UNIVERSITY OF SCIENCE AND TECHNOLOGY HANOI

Department of Applied Engineering and Technology

BACHELOR THESIS

Design Intelligent Controller for Quadruped Robot via Digital Twin

LẠI ĐỨC TUẤN
tuanld22bi13445@usth.edu.vn

Major: Mechatronic Engineering Technology

Supervisor: **Dr. NGUYEN Van Truong**
Dr. DUONG Viet Anh

Hanoi, 30/06/2025

Acknowledgment

First of all, I would like to express my gratitude to my family, who have supported me throughout my journey. Thanks to my father, who encourages me to study engineering. Thanks to my mother, who taught me to believe in myself.

I sincerely appreciate Dr. Nguyen Van Truong for his big belief and support in me, for providing the opportunity to be part of such a far-reaching project. His broad knowledge of system modeling and control and his very professional and friendly way of support had a wide influence on the results and also on our education. He persistently helped us with many complex questions and motivated us to expand the scope of the project in the right direction.

Many thanks to Mr. Duong Dai Nhan for the supervision and coaching of the whole focus project. Not only was he able to keep the very important overview, with a lot of effort and knowledge about this report, but also helped define and manage all interconnections in order to bring all parts together.

Great thanks to Irobot Lab's Spot dogs team for the support and corporate of the whole project. This project cannot be finished without your generous help. We are very grateful to all members of the team and are very happy to have spent time working together on this project. We have formed a very capable team and helped each other realize our ideas.

Finally, I greatly thank to all of my colleagues who helps me throughout my three years in USTH.

Contents

List of Figures

List of Tables

List of Abbreviations

MDP	Markov Decision Process
RL	Reinforcement Learning
DT	Digital twin
TD3	Twin Delayed Deep Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
PPO	Proximal Policy Optimization
URDF	Unified Robot Description Format
UART	Universal Asynchronous Receiver/Transmitter
IMU	Inertial Measurement Unit
FL	Front Left
FR	Front Right
BL	Back Left
BR	Back Right
PM	Physical Model
VM	Virtual Model
ADC	Analog Digital Converter
CAD	Computer-Aided Design
VRI	Virtual-Real Interaction

Abstract

Quadruped robots represent a significant advancement in mobile robotics due to their ability to traverse uneven and unstructured terrains. Recent developments in reinforcement learning (RL), particularly the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, have enabled more adaptive and autonomous locomotion strategies for these systems. However, bridging the gap between simulation-trained policies and real-world deployment remains a major challenge.

This thesis explores the use of TD3 to train a quadruped robot, named SpotDog, in a PyBullet simulation environment. A digital twin framework is developed to synchronize and model the robot's physical behavior in real time, facilitating sim-to-real transfer. The research aims to achieve stable and efficient locomotion by combining learning-based control with a dynamic digital representation of the robot.

Experimental results demonstrate successful training of walking behaviors in simulation and effective interaction between the simulated and physical models. The outcomes suggest that the integration of reinforcement learning and digital twin technologies can significantly enhance the adaptability and deployment of quadruped robots in real-world scenarios.

1 Introduction

1.1 Background and motivation

Quadruped robots [1] represent a significant advancement in the field of mobile robotics due to their ability to navigate rough, uneven, or unstructured terrains, which are conditions where wheeled or tracked robots often fail. Their legged structure allows for greater maneuverability and adaptability, making them suitable for difficult tasks such as exploration, search and rescue, and industrial inspection in hazardous environments.

The growing interest in quadruped robots is fueled by progress in mechanical design, sensor integration, and artificial intelligence, particularly machine learning. For instance, Boston Dynamics' Spot (Figure.1a) has demonstrated the real-world applicability of such robots. Meanwhile, research institutions and robotics laboratories have begun to explore reinforcement learning (RL) as a promising approach to achieving dynamic, flexible, and adaptive locomotion.

Traditional control methods, while effective under certain conditions, are often limited by the need for accurate system modeling and extensive parameter tuning. This has led to increasing interest in learning-based approaches, such as Deep Reinforcement Learning, which allow robots to learn control policies directly through interaction with their environment. Among these methods, the Twin Delayed Deep Deterministic Policy Gradient algorithm stands out for its performance in continuous control tasks.

