action a' in state $S_{t+1} = s'$ is $\pi(a' \mid s')$. On the other hand, in the case of deterministic policy, either $P(s', r \mid s, a)$ or $\pi(a' \mid s')$

simply equals to 1. So the Q-function with infinite-horizon of a stochastic policy can be expressed as:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t|s_t = s, a_t = a] \\ &= \mathbb{E}_{\pi}[r_t + \gamma G_{t+1}|s_t = s, a_t = a] \\ &= \sum_{s'}\sum_{r} P(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s')\mathbb{E}_{\pi}[G_{t+1}|s_{t+1} = s', a_{t+1} = a']] \\ &= \sum_{s',r} P(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s')Q^{\pi}(s', a')] \end{aligned} \tag{3.10}$$

The Equation 3.9 and Equation 3.10 are the *Bellman Equations* for v_π and q_π, which directly represents the relationship between the state value or action value of a state and its next state.

Lastly, in some algorithms like Policy Gradient methods and Actor-Critic methods[35], we introduce the concept of the advantage function to evaluate how much better or worse it is to take a particular action a in the specific state s compared to a randomly chosen action, according to the policy π:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{3.11}$$

Since it measures an action's relative benefit above the average action in a particular state, the advantage function is a common tool in reinforcement learning. By using this differential value in further computations, the variance of estimates is greatly decreased, improving the stability and effectiveness of learning algorithms.

With the knowledge of the basic concepts of RL, we can now better understand the categorization of RL algorithms. It is noteworthy that these categorizations reflect multiple dimensions, indicating from various perspectives what types of problems an algorithm is suited for. These categorizations are not mutually exclusive, meaning an algorithm can fall into multiple categories simultaneously. The next paragraphs will focus on a few of the more fundamental categories, laying the groundwork for the subsequent section on the algorithm used in this research.

3.3.2 Model-based Methods vs. Model-free Methods

One of the major distinctions in RL algorithms is whether there is a model of the environment. *Model-based* and *model-free* methods are the two main groups of RL

techniques that result from this divide.

Model-based methods incorporate an environment model, which can be used to estimate the result of taking an action without directly interacting with the environment. For stochastic state spaces, model-based methods often have a predefined transition probability function P such that based on the current state s and action a, we can sample the next state s' and derive the associated reward r from the probability distribution $P(s', r \mid s, a)$. In cases where the state space is deterministic, this transition probability function simplifies to directly mapping each state-action pair to a single next state and reward. By being able to compute values of different possible options before actually taking an action, model-based algorithms enable agents to plan ahead of time and thus increase learning efficiency. In some approaches such as the AlphaZero algorithm[50], the environment model is given to the agents, while in scenarios where the environment model is unknown, agents have to learn the model[16], which is a tough task when the environment is complex, especially when the environment is not stationary.

In contrast, algorithms that do not rely on environment models are referred to as *model-free* methods. These algorithms, including approaches like Deep Q-Networks (DQN) [36] and Trust Region Policy Optimization (TRPO)[47], cannot directly use transition probabilities and rewards from an environment model to calculate values of the returns as described in Equations 3.9 and 3.10 do. Instead, they implicitly use the Bellman equation to estimate the values and update these estimations by rewards received when interacting with the environment.

There are advantages and disadvantages to both model-based and model-free methods. Model-based methods enable prediction and planning ahead, which can result in more effective learning. However, such efficiency is often based on the premise of having an accurate model of the environment, which is difficult to obtain or learn especially when the environment is complex. Otherwise, the reliance on an environment model may be counterproductive. For example, an inaccurate environment model could provide incorrect predictions of values and lead to poor decision-making. Then, one bad decision after another can snowball an agent's performance further off track. Additionally, generalization to different environments may be restricted by dependence on a particular model.

On the other hand, with model-free methods, although the learning process may be less efficient without a model, the agent learns directly from its interactions with the environment rather than trying to first learn or act according to a model that

is not necessarily perfect. This makes it easier for the agent to avoid errors and be more flexible in adjusting its policies as the environment changes, making model-free methods more suitable for complex environments.

3.3.3 On-policy Methods vs. Off-policy Methods

Another way to classify RL algorithms is by making a distinction between *on-policy* methods, which directly learn a policy that makes decisions, and off-policy methods, which learn a policy based on experience generated by a separate policy or from other kinds of data sources.

On-policy approaches such as State-Action-Reward-State-Action (SARSA) [45] use the feedback from actions taken by the policy to update that policy itself. For example, with SARSA, the action values are updated in the following way:

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha\left[r + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)\right] \tag{3.12}$$

where π is the current policy, s and a are the current state and action, s' and a' denote the next state and action, r is the reward received, α is the learning rate, and γ refers to the discount factor.

In contrast, *off-policy* algorithms such as the Q-learning method [63] train the target policy with actions that are either taken from another policy defined as a behaviour policy, or that come from different forms of data sources, as the definition of the Q-learning algorithm shows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{3.13}$$

where $\max_{a'} q(s',a')$ refers to the maximum Q-value among all possible actions a' in the next state s', regardless of the current policy π. This equation illustrates the essence of off-policy methods, which use data from a greater variety of sources to optimize the target policy, allowing for a wider range of exploration. For instance, let us use the behaviour policy as an example to explain off-policy methods. The behaviour policy is not a perfect one, otherwise there is no need to learn a target policy. As a result, such a policy can take sub-optimal actions which enables the target policy to learn how to deal with more possibilities.

Both on-policy and off-policy approaches also have merits and demerits in terms of training efficiency, accuracy and stability, and storage space requirements. For on-

policy algorithms, directly updating the policy that makes decisions leads to a simple, coherent and efficient training process. However, as the policy gradually improves, it may lose opportunities of exploration. Specifically, when the policy performs better, it may learn to avoid taking actions that seems to be less optimal in the short term, but such actions may actually be part of an optimal trajectory from a long-term perspective.

Although off-policy algorithms improve this by separating behaviour policy and target policy, the fact that the behaviour policy which makes decisions cannot be affected by reward signals gives birth to other problems. When the target policy uses the feedback to improve itself, the next action derived from the behaviour policy may be less favourable that it is unlikely, if not possible, for the target policy to take such an action. This leads to the instability of training. Besides, off-policy methods also suffer from problems such as the need for more memory to process gathered experience and the possibility of overfitting data sources.

3.3.4 Value-based Methods vs. Policy-based Methods

Last but not least, RL algorithms can be categorized by whether they are *value-based* or *policy-based*. As mentioned in previous paragraphs, value functions can be used to represent how good a state or a state-action pair is. Value-based methods then aim to learn optimal value functions. For example, in the Q-learning method[63], an estimation of the action-value function, denoted as $\hat{Q}^{\pi}(s, a)$, is updated when the agent interacts with the environment and receives rewards, and the estimation is expected to converge to the optimal Q-function, Q^*:

$$Q^*(s, a) \doteq \max_{\pi} Q^{\pi}(s, a), s \in \mathcal{S}, a \in \mathcal{A}, \tag{3.14}$$

indicating the expected return when an agent is in state s and takes action a, and then always takes actions based on optimal policy. There are also approaches such as the Value Iteration algorithm [53] that learn an optimal state-value function V^*:

$$V^*(s) \doteq \max_{\pi} V^{\pi}(s), s \in \mathcal{S}, \tag{3.15}$$

which calculates the expected return when the agent is in state s and then always acts based on the optimal policy.

Policy-based methods aim to approximate the optimal policy π_* with a parameter-

ized policy function π_θ, where θ represents the set of parameters. Most policy-based methods such as REINFORCE [65] and TRPO [47] update the policy parameters using the widely adopted policy gradients theorem[54], which is based on the concept of gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)\big|_{\theta_t} \tag{3.16}$$

where θ_t denotes the current policy's parameter vector, θ_{t+1} refers to the updated parameter vector, and α is the learning rate. $\nabla_\theta J(\pi_\theta)$ is referred to as policy gradient, which is the gradient of the performance measure J with respect to the policy parameter, and $\big|_{\theta_t}$ indicates that this policy gradient is evaluated based on parameter vector θ_t. A simple metric can be the expected return of the agent's trajectory G_t. Assuming that the policy π_θ is stochastic, then we can compute the policy gradient as follows:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}(G_t) && (3.17)\\
&= \nabla_\theta \int P(\tau|\theta) G_t \, d\tau && (3.18)\\
&= \int \nabla_\theta P(\tau|\theta) G_t \, d\tau && (3.19)\\
&= \int P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) G_t \, d\tau && (3.20)\\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta) G_t]. && (3.21)
\end{aligned}$$

Since there are multiple possible trajectories, we compute the expected return value for all trajectories, $\mathbb{E}_{\tau\sim\pi_\theta}(G_t)$, to ensure that the policy can cope with the changing environment. In Equation 3.18, we expand $\mathbb{E}$ by the mathematical definition of expectation value, where $P(\tau|\theta)$ indicates that the probability distribution over all possible trajectories depends on the policy parameter θ. In typical reinforcement learning scenarios where neural networks are used to approximate policy functions, we can assume that $P(\tau|\theta)$ is well-behaved. This means that $P(\tau|\theta)$ is continuous and differentiable with respect to θ, and the rewards and probabilities are bounded. Given these conditions, we can apply the Linearity of Differentiation to move the derivative inside the integral from Equation 3.18 to Equation 3.19. Then, to facilitate the subsequent calculations, we use the log-derivative trick such that $\nabla_\theta \log P(\tau|\theta) =$

$\frac{1}{P(\tau|\theta)}\nabla_\theta P(\tau|\theta)$, which results in Equation 3.20. Finally, Equation 3.21 is derived again by the definition of the expectation value. The following problem is then to compute $\nabla_\theta \log P(\tau|\theta)$:

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \log[P(s_0)\prod_{t=0}^{T} P(s_{t+1}, r_t|s_t, a_t)\pi_\theta(a_t|s_t)] \tag{3.22}$$

$$= \nabla_\theta[\log P(s_0) + \sum_{t=0}^{T}[\log P(s_{t+1}, r_t|s_t, a_t) + \log \pi_\theta(a_t|s_t)]] \tag{3.23}$$

$$= \nabla_\theta \log P(s_0) + \sum_{t=0}^{T} \nabla_\theta \log P(s_{t+1}, r_t|s_t, a_t) + \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{3.24}$$

Since the derivation is computed with respect to θ and the probability of any state only depends on the environment, the derivatives of $\log P(s_0)$ and $\log P(s_{t+1}, r_t|s_t, a_t)$ are 0. As a result, the policy gradient can be written as:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta) G_t] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t]\end{aligned} \tag{3.25}$$

We can use a function other than G_t to compute the performance of the policy and come up with results similar to (3.25). Thus the general form of the policy gradient can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t] \tag{3.26}$$

where Φ_t can be any function such as Q-function $Q^{\pi_\theta}(s, a)$ and advantage function $A^{\pi_\theta}(s, a)$.

There are also some methods that do not rely on policy gradients, such as Evolutionary Strategies (ES) [4] which optimize policies with mechanisms similar to biological evolution.

Depending on how the action space is designed, value-based approaches and policy-based approaches are suitable for different types of scenarios and also have

their own advantages and disadvantages. In the cases where the action space is discrete and finite, value-based methods are simpler and can be utilized to their full advantage. Imagine an algorithm whose goal is to learn a q-function, and given a state, if the available actions are finite, then it is convenient to compare the value of each possibility by means of the estimated q-function. In contrast, when the action space is continuous and deriving values for all possible action choices becomes computationally prohibitive or even difficult to realize, policy-based algorithms turn out to be a better choice. However, applying policy-based methods also means that training may become more unstable. With value-based methods, the value update for a state or a state-action pair relies on the existing return estimation of subsequent trajectory and the actual reward received by the agent when it transits to this state (and takes a certain action), allowing the estimated value function to gradually converge towards the optimal one, so the policy is also incrementally and implicitly updated. On the other hand, with policy-based methods, since the policy function directly computes the next action or probability distribution of the action,, adjustments to the policy parameters may affect the choice of a subsequent series of actions, and significantly different trajectories in turn cause the policy to make more changes, leading to training instability.

3.4 Extension to Multi-Agent Systems: Stochastic Games

The previous section introduced the definition of RL and some common categorizations of algorithms, and compares the strengths and weaknesses of different categorizations of algorithms in various situations. These concepts also apply to multi-agent scenarios, which will be introduced in this section.

Similar to the example at the beginning of section 3.2, now imagine a plane with two agents, A and B, starting from different locations and each with a goal area to reach. For agent A, the walking strategy used in the case of single agent may no longer be optimal, as it will then have to avoid not only obstacles to reach the end point, but also agent B. Thus how agent B acts will affect agent A's actions, and vice versa.

As an extension of the MDP to a multi-agent scenario, MARL is more like a Markov game [33] that each agent's actions affect not only the environment but also

other agents. A Markov game, also known as a stochastic game, can be defined by a tuple $< N, \mathcal{S}, \{\mathcal{A}^i\}_{i\in N}, P, \{R^i\}_{i\in N}, \gamma >$, where N denotes the number of agents and each of them has their own set of actions $\mathcal{A}^i$, so the action space can be represented as: $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^N$. Similarly, R^i refers to the reward function of agent i. $\mathcal{S}$ is the state space containing all possible states observed by all agents, and P denotes the transition probability between any states, given any joint action $a \in \mathcal{A}$ of all agents. γ denotes the discount factor that is similar to the one in an MDP.

In the case where partial observations make up the complete state space when put together, this type of Markov game can be defined as a Partially Observable Stochastic Game (POSG). Besides the tuple mentioned in the last paragraph, we add $\{\mathcal{O}^i\}_{i\in N}$ to describe the game, which denotes the observation set for each agent i and where the joint observation set is represented as $\mathcal{O}$. As a result, each agent has its own policy : $\pi^i \in \Pi^i : \mathcal{O} \to \Delta(\mathcal{A})^i$, which means that for all possible policies for agent i in the set Π^i, the policy π^i maps every possible observation from $\mathcal{O}$ to a probability distribution over the action space $\mathcal{A}^i$ [66].

As introduced in the previous section, RL algorithms can also be categorized in different dimensions when applied to multi-agent scenarios. However, in MARL, multiple agents make the structure of the training and the strategy of the agents' actions more complex, thus introducing more categorization dimensions that need to be decided on a case-by-case basis. The next few subsections list three categorization settings that are relevant for scenario design.

3.4.1 Cooperative Settings vs. Competitive Settings

When a group of agents share a common reward signal and act with the objective of maximizing it, such a scenario can be described as having a *cooperative* setting [53]. For example, if several agents simulating farmers aim to learn how to finish sowing a plot of land quickly, they have no conflict of interest at this point, and shortening the total elapsed time is their common goal. In a *competitive* scenario, on the other hand, the agents each receive different reward signals and learn to maximize their own reward signals, turning the scenario into a game. Maximizing their own interests may be at the expense of other agents [53]. For instance, in the example of two agents walking towards their destinations mentioned earlier in this section, agent A needs to reach its destination as fast as possible while avoiding bumping into B. In order to go around B, A needs to predict B's possible behaviour and adjust its route, and the

same is true for B. There are also scenarios where the two paradigms are mixed, such as a soccer game where teammates on the same team are in a cooperative setting, while two competing teams are in a competitive setting with one another.

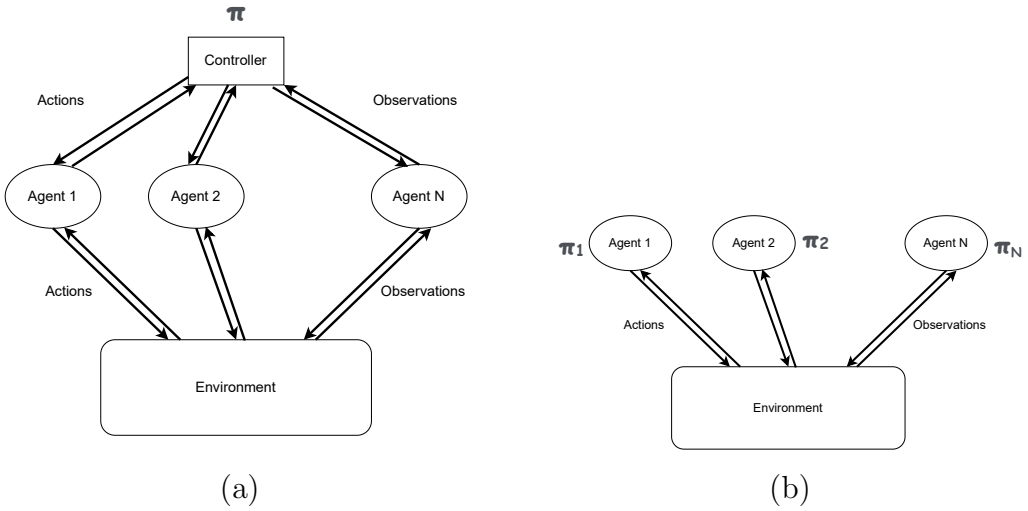
3.4.2 Centralized vs. Decentralized

When categorized in terms of who makes decisions for the individual agents, the learning procedure can usually be divided into *centralized* and *decentralized* settings, and each of these methods has its own merits and demerits. Figure 3.1 depicts the differences between these two types of algorithms. In centralized learning, a central controller has access to the states and observations of all agents, and determines policies for all agents [67]. The controller has a comprehensive view of the entire environment, which allows it to obtain policies quicker given more necessary information, but as the number of agents grows, the computational complexity also increases. Furthermore, agents do not always communicate with each other, especially in many real-world scenarios such as human crowds. In contrast, with decentralized setting, agents train their own policies independently without exchanging information. Although decentralized methods enable agents to adapt to changes quicker without waiting for others, and the framework is more relevant to some cases in reality, the problem of non-stationarity cannot be ignored. While an agent treats other agents as part of the environment and try to learn based on its local observation, those other agents are also changing the environment and learning. The policies of each agent which determine their actions can update frequently so the training process becomes complex and may even have the problem of non-convergence [55].

There are, of course, ways to compensate for the lack of stability while retaining the advantages of decentralized learning. This research uses an approach called *parameter sharing* [56]. Each agent learns a policy independently, but all policies share parameters, i.e., each agent has an identical neural network. In this way, each agent only needs to deal with its own observations, and the whole training process only needs to learn the number of parameters for one neural network, which alleviates the non-stationary problem while reducing the overall computational cost.

3.4.3 Homogeneous vs. Heterogeneous Agents

There is another categorization that depends on whether the agents are identical or not. For *homogeneous* agents, they usually have the same state space, action space,



(a) (b)

Figure 3.1: Differences between centralized learning and decentralized learning. Figure (a) shows the framework of centralized learning, where the central controller makes decisions for every agent based on shared observations. Figure (b) illustrates the framework of decentralized learning, where there is no central controller and every agent makes decisions based on its individual observations and its own policy. In the case of a parameter sharing model, all agents share the same set of policy parameters, which means that $\pi_1 = \pi_2 = \cdots = \pi_N$.

and reward functions, which means that they will react in the same way to the same environment, such as soldiers in a formation at a military parade. *Heterogeneous* agents, on the other hand, may have different action spaces or reward functions, as they literally do. For instance, a synthetic crowd of people of different ages or personalities may have various average walking speeds and stride lengths [66]. When agents are homogeneous, they are relatively easy to train, whereas heterogeneous agents are more likely to be flexible in different scenarios, despite the complexity of their training.

3.5 Key Challenges in MARL Implementation

Although there is a large body of literature that continuously improves the algorithms of RL to cope with various complex problems, there are still unavoidable challenges in the field of RL today.

3.5.1 Reward Engineering

One of the benefits of RL over supervised learning, which necessitates the preparation of a large amount of accurate data to be used as criteria for judging, is that the engineer designing reward signals does not need to be an expert on what makes a correct action. To train a bipedal agent to walk toward its destination, for instance, it is not essential to know the precise orientation and speed of the agent's body or feet at each moment. Instead, the agent is guided to gradually master the laws of action, or the policy, by generating rewards based on the amount of time spent, the ability to reach the destination and many other factors. However, with this advantage comes the challenge of reward engineering.

One of the difficulties is that it is not easy to design rewards that match the designer's desire and ensure the agent do what it is supposed to do. In the case of this study, if the agent is a particle that can only move in four directions at a fixed speed, and there are no obstacles or multiple agents in the environment, training it to reach the goal in the fastest possible time would only require considering a time penalty and a reward for reaching the goal. However, a bipedal agent in a multi-agent scenario requires much more complex considerations. How an agent moves its body and feet, and whether it stops, slows down, or walks around an obstacle depends not only on how much time it takes, but also on how much energy it consumes to perform the action. Thus, its energy consumption, time, penalties for colliding with obstacles and rewards for reaching its destination need to be balanced in order for the agent to learn the desired decision-making policy. Otherwise it may obtain rewards in unexpected ways. For example, when the penalty for collision is too small and the reward for reaching the destination too large, the agent may prefer to hit the wall or other agents in order to reach its destination.

Another problem is properly assessing the progress in reaching the goal. Assume that an agent reaches its destination and receives the corresponding reward signal. Although the last state-action pair when it arrives at the destination directly triggers the reward signal, it is clear that the whole trajectory which contributes to this result. However, since the agent is assigned random destinations, an agent may spend different amounts of time and travel different distances to receive the reward signal for reaching its target. This leads to the target reward being too sparse to judge whether the agent's progress toward the goal has improved.

To address these challenges, Sutton and Barto [53] proposed a number of ap-

proaches, most of which were also utilized in this research. For example, we can tune the reward function with a trial-and-error search until we get an ideal result. We can also automate this search process by using an optimization algorithm to evaluate a set of candidates with an objective function which represents the designer's actual intent. Admittedly, the large amount of computation required for trial and error is a drawback of this approach.

One way to make the reward signal less sparse is to set rewards for sub-goals. For example, in this project, a reward signal can be set for "the agent is closer to the destination than it was a moment ago", which is triggered at some point along the way, so that the agent can learn the process of approaching the goal. The specific design will be introduced later. However, there is a risk that the agent may focus too much on the reward of the sub-goal and completely disregard the final goal. One solution could be to set the reward value of the final goal to be significantly larger, so that even if the reward can be obtained along the way, the agent still learns to obtain the larger reward.

Another way to mitigate the problem posed by sparse rewards is reward shaping, where some more frequent reward signals are added at first to speed up the training process, and then the reward function is gradually changed back to the original one. Although it can improve efficiency, designing these extra rewards in a reasonable way poses a new challenge, and an improper design may slow down the training.

Last but not least, there is an approach to fundamentally avoid the dilemma of reward engineering called imitation learning, i.e., mimicking the behaviours of the expert. Although it sounds a bit like supervised learning, imitation learning involves learning a policy to make a sequence of decisions, so it is still a form of RL. The advantage of this approach is its efficiency and its ability to reduce the cost of trial and error in designing the reward function, but it also means that the learned policy will be limited by the performance of the expert. The agent will not be able to explore better solutions than the expert's behaviour, and it may not be flexible enough to deal with different scenarios because the actions given by the expert are not generalized enough.

3.5.2 Balancing Exploration and Exploitation

Another challenge that RL must face is balancing exploration and exploitation. As mentioned in previous sections, in RL the agent learns a policy through interactions

with the environment, so the agent needs to explore new actions to discover potentially better options than those previously experienced. After accumulating sufficient knowledge through exploration, the agent exploits this information to refine its policy by taking actions it believes will obtain the highest reward signals. If the agent spends too much time exploring, training becomes inefficient. On the other hand, focusing too much on exploitation without adequate exploration may result in training converging at local optimal solutions.

One of the common approaches to balance exploration and exploitation is the epsilon-greedy algorithm[53], in which the agent spends most of its time exploiting experience, with only a small probability ϵ that it will choose to explore. Another widely used algorithm is softmax selection[53], where the agent chooses which action to take based on the Boltzmann-distributed probability of the estimated state-action value:

$$P(a) = \frac{e^{\frac{Q^{\pi}(s,a)}{\tau}}}{\sum_{b \in \mathcal{A}} e^{\frac{Q^{\pi}(s,b)}{\tau}}} \tag{3.27}$$

where τ refers to a parameter that determines the tendency to explore. High values of τ lead to a more uniformly distributed probability across actions, indicating that the agent tends to explore more, while low values of τ mean the agent is inclined to choose the action with a high estimated state-action value.

3.5.3 Non-stationary Problem in MARL

The problem of non-stationarity in a multi-agent scenario has already been introduced in section 3.4.2. As each agent adjusts its policy, it simultaneously changes the environment's dynamics for other agents. Every agent attempts to adapt to the changing strategies and actions of other agents, and this type of behaviour leads to further adaptations by others. In this vicious circle, agents are unable to converge on a stable policy, which affects training efficiency. In addition to parameter sharing mentioned in section 3.4.2, there are several more commonly utilized approaches to mitigate non-stationary problems.

One method is Centralized Training with Decentralized Execution (CTDE) [34]. As the name suggests, a central controller is used in the training process to synthesize the observations, actions and rewards from all agents to optimize a shared policy. During the execution phase, each agent makes decisions independently based only on

its own observations and the policy learned during training. The advantage of CTDE is its operational efficiency during the execution phase. Compared to centralized execution, this approach is more suitable for real-time scenarios that require quick decision-making. However, centralized training still requires complex computations. Furthermore, if information essential enough to influence decision-making can only be obtained through a global view, the decentralized execution of the agent may not be able to achieve excellent performance based on its own observation alone.

Another way to address this is to adjust the learning rate during training. Learning rate, usually denoted as α, is a hyper-parameter that controls the speed at which newly acquired experience changes existing policies. Traditionally, a higher learning rate is used at the beginning of training to allow the agent to learn faster, and as training proceeds, the learning rate can be decreased by linear decay, exponential decay, etc., to slow down the policy updates, prompting a gradual convergence of the policy. There are other ways of adjusting the learning rate, such as applying Adaptive Moment Estimation (Adam)[27], which adjusts the learning rate for each parameter in the policy according to the estimation of the first and second moments of the gradients. While adjusting the learning rate can significantly mitigate the problems associated with non-stationary environments, designing learning rates that can accommodate both policy updates and stable learning remains a challenge.

3.6 Discussion

This chapter starts with the concept of an MDP, provides an introduction to the nature of RL and the categorization of common algorithms, and then extends to the multi-agent scenario, further introducing stochastic games and MARL, as well as enumerating a number of categorizations in multi-agent scenarios in terms of the dimensions of training architectures and decision-making strategies.

In order to address these problems, careful consideration and selection of appropriate algorithms and training structures are needed during the research process. In this study, we chose to use the PPO algorithm, designed a competitive framework, and used a shared-parameter training method, which will be described in detail in the subsequent sections.

Chapter 4

Biomechanical Modeling of Human Bipedal Locomotion

This chapter introduces the biomechanical modelling of bipedal locomotion. By introducing the concept of the 3D inverted pendulum model, it provides an intuitive and straightforward to calculate the trajectory of the feet and centre of mass (COM) as the agent takes each step, ensuring both realism in the synthesized crowd and computational efficiency.

4.1 Introduction

The use of a bipedal model as an agent is one of the primary focuses of this study. The analysis of human walking gait has been well studied, and bipedal movement is far more complex than particle movement. When an agent is modelled as just a particle, it is possible to easily determine its direction, speed, and duration of movement, but that is the extent of the information. For instance, when a person wants to change their orientation, it is nearly impossible to rotate in place without moving their feet. At this point, a biomechanically-based model of bipedal locomotion can compute how the movement of the two feet corresponds to the movement of the body's centre of mass (COM). However, for a particle-based agent, it either rotates in place—which significantly reduces realism—or it simulates only the trajectory of the body (i.e., the centre of mass), without accounting for other parts, like the feet, potentially complicating the computation. For example, in a scenario where two crowds of people meet and go in different directions, or where people meet head-on at

a narrow entrance/exit, there may be a scenario where one foot of person A is located between the feet of person B. In this case, A's foot may be located beneath B's body, but they do not collide. The scenario described above cannot be accurately modelled using a particle-based agent. In conclusion, the bipedal model contains much richer information compared to the particle-based model and can more realistically simulate human walking trajectories. This increased realism allows for more accurate decision-making in gait planning.

It is important to understand the basic concepts of the human gait cycle, as this will aid in comprehending the principles involved in constructing the bipedal locomotion model required for this study. Human walking consists of a series of repeated gait cycles, during which each foot alternately touches and leaves the ground. One foot remains on the ground to support the body while the other swings forward in the direction of walking, then touches the ground. At this point, both feet support the body until the initial supporting foot leaves the ground, and the body once again relies on a single foot for support. During this process, the human body, or COM, moves vertically in the sagittal plane, laterally in the frontal plane, and with slight rotation in the transverse plane, all coordinated with the lower limb movements. Although the bipedal locomotion model used in this study simplifies the actual human walking process, understanding the major phases of the gait cycle is crucial for recognizing why the 3D inverted pendulum can be employed for analogy and analysis.

The biomechanical model used in this study, the 3D inverted pendulum model, provides a simplified yet effective representation of the trajectory of the COM when supported by a single foot. It assumes that the human centre of mass and this leg form an inverted pendulum during the phase when the person is supported by a single leg. Furthermore, as the centre of gravity shifts to the left and right sides of the body in the horizontal plane during walking, the trajectory of the centre of mass during a segment of inverted pendulum motion is projected onto the horizontal plane. This results in a parabolic segment, which is approximated using a second-order Taylor expansion. Simplified computation using a local coordinate system then provides the formula for the COM trajectory in this study. To simplify the calculation while preserving the basic characteristics of human gait, this study makes several assumptions, such as a fixed distance between the COM and the feet, a fixed COM height, and the use of a unified modelling framework to analyze the COM trajectory.

In the following sections, we first introduce the basic concepts of the gait cycle and describe its main phases. Next, we present the concept of the 3D inverted pendulum,

discuss the assumptions required to reduce computational complexity, and provide a detailed derivation of the formulas used to calculate the COM trajectories in this study, laying the foundation for practical applications in the later sections.

4.2 Fundamental Principles of Human Gait Cycle

The process of human walking can be summarized as a repetition of the gait cycle, which refers to the time duration from one certain walking event to the next time when it is repeated. As shown in the Figure 4.1, for instance, suppose a gait cycle begins when the left heel just touches the floor, which can be defined as an *initial contact* [64]. With the left foot fully on the ground, the right foot pivots on its front and lifts. After the right toes leave the ground, the right limb swings toward the direction of movement and strikes the ground with its heel. Following this, the right limb replicates the movements of the left one and vice versa. Finally, the gait cycle ends right before the left initial contact happens again.

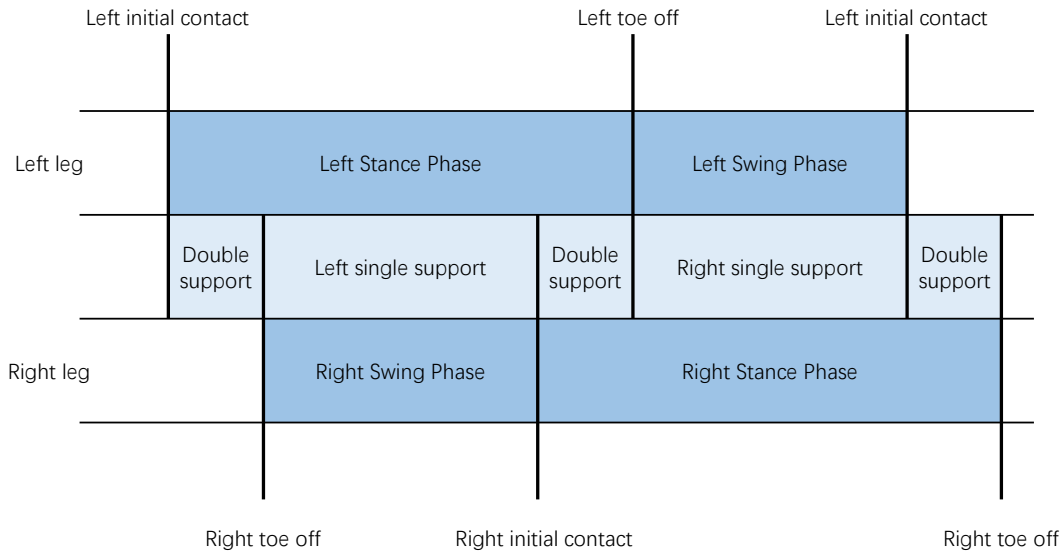


Figure 4.1: Normative human bipedal gait cycle.

The gait cycle includes two major phases: *stance*, when the foot touches the ground; and *swing* as the foot moves forward [6]. The COM is supported by two feet when they are both on the ground, while in the *single support phases*, one of the limbs swings in the air.

Figure 4.2 depicts terms describing foot placements in the field of gait analysis. *Stride Length* refers to the distance between two consecutive placements of the same foot, while *Step Length* is the distance between two successive placements of different feet.

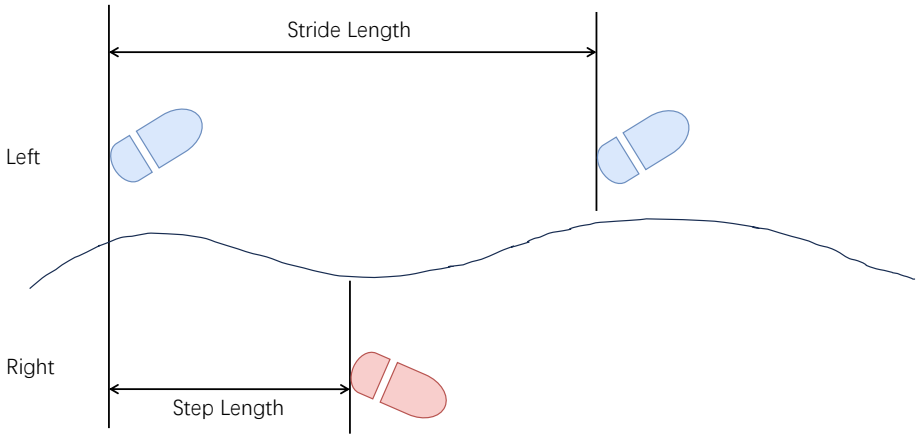


Figure 4.2: Step length and stride length in normative human bipedal gait cycle.

4.3 Mathematical Framework for 3D Inverted Pendulum Model

The locomotion of an agent in this study is formulated on the idea of a 3D linear inverted pendulum[24]. This section will explain the definition of a gait cycle and the inverted pendulum model, and go into depth about the math.

4.3.1 Model Simplification and Key Assumptions

Before delving into the concept of the 3D inverted pendulum model, it is important to emphasize a few assumptions that this research introduces to reduce computational complexity.

Firstly, this study presupposes that the length of the leg stays unchanged. People frequently bend their knees in the gait cycle, leading to slight adjustments in the

length between their COM and feet. Following the analogy of the inverted pendulum model, we ignore the change in leg lengths based on a small-angle approximation.

Secondly, this research assumes that the COM remains at a fixed height. During a gait cycle, the COM, which is approximately aligned with the height of the hip from the ground, fluctuates when the walking posture changes. As the variation in height is relatively small, the trajectory of the COM is calculated by projecting it onto a 2D horizontal plane, and for the purpose of simplifying both agent movement animation and computations, this assumption is made in this study.

Thirdly, the 3D inverted pendulum model only analyzes the COM trajectory during the single support phase, whereas the trajectory during the short period of double support phase may be smoother and would need to be calculated in some other way. In this research, we use a unified modelling framework that ignores the mathematical expression for the double support phase and use only the formulas derived from the 3D inverted pendulum model.

4.3.2 Derivation of the 3D Inverted Pendulum Dynamics

The inverted pendulum model is an appropriate analogy to represent the dynamics of bipedal locomotion [29]. As shown in Figure 4.3, in this model, when the pelvis is in a single support phase, the stance leg is comparatively straight, so the COM can be considered as oscillating around the stance foot like an inverted pendulum when the opposite foot swings. As the COM rises and falls, its trajectory in the sagittal plane resembles a sequence of circular arcs [42]. Furthermore, from the perspective of the transverse plane, when one foot initially contacts the floor, the COM begins to move laterally toward that side and reaches maximum lateral displacement at the midpoint of that foot's stance, then starts to shift toward the other side. The movements repeats, so the COM trajectory appears as a sinusoidal curve [39].

As a result, combining the trajectories from these two planes indicates that the actual COM trajectory is a multidimensional curve. In this study, the movement of COM is considered as a 3D linear inverted pendulum model[24], and this curve will be projected onto the 2D horizontal plane and approximated by parabolas.

Figure 4.3: COM movements on the sagittal plane during the single support phase based on the inverted pendulum analogy. The black horizontal line represents the ball of the foot, while the long vertical lines in red, black, and blue indicate the beginning, middle, and end of the leg during the single support phase. The yellow dotted line represents the COM trajectory, which moves up and down like an inverted pendulum as the leg rotates.

Suppose a limb is in its single support phase, and construct a 3D coordinate system originating from its foot, then the leg can be treated as a mass-less pole with length r, and the point $p = (x, y, z)$ on its tip is considered as the COM with mass m. As the inverted pendulum rotates around the origin, the position of p is determined by a set of state variables $s = (\theta, \phi, r)$, as shown in Figure 4.4.

$$x = r \sin \phi \tag{4.1}$$

$$y = -r \sin \theta \tag{4.2}$$

$$z = rD \tag{4.3}$$

$$D = \sqrt{1 - \sin^2 \theta - \sin^2 \phi} \tag{4.4}$$

The Jacobian matrix J, describing how small changes in s would affect p, is then defined as:

$$J = \frac{\partial p}{\partial s} = \begin{pmatrix} 0 & r \cos \phi & \sin \phi \\ -r \cos \theta & 0 & -\sin \theta \\ -r \cos \theta \sin \theta / D & -r \cos \phi \sin \phi / D & D \end{pmatrix} \tag{4.5}$$

Accordingly, J^{-1} indicates how small changes in position p in Cartesian coordinates would change the state variables s.

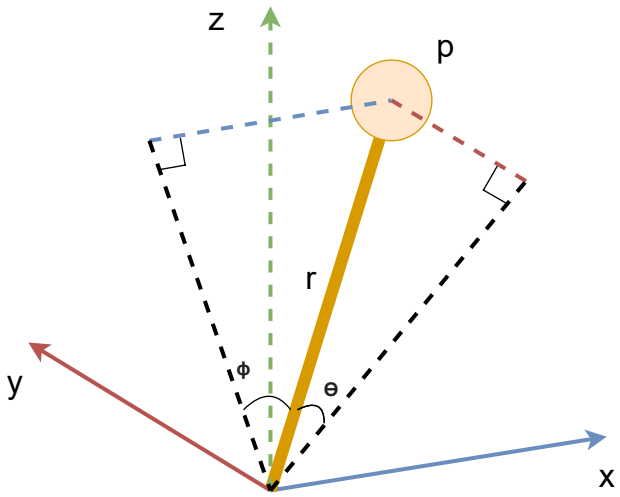


Figure 4.4: 3D inverted pendulum

Then, consider $(\tau_\theta, \tau_\phi, f)$ as the actuator torques and force corresponding to $s = (\theta, \phi, r)$, which would affect the acceleration of COM in a Cartesian coordinate system. With these torques, force, and gravity, the equation of motion is:

$$m \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = (J^T)^{-1} \begin{pmatrix} \tau_\theta \\ \tau_\phi \\ f \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} \tag{4.6}$$

where $\ddot{x}, \ddot{y}$ and $\ddot{z}$ denotes accelerations along x, y, z axes and g is the gravity acceleration.