This project is motivated by the desire to explore how TD3 [2] can be used to control a quadruped robot, namely SpotDog, developed by IrobotLab team, in a simulated environment using PyBullet. The goal is to train the robot to walk autonomously and stably using an adaptive, learning-based approach. Then, the trained policy transfers to the real model to test simulation to reality transference. Furthermore, a digital twin framework is implemented to mirror the real-time behaviors of SpotDog, enabling synchronized testing, monitoring, and debugging.

This digital twin plays a crucial role in evaluating the fidelity of the sim-to-real transfer and enhancing the reliability of policy deployment in real-world environments.

(a) Boston's Spot dog (b) MIT's mini cheetah

Figure 1: Examples of Quadruped robot

1.2 Literature review

1.2.1 Reinforcement Learning for Quadruped robot locomotion

In recent years, reinforcement learning (RL) has emerged as a powerful approach for training quadrupedal robots to perform agile and adaptive locomotion. Unlike classical control methods, which require hand-crafted gait patterns and extensive tuning, RL enables the autonomous development of locomotion strategies by interacting with the environment and learning from experience.

One of the most influential studies in this field is by Tan et al., where deep reinforcement learning was used to train agile trotting and galloping behaviors in simulation, followed by successful deployment on real quadruped hardware using domain randomization to bridge the sim-to-real gap [3]. Building on this foundation, researchers have proposed more sophisticated architectures to enhance performance and generalization.

For instance, Jain et al. introduced a hierarchical RL framework that separates high-level decision-making from low-level motor control, allowing the robot to exhibit emergent behaviors like turning and speed modulation [4]. Similarly, Li et al. integrated trajectory generation with reinforcement learning, enabling quadrupeds

to navigate unstructured terrains by adjusting foot placement and stride length in real time [1].

Vision-based locomotion is another promising direction. Yang et al. proposed the LocoTransformer, a cross-modal transformer architecture that fuses proprioceptive data and depth images to generate robust walking strategies over uneven terrain [5]. This approach demonstrated significant improvement in anticipatory behavior and obstacle avoidance.

More recently, decentralized and multi-agent learning strategies have gained attention. Zhang et al. treated each leg of the quadruped as a separate agent under a shared critic in the MASQ framework, leading to faster convergence and increased gait diversity [6].

Additionally, safety-aware and risk-sensitive approaches have been proposed to ensure robust performance in dynamic or uncertain environments. For example, Smith et al. applied distributional reinforcement learning to minimize the likelihood of catastrophic failures during high-speed locomotion or recovery from slips [7].

Overall, these advancements illustrate that RL is a highly effective tool for enabling complex and adaptive behaviors in quadruped robots. By leveraging simulation, hierarchical control, vision integration, and multi-agent learning, researchers continue to push the boundaries of robot agility and robustness in real-world applications.

1.2.2 Digital Twin

The concept of a digital twin (DT) — a synchronized virtual model of a physical system — has shown increasing relevance in robotic locomotion, yet its application to quadrupedal robots remains underexplored. While DT-driven control and learning have seen growth in industrial manipulators [8], their benefits for legged robots are now being realized through emerging research combining DTs with reinforcement learning (RL).

Hybrid Control and Digital Twin Integration Du et al. propose a framework for quadruped locomotion that integrates model-predictive control (MPC) and neural adaptive sliding mode control (SMC), verified using a DT in Webots [9]. The DT provides a testbed for tuning body-leg coordination, demonstrating accurate trajectory tracking and control stability before deployment.

Sim-to-Real and DT-Driven RL The established sim-to-real strategy, widely used in RL-based quadruped locomotion, relies on domain randomization to bridge simulation and reality [3, 4]. However, a DT enables ongoing synchronization between the virtual and physical robot during training, allowing real-time policy refinement. Sun et al. developed such an online DT-RL framework—demonstrated on a collaborative manipulator—for real-time obstacle avoidance, suggesting applicability to legged systems [8].

Direct DT-RL Methods While most DT-RL applications target industrial robots or AMRs [8, 10], Ahmad Khan Jadoon & Ekpanyapong (2025) pioneered a lightweight DT-based RL for ANYmal-like quadrupeds, using RaiSimGymTorch to train control policies in sim before real-world deployment [11]. This work highlights the DT's role as a simulation stand-in that accelerates learning using modest computational resources.