For ease of calculation, both sides of Equation 4.6 are pre-multiplied by J^T:

$$m \begin{pmatrix} 0 & -r\cos\theta & -r\cos\theta\sin\theta/D \\ r\cos\phi & 0 & -r\cos\phi\sin\phi/D \\ \sin\phi & -\sin\theta & D \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} \tau_\theta \\ \tau_\phi \\ f \end{pmatrix} - mg \begin{pmatrix} -r\cos\theta\sin\theta/D \\ -r\cos\phi\sin\phi/D \\ D \end{pmatrix} \tag{4.7}$$

For the first row in Equation 4.7, multiply both sides by $D/\cos\theta$ and we can get:

$$m(-rD\ddot{y} - r\sin\theta\ddot{z}) = \frac{D}{\cos\theta}\tau_\theta + r\sin\theta - mg \tag{4.8}$$

Using Equations 4.2 and 4.3 to substitute some of the terms in the equation, the

dynamics along the y-axis can be summarized as:

$$m(-z\ddot{y} + y\ddot{z}) = \frac{D}{\cos\theta}\tau_\theta - mgy \tag{4.9}$$

Similarly, using the second row in Equation 4.7 with Equation 4.1 and Equation 4.3, we derive an equation that describes the dynamics along the x-axis:

$$m(z\ddot{x} - x\ddot{z}) = \frac{D}{\cos\phi}\tau_\phi + mgx \tag{4.10}$$

In this study, the agents are constrained to walk on a flat plane. Based on the assumption in Section 4.3.1, suppose the movement of the COM is on a plane with a normal vector $(k_x, k_y, -1)$, and the plane intersects the z-axis at $z = z_3$, then the plane can be represented as:

$$z = k_x x + k_y y + z_3, \tag{4.11}$$

and the second derivatives of (4.11) is:

$$\ddot{z} = k_x\ddot{x} + k_y\ddot{y}. \tag{4.12}$$

With these constraints applied to Equations 4.9 and 4.10, and defining:

$$\mu_{theta} = \frac{D}{\cos\theta}\tau_\theta, \tag{4.13}$$

$$\mu_{phi} = \frac{D}{\cos\phi}\tau_\phi, \tag{4.14}$$

for input-nonlinearity compensation, we get:

$$\ddot{y} = \frac{k_x}{z_3}(\ddot{x}y - x\ddot{y}) - \frac{1}{mz_3}\mu_\theta + \frac{g}{z_3}y \tag{4.15}$$

$$\ddot{x} = \frac{k_y}{z_3}(x\ddot{y} - y\ddot{x}) + \frac{1}{mz_3}\mu_\phi + \frac{g}{z_3}x \tag{4.16}$$

In this study, the agents walk on a horizontal plane, so $k_x = k_y = 0$. Additionally, there is no torque applied to the COM because the agents move using passive dynamics, so $\tau_\theta = \tau_\phi = 0$, and correspondingly, $\mu_{theta} = \mu_{phi} = 0$. Finally, the assumptions in Section 4.3.1 indicate that $z_3 = r$. As a result, the updated Equations 4.16) and 4.15 are:

$$\ddot{x} = \frac{g}{r}x \tag{4.17}$$

$$\ddot{y} = \frac{g}{r}y \tag{4.18}$$

We can observe that Equations 4.17 and 4.18 are homogeneous second-order differential equations, which can be solved by solving their characteristic equation. Consider Equation 4.17 as an example: the solutions of its auxiliary equation $a^2 - \frac{g}{r} = 0$ are $a = \pm k$, where $k = \sqrt{\frac{g}{r}}$. Thus, the general form of the solution to $x(t)$ is:

$$x(t) = c_1 e^{kt} + c_2 e^{-kt} \tag{4.19}$$

and its first derivative describing velocity:

$$\dot{x}(t) = c_1 k e^{kt} - c_2 k e^{-kt} \tag{4.20}$$

Along x-axis, given an initial position $x(0) = x_0$ and related velocity $\dot{x}(0) = v_{x_0}$, c_1 and c_2 can be solved and the solution of $x(t)$ is:

$$x(t) = \frac{(kx_0 + v_{x_0})e^{kt} + (kx_0 - v_{x_0})e^{-kt}}{2k} \tag{4.21}$$

By using a second-order Taylor approximation, it is simplified to:

$$x(t) = x_0 + v_{x_0}t + \frac{1}{2}x_0 k^2 t^2 \tag{4.22}$$

and we can get $y(t)$ in a similar way:

$$y(t) = y_0 + v_{y_0}t + \frac{1}{2}y_0 k^2 t^2 \tag{4.23}$$

Equations 4.22 and 4.23 approximate a parabolic curve on 2D plane in the form of parametric representation. To make it easier to calculate and analyze, and meanwhile retain the characteristics of a 2D parabola, we set up a local coordinate system and express this curve with the simplest parabola formula, as mentioned in [52]:

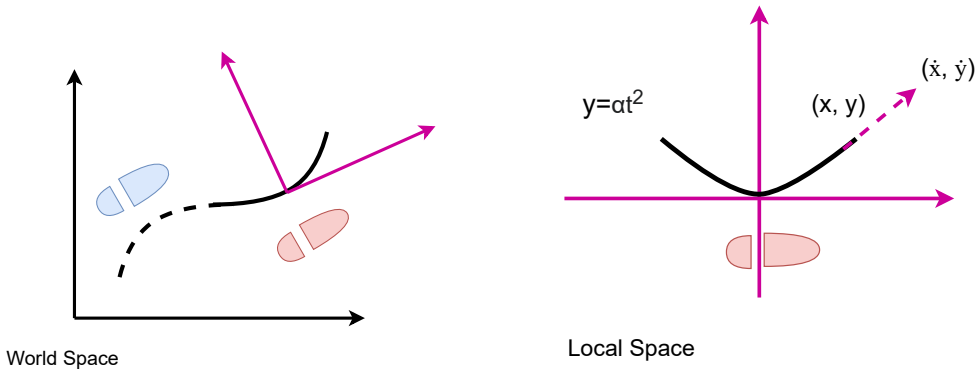
$$x(t_l) = v_x^l t_l \tag{4.24}$$

$$y(t_l) = \alpha t_l^2 \tag{4.25}$$

$$\dot{x}(t_l) = v_x^l \tag{4.26}$$

$$\dot{y}(t_l) = 2\alpha t_l \tag{4.27}$$

where α is a non-negative coefficient defining the parabola's curvature, and the superscript l denotes parameters in the local coordinate system. The local time parameter t_l represents the re-parameterized time in this coordinate system. Figure 4.5 illustrates how this local parameterization relates to the world space trajectory for consecutive left and right steps. The original curve is transformed and re-parameterized as a canonical parabola in the local coordinate system, allowing efficient computation of global positions through coordinate transformation.



(a) (b)

Figure 4.5: Step trajectories in world and local space [52]. (a) World space representation showing foot positions and COM trajectories for consecutive steps. The dashed line indicates the COM trajectory during the left step, and the solid line during the right step. (b) Local space representation of the right step.

4.4 Discussion

This chapter presents a biomechanical model for bipedal locomotion based on the 3D inverted pendulum concept. The model simplifies the complex human gait cycle by projecting the trajectory of the centre of mass (COM) onto the horizontal plane, representing it as a curved segment approximated by a parabolic equation. This approach provides a biomechanically sound and computationally efficient way to simulate human-like movement. The integration of this model with the MARL framework, which will be discussed in the next chapter, forms a vital component of this study.

Chapter 5

Learning to Step in Multi Agent Environments

Building on the theoretical foundation established in the previous chapters, this chapter transitions into practical application by detailing the creation of the agent model for this study, the algorithms used to generate the training environment, the selection and application of the PPO algorithm within the MARL framework, and the construction of a function to compute the agent's bipedal locomotion using the neural network's output.

5.1 Introduction

In this work, many agents were dropped into a randomly constructed maze and instructed to navigate in the direction of a specified destination point while learning to avoid collisions. Every agent in this study has the same body size, activity plan, and biomechanical limitations. The introduction of diversity, such as considering agents with mobility disabilities, will be discussed in a future study. In this study, the mean height and weight values for Canadians were used for the agents, along with corresponding mean values for the proportions of major body parts, such as shoulder width and leg length. For natural walking, the mean step speed was used as mentioned in [64]. The training environments in this study were randomly generated mazes, with complexity and density controlled by two parameters. New environments were randomly generated as the training progressed to prevent overfitting.

The decision to use the PPO algorithm in this study is based on its robustness

in handling dynamic environments and continuous action spaces. According to the taxonomy of RL algorithms introduced in Chapter 3, PPO is a model-free, on-policy, policy-based algorithm. For a study like this, where there is no existing model reference and both the environment and action space are complex, these properties make PPO highly suitable. In addition, this study uses the decentralized learning framework mentioned in Section 3.4.2 and implements parameter sharing, meaning all agents share the same set of policy parameters, but each agent has access only to its own observations of the environment and makes independent decisions. This framework reduces computational requirements compared to centralized learning, which relies on a central controller to consolidate the observations of all agents to make decisions. Moreover, it enhances the stability of policy updates by sharing parameters. In addition, the tuning of the reward function is a major challenge in the field of RL. In this study, alongside considering rewards and penalties for time spent, collisions with obstacles, and reaching the destination, we also incorporate the energy formula proposed in [52] when designing the reward function, ensuring that the agent's policy adjustments are as realistic as possible.

The platform used for training in this study is Unity, and the ML-Agents toolkit was utilized [22]. This toolkit conveniently allows the selection of the PPO algorithm, adjustment of hyperparameters for training, and configuration of the observation and action spaces for each agent. The output of the neural network obtained using this toolkit is a vector representing the agent's actions. The values of the individual elements in this vector need to be linearly transformed to obtain the variables required for this study, such as the agent's speed, step duration, and so on. After obtaining the variable values through the AI model, the 3D inverted pendulum model is used to calculate the specific movements of each agent's feet and COM, which will be explained in detail in this chapter.

This chapter will first introduce the modelling of the agents, including the dimensions of the main body parts, the simulation of the field of view, and the biomechanical constraints. Next, it describes the mathematical formulation of steps in our framework, introducing the model parameters and derivation methods needed to generate agent locomotion. Then, it details how these parameters are used to compute trajectories for both the agent's COM and feet movements. After explaining the agent model construction, this chapter discusses the training methodology in depth. First, the Proximal Policy Optimization (PPO) algorithm and its integration with parameter sharing are presented, along with the rationale for its selection. This is followed

by a description of the procedurally generated training environment and maze algorithm, and then a detailed explanation of the observation space, action space, reward function, and hyperparameter settings used in this research.

5.2 Agent Physical Model and Constraints

This section begins with a description of the agent's body size and the colliders that constitute the model, followed by a biomechanical explanation of the constraints on speed, footfall amplitude, and foot orientation during walking. Finally, the agent's field of view, which is a straightforward system with multiple resolutions, is introduced.

5.2.1 Anthropometric Parameters and Body Structure

In this study, all agents share the same physical model size, which means that their height, weight, shoulder width, leg length and the structure of the field of view are the same. In future works, when diversity is introduced in further study, these elements may vary.

Since FootStepRL emphasizes the trajectories of agents' bodies and feet, the physical model of an agent is concisely composed of three main parts: body, left foot and right foot. As shown in the Figure 5.1, the body of an agent is a capsule with a collider equal in size to its appearance. The cuboid located on the upper side is only a decoration for indicating the orientation of the agent, so it does not have a collider, and its position is not crucial.

The centre of the capsule is treated as the agent's COM; therefore, the leg length is the distance from the centre of capsule to the ground. Since the centre of the capsule representing the torso is to be used to correspond to the COM, the distance from the highest point of this capsule to the ground does not equal the actual human height. The length of this capsule does not affect the training results because the agent's field of view does not start at the position of the "human eye".

According to the latest data available provided by NCD Risk Factor Collaboration (NCD-RisC)[7], in 2019, the average height of 19-year-old males in Canada was 178.7 cm, while Canadian females at the same age are 164.7 cm tall on average. So the agent height h is determined by the approximate mean value of the two, that is, 1.7 m; in the program, it corresponds to 1.7 units. According to the Body Mass

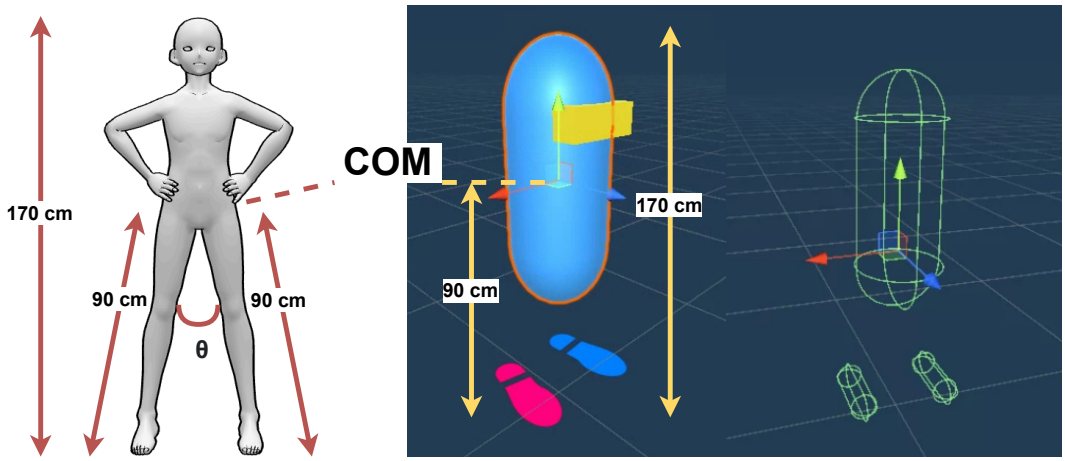


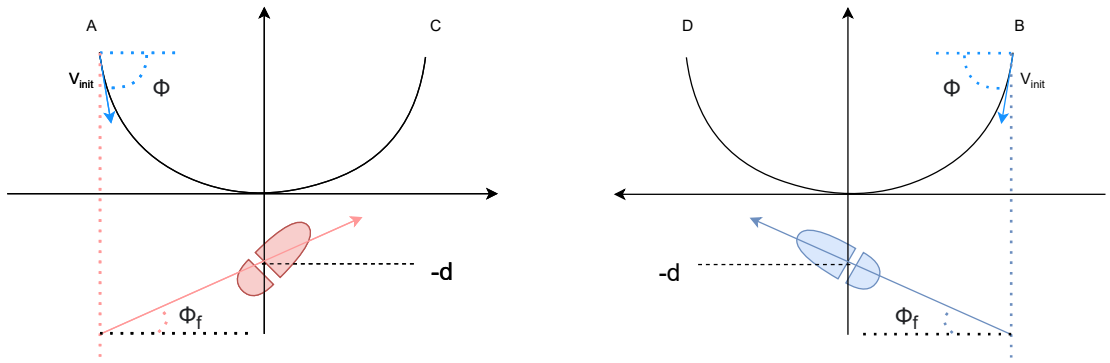
Figure 5.1: Human 3D model and agent representation, as well as the colliders of the human agent. The body consists of a capsule with a radius of 0.17 m. The centre point of the capsule represents the position of the COM, which is 0.9 m above the ground. The cube on the capsule body are used to cue the agent's orientation, and do not represent the actual eye positions. The feet of the agent are represented by two capsules. The rightmost figure displays the three capsules that form the agent's collision body.

Index (BMI) range ($18.5\,\mathrm{kg/m^2} - 24.9\,\mathrm{kg/m^2}$) for normal weight defined by World Health Organization[40], this research uses the average of the range to get a reasonable weight, which is approximately 62 kg.

In accordance with the general body proportions for males and females in relation to their height, calculated by Renato Contini[9], some body segment sizes can be derived. The proportion for each segment differs slightly between males and females, so this research takes their average. For the capsule relative to the human body, the cylinder radius corresponds to the shoulder radius, which is about $0.1 \times h = 0.17\,\mathrm{m}$. Similarly, distance from COM to the ground is equal to $0.53 \times h \approx 0.9\,\mathrm{m}$. As for the agent's feet, the capsules used to represent them have a cylinder radius of $0.02 \times h = 0.034\,\mathrm{m}$ long and the total length of a foot capsule (calculating by adding the cylinder height and twice the cylinder radius) is $0.15 \times h = 0.255\,\mathrm{m}$. When an agent is in a standing position, the distance from the centres of both capsules representing the feet to the body's mid-line equals the radius of the capsule representing the body.

In this research, agent movements are expressed by parabolic segments and corresponding foot placements. For example, a parabolic segment protruding to the right side of the body depicts the trajectory of the COM during the stance phase

of the right foot, and vice versa, so the terms from gait analysis, such as *stride* and *step*, cannot precisely describe these curves. Therefore, we define s as a step unit representing one parabolic curve, where s_L and s_R denote left and right step units, respectively, indicating which foot is in its stance phase.



(a) (b)

Figure 5.2: Left and right step in local space. (a) For s_R, the COM trajectory begins at point A and ends at C. (b) For s_L, the COM trajectory begins at point B and ends at D.

Figure 5.2 shows right and left steps in the local space. Regardless of the shape of the parabola, the related foot is always placed at $(0, -d)$, where d equals half of the shoulder radius, which in this research refers to the cylinder radius of the agent body. ϕ_f represents the foot orientation measured measured between the foot's midline and the x-axis.

5.2.2 Biomechanical Movement Constraints

The research focuses on agents walking at their free speeds. The study models agents based on typical adults, and to generalize across genders, the parameters such as weight, height and body size are determined by averaging data from human adults. The related biomechanical constraints are thus also affected by average values within genders.

Speed Range

The Principle of Least Effort (PLE)[15], which assumes that an organism would naturally choose to take actions that involve the least amount of energy expenditure,

serves as the basis for the walking speed limits in this study. In line with previous studies on human gait analysis[64], the average walking speed on a flat, obstacle-free terrain for an adult without mobility problems is established at 1.33 m/s. Considering the different situations in the real world that human crowds have to deal with when walking, and the dynamics of the corresponding MARL training environments, the agents are supposed to be capable of modifying their speeds according to their walking strategies. Therefore, the scalar value of the initial speed of each step, v_{init} in Figure 5.2 (a), is set in range $[0, 2 \times 1.33 = 2.66]$ m/s to accommodate situations in which agents might need to stop or accelerate. The benchmark of 1.33 m/s represents the horizontal speed on a macroscopic perspective, abstracting away the details of the rise, fall and tilt of the COM, which is similar to the motion of an inverted pendulum. On the other hand, this research approximates COM trajectory by projecting it onto the horizontal plane as segments of parabolas, simplifying the minor velocity fluctuations caused by the up-and-down undulations. Although the microscopic velocity of COM, i.e., the slope of the parabola, varies from moment to moment at each location, the macroscopic average speed is closely related to this established average, which explains why this research sets v_{init} in this way.

Angle Between Legs

When walking naturally, humans maintain a relatively stable step length that may decrease when slowing down or stopping. Therefore, agents in this study need to learn to keep their step lengths within an appropriate range. The length of each step in this study is defined as the span of its parabolic trajectory - specifically, the distance between the start and end points of the parabola segment. According to [64], the average step length l_s for young adults is 0.73 m. Hence, if a step length exceeds l_s, the agent receives a penalty.

Foot Orientation

The foot orientation angle ϕ_f is signed, considered positive when the foot rotates outward from the midline of the body, and negative when it rotates inward. Figure 5.2 shows examples of ϕ_f for a right step and a left step in the local space. This study assumes that during casual walking, the foot orientation aligns with the direction of body movement - that is, parallel to the x-axis of the local coordinate system.

5.2.3 Multi-Resolution Synthetic Vision System

The human eye recognizes objects in the environment by capturing light, while the agent in this study emits rays outward from an egocentric perspective to detect any obstacles. Let Ω represent the set of all such rays that constitute the field of view (FOV), consisting of rays in both transverse and longitudinal directions. Although the FOV simulates human vision, it is not strictly equivalent, so the rays do not emanate from a human-eye-like position.

The horizontal rays are supposed to sense distant, large items. For the purpose of this study, such entities include the bodies of other agents and walls that are taller than the agent's COM, ensuring that they are detectable even when rays are projected from the agent's COM. The set Ω_h consists of 18 rays, each 3 m long and evenly spaced, assembling into a sector shape that spans 160°. For each ray $r \in \Omega_h$, two values are stored: its direction vector $\mathbf{v}_r$ and a normalized distance to obstacle η_h:

$$\eta_h = \frac{d_o}{r_f} \tag{5.1}$$

where d_o denotes the distance from a ray's starting point (COM) to the obstacle, and r_f is the full length of the ray. If there is no obstacle detected, $\eta_h = 1.0$.

Accordingly, the task of the vertical rays is to discern obstacles closer to the agent, which in the context of this research are other agents' bodies and feet. In comparison to Ω_h, the set of vertical rays Ω_v produces more detailed results through denser arrangement. For each ray in Ω_h, there are four vertical rays aligned along its length. Given how Ω_h splits the sector-shaped field into 17 equal intervals, the distance between vertical rays along one horizontal ray is determined by dividing the sector radius by 17, ensuring a consistent and logical spacing. Furthermore, these rays originate from a position higher than the agents and shoot downwards, ensuring that they can detect other agents' bodies. Using the projection of an agent's COM onto the x-z plane as the origin of a local coordinate system, the local positions of the projections of each vertical ray onto the same plane are recorded. For each ray $r \in \Omega_v$, the normalized distance η_v is calculated:

$$\eta_v = \frac{d_o}{d_g} \tag{5.2}$$

where d_g denotes the length of the ray to the ground. Moreover, to simulate human vision, for rays aligned with the same horizontal ray, if a vertical ray closer to the

agent encounters an obstacle with a height of h_o, and a farther ray either detects no obstacle or catches one with a height less than h_o, the corresponding η_v for the farther ray is set to zero, reflecting the blind spot in human vision.

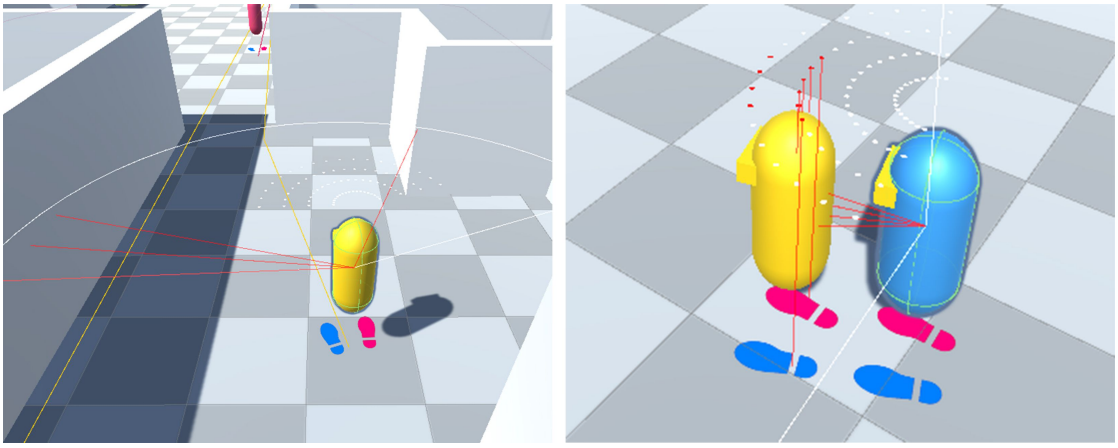


Figure 5.3: The field of view (FOV) extent is represented by a sector with a white border. Rays are shown in red when they detect an obstacle. The image on the left displays several horizontal rays detecting walls, while the image on the right shows vertical rays close to the agent detecting obstacles. When no obstacles are detected, only the starting point of the ray is displayed as a white dot. If a tall obstacle is detected, its height is evaluated to determine if it creates a blind spot by blocking the view of objects behind it. In such cases, the area behind the obstacle becomes a blind spot in the agent's field of view, indicated by a red dot.

The whole FOV is rendered in real time in the program. As shown in the Figure 5.3, the outline of the sector is drawn with white line, and the starting points of vertical rays are plotted in white. Rays that hit an obstacle are drawn in red, and for a vertical ray in blind spot, its starting point turns red.

5.3 Bipedal Motion Parameterization

Building on the 3D inverted pendulum model and the parabolic approximation of gait trajectories introduced in Chapter 4, this section presents the mathematical framework for generating agent steps. It introduces two key components: the step parameter set s that defines the agent's state during each step, and the motion specification μ that determines how the step is executed. The section then details the algorithmic process that computes these parameters.

5.3.1 Footstep Parameter Space

The trajectory of each step is defined by a set of parameters that describe the agent's state, motion characteristics, and coordinate transformations. Table 5.1 presents the complete parameter space.

Table 5.1: Footstep Parameter Space

Parameter	**Description**
State Parameters:	
$\mathbf{p}$	Position vector of COM at step end
$\mathbf{v}$	Velocity vector of COM at step end
$\sigma \in \{L, R\}$	Step side indicator (Left or Right)
$\mathbf{p}_f$	Position vector of stance foot
$\mathbf{d}_f$	Direction vector of stance foot
Motion Parameters:	
v_0	Initial speed of COM
ϕ	Angle between initial velocity and x-axis
t_{total}	Step duration
$\Delta\phi_f$	Change in foot orientation from previous step
Auxiliary Parameters:	
v_x^l	Initial x-velocity in local space
α	Parabolic trajectory coefficient
$\mathbf{T} = (x_T, y_T)$	Translation vector for local-world conversion
ϕ_w	World space orientation angle
t_{trans}	Orientation transition duration ($\frac{1}{5}t_{total}$)

Let s_t represent the parameter set for step t. The next step s_{t+1} is computed as a function of the current step s_t and motion parameters μ:

$$s_{t+1} = f(s_t, \mu) \tag{5.3}$$

For the initial step s_0, this study defines default parameters with the agent starting

from rest:

$$s_0 = \{\mathbf{p} = \mathbf{p}_0, \mathbf{v} = \mathbf{0}, \sigma = R, \mathbf{p}_f = \mathbf{p}_{f0}, \mathbf{d}_f = \mathbf{0}, \alpha = 0, \mathbf{T} = \mathbf{0}, \phi_w = 0, t_{trans} = 0\}$$

where $\mathbf{p}_0$ is the initial COM position and $\mathbf{p}_{f0}$ is the initial right foot position.

5.3.2 Asymmetric Step Parameterization

The computation process varies slightly between left and right steps. As shown in Figure 5.2, while the parabolic segments are equivalent, their parameterizations differ. Both share the same duration t_{total}, angle ϕ between initial velocity and x-axis, and initial speed magnitude v_0. In the local coordinate system, with the stance foot at $(0, -d)$, the trajectories are defined as follows: For a right step ($\sigma = R$) starting at point A and ending at C, the initial velocities are:

$$v_{A_x} = v_0 \cos\phi \tag{5.4}$$

$$v_{A_y} = -v_0 \sin\phi \tag{5.5}$$

For a left step ($\sigma = L$) starting at point B and ending at D, initial velocities are:

$$v_{B_x} = -v_0 \cos\phi \tag{5.6}$$

$$v_{B_y} = -v_0 \sin\phi \tag{5.7}$$

While both steps share the time interval $[0, t_{total}]$, their local time parameterization t_l differs on the step type. For actual time t_a, the local t_l for right steps is:

$$t_l = t_a - \frac{t_{total}}{2} \tag{5.8}$$

while the local t_l for left steps is:

$$t_l = \frac{t_{total}}{2} - t_a \tag{5.9}$$

5.3.3 Step Generation Implementation

Given the asymmetric parameterization described above, this section presents the algorithm for generating successive steps. While the process varies slightly between

left and right steps, the right step generation is described as a representative example. Algorithm 1 details how a right step s_{t+1} is computed from a left step s_t and motion parameters μ.

Algorithm 1 Compute Next Step Parameters

Require: Previous step velocity $\mathbf{v}_t$, initial speed v_0, orientation angle ϕ
Ensure: Next step parameters s_{t+1}
Calculate initial velocity in world space:
  $\mathbf{v}_{init} \leftarrow f(\mathbf{v}_t, v_0)$
Compute initial velocity in local space:
  $\mathbf{v}_l \leftarrow (v_0 \cos(\phi), -v_0 \sin(\phi))$
  Set $v_x^l \leftarrow v_{l_x}$
Calculate parabolic coefficient α using Equation 4.27
Compute initial COM position in local space:
  $\mathbf{p}_l \leftarrow f(\alpha)$
Determine world orientation:
  $\phi_w \leftarrow f(\mathbf{v}_l, \mathbf{v}_{init})$ ▷ v_{init_y} constant on horizontal plane
Compute translation vector $\mathbf{T}$:
  Using:
    • ϕ_w (new orientation)
    • $\mathbf{p}_l$ (initial COM position)
    • p_x, p_z components of $\mathbf{p}_{t+1}$
Calculate endpoint parameters in local space:
  $\mathbf{p}_l^{end} \leftarrow (-p_{l_x}, p_{l_y})$
  $\mathbf{v}_l^{end} \leftarrow (v_{l_x}, -v_{l_y})$
Transform $\mathbf{p}_l^{end}$ and $\mathbf{v}_l^{end}$ to world space
For right foot:
  Compute position $\mathbf{p}_f$
  Calculate direction $\mathbf{d}_f$ using previous orientation and $\Delta\phi_f$
Update step parameters:
  Set $\sigma \leftarrow R$
  Set $t_{trans} \leftarrow \frac{1}{5} t_{total}$
return s_{t+1}

5.4 Motion Synthesis System

This section examines how the position and orientation of an agent's COM and feet are updated during simulation, using a right step as an example. Consider consecutive steps s_t^L and s_{t+1}^R representing a left step followed by a right step, with motion parameters μ such that $s_{t+1}^R = f(s_t^L, \mu)$.

5.4.1 Center of Mass Trajectory Generation

The COM position $\mathbf{p}(s, t_l)$ in 3D world space is computed as a function of the current step parameters s and local time t_l. The local time is derived from actual time t_a and depends on the step side σ:

$$t_l = \begin{cases} t_a - \frac{1}{2}t_{total}, & \text{if } \sigma = R \\ \frac{1}{2}t_{total} - t_a, & \text{if } \sigma = L \end{cases} \tag{5.10}$$

The COM position in local coordinates is calculated as follows:

$$x_l = v_x^l \cdot t_l \tag{5.11}$$

$$y_l = \alpha \cdot t_l^2 \tag{5.12}$$

These local coordinates are then transformed to world space using:

$$\mathbf{p}_w = \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\phi_w) & 0 & \sin(\phi_w) & x_T \\ 0 & 1 & 0 & 0 \\ -\sin(\phi_w) & 0 & \cos(\phi_w) & y_T \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_l \\ y_w \\ y_l \\ 1 \end{pmatrix} \tag{5.13}$$

where y_w is kept constant to maintain horizontal COM movement. The world coordinates are thus:

$$x_w = x_l \cos(\phi_w) + y_l \sin(\phi_w) + x_T \tag{5.14}$$

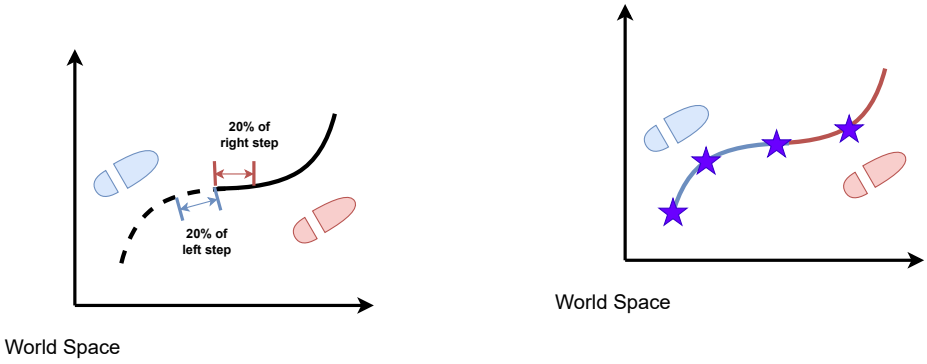
$$z_w = -x_l \sin(\phi_w) + y_l \cos(\phi_w) + y_T \tag{5.15}$$

5.4.2 COM Orientation Control

The agent's orientation changes gradually to align with its movement direction, defined by the vector from the parabola's start to end point. his reorientation occurs through linear interpolation, beginning when 20% of the step duration remains in step s_t^L (at t_{trans}^L) and completing at t_{trans}^R in step s_{t+1}^R, as shown in Figure 5.4a.

5.4.3 Foot Placement and Orientation Updates

To maintain computational efficiency while focusing on trajectories, foot translations are implemented using linear interpolation, omitting detailed leg animations. For



(a) (b)

Figure 5.4: Figure (a) shows that the COM begins a reorientation of facing with 20% of the total time remaining in the previous step s_t^L and completes the facing shift just after the next step s_{t+1}^R has travelled 20% of the total time. Figure (b) shows that the agent collects a set of observations at the beginning and the middle point of a left step, and the beginning and the middle point of the next right step, marked with stars.

example, the right foot maintains its position until 50% of the duration of step s_t^L has elapsed, then linearly interpolates to its new position $\mathbf{p}_f$ and orientation $\mathbf{d}_f$, as defined by s_{t+1}^R, by the end of step s_t^L.

5.5 MARL Training Implementation

5.5.1 Proximal Policy Optimization Algorithm Configuration

The choice of RL algorithm largely determines the accuracy, stability and efficiency of the training process. This research employs the Proximal Policy Optimization (PPO) algorithm [48], which is a model-free, on-policy algorithm that extends policy gradient methods. Standard policy gradient approaches often encounter frequent or excessively large updates, which results in unstable training. To address this issue, PPO introduces a clipped surrogate objective function:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\rho_t(\theta)\hat{A}_t, \text{clip}\left(\rho_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{5.16}$$

where $L^{\text{CLIP}}(\theta)$ refers to the clipped objective function corresponding to the policy parameter θ, $\hat{\mathbb{E}}_t[\dots]$ denotes the empirical expectation of a set of sampled data at

time t. $\hat{A}_t$ is an estimator of the advantage function at time t, indicating how much better or worse an action is compared to the average action in that state. We use estimators for $\hat{\mathbb{E}}$ and A since it would be computationally impossible and inefficient to calculate these values precisely across potentially infinite states and actions. The ratio $\rho_t(\theta)$ is defined as:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{5.17}$$

which equals the probability of taking action a_t in the state s_t under the new policy π_θ compared to the probability under the old policy $\pi_{\theta_{\text{old}}}$. ϵ is a hyperparameter that limits the policy update extent, and the function *clip()* bounds the value of $\rho_t(\theta)$ in the range $[1-\epsilon, 1+\epsilon]$. The policy gradient is the derivative of $L^{\text{CLIP}}(\theta)$ with respect to θ, and the policy π_θ is updated through stochastic gradient ascent (SGA) [5]. Specifically, data collected from the agent's interaction with the environment is strategically divided into n mini-batches. For each mini-batch, the PPO algorithm computes the clipped surrogate objective function and the corresponding policy gradient. Then, these gradients are aggregated—typically through averaging—to produce a single gradient that is used to update the policy parameters. This aggregated gradient reflects the overall best direction for adjusting the policy parameters, considering all available sampled data under the current policy. Using this mini-batch approach improves the stability of policy changes and makes computing enormous datasets efficient. It also guarantees diversity in the training process because each update is derived from several batches of data.

PPO limits the magnitude of each update to the policy with a simple clipping mechanism, which improves stability and lessens the possibility of abrupt changes. Moreover, the use of an advantage function in the objective function ensures that the update is related to the relatively optimal action under the current policy. Furthermore, the objective function's minimization aspect balances exploration with exploitation. According to the experiments in the paper [48], PPO is proven to perform better than other online policy gradient methods.

5.5.2 Procedurally Generated Training Environments

The study uses a maze algorithm with density and complexity parameters that can be adjusted to generate different types of environments in a way that affords domain randomization in the navigation problem. Henceforth, we shall designate this gener-

ated space as the training environment. It is essential to note that all objects in the training environment, such as the walls and the agents, are non-physical and only have collision detectors that are the same shape as they are. Because of this design decision, the agents are not able to physically adjust their routes upon collision, which might cause unintended behavioural changes during training and complicate the training process. Rather than being prohibited from passing through these objects, agents are rewarded negatively each time they collide with obstacles, which encourages them to develop avoidance strategies. Nevertheless, as there are no physical barriers to prevent the agents from wandering out of the training environment, this could lead to unpenalized exploration, confusing the training process. To address this issue, a thick, virtual barrier encircles the environment's edge. Even if this barrier is still passable and non-physical, it has a collision detector that can constantly penalize the agents for trying to cross it, thus gradually teaching the agents to stay inside the environment's bounds and reducing the risk of interference with the training process.

The algorithm is a variant of Prim's algorithm [43], which is a greedy algorithm for determining a minimum spanning tree for a weighted, directed graph. In this algorithm, the environment dimensions are modified to guarantee that the width and length of the environment have an odd number of cells each. Such an arrangement makes it easier to calculate and build the walls and paths inside the environment. The algorithm begins by converting user-defined density and complexity, which are floating-point numbers within a range of $[0, 1]$, into practical integer values, denoted as n and p. Here, n represents the number of distinct sections within the environment, which we also call islands. p corresponds to the number of wall sections within each island, with each section consisting of three connected cells.

The training environment is structured as a 2D array M, with each cell initially set to zero, indicating open paths. The exterior wall of the environment is represented by the perimeter cells of the array, which are set to one. To avoid confusion during training, additional thick walls are manually added adjacent to these borders in the training environment, ensuring that the agents will be penalized accordingly if they walk out of the environment.

A sequence of steps that define the islands and their enclosures comprise the interior structure of the environment. For each island, an initial cell with even coordinates is randomly selected as the starting point of a wall section. Then the algorithm locates potential "neighbouring cells", which are two steps away in each cardinal direction. One of these "neighbours" is randomly chosen, and it becomes the endpoint of the

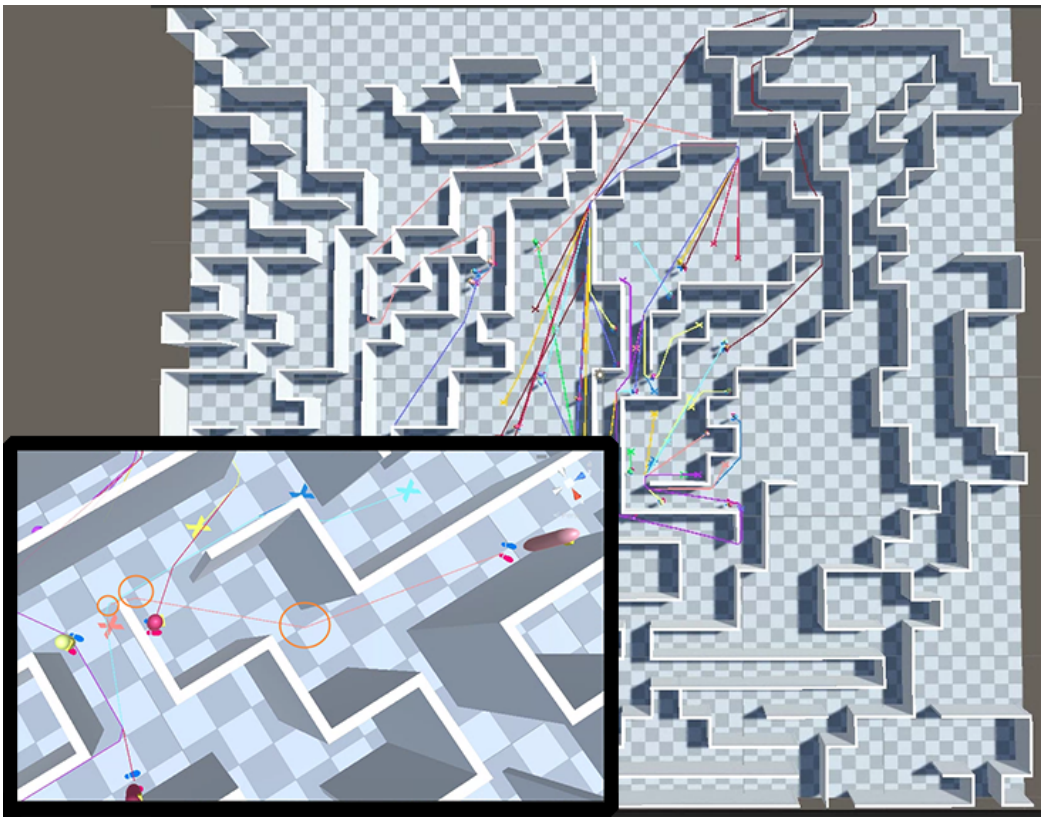


Figure 5.5: An overview of the training environment, which is a $101 \times 101\,\mathrm{m}^2$ with complexity of 0.1 and density of 0.03. The picture in the lower left corner shows the A* path and the waypoints of an agent.

wall section only if its value is zero, otherwise it may result in duplicate wall generation or the formation of an impassable closed space. Once a valid endpoint has been found, the values of this cell and the cell between it and the starting point become one, after which the process is repeated with this endpoint as the new starting point. This procedure is done p times for each island, and after completing this cycle n times, the environment will be filled with islands of a particular density and complexity. The pseudo-code for this maze-like training environment generation approach can be seen in Algorithm 2.

As shown in Figure 5.5, the training environment is constructed on a $101 \times 101\,\mathrm{m}^2$ flat plane, which is divided by 101 unit cells, each of which is of size $1 \times 1\,\mathrm{m}^2$. The walls in the training environment are 3 m high and 0.2 m thick. Each piece of the wall is built from the centre of a unit cell. According to the training environment generation algorithm, one wall segment occupies 3 unit cells, so the narrowest corridor is 1.8 m wide. At the beginning of a training episode, a training environment with user-defined

Algorithm 2 Generate Maze-like Training Environment

Require: width, height, complexity, density
Ensure: Maze grid M as a 2D array
Note: Borders are not initialized here as they are manually added to ensure thickness.