Broader DT-RL Frameworks Beyond legged robots, DT-RL frameworks for mobile robots in production environments show how synchronized state updates enable DRL policies to dispatch AMRs effectively [10]. These methods underscore the potential for real-time learning in quadrupeds operating in dynamic terrains.

Challenges and Potential Integrating DTs with quadruped RL remains challenging: building high-fidelity DTs is costly, synchronization latency must be minimized, and safety guarantees are required before real-world deployment. Nonetheless, these studies indicate that DTs offer a pathway to continuous adaptation, iterative validation, and robust sim-to-real transfer unavailable in offline-only pipelines.

In conclusion, leveraging DTs in quadruped robot locomotion is an emerging area that builds upon established RL and sim-to-real approaches. DTs enhance the training loop by enabling real-time feedback and iterative refinement, potentially reducing development time and improving robustness. Future work should focus on standardized DT architectures for legged systems, frameworks for live synchronization, and incorporation of perception-driven DTs for complex terrain navigation.

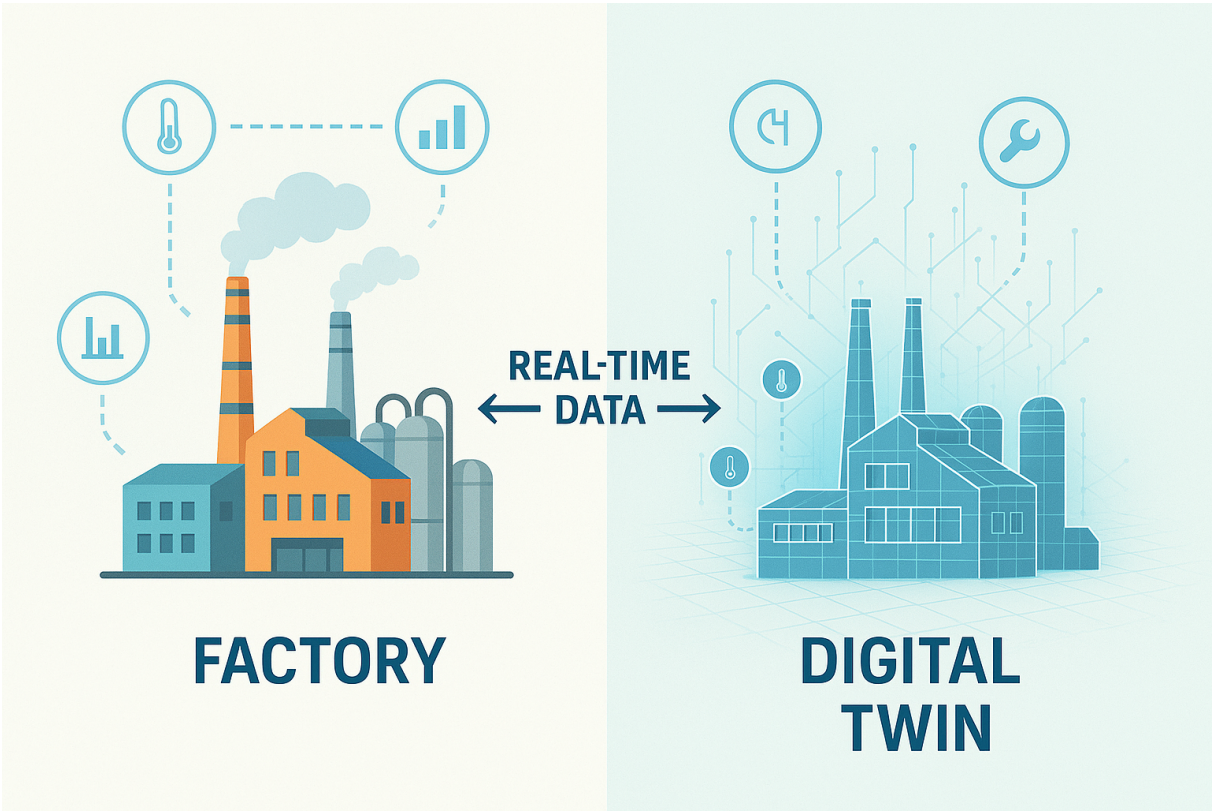

Figure 2: Digital twin in Industry

1.3 Objective & Expected outcome

After reviewing and organizing the problems, this thesis focuses on the following key aspects:

- First, a SpotDog model is built and simulated on the PyBullet platform. This serves as the foundation for applying reinforcement learning (RL) to train quadruped robot locomotion across different environments before deployment in the real world.