$shape \leftarrow ((height//2) \times 2 + 1, (width//2) \times 2 + 1)$
$p \leftarrow \text{int}(complexity \times (5 \times (shape[0] + shape[1])))$
$n \leftarrow \text{int}(density \times ((shape[0]//2) \times (shape[1]//2)))$
Initialize environment array M of dimensions $shape$ with zeros
Fill borders of M with 1
for $i \leftarrow 0$ to $n - 1$ **do**
    $x, y \leftarrow$ random even position within $shape$
    $M[y][x] \leftarrow 1$
    for $j \leftarrow 0$ to $p - 1$ **do**
        $neighbours \leftarrow$ empty list
        if $x > 1$ **then**
            $neighbours$.append$((y, x - 2))$
        end if
        if $x < shape[1] - 2$ **then**
            $neighbours$.append$((y, x + 2))$
        end if
        if $y > 1$ **then**
            $neighbours$.append$((y - 2, x))$
        end if
        if $y < shape[0] - 2$ **then**
            $neighbours$.append$((y + 2, x))$
        end if
        if $neighbours \neq$ empty **then**
            $y\_, x\_ \leftarrow$ random element from $neighbours$
            if $M[y\_][x\_] == 0$ **then**
                $M[y\_][x\_] \leftarrow 1$
                $M[(y\_ + (y - y\_)//2)][(x\_ + (x - x\_)//2)] \leftarrow 1$
                $x, y \leftarrow x\_, y\_$
            end if
        end if
    end for
end for

complexity and density is generated on the plane. After that, 60 agents are randomly placed in different locations in the environment, each assigned a distinct, randomly located endpoint, which is a circular area with a radius of 0.25 m. For convenience of visualization while debugging, these target areas are indicated by spheres coloured to match the associated agent. Since the study focuses on developing steering algorithms rather than high-level route planning approaches, the agents are only required to learn how to walk with two feet and perform actions like pauses, diversions and backtracking in response to obstacles. As a result, the A* path-finding method is used to map routes from each agent to their assigned endpoint region, and each corner of the path is marked as a temporary target area that agents need to pass through one after the other. This area is also a circular region with a 0.25 m radius. When an agent reaches the last target area along the A* computed path, a new random endpoint is selected for the agent to continue the training. This cycle repeats continuously until a training cycle is completed. To provide a dynamic training environment and avoid overfitting of the agents to a specific layout, each training episode begins with a randomization of the environment layout, agent starting positions and endpoints.

5.5.3 Agent Observation Space Design

Each agent's policy decision is based on observations collected during a consecutive pair of left and right steps (s_t^L and s_{t+1}^R). To replicate realistic human behaviour, agents operate with partial observability of their environment. The observation space includes both constant parameters and dynamic measurements. The constant parameters comprise six physical properties: agent mass (m), COM height (h_c), body radius (r_b), maximum stride length (l_{max}), and maximum initial velocities for left and right steps (v_{max}^L, v_{max}^R). While these parameters are uniform across all agents in this study, they can be varied in future work to investigate different walking policies by altering body size, stride length, and mobility.

In addition to this information, agents also collect data that humans may notice when they walk. Dynamic observations are collected at four points during the step pair: at the start and midpoint of both s_t^L and s_{t+1}^R, as shown in Figure 5.4(b). At each observation point, the agent's field of view measurements Ω are first collected, comprising 18 horizontal ray distances η_h^i, where $i = 1, \ldots, 18$ and 72 vertical ray distances η_v^j, where $j = 1, \ldots, 72$ (4 vertical rays per horizontal ray). As introduced in Section 5.2.3, since the FOV geometry relative to COM is constant across agents,

these measurements sufficiently capture environmental conditions without requiring additional ray position information. The agent then records a target vector $\mathbf{v}_t \in \mathbb{R}^2$ from COM to the current target location, representing relative position on the horizontal plane. Next, relative foot positions $\mathbf{p}_f^L, \mathbf{p}_f^R \in \mathbb{R}^3$ with respect to COM are measured. Finally, two step parameters are recorded: a step indicator $\sigma \in 0, 1$ (0 for left, 1 for right) and the elapsed time Δt since the last observation.

Each observation point thus yields 100 measurements: 90 for FOV $(18 + 72)$, 2 for target position, 6 for foot positions, and 2 for step parameters. The complete observation space consists of 406 dimensions: 6 physical constants and 4 sets of 100 measurements. The usage of relative positions ensures that this information is not affected by environmental changes. Furthermore, by making multiple observations during step pairs, it is possible to implicitly infer the relative velocity between the agent and an obstacle or target, thus allowing the agent to respond flexibly to dynamic environments.

5.5.4 Action Space Parameterization

The PPO algorithm produces a continuous action vector with components in $[0, 1]$, which we linearly transform to match our motion parameters. The action vector $\mathbf{a} \in \mathbb{R}^8$ contains motion parameters for both left and right steps, where $\mathbf{a}_{1:4}$ and $\mathbf{a}_{5:8}$ correspond to left and right motion parameters μ^L and μ^R, respectively. Based on the biomechanical constraints described in Section 5.2.2, each motion parameter set contains four values within specific ranges, as shown in Table 5.2.

Table 5.2: Motion Parameter Ranges

Parameter	**Range**
v_0	$[0, 2.66]$
ϕ	$\left[0, \frac{\pi}{2}\right]$
t_{total}	$[0.5, 1]$
$\Delta\phi_f$	$\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$

5.5.5 Multi-Objective Reward Function Design

There are fixed-value reward signals that are used to reward agents for passing through the target area or penalizing collisions with obstacles. However, these rewards are

too sparse, so in order to evaluate the walking process before the agent receives these reward signals, a multi-objective reward function is used to guide agents' behaviour:

$$R = R_g + R_d + R_e + R_p + R_s + P_t + P_c \tag{5.18}$$

where each term is explained in detail below.

The goal reaching reward R_g is a fixed-value reward applied every time the agent reaches the waypoint or the goal. To encourage the agent to approach the goal, a continuous reward related to the distance between the agent and the goal is applied at every timestep:

$$R_d = w_d \cdot e^{-\beta \cdot d_t} \tag{5.19}$$

in which d_t refers to the distance from the agent to its current target, either a waypoint or the final goal, and β is a scalar set to 0.2 that controls the decay rate of the exponential function. If the agent is close to the target, the proportion is close to zero and R_d becomes larger.

R_e promotes energy-efficient movements, which relate to the work of ground reaction force, as shown in Figure 5.6:

$$R_e = w_e \cdot e^{-\frac{m}{2}\left|(v_0)^2 - (v_{\text{prev}} cos(2\theta))^2\right|} \tag{5.20}$$

where v_{prev} denotes the velocity of the centre of mass at the end of the previous step, and v_0 represents the desired velocity at the beginning of the next step. In this footstep model, the loss of momentum at the beginning of a step leads to the result that the agent's speed reduces from v_{prev} to $v_{\text{prev}} cos\,(2\theta)$. Then, to choose an ideal speed for the next step, additional work is required and is computed as $\frac{m}{2}\left|(v_0)^2 - (v_{\text{prev}} cos\,(2\theta))^2\right|$. his value with a smaller magnitude indicates that the agent uses less effort to execute its next step, and the more natural its behaviour looks. It is worth noting that these analyses of energy are based on biomechanics and are estimated with simple trigonometry, so v_0 and v_{prev} in this case are not horizontally oriented, just as they are in reality when a human walks. However, based on the assumptions in Section 4.3.1, when these velocities are applied to compute the COM's movement, they still remain horizontal.

R_p discourages rapid changes of the momentum P over the length of the step:

$$R_p = w_p \cdot e^{-\frac{dP}{dt} \cdot L} = w_p \cdot e^{-ma \cdot l_{\text{arc}}} \tag{5.21}$$

where l_{arc} refers to the length of the curvature, so $ma \cdot l_{\text{arc}}$ grows when the agent is walking along the trajectory of high curvature.

R_s is used to limit the step length of the agent's walk:

$$R_s = \begin{cases} w_s \cdot (l_{\text{step}} - l_{\text{s}})^2 & \text{if } l_{\text{step}} > l_{\text{s}} \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.22}$$

where l_{step} is the current step length, and l_{s} denotes the average step length for young adults, set to $0.73, \text{m}$. This piecewise function evaluates to zero if the step length is within the allowed limit, but it becomes a penalty if the step length exceeds this limit. The excess length is calculated as $l_{\text{step}} - l_{\text{s}}$ and weighted by a negative weight w_s.

In addition to these rewards, penalties are implemented in this study to encourage faster target finding and collision avoidance. To incentivize quick goal finding we consistently penalize time with P_t:

$$P_t = w_t \tag{5.23}$$

where w_t is a negative float that represents the weight of this penalty. With P_t, the agent will be penalized less if it can find the quickest route to reach the goal. The agent receives this reward signal at every time step.

Above all, the agent is penalized with a collision penalty P_c, a fixed-value penalty applied whenever the agent collides with a wall or other agents.

5.5.6 Training Hyperparameter Configuration

Most of the hyperparameters used for training in this study are based on standard configurations that are commonly used for PPO.

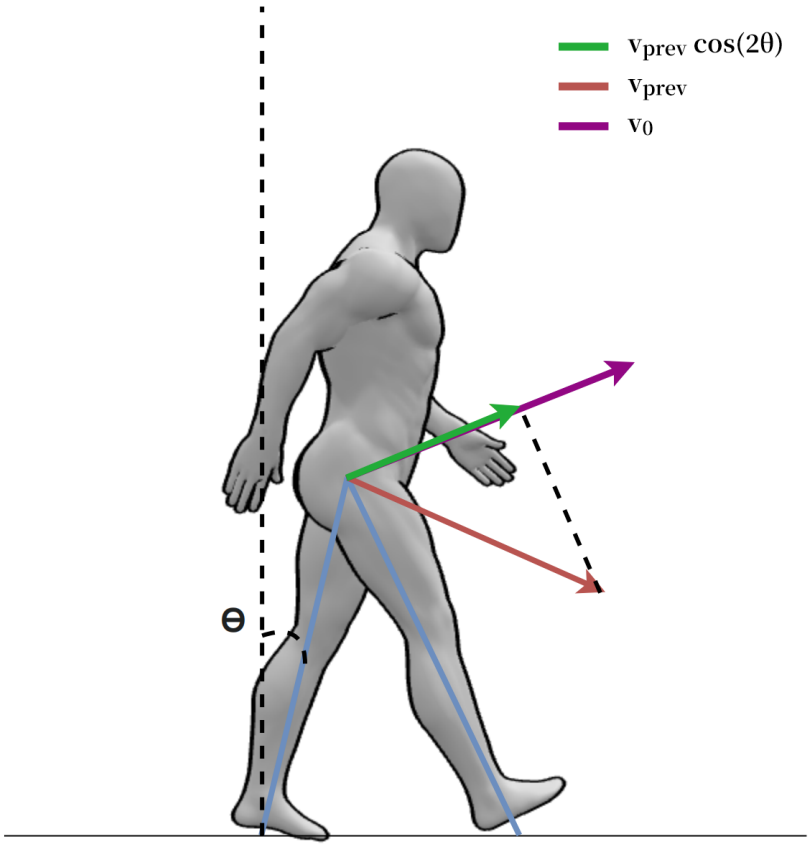


Figure 5.6: The pendulum model of a walking human, which is used to compute R_e, is described in [52].

Table 5.3: Hyperparameters for PPO Training Configuration

Parameter	**Description**
Trainer Type	Indicates algorithm that is being used; in this case, PPO.
Batch Size	The number of training cases considered in one iteration, set to 2048.
Buffer Size	The size of the buffer used to store training data, here 20480.
Learning Rate	The rate at which the algorithm updates its weights in the network, set at 0.0003.
Beta	A hyperparameter for the optimization algorithm, set at 0.01.
Epsilon	Used for the PPO clipping mechanism, set at 0.2.
Lambda	Factor for trade-off between bias and variance in the Generalized Advantage Estimation algorithm, set at 0.99.
Number of Epochs	The number of times the learning algorithm will work through the collected batch of data, here 3.
Learning Rate Schedule	Determines how the learning rate changes over training epochs, set to linear (the learning rate decreases linearly over the training).
Normalize	Whether to normalize input variables for the network, set to true.
Hidden Units	Number of units in each hidden layer of the neural network, here 512.
Number of Layers	The number of layers in the neural network, set to 3.
Gamma	The discount factor for calculating the future discounted reward, set at 0.995.
Strength	A parameter for adjusting the strength of the reward signal, set at 1.0.
Maximum Steps	The total number of training steps before ending the session, set to 3500000.

Continued on next page

Table 5.3 – continued from previous page

Parameter	Description
Time Horizon	The number of steps in the future to look while calculating returns, set to 1000.
Seed	The seed for random number generation to ensure the training can be reproduced, set at 20.

5.6 Training Results

After iterative training and tuning of the reward function, the resulting model from this study is presented below. Table 5.4 lists the weights of each continuous component in the reward function, along with the discrete reward values.

Table 5.4: Reward Function Settings

Symbol	Value
R_g	100
w_e	0.005
w_p	0.005
w_d	0.06
w_s	-0.005
w_t	-0.06
P_c	-3.25

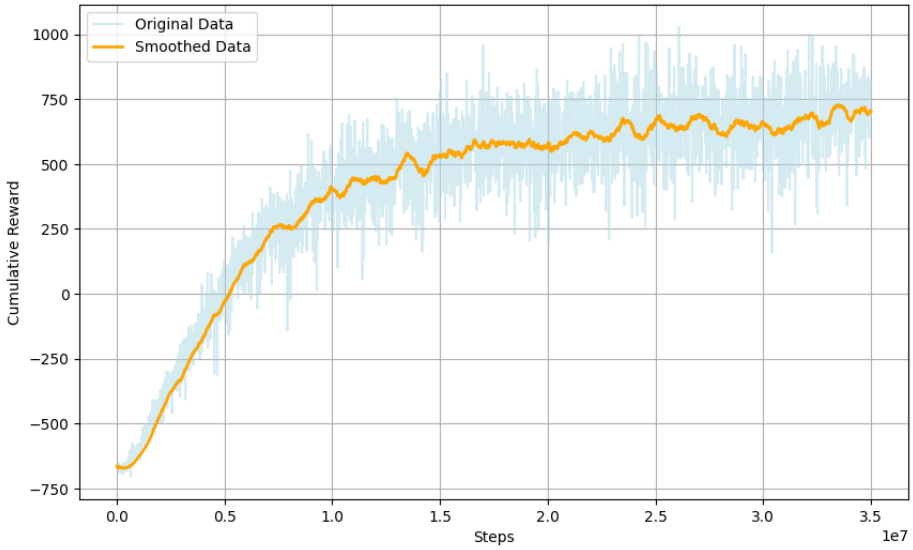
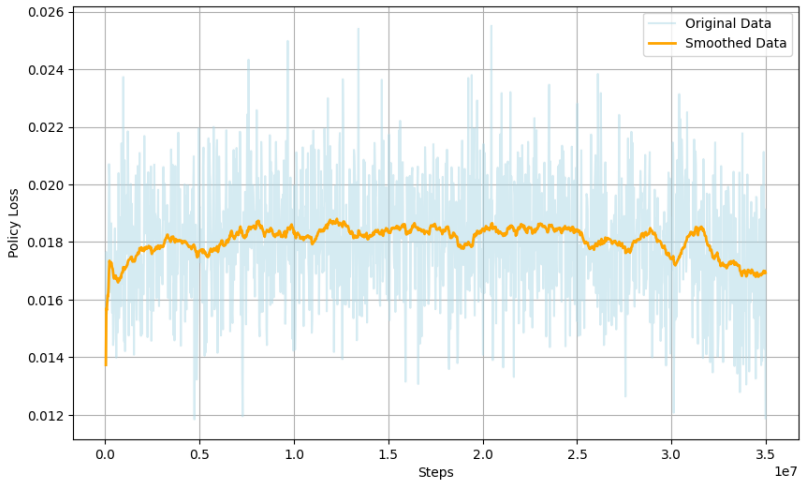


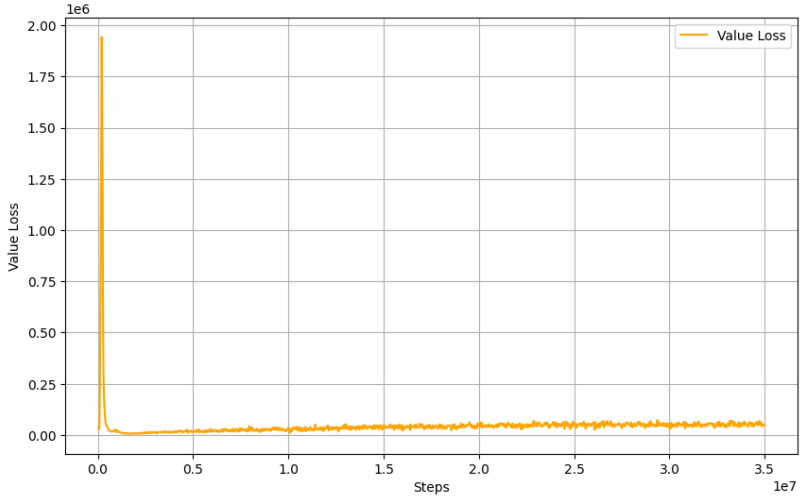
Figure 5.7: Training process for the agent (part 1). The plot of cumulative reward over training episodes shows how the agents' performance and learning progress have improved over time.

Figures 5.7 and 5.8 illustrate the training process. Figure 5.7 shows the cumulative reward during training, reflecting the agents' performance improvement over time. There is a period of rapid growth in the early stages, followed by a levelling off that remains upward-trending, suggesting a stabilization of policy updates. Figure 5.8a represents the policy loss during training. In PPO, policy loss measures the extent to which the agent's actions deviate from the previous policy, with clipping employed to prevent excessive updates, balancing exploration and stability. The policy loss graph fluctuates over time, showing a slight downward trend in the middle and late stages of training. This indicates that the agent is continually exploring and adapting during training. However, the relative stability suggests that there may be room to fine-tune hyperparameters or adjust the reward function structure to achieve smoother convergence. Figure 5.8b represents the value loss, which measures the error in estimating the value function based on the expected reward. The graph shows a rapid decrease in value loss early in training, followed by an increase and eventual stabilization. This reflects that the agent quickly approximates the reward in the early stages, then fine-tunes its performance for more complex situations in the later stages of training.

Figures 5.9 and 5.10 represent the average value of each reward or penalty received by an agent during a training episode. Figure 5.9a shows the change in rewards



(a) Policy loss



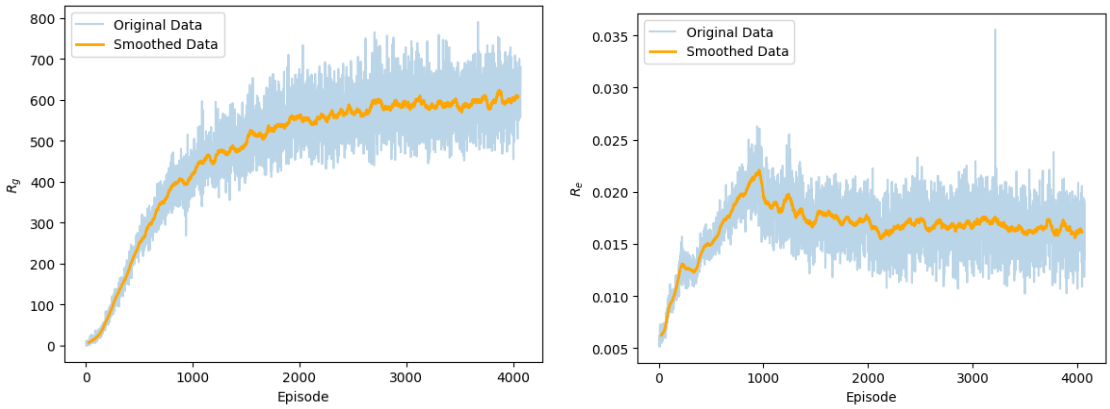
(b) Value loss

Figure 5.8: Training process for the agent (part 2). (a) Policy loss during training, which illustrates the variations that occur when policy updates are implemented to enhance the stability of decision-making. (b) Value loss during training, which reflects the adjustments in the value function's estimate of expected rewards. The graph indicates that value loss initially converges quickly and then tends to stabilize.

the agent receives for reaching the target, which rises rapidly in the early stages and then levels off, indicating that the agent's strategy for reaching the destination has stabilized. Figure 5.9b, Figure 5.9c, Figure 5.9d and Figure 5.10a represent the average values of R_e, R_p, R_d and R_s obtained by an agent during each training episode. They all show rapid growth at the beginning, followed by levelling off, indicating quick adaptation by the agent in the early stages of training, with continuous fine-tuning thereafter. Figure 5.10b shows the time penalty, which is applied at every time step except during program initialization or when the agent requests a decision from the neural network. This penalty reflects the time cost of regular actions, explaining why the average P_t value varies in each training cycle. Figures 5.10c and 5.10d depict the penalties received by the agent for colliding with walls and for colliding with other agents. These values fluctuate over time without a clear tendency to decrease, indicating that the reward function structure still has room for adjustment to better achieve the goal of collision avoidance.

5.7 Discussion

This chapter describes the integration of a biomechanical bipedal locomotion model with the MARL framework, which forms the core of the new approach to crowd simulation proposed in this study. By detailing the construction of the agent model, the mathematical framework for computing step trajectories, and the design of the training environment, observation space, action space, and reward function in MARL training using the PPO algorithm, this chapter demonstrates the model's consideration of biomechanical constraints and model flexibility and robustness. In the next chapter, a comprehensive evaluation of this model will be presented in conjunction with comparisons with traditional models.



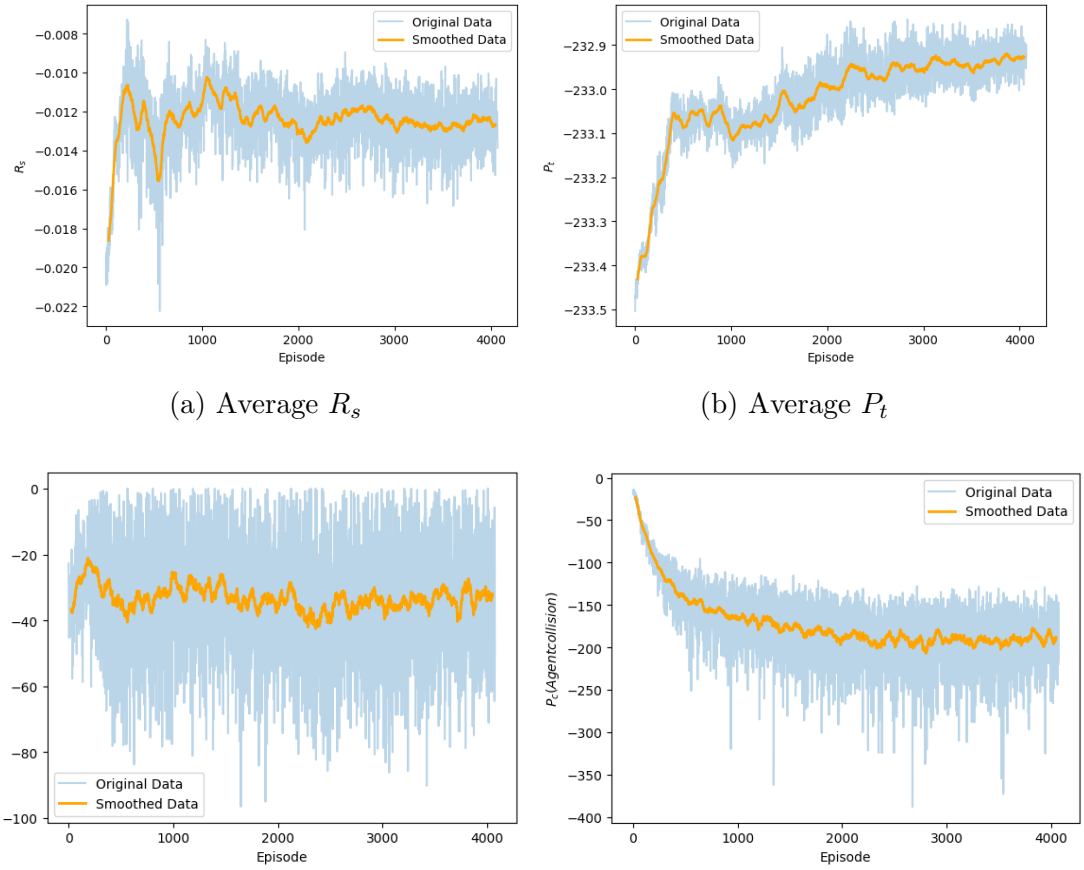
(a) Average goal reaching reward

(b) Average R_e

(c) Average R_p

(d) Average R_d

Figure 5.9: Average rewards and penalties for an agent across training episodes (part 1)



(a) Average R_s

(b) Average P_t

(c) Average wall collision penalty

(d) Average agent collision penalty

Figure 5.10: Average rewards and penalties for an agent across training episodes (Part 2)

Chapter 6

Experimental Validation and Comparative Analysis

This chapter aims to assess the walking efficiency, collision avoidance, and the ability to simulate behaviours specific to bipedal actions, such as side-stepping, during walking of bipedal agents trained in a simulated maze using the PPO algorithm, and to compare their performance with existing algorithms.

6.1 Introduction

Designing a fair and reasonable evaluation framework to analyze and compare the performance of steering algorithms is challenging. In general, steering algorithms can be evaluated using two main approaches: data-driven methods such as [23] and synthetic comparative methods like [51].

One approach is to use real-world data by analyzing the group behaviour of people and comparing it with the results generated by the algorithms. This method has several drawbacks. Real populations are difficult to predict and reproduce accurately. While studies indicate that human walking behaviour tends to follow certain patterns, such as the Principle of Least Effort (PLE)[15], it is also influenced by numerous internal and external factors, such as physical condition and environmental disturbances. As a result, reproducing group behaviour perfectly in reality is challenging, and discrepancies between simulated results and real-world data do not necessarily indicate poor algorithm performance. Additionally, real-world data may not encompass a sufficiently diverse range of scenarios to comprehensively evaluate

the algorithm.

On the other hand, the synthetic comparative method evaluates algorithms solely through simulated scenarios. While this approach avoids the drawbacks of the data-driven method, it remains challenging to design scenarios that are representative enough to produce convincing evaluation results, i.e., ensuring that the agent's behaviour in these simulations accurately reflects real-world situations.

In addition to selecting the evaluation method, tuning the parameters of each algorithm is crucial to ensuring fairness in the evaluation process. Different algorithms operate on distinct computational principles, resulting in varying model structures and parameter requirements. For example, the force-based Social Force model requires tuning the magnitude of repulsive forces exerted on the agent by different types of obstacles, such as other agents and static objects. Similarly, the velocity-based ORCA algorithm requires tuning parameters such as the minimum reaction time for the agent to respond to an obstacle. Previous studies have explored this issue, and SteerFit [3] has proposed a set of parameter optimization methods that ensure fair model comparisons by aligning parameters with real-world benchmarks.

The evaluation framework proposed in this study aims to provide a fair performance assessment using commonly employed conventional models and typical synthetic scenarios. The first baseline model selected for this study is the modified Social Force model[37], a classic force-based steering algorithm widely used in urban design, emergency evacuation planning, and the entertainment industry. The second baseline model is the velocity-based ORCA algorithm[57], which is widely applied not only in the aforementioned fields but also in robotics and autonomous driving. To select test scenarios, this study employs five typical scenarios from SteerBench[51], a comprehensive evaluation framework, to assess the model's performance ranging from simple one-to-one situations to complex many-to-many interactions. To ensure parameter fairness, the evaluation sessions uniformly tune the internal parameters of each model to optimize agent effort. Finally, this study will analyze and evaluate the test results from both quantitative and qualitative perspectives, focusing not only on local navigation performance—such as obstacle avoidance and walking efficiency—but also on whether the bipedal locomotion exhibited by this model offers richer human-like walking behaviours compared to particle-based agents and algorithms.

This chapter outlines the evaluation process and presents the analyzed results in the following order. First, the baseline models and the rationale for their selection in this study are discussed, along with the internal parameter values they use. Next, the

selection of experimental scenarios for the assessment is explained. Following that, the quantitative metrics selected for the assessment are introduced. The chapter then presents the statistics and analysis of the test results for each model in each test scenario from a quantitative perspective, followed by a qualitative performance analysis of the proposed model. Additionally, the computational efficiency of the model is evaluated. Finally, based on the evaluation results, the chapter discusses the experimental failures of the model, the potential reasons for these failures, the model's shortcomings, and possible directions for future work.

6.2 Baseline Models and Metrics

This section will first introduce three baseline models, then the five test scenarios, and finally the quantitative metrics for each scenario.

6.2.1 Baseline Models

This research uses two representative steering algorithms as the baseline models. The first one is an improved version of the Social Force model[37], a traditional force-based approach that steers agents to avoid collisions by applying social attraction and repulsion to agents. The second algorithm is ORCA[57], which according to Section 2.4 is a velocity-based algorithm that computes new velocity for each agent by solving optimization problems and assuming that all the other agents will do the same thing. In the following sections, I will use *SF2* and *ORCA* to represent the first two models.

Although all of these algorithms are steering algorithms, their performance depends on the tuning of their internal parameters. To standardize the analysis criteria, this study assumes that all three algorithms, including the FootStepRL model, aim to optimize walking energy consumption. For SF, since the original paper [37] optimized the SF algorithm using walking data collected from real-life volunteers, following the Principle of Least Effort (PLE) [15]—which suggests that people tend to walk with minimal effort—this study adopts the ideal parameters derived from the original paper, as shown in Table 6.1. For ORCA, this study uses the parameter settings derived from SteerFit [3], which represent the optimal parameters for optimizing PLE Quality, as shown in Table 6.2. For the FootStepRL model, as discussed in Section 5.5.5, energy consumption has been considered in the design of the reward functions.

Table 6.1: Parameters Settings of SF2

Parameter Name	**Value**
$DesiredSpeed$	1.29
$RelaxationTime$	0.54
$RelativeMotionDirectionWeight$	2
A	4.5
γ	0.35
n	2.0
$n\prime$	3.0

Table 6.2: Parameters Settings of ORCA

Parameter Name	**Value**
$MaxSpeed$	1.52
$NeighborDistance$	12.08
$TimeHorizon$	2.72
$TimeHorizonObstacles$	11.81
$MaxNeighbors$	15.03

6.2.2 Experimental Setup

In this experiment, five typical scenarios from SteerBench [51] are selected to verify whether the training model can flexibly cope with occasions that may occur in reality. In addition, we use other algorithms in the same scenarios for comparison. As is shown in Figure 6.1, circles represent agents, each assigned a final goal location marked by a cross. In scenarios with only two agents, the agent and its target location are represented by the same colour. In scenarios with multiple agents, colours are not carefully distinguished to avoid confusion caused by excessive colours.

One on One: two face-to-face agents are placed in one closed environment without obstacles. Each agent's target is set at the initial location of the other agent.

One on One Obstacle: based on the *One on One* scene, there is a wall obstacle between the agents.

Diametric Goals: 8 agents are placed in a closed environment with no other obstacles. They are divided into two people a group, and each agent in one group is assigned a target at the initial location of the other agent. The path they are supposed to walk through is in the shape of a cross.

Two Way Crowd: based on the *One on One* scene, there are 10 agents on both sides of the environment.

One Way Bottleneck Egress: a hallway-like environment, where 10 agents are initialized on the left side and their targets are on the right. There is a bottleneck egress between the agents and the goals.

In the test scenarios, all objects have no physical entity and can be passed through. Each agent's A* route to the assigned final target area is marked with a straight line of the same colour as the agent, and when the agent passes through its current target area, the line up to that relay point is no longer displayed. Furthermore, the path of the agent COM will be drawn as the agent moves, to study the actual trajectory of the agent. The test program will record the position of each agent's feet and the point in time when the position was changed, the number of times the agent was penalized by collisions, the actual distance travelled, and the time elapsed.

6.2.3 Quantitative Metrics

This section will explain the definition of each quantitative metric. Since each test scenario has a different focus, the metrics used in each scenario are not the same.

Flow Rate: How quickly agents accomplish their final goal during the simulation. This metric enables us to compare the time taken to reach the goal and the completion rates across different models. In this study, the flow rate is defined as:

$$F(A) = \frac{|A|}{T_f} \tag{6.1}$$

where F refers to flow rate, A represents the set of agents, and T_f is the total time of the simulation, from the beginning of the simulation to the time when the last agent reaches its assigned target area.

Path Efficiency: Actual length of agent's trajectory divided by the desired trajectory length when collision avoidance is not required. For agents with FootStepRL model, the actual length of trajectory is the summation of each step length (i.e. for each parabolic segment, the distance between the starting point and the ending point of the parabola). This criterion enables the study of the extent to which the agent deviates from the desired route, particularly when it encounters unexpected situations, such as obstacle avoidance.

Effort: Metabolic expenditure of the agents during walking. This computation reflects the extent to which the agent attempts to optimize its effort consumption during the walk, similar to how humans do. This metric is calculated by the sum of all agents' effort over the integral of their path:

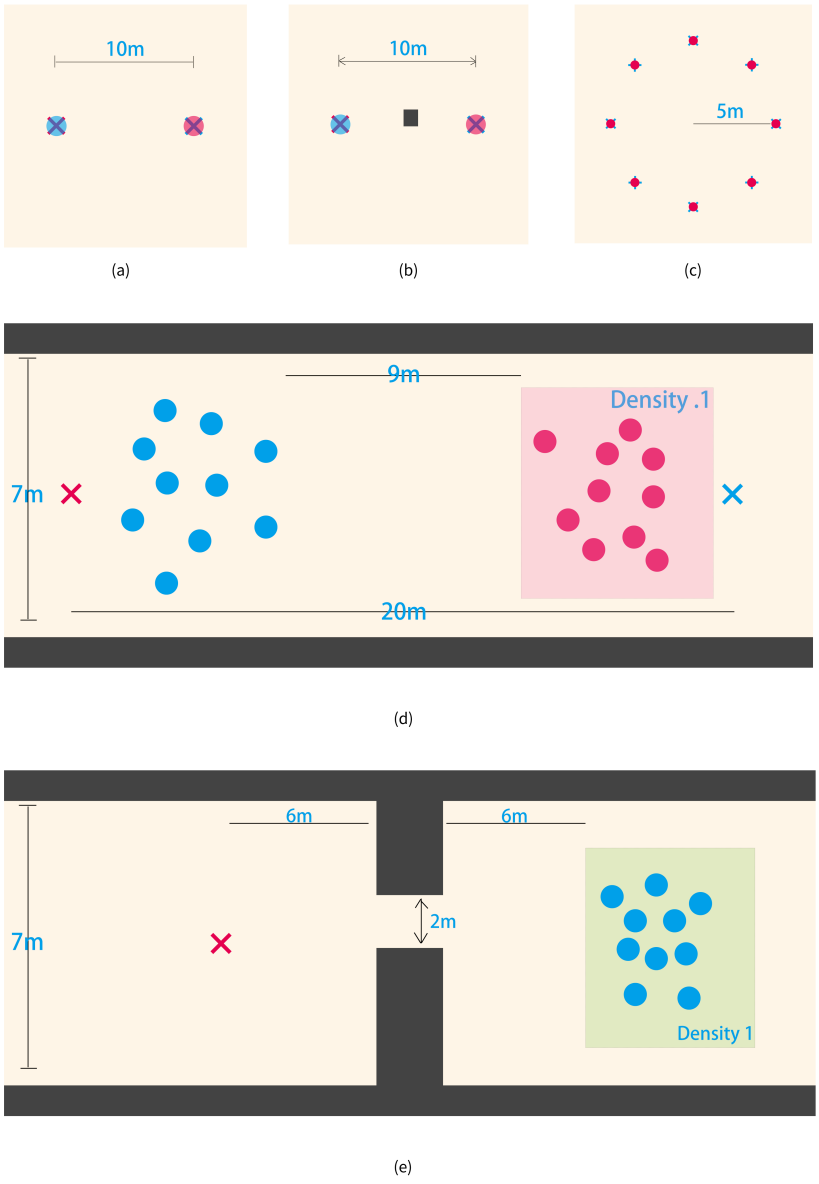


Figure 6.1: Five scenarios for evaluation. From (a) to (e): One on One; One on One Obstacle; Diametric Goals; Two Way Crowd; One Way Bottleneck Egress.

$$E(A) = \sum_{a \in A} m_a \int \left(e_s + e_w |v_a|^2\right)\, dt \tag{6.2}$$

where m_a is the mass of each agent, which is standardized to $62Kg$ in this study. The biological constants e_s and e_w are empirically averaged to be $2.23J/Kgs$ and $1.26Js/Kgm^2$ for human walking. v_a is the velocity of the agent at time t.

Number of Collisions: Total number of collisions. This metric describes the agent's ability to avoid obstacles.

6.3 Results

This section analyzes the performance of the baseline and FSRL models using both quantitative and qualitative metrics.

6.3.1 Quantitative Results

In the One on One scenario and Diametric Goals scenario, a significant observation surfaced. ORCA demonstrates deadlock behaviour when the initial placements of the agents are completely symmetrical, spinning about a vortex for long periods of time before discovering feasible dispersion paths, and slowing to a near-stop around the centre of the circle. To get useful comparison measurements, this requires adding a tiny amount of position noise to the experimental setup.

Table 6.3 records the flow rate for the three models in each scenario, along with the average effort per agent, average path efficiency, and total number of collisions for all agents. Figures 6.2 through 6.6 display box plots for the time to goal, effort, path efficiency, and number of collisions for the ORCA, SF2, and FootStepRL models across the five test scenarios. Because flow rate is an aggregate metric that cannot be represented by a box plot, it is depicted using the time to goal for each agent. The flow rate for each steering model in each scenario is displayed using a bar chart, as shown in Figure 6.7.

To evaluate the differences between models, I conduct a statistical analysis using the Kruskal-Wallis test [28] and Conover's post hoc test [8] with Holm's correction [20]. The significance level for the p-value is set to 0.05. If the Kruskal-Wallis test indicates significance ($p < 0.05$), the post hoc test is performed to identify specific pairs of models with significant differences. In the figures, the level of statistical

Table 6.3: Quantitative Metrics for Different Models Across Test Scenes

Metric	ORCA	SF2	FootStepRL
One on One			
Flow Rate	0.151704	0.13409	0.108712
Effort	4162.4515	3913.091	5021.985
Path Efficiency	0.9976333	0.99820715	0.85235697
Number of Collisions	0	0	0
One on One Obstacle			
Flow Rate	0.153135	0.13441	0.0728
Effort	4144.381	3771.114	6233.833
Path Efficiency	0.9967443	0.998041	0.717511617
Number of Collisions	2	1	2
Diametric Goals			
Flow Rate	1.15942	1.06667	0.046118
Effort	2015.0775	1862.3595	3044.125875
Path Efficiency	0.99310795	0.982755113	0.726413618
Number of Collisions	17	0	43
Two Way Crowd			
Flow Rate	0.095433	0.09069	0.029118
Effort	4830.7579	4531.74805	6696.9768
Path Efficiency	0.996117005	0.977486435	0.789009348
Number of Collisions	0	0	229
One Way Bottleneck Egress			
Flow Rate	0.09434	0.0795	0.027676
Effort	3100.2912	2793.6844	5585.0599
Path Efficiency	1.0387385	1.01495895	0.672187661
Number of Collisions	0	4	162

significance between models is marked by asterisks: $*$ indicates $p < 0.05$, $**$ indicates $p < 0.01$, and $***$ indicates $p < 0.001$.

Results are follows:

One on One: The Kruskal-Wallis test reveals no significant difference in **Time to Goal** among the models ($H = 4.706$, $p = 9.509 \times 10^{-2}$), as $p \geq 0.05$. As a result, no post hoc test is conducted for this metric.

Similarly, for **Effort**, the Kruskal-Wallis test indicates no significant difference ($H = 4.571$, $p = 1.017 \times 10^{-1}$).

For **Path Efficiency**, the Kruskal-Wallis test also shows no significant difference ($H = 4.571$, $p = 1.017 \times 10^{-1}$).

For **Collisions**, all values are identical across models, making it unsuitable for

the Kruskal-Wallis test.

One on One Obstacle: The Kruskal-Wallis test reveals no significant difference in **Time to Goal** among the models ($H = 4.571$, $p = 1.017 \times 10^{-1}$).

For **Effort**, the Kruskal-Wallis test also indicates no significant difference ($H = 4.571$, $p = 1.017 \times 10^{-1}$).

Similarly, the Kruskal-Wallis test for **Path Efficiency** shows no significant difference among the models ($H = 3.714$, $p = 1.561 \times 10^{-1}$).

Lastly, for **Collisions**, no significant difference is found ($H = 2.000$, $p = 3.679 \times 10^{-1}$).

Diametric Goals: The Kruskal-Wallis test reveals a significant difference in **Time to Goal** among at least one of the models ($H = 21.256$, $p = 2.422 \times 10^{-5}$). Using Conover's multiple comparisons post hoc test, it is identified that the model FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ significance level. Specifically, significant differences are observed between the following pairs: ORCA and FootStepRL ($p = 9.24 \times 10^{-13}$), ORCA and SF2 ($p = 1.65 \times 10^{-7}$), and SF2 and FootStepRL ($p = 1.65 \times 10^{-7}$).

Effort: The Kruskal-Wallis test also reveals a significant difference among at least one of the models ($H = 20.480$, $p = 3.571 \times 10^{-5}$). Conover's post hoc test identifies that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ level. Significant differences are found between the following pairs: ORCA and FootStepRL ($p = 3.61 \times 10^{-6}$), ORCA and SF2 ($p = 3.61 \times 10^{-6}$), and SF2 and FootStepRL ($p = 4.49 \times 10^{-11}$).

Path Efficiency: The Kruskal-Wallis test indicates a significant difference among the models ($H = 20.525$, $p = 3.492 \times 10^{-5}$). Conover's test shows that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ level. The following pairs show significant differences: ORCA and FootStepRL ($p = 3.72 \times 10^{-11}$), ORCA and SF2 ($p = 3.12 \times 10^{-6}$), and SF2 and FootStepRL ($p = 3.12 \times 10^{-6}$).

Collisions: the Kruskal-Wallis test reveals a significant difference among models ($H = 15.742$, $p = 3.817 \times 10^{-4}$). Conover's post hoc test shows that FootStepRL differs significantly from SF2 at the $p < 0.001$ level. The significant pairwise differences are as follows: ORCA and SF2 ($p = 2.84 \times 10^{-4}$), and SF2 and FootStepRL ($p = 5 \times 10^{-6}$).