- Second, the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is tested to train the SpotDog's locomotion while walking on a sloped terrain.

- Third, the trained policy is deployed on the real robot model to evaluate the effectiveness of the digital twin-driven reinforcement learning approach.

- Finally, real-time parallel interaction between the virtual and physical models is tested to analyze the reality gap. This is a foundational step toward digital twin research for quadruped robots.

The expected outcomes of this research include:

- Developing a virtual model that closely resembles the real-world system.

- Successful training of quadruped robot locomotion using the TD3 algorithm.

- Achieving successful sim-to-real transfer by applying the trained policy to the physical quadruped robot.

- Achieving smooth and precise interaction between the virtual model and the physical model.

1.4 Structure of thesis

This thesis is organized into five sections, outline as follows:

- **Section 1**

  This section presents the background and motivation for the research, the challenges in development of quadruped robot, the objective and expected outcome.

- **Section 2**

  This section introduces the quadruped robot dynamics model and the fundamental theories of RL.

- **Section 3**

  This section details the setup parameters for training RL. It is an essential step for training a quadruped robot.

- **Section 4**

  This section illustrates the digital twin system setup.

- **Section 5**

  This section discusses the experiment results, highlighting the strength and limitations of the proposed methods.

- **Section 6**

  This section summarizes the key findings, contributions, and future planning for researching in quadruped robot.

2 Preliminaries

2.1 Kinematics Model

2.1.1 Body frame Coordination

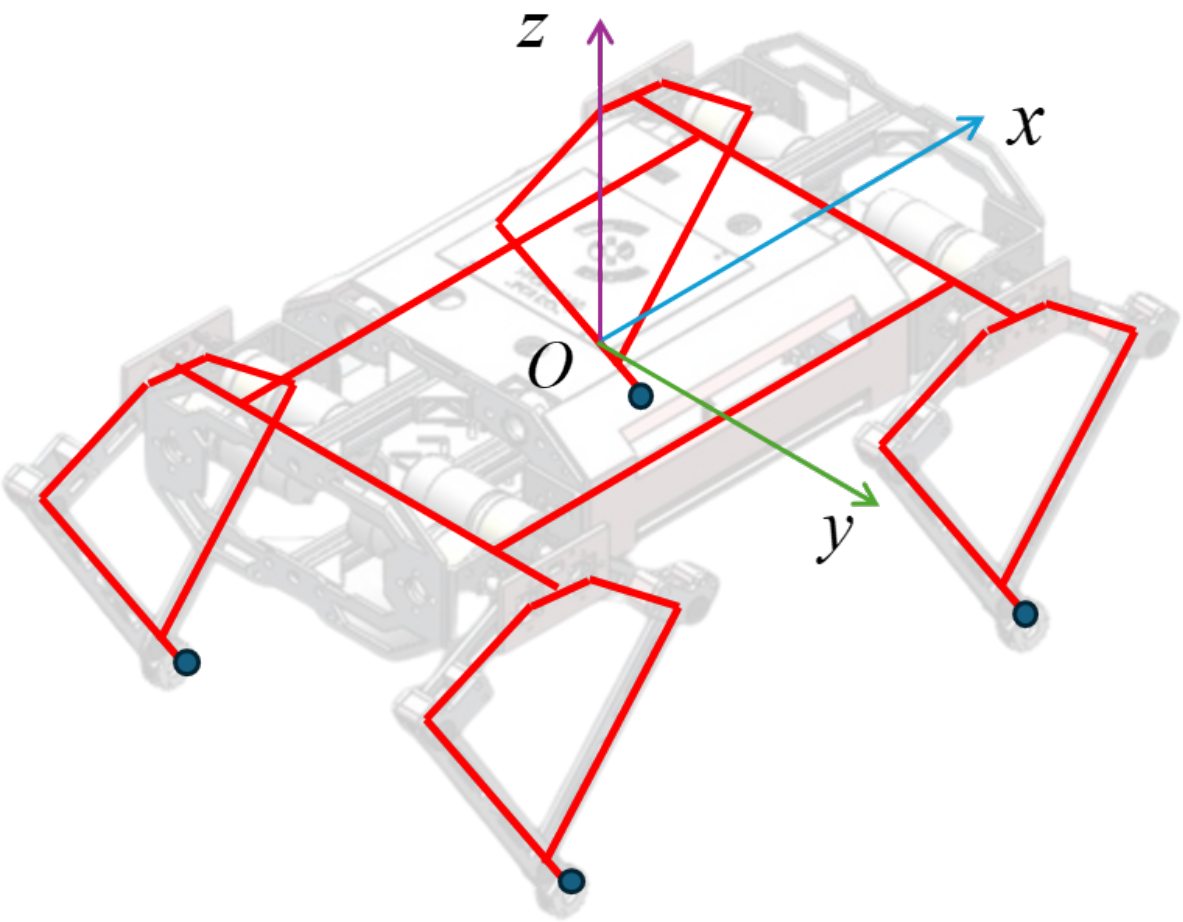

Figure 3: Spot Dog frame

Table 1: Robot parameters

Category	Description	Symbol
Dimensions	Length of the robot	L
	Width of the robot	W
Coordinate Systems	Coordinate of the body center	$[x_m, y_m, z_m]$
	Local coordinate of each leg	$[x_0, y_0, z_0]$
Variables	Yaw angle of the robot	θ
	Pitch angle of the robot	ψ
	Roll angle of the robot	ω

The robot model is represented and analyzed as shown in Fig.3, with its parameters summarized in Table 1. When the robot body undergoes rotation or translation, the position of each leg relative to the body changes, since the body frame is used as the reference coordinate system, while the legs remain in contact with the ground.

As a result, the coordinates of the legs relative to the body change over time. These coordinates can be calculated by applying appropriate rotation matrix transformations.

Let point M have coordinates (x_m, y_m, z_m) defined in space with respect to the body's center as the origin. Let Roll, Pitch, and Yaw represent the rotational motions around the x-, y-, and z-axes, respectively. The rotation matrices that represent these rotations are:

- $R_x(\omega)$: Rotation matrix about the x-axis (Roll)
- $R_y(\psi)$: Rotation matrix about the y-axis (Pitch)
- $R_z(\phi)$: Rotation matrix about the z-axis (Yaw)

These matrices can be combined to transform the coordinates of the legs with respect to the rotated body frame.

$$R_x(\omega) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) & 0 \\ 0 & \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$R_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$\mathrm{R}xyz = \mathrm{R}x\mathrm{R}y\mathrm{R}z \tag{4}$$

$$T_M = Rxyz \times \begin{bmatrix} 1 & 0 & 0 & \mathrm{Xm} \\ 0 & 1 & 0 & Ym \\ 0 & 0 & 1 & Zm \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The kinematic equations of the body coordinate center (x_m, y_m, z_m) and the local coordinate system of each leg (x_0, y_0, z_0) are defined by the transformation matrices given in Eq.(6)-(9). The position and orientation of each leg can be computed based on the position and orientation of the robot body.

$$T_{FR} = T_M \times \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) & -\mathrm{L}/2 \\ 0 & 1 & 0 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) & W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$T_{FL} = T_M \times \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) & \mathrm{L}/2 \\ 0 & 1 & 0 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) & W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$T_{BR} = T_M \times \begin{bmatrix} \cos(-\pi/2) & 0 & \sin(-\pi/2) & \mathrm{L}/2 \\ 0 & 1 & 0 & 0 \\ -\sin(-\pi/2) & 0 & \cos(-\pi/2) & -W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$T_{BL} = T_M \times \begin{bmatrix} \cos(-\pi/2) & 0 & \sin(-\pi/2) & -\mathrm{L}/2 \\ 0 & 1 & 0 & 0 \\ -\sin(-\pi/2) & 0 & \cos(-\pi/2) & -W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

The Spot Dog robot has four identical legs; therefore, only the front right leg is analyzed, and the results apply to the other legs as well. For the research model, I consider using a 2-dof parallel manipulator mechanism for the leg, featuring two active joints controlled by actuators and three freely passive joints, as shown in figure.4. Each parameter is denoted:

- $q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T$ as joint angular.
- $l = \begin{bmatrix} l_1 & l_2 & l_3 & l_4 & l_5 \end{bmatrix}^T$ as the length of the links.
- $X(x, y)$ as position of the feed.