Two Way Crowds: The Kruskal-Wallis test reveals a significant difference in

Time to Goal among at least one of the models ($H = 27.388$, $p = 1.129 \times 10^{-6}$). Using Conover's post hoc test, it is identified that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ significance level. Specifically, significant differences are observed between the following pairs: ORCA and FootStepRL ($p = 9.45 \times 10^{-9}$), ORCA and SF2 ($p = 4.1 \times 10^{-3}$), and SF2 and FootStepRL ($p = 3.55 \times 10^{-4}$).

Effort: The Kruskal-Wallis test reveals a significant difference among the models ($H = 31.067$, $p = 1.795 \times 10^{-7}$). Conover's post hoc test indicates that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ level. Significant pairwise differences are found between the following pairs: ORCA and FootStepRL ($p = 1.45 \times 10^{-7}$), and SF2 and FootStepRL ($p = 1.79 \times 10^{-9}$).

Path Efficiency: The Kruskal-Wallis test indicates a significant difference among the models ($H = 49.616$, $p = 1.683 \times 10^{-11}$). Conover's test shows that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ level, with significant differences observed between ORCA and FootStepRL ($p = 6.41 \times 10^{-24}$), ORCA and SF2 ($p = 1.01 \times 10^{-10}$), and SF2 and FootStepRL ($p = 5.98 \times 10^{-13}$).

Collisions: the Kruskal-Wallis test reveals a significant difference among the models ($H = 52.152$, $p = 4.734 \times 10^{-12}$). Conover's post hoc test shows significant differences between ORCA and FootStepRL ($p = 9.28 \times 10^{-25}$), and SF2 and FootStepRL ($p = 9.28 \times 10^{-25}$), both at the $p < 0.001$ level.

One Way Bottleneck Egress: The Kruskal-Wallis test reveals a significant difference in **Time to Goal** among at least one of the models ($H = 25.568$, $p = 2.805 \times 10^{-6}$). It is identified with Conover's post hoc test that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ significance level. Specifically, significant differences are observed between the following pairs: ORCA and FootStepRL ($p = 1.49 \times 10^{-13}$), ORCA and SF2 ($p = 1.93 \times 10^{-7}$), and SF2 and FootStepRL ($p = 1.93 \times 10^{-7}$).

Effort: The Kruskal-Wallis test reveals a significant difference among the models ($H = 22.888$, $p = 1.071 \times 10^{-5}$). Conover's post hoc test indicates that FootStepRL differs significantly from both ORCA and SF2 at the $p < 0.001$ level, with significant pairwise differences found between the following pairs: ORCA and FootStepRL ($p = 3.88 \times 10^{-6}$), ORCA and SF2 ($p = 5.04 \times 10^{-4}$), and SF2 and FootStepRL ($p = 4.42 \times 10^{-10}$).

Path Efficiency: The Kruskal-Wallis test indicates a significant difference among

the models ($H = 20.493$, $p = 3.548 \times 10^{-5}$). Conover's post hoc test identifies that FootStepRL differs significantly from ORCA and SF2 at the $p < 0.001$ level, with significant differences between the following pairs: ORCA and FootStepRL ($p = 7.62 \times 10^{-8}$) and SF2 and FootStepRL ($p = 6.50 \times 10^{-6}$).

Collisions: the Kruskal-Wallis test reveals a significant difference among the models ($H = 23.371$, $p = 8.414 \times 10^{-6}$). Conover's post hoc test indicates significant differences between the following pairs: ORCA and FootStepRL ($p = 2.95 \times 10^{-10}$), ORCA and SF2 ($p = 0.017$) at the $p < 0.05$ level, and SF2 and FootStepRL ($p = 6.62 \times 10^{-8}$) at the $p < 0.001$ level.

According to the Kruskal-Wallis test and Conover's post hoc test, the test results are not significantly different in the test scenario One on One and One on One Obstacle, while in the remaining three scenarios (Diametric Goals, Two Way Crowds, and One Way Bottleneck Egress) which are more crowded and complicated, the performance of FootStepRL is significantly different from the other two baseline models in terms of flow rates, effort, path efficiency and number of collisions. It can also be observed in Figure 6.7 that the FootStepRL model has a lower flow rate than the other two models in every scenario, and is significantly lower than them in scene Diametric Goals.

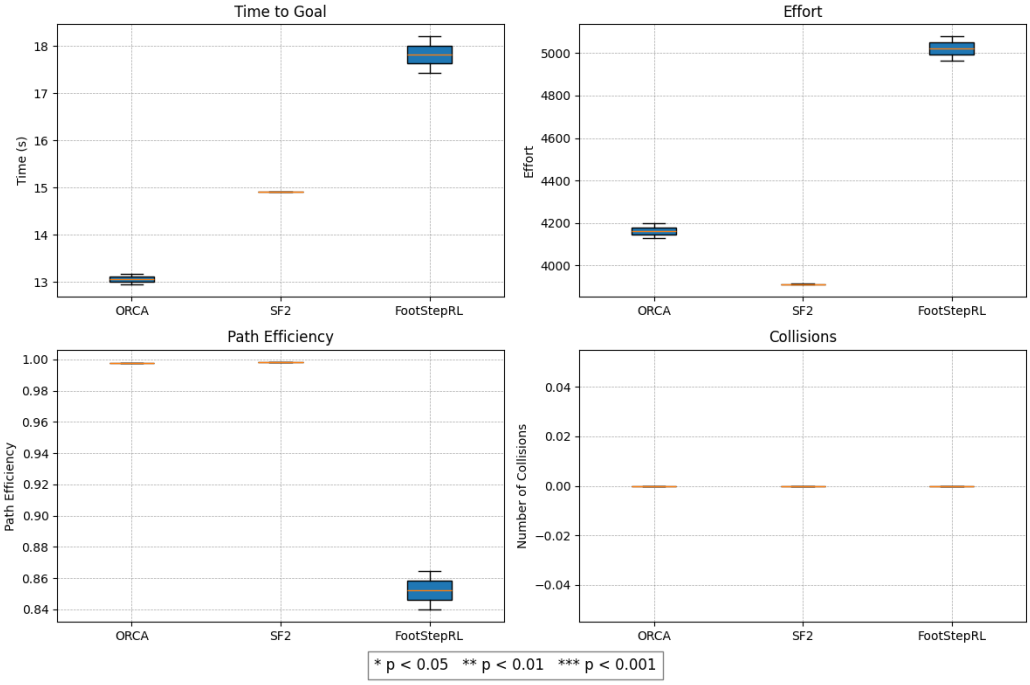


Figure 6.2: Boxplots of time to goal, effort, path efficiency, number of collisions in One on One scenario

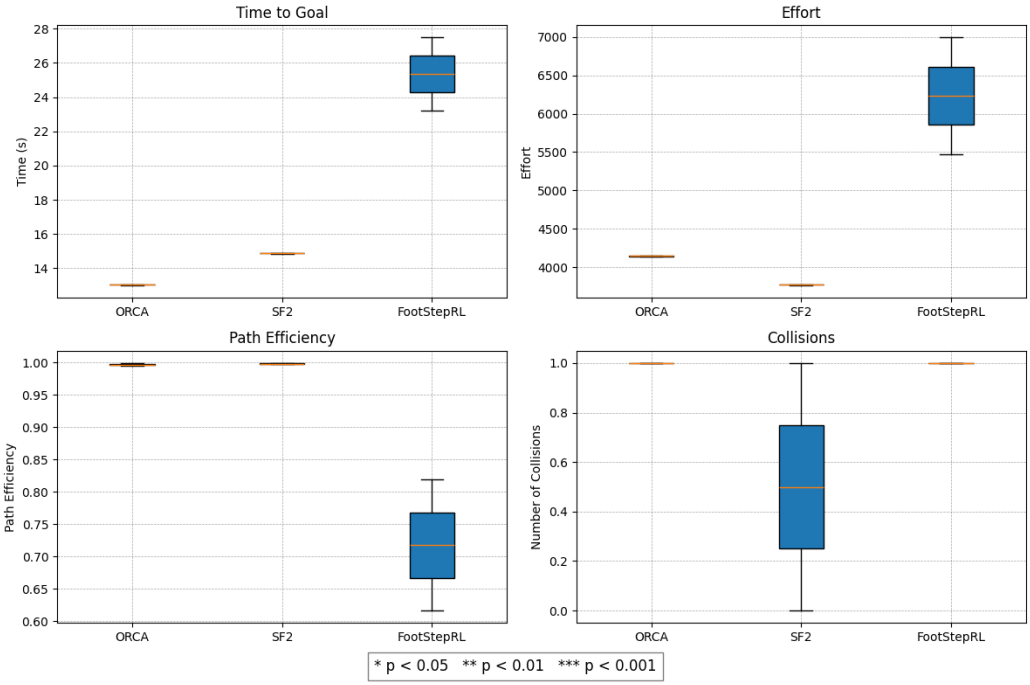


Figure 6.3: Boxplots of time to goal, effort, path efficiency, number of collisions in One on One Obstacle scenario

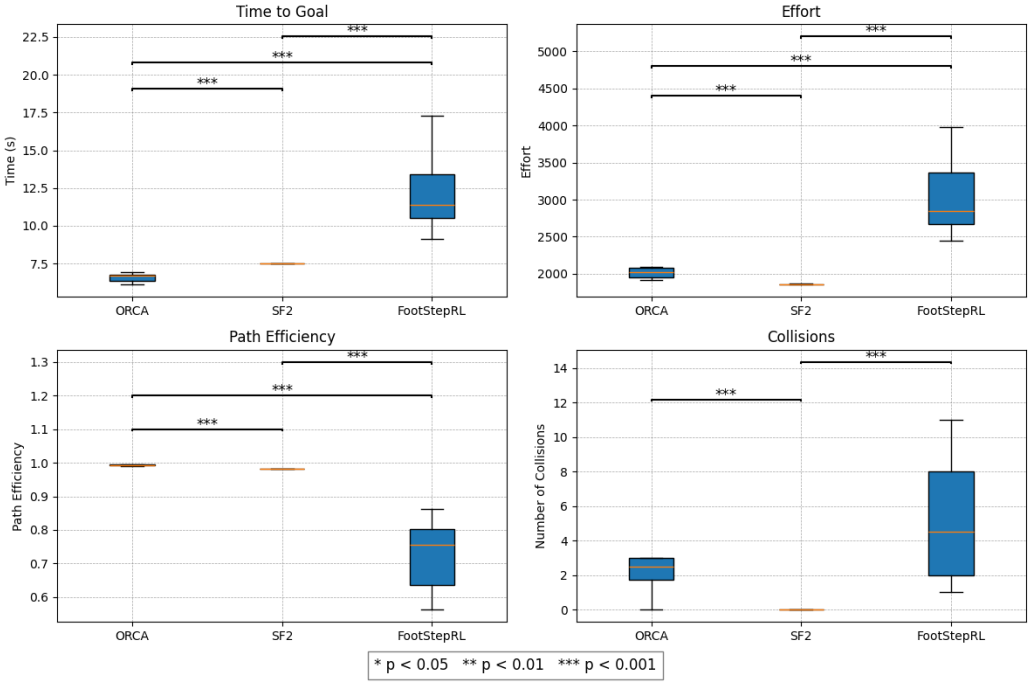


Figure 6.4: Boxplots of time to goal, effort, path efficiency, number of collisions in Diametric Goals scenario

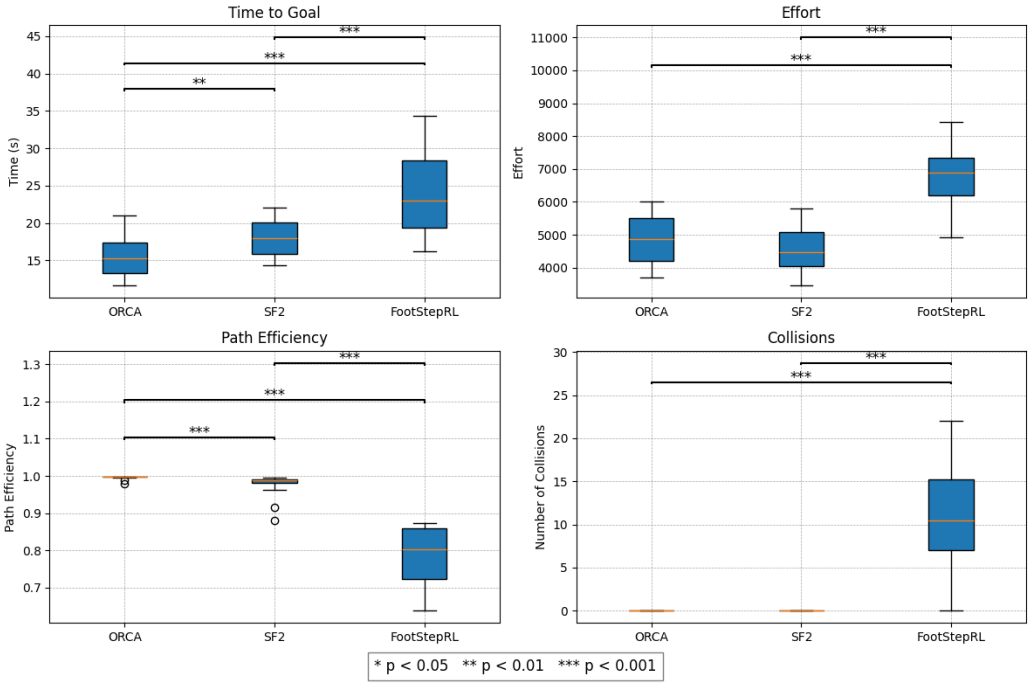


Figure 6.5: Boxplots of time to goal, effort, path efficiency, number of collisions in Two Way Crowds scenario

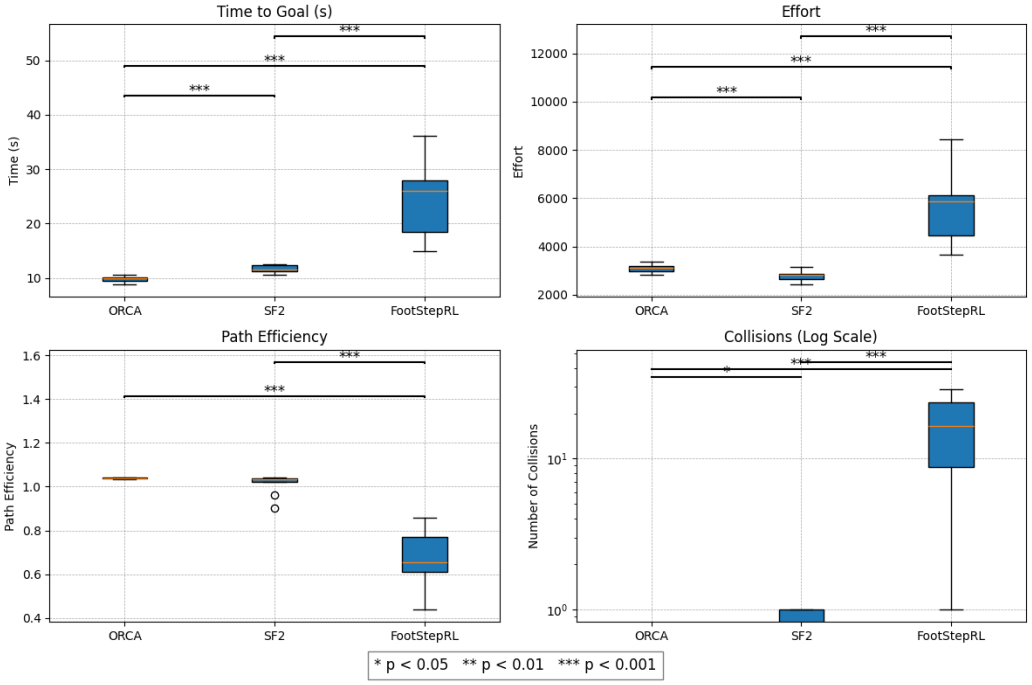


Figure 6.6: Boxplots of time to goal, effort, path efficiency, number of collisions in One Way Bottleneck Egress scenario

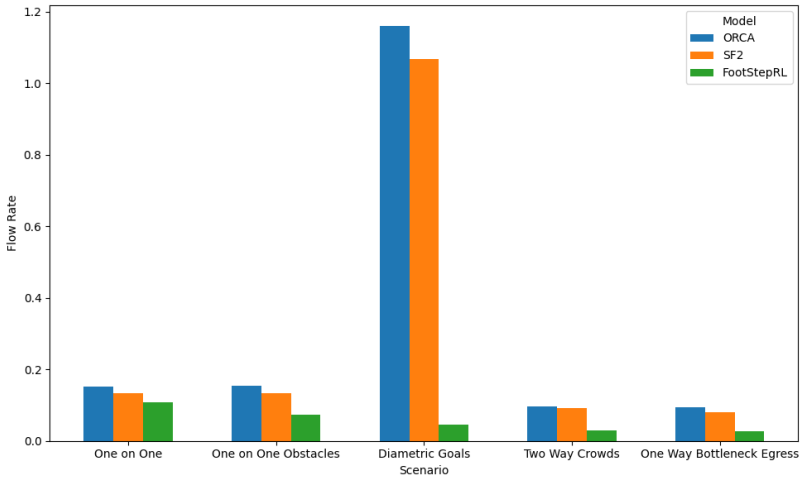


Figure 6.7: Group bar chart of flow rates for each model in each scenario

6.3.2 Qualitative Analysis

This section will analyze the navigation patterns and bipedal locomotion characteristics of agents with FootStepRL model. Figure 6.8 shows the screenshots of the 5 test scenarios with the COM trajectories of each agent.

Throughout the test scenes, the FootStepRL model demonstrates unique traits in both bipedal locomotion and navigation behaviours. Agents exhibit anticipatory behaviour in navigation by modifying their paths subtly and early before they come into direct contact with obstacles. This is especially noticeable in the One on One and One on One Obstacle scenarios when agents avoid collisions by maintaining arc-like, smooth trajectories.

In multi-agent scenarios, the model shows emergent crowd behaviours similar to human pedestrians. In the scene of Diametric Goals, in addition to early deviations from desired paths, the agents perform circular movement patterns around the centre of the circle and then tend to walk towards their assigned goal areas. In the Two Way Crowds scene, lane formation can be observed that the larger crowd is divided into smaller ones, passing between the gaps of the oncoming crowd. In addition, in the scenario of Bottleneck Egress, if the ideal path for the agents is a straight line through the egress, they pass through the egress earlier, while the agents on either side of them gradually slow down and take shorter steps as they walk closer to the wall, performing collision avoidance behaviours for both the wall and the agents beside them. It is worth noting that in crowded spaces, the waiting behaviour of the agents is not a complete stopping of movement, but rather an adjusting of their orientation or position with very small footsteps.

In the test scenarios mentioned above, the agents also exhibit traits of bipedal locomotion. When the agent is in the state of naturally walking towards its current goal area, its walking speed, footstep length, and foot orientation are essentially unchanged. It can be observed from the COM trajectory that the parabolic shape of the curve segments is approximately the same shape. The model also learns to perform more complex bipedal steering behaviours such as turning around. The agent performs a large turn by making large footsteps with its outside foot while having compensatory small steps with inside foot. During this process, the orientation of the feet also goes through several changes. In addition to this, in scenarios with limited space such as near a wall or corner, the agent will demonstrate a side-step-like behaviour to steer, as is shown in Figure 6.9.

To summarize, the experiments evidence that the FootStepRL model performs space-time planning and collision avoidance behaviour for both static and dynamic obstacles, as well as keeping the speed of movement as consistent as possible. Moreover, the agent using this model can perform behaviours that are characteristic of bipedal locomotion, which is not possible with particle-like agents.

6.4 Computational Efficiency

The computational efficiency of the model is also in this study. The experimental environment is a $1000 \times 1000\,\mathrm{m}^2$ plane. A specified number of agents are spawned at random locations within the plane and assigned a randomly located target. The agents are allowed to walk freely for one minute, and the average frame per second (FPS) is calculated over this time period. In this study, FPS is tested across scenarios with varying numbers of agents, ranging from one to 5000. As is shown in Figure 6.10, the model maintains interactive frame rates (> 30 FPS) with up to 3000 agents, indicating that while the absence of rigid body reduces collision accuracy, it significantly improves computational performance.

6.5 Limitations and Failure Cases

Although the present model has now demonstrated the possibility of incorporating bipedal locomotion and MARL, this research still has limitations in the design of the agent model and the design of the training schemes, resulting in some deficiencies in the training results.