2.1.2 Forward Kinematic

Forward kinematics uses the given q to calculate the end effector at the foot. From [12], the forward kinematic is rewrote:

$$x = \frac{1}{2} \sum_{i=1}^{4} l_i \cos q_i + \frac{l_5}{2} \tag{10}$$

$$y = \frac{1}{2} \sum_{i=1}^{4} l_i \sin q_i \tag{11}$$

where q_i and $l_i (i = 1, 2, 3, 4)$ are the joint angles and link length, respectively. From Eq.11, two passive joints q_3 and q_4 are depending on two active joints q_1 and q_2 as computed functions follow:

$$q_1' = \arccos\left(\frac{l_3^2 + (a^2 + b^2) - l_4^2}{2l_3\sqrt{a^2 + b^2}}\right) + \arctan\left(\frac{b}{a}\right) \tag{12}$$

$$q_2' = \pi - \arccos\left(\frac{l_4^2 + (a^2 + b^2) - l_3^2}{2l_4\sqrt{a^2 + b^2}}\right) + \arctan\left(\frac{b}{a}\right) \tag{13}$$

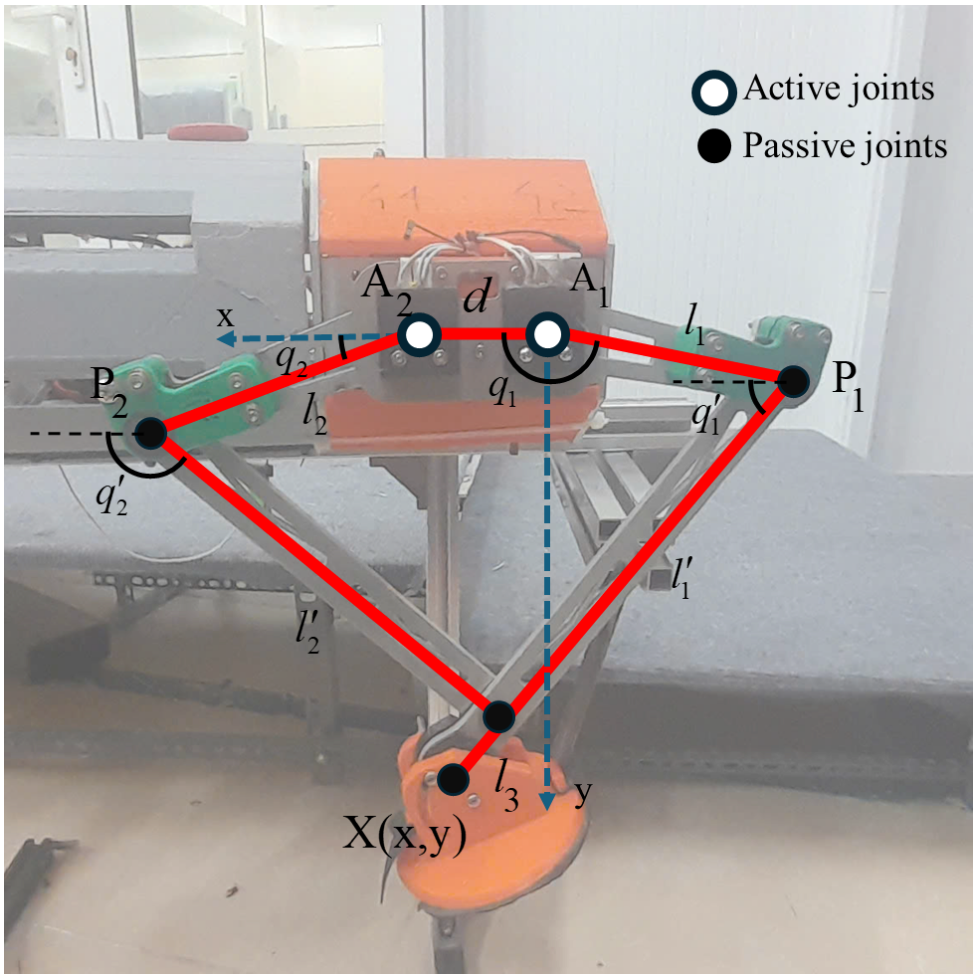

Figure 4: 5-bar mechanism leg

with

$$a = -l_1 \cos q_1 + l_2 \cos q_2 + l_5$$

$$b = l_2 \sin q_2 - l_1 \sin q_1$$

2.1.3 Inverse Kinematic

The inverse kinematic equation computes the active joint q from the given end-effector position $X(x, y)$. For our model, using the Cosine theorem and trigonometric relations, the inverse kinematic function can be obtained:

$$q_1 = \arctan\left(\frac{y}{x}\right) + \arccos\left(\frac{l_1^2 + (x^2 + y^2) - l_3^2}{2l_1\sqrt{x^2 + y^2}}\right) \tag{14}$$

$$q_2 = \pi - \arctan\left(\frac{y}{l_5 - x}\right) - \arccos\left(\frac{l_2^2 + ((l_5 - x)^2 + y^2) - l_4^2}{2l_2\sqrt{(l_5 - x)^2 + y^2}}\right) \tag{15}$$

2.2 Walking Control Strategy

2.2.1 Gait planning

Gait planning is a crucial step in designing a stable QR walk. In general, there are 3 fundamental gaits noticed by animals for the quadruped model, as shown in Fig.5. Those are:

- trot: diagonal pairs of legs move at the same time;
- pace: lateral pairs of legs move at the same time;
- bound: the front legs move at the same time, as do the rear.

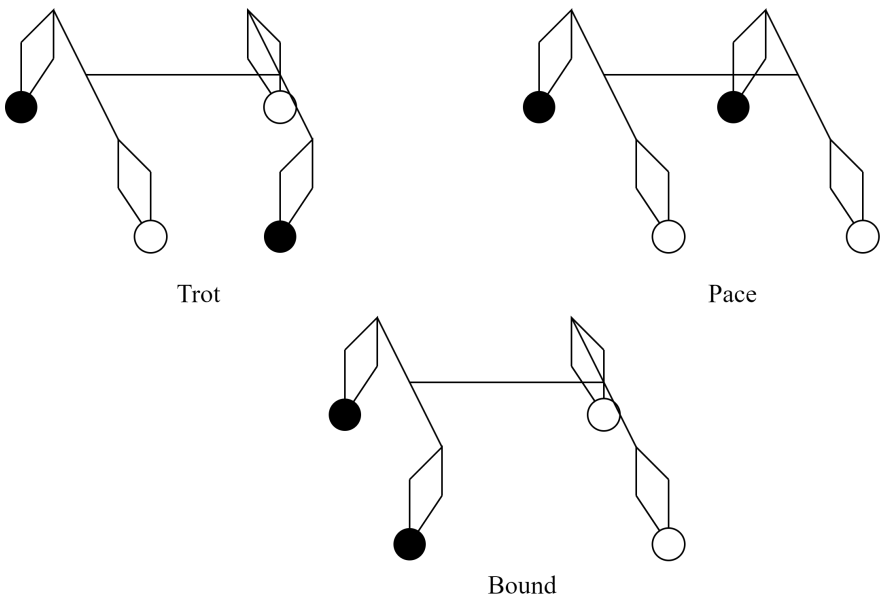

Figure 5: Basic symmetric gaits for quadruped robot.

In this research, trot gait is chosen for experiment. For stability, the walk must satisfy the follow conditions:

$$\phi = 0.5$$
$$q(0) = -q(T_0)$$
$$\dot{q}(0) = -\dot{q}(T_0)$$
$$T_0 = (1-\phi)T = 0.5T$$

where the duty factor,$\phi(0 < \phi < 1)$, is defined as the ratio of the period in which the leg contacts the ground over the cycle of the walk,T; T_0 is a swing phase time period.

2.2.2 Swing Phase Trajectory

After determining the robot's main gait, the next step is to plan the motion trajectories for each leg to ensure the robot moves as intended. The most common approach in leg trajectory planning is to create separate trajectory functions for the swing leg (leg in the air) and the stance leg (leg on the ground), and then connect them to form a closed-loop trajectory over one movement cycle of the robot.

A key aspect of this control method is the accurate determination of the foot end-point state at the moment of ground contact. The stability of the robot's body largely depends on this contact, along with the frictional forces acting on the foot when it touches the ground. To achieve both stability and motion optimization, various leg trajectory designs can be employed.

In this study, for optimal model, the elliptical trajectory is implemented for the swing phase of the model's leg. The trajectory takes form of:

$$\begin{cases} x_{toe} = x_C + R\sin\left(\frac{2\pi}{T} + \varphi\right) \\ y_{toe} = y_C + R\cos\left(\frac{2\pi}{T} + \varphi\right) \end{cases} \tag{16}$$

where $[x_{toe}, y_{toe}]^T$ are the coordinates of the trajectory that the toe should be place at, $[x_C, y_C]^T$ is the ellipse center, The designed trajectory graph is shown in Fig.6.

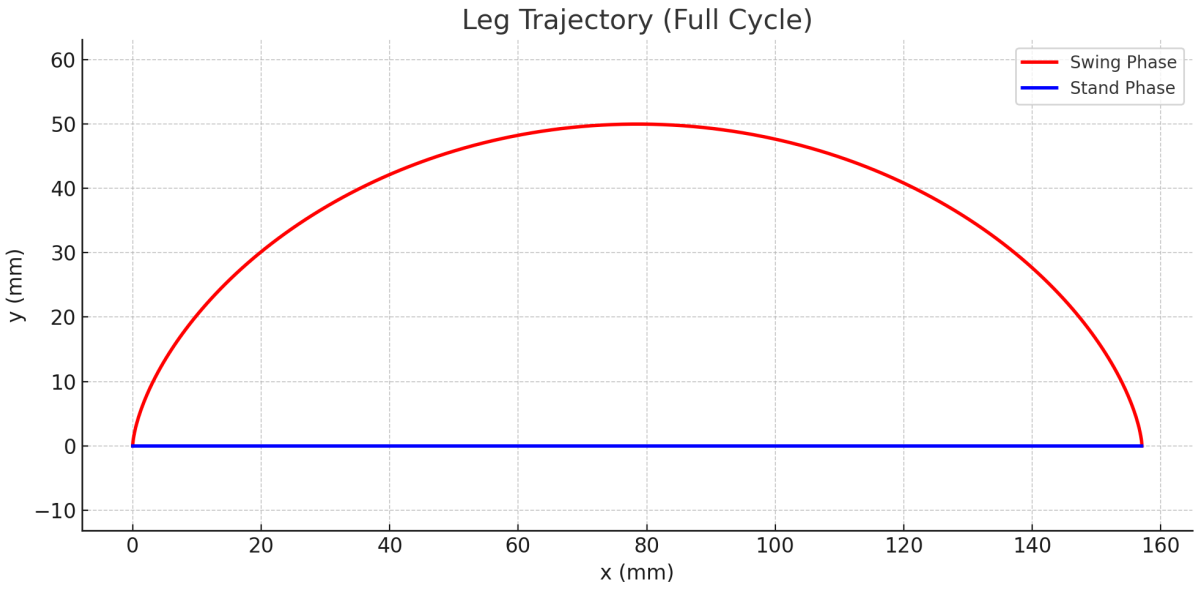

Figure 6: Leg trajectory full cycle

2.3 Markov Decision Process

Markov Decision Process (MDP) [13] is a mathematical framework for sequential decision-making, commonly used to model the interaction between intelligent agents and stochastic environments. In the context of training a quadruped robot, the MDP is represented as a quintuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is the **state space**, representing the environment's current condition as perceived by the quadruped robot. This includes information such as body orientation, tilt angles, foot contact points, and joint positions.
- $\mathcal{A}$ is the **action space**, which contains control commands such as the angular velocity and target position for each leg joint. These actions are typically generated by a policy network (e.g., an Actor Network in actor-critic methods).
- $\mathcal{P}(s'|s, a)$ is the **state transition probability**, defining the likelihood of transitioning to a new state s' given the current state s and action a. This captures the robot's dynamic behavior and the physics of the environment.
- $\mathcal{R}(s, a)$ is the **reward function**, providing a scalar feedback signal based on the effectiveness of the selected action in achieving locomotion objectives (e.g.,

forward movement, stability, or energy efficiency).

- $\gamma \in [0, 1]$ is the **discount factor**, which balances the importance of immediate and future rewards. A value close to 1 encourages long-term planning, while a value near 0 emphasizes short-term gains. It also ensures convergence by reducing the impact of future rewards in infinite-horizon tasks.

In each episode of training, the quadruped robot interacts with the environment by observing its current state, selecting an action based on its policy, and receiving the next state and corresponding reward. Over time, the policy is optimized through reinforcement learning algorithms (e.g., TD3, PPO) to improve locomotion performance under various terrains and dynamic conditions.

The agent-environment