First, the simplified agent model design resulted in unexpected training outcomes. Since this study focuses on the bipedal steering algorithm, the legs were not represented to avoid adding computational complexity to the agent's movements. The absence of a collider at the leg positions caused some training models to exhibit undesired behaviour, where agents avoiding collisions bypass each other by passing through the gaps between their torsos and feet instead of making the expected detour.

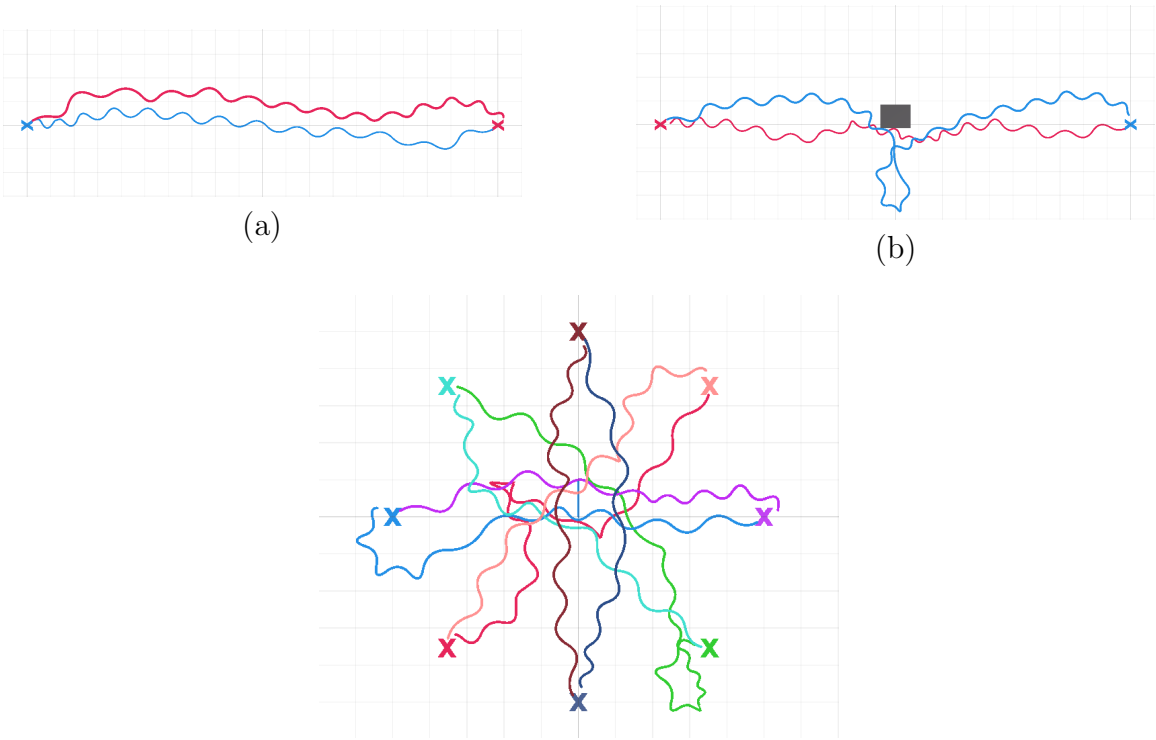
Second, while the agent performs reasonably well in simple scenarios such as One on One and One on One Obstacle, it encounters issues with navigating decisions in crowded scenarios. For example, in Figure 6.8 (b), the blue agent observing the wall and the oncoming agent near the obstacle chooses to avoid the obstacles by immediately making a small detour in the direction of the open space, resulting in the circle-like trajectory on the figure. In the scenario of Diagonal Goals, the agent's circling behaviour around the centre is not 100% collision-free. After a collision, the agent's decision-making becomes confused, causing the trajectory to lose its desired shape formation, and the agent may circle around the area near the target area before entering it. Furthermore, in Bottleneck Egress scenario, when walls only detect collisions and have no physical properties, we can observe that the agents which cause

congestion near the exit may either remain in place for a short period or attempt to walk through the wall. These failure cases may be due to the agents being too spread out in the maze environment during training, resulting in a lack of experience in handling dynamic obstacles. Another possibility is that the continuity reward R_d, which is related to the distance between the agent and the goal area and was introduced to avoid sparse rewards for reaching the target region, is too large compared to the time loss penalty P_t. This imbalance may have led the model to adopt an unexpected strategy, such as circling very close to the target region to compensate for earlier collision penalties.

In the training design, the reward function of this model consists of multiple weighted components, making the tuning of the reward function particularly challenging. As mentioned in Section 3.5.1, the training process of RL is often like a black box, where simply adjusting the weight of an individual component in a complex reward function does not immediately result in the desired outcome and may even worsen the performance. This has led to a long training cycle in this study, and complete collision avoidance, as seen in heuristic algorithms such as [2, 52], has yet to be achieved.

6.6 Discussion

This chapter provides a comprehensive assessment of the FootStepRL model, comparing its performance with baseline algorithms in several common crowd simulation scenarios. Quantitative metrics including success rate, path efficiency, and number of collisions demonstrate the potential of this approach which combines MARL and bipedal locomotion. Compared to traditional particle-based models, the FootStepRL model allows the agent to exhibit richer steering behaviours with both feet, such as turning around and side-stepping, as well as slowing down movement by reducing stride length. However, there are still limitations in performance in dense crowds. The reward engineering remains a challenge due to the complexity of bipedal locomotion, which is a problem to be tackled in future research. Despite this, the overall performance of FootStepRL suggests that it has the potential to advance the field of crowd simulation by providing a more nuanced representation of human movement in complex, dynamic environments.



(a)

(b)

(c)

(d)

(e)

Figure 6.8: Screenshots of COM trajectories in each test scenario. From (a) to (e): (a) One on One (b) One on One Obstacle (c) Diametric Goals (d) Two Way Crowds (e) One Way Bottleneck Egress

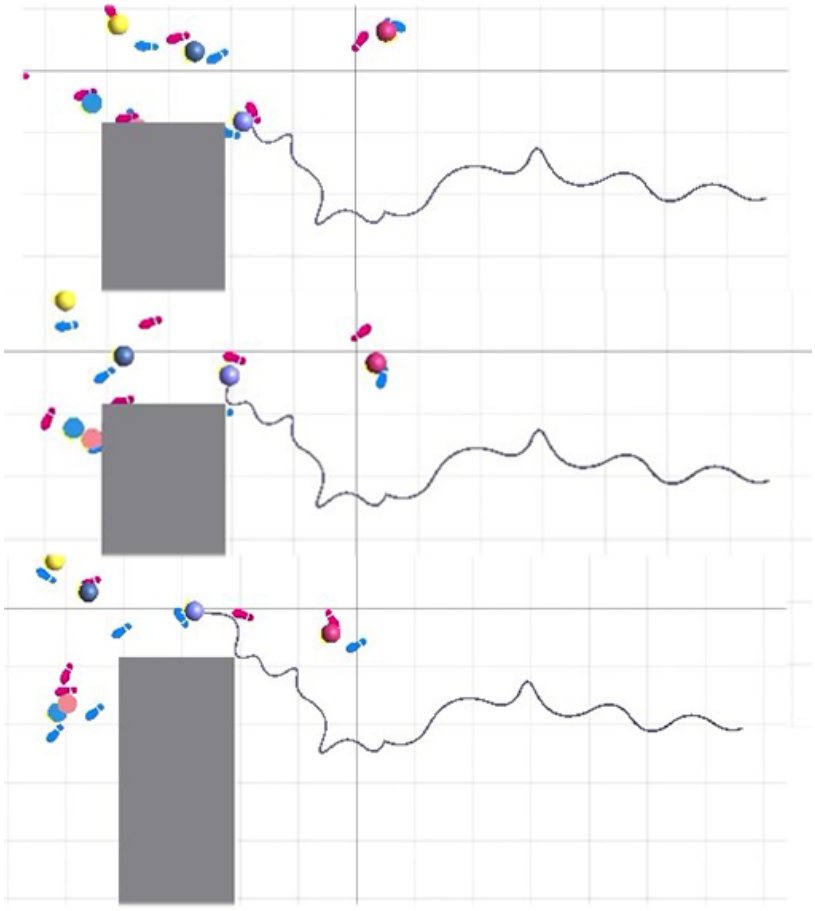


Figure 6.9: Screenshots of a sidestepping behaviour during the test scenario. The purple agent takes a lateral step with the right foot to get within range of the exit, followed by a step with the left foot to fully enter the exit.

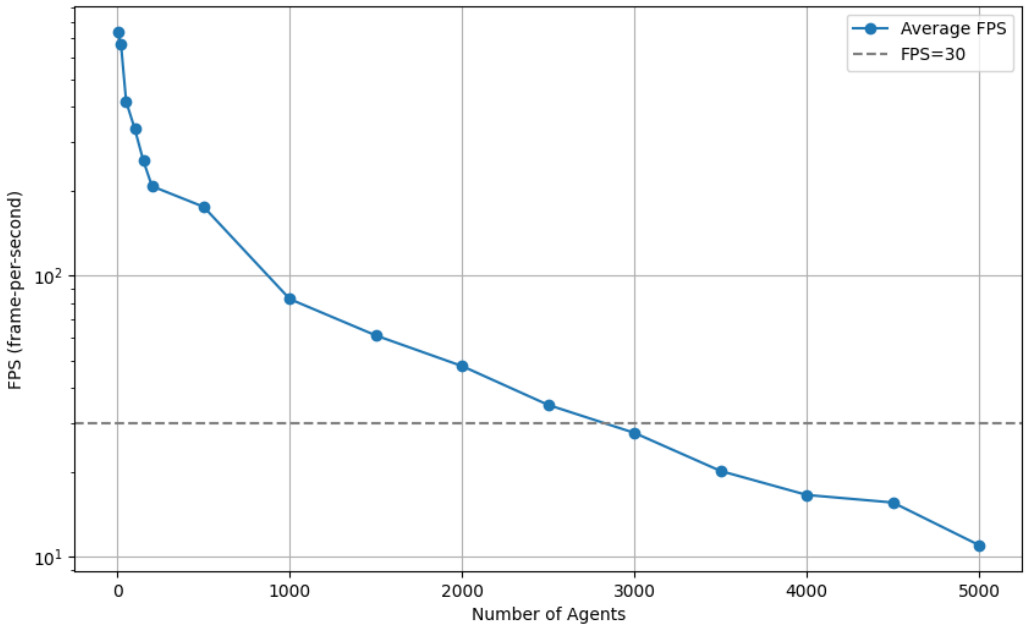


Figure 6.10: Mean frame per second (FPS) of FootStepRL with a different number of agents. The dashed line indicates an FPS of 30. The figure shows that the model's FPS drops below 30 as the number of agents approaches 3,000.

Chapter 7

Conclusion

This research has introduced a new method for simulating human crowds that blends biomechanical modelling of bipedal movement with Multi-Agent Reinforcement Learning (MARL). By overcoming the drawbacks of conventional particle-based algorithms and preserving computational viability, the study sought to create a steering model that is more realistic and effective by combining the PPO algorithm with the 3D inverted pendulum model.

7.1 Summary of Findings and Contributions

The major findings and contributions of this research can be summarized from the following perspectives.

First, from the standpoint of integrating theoretical frameworks, this research has illustrated the possibility of combining MARL with biomechanical modelling, especially the 3D inverted pendulum theory, for crowd simulation. The study demonstrates that this combination can capture intricate bipedal movements like turning and side-stepping, which are challenging to do with conventional particle-based methods. Although previous studies have proposed steering algorithms for modelling agents based on 3D inverted pendulum theory[52], their heuristic computations present challenges for efficient real-time rendering and do not always result in optimal solutions. The introduction of MARL helps address this problem. Through policy exploration during training, the agent can respond more flexibly to diverse environments.

Second, the research has implemented several technique components to design the

crowd simulation system. A multi-resolution synthetic vision system has been used in the study to help agents observe their surroundings. Horizontal rays provide a rough view of the slightly distant environment, while denser vertical rays are used to more accurately detect obstacles directly ahead. Such a way to simulate the field of view is computationally efficient. Additionally, this study has developed a comprehensive reward function that considers both biomechanical efficiency and navigation objectives. Moreover, the research has implemented randomized mazes as training environments that are procedurally generated and have adjustable density and complexity. The complexity and randomness of the training environment ensure that agents can explore and update policies across multiple environments, reducing the risk of overfitting to a single scenario.

Third, the research has provided empirical evidence for both the strengths and weaknesses of this approach. Quantitative and qualitative analysis of the FootStepRL model indicates that the agents manage to learn goal-reaching behaviours and local space-time planning strategies to avoid collisions. It demonstrates the feasibility of tuning reward functions to enable agents to learn obstacle avoidance, making a physics-free modelling setup possible and significantly reducing the program's running cost. The model also learns to generate complex bipedal steering behaviours, such as turning and decelerating by narrowing its stride and slowing its foot movements. As a result, this model simulates bipedal movements with greater fidelity than traditional particle-based agent algorithms.

7.2 Limitations and Future Work

Even though the study has made great strides in integrating MARL with biomechanical modelling, a number of drawbacks and potential topics for further research have been identified.

Despite the fair performance in simple scenarios, the model's performance degrades significantly in highly crowded situations. Furthermore, the absence of leg modelling in the agent's body affects the training results, potentially allowing agents to discover unintended avoidance strategies, such as passing through gaps between legs. Additionally, the complexity of the reward function makes it difficult to tune for optimal policy.

Based on the assessment results of this model, several intriguing avenues for further investigation have been suggested.

First, improvements can be made to the agent's model and animation. For example, biomechanical modelling of the entire body can be integrated into the model, particularly the legs and hands. Modeling the legs prevents the agent from learning unintended behaviours, such as crossing through gaps between the legs to avoid obstacles, while the oscillation of the hands during walking influences the agent's obstacle-avoidance strategy. Additionally, the accuracy of the FOV can be enhanced. Current multi-resolution FOVs use rays that leave gaps between them, potentially leading to incorrect obstacle perception. Future research could explore replacing these rays with adjacent 3D colliders, such as cubes. Future work could also introduce heterogeneous agents. This involves not only varying the agents' body sizes but also diversifying their physical attributes, such as maximum walking speed and stride length. Furthermore, the heterogeneous setup can be parameterized, enabling the steering model to flexibly simulate diverse populations based on user requirements, thereby adapting to more complex scenarios and promoting accessible design.

On the other hand, improvements could be made to the training process. The reward function can be further optimized to explore strategies that enhance the agent's obstacle avoidance performance. Additionally, curriculum learning can be introduced, enabling the agent to gradually adapt to increasingly complex environments in a stepwise manner. Furthermore, although data-driven algorithms depend on both data quality and diversity, combining them with MARL could be beneficial to incorporate imitation learning from human motion data.

7.3 Conclusion

This study has shown that MARL and biomechanical modelling may be used to create crowd simulations that are more realistic while still being computationally feasible. The framework created in this study offers a strong basis for further research in crowd simulation and related topics, even though there are still obstacles to be addressed. An important advancement in the field of crowd simulation is the capacity to handle intricate environmental interactions and produce more realistic human movement patterns.

Bibliography

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[2] Alejandro Beacco, Nuria Pelechano, and Mubbasir Kapadia. Dynamic footsteps planning for multiple characters. In *Proceedings of the Congreso Espaol de Informtica Grfica*, 2013.

[3] Glen Berseth, Mubbasir Kapadia, Brandon Haworth, and Petros Faloutsos. Steerfit: Automated parameter fitting for steering algorithms. In *Simulating Heterogeneous Crowds with Interactive Behaviors*, pages 229–246. AK Peters/CRC Press, 2016.

[4] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1:3–52, 2002.

[5] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

[6] Armin Bruderlin and Thomas W Calvert. Goal-directed, dynamic animation of human walking. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 233–242, 1989.

[7] NCD Risk Factor Collaboration. Evolution of height over time. `https://ncdrisc.org/height-mean-line-from-map.html`. Accessed: 2023-07-30.

[8] William Jay Conover. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

[9] Renato Contini. Body segment parameters, part ii. *Artificial limbs*, 16(1):1–19, 1972.

[10] John Conway et al. The game of life. *Scientific American*, 223(4):4, 1970.

[11] Camille Dupont, Luis Tobias, and Bertrand Luvison. Crowd-11: A dataset for fine grained crowd behaviour analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 9–16, 2017.

[12] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 560–565. IEEE, 1993.

[13] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998.

[14] Julio Godoy, Tiannan Chen, Stephen J Guy, Ioannis Karamouzas, and Maria Gini. Alan: adaptive learning for multi-agent navigation. *Autonomous Robots*, 42:1543–1562, 2018.

[15] Stephen J Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming C Lin, and Dinesh Manocha. Pledestrians: A least-effort approach to crowd simulation. In *Symposium on computer animation*, pages 119–128, 2010.

[16] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[17] Brandon Haworth, Glen Berseth, Seonghyeon Moon, Petros Faloutsos, and Mubbasir Kapadia. Deep integration of physical humanoid control and crowd navigation. In *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games*, pages 1–10, 2020.

[18] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[19] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[20] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[21] Kaidong Hu, Brandon Haworth, Glen Berseth, Vladimir Pavlovic, Petros Faloutsos, and Mubbasir Kapadia. Heterogeneous crowd simulation using parametric reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[22] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[23] Julio Cezar Silveira Jacques Junior, Soraia Raupp Musse, and Claudio Rosito Jung. Crowd analysis using computer vision techniques. *IEEE Signal Processing Magazine*, 27(5):66–77, 2010.

[24] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 1, pages 239–246. IEEE, 2001.

[25] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 215–223, 2009.

[26] Ioannis Karamouzas, Peter Heil, Pascal Van Beek, and Mark H Overmars. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games: Second International Workshop, MIG 2009, Zeist, The Netherlands, November 21-24, 2009. Proceedings 2*, pages 41–52. Springer, 2009.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[29] Arthur D Kuo. The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective. *Human movement science*, 26(4):617–656, 2007.

[30] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–118, 2007.

[31] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer graphics forum*, volume 26, pages 655–664. Wiley Online Library, 2007.

[32] Alex Tong Lin, Guido Montúfar, and Stanley J Osher. A top-down approach to attain decentralized multi-agents. In *Handbook of Reinforcement Learning and Control*, pages 419–431. Springer, 2021.

[33] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[34] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[35] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[37] Mehdi Moussaïd, Dirk Helbing, Simon Garnier, Anders Johansson, Maud Combe, and Guy Theraulaz. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society B: Biological Sciences*, 276(1668):2755–2762, 2009.

[38] Soraia Raupp Musse and Daniel Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Computer Anima-*

tion and Simulation'97: Proceedings of the Eurographics Workshop in Budapest, Hungary, September 2–3, 1997, pages 39–51. Springer, 1997.

[39] Michael S Orendurff, Ava D Segal, Glenn K Klute, Jocelyn S Berge, Eric S Rohr, and Nancy J Kadel. The effect of walking speed on center of mass displacement. *Journal of Rehabilitation Research & Development*, 41(6), 2004.

[40] World Health Organization. A healthy lifestyle - who recommendations. `https://www.who.int/europe/news-room/fact-sheets/item/a-healthy-lifestyle---who-recommendations`, May 2010. Accessed: 2023-07-30.

[41] Andreas Panayiotou, Theodoros Kyriakou, Marilena Lemonari, Yiorgos Chrysanthou, and Panayiotis Charalambous. Ccp: Configurable crowd profiles. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022.

[42] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 2012.

[43] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[44] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[45] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[46] Thomas C Schelling. Dynamic models of segregation. *Journal of mathematical sociology*, 1(2):143–186, 1971.

[47] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[49] Jing Shao, Kai Kang, Chen Change Loy, and Xiaogang Wang. Deeply learned attributes for crowded scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4657–4666, 2015.

[50] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[51] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. Steerbench: A benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds*, 20(5–6):533–548, Feb 2009.

[52] Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds*, 22(2-3):151–158, 2011.

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

[54] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

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

[56] Justin K Terry, Nathaniel Grammel, Sanghyun Son, Benjamin Black, and Aakriti Agrawal. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.

[57] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research: The 14th International Symposium ISRR*, pages 3–19. Springer, 2011.

[58] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008.

[59] John Von Neumann, Arthur Walter Burks, et al. Theory of self-reproducing automata. 1966.

[60] Qi Wang, Junyu Gao, Wei Lin, and Yuan Yuan. Learning from synthetic data for crowd counting in the wild. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8198–8207, 2019.

[61] Charles W Warren. Global path planning using artificial potential fields. In *1989 IEEE International Conference on Robotics and Automation*, pages 316–317. IEEE Computer Society, 1989.

[62] Charles W Warren. Multiple robot path coordination using artificial potential fields. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 500–505. IEEE, 1990.

[63] Christopher J. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[64] Michael W Whittle. *Gait analysis: an introduction*. Butterworth-Heinemann, 2014.

[65] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[66] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

[67] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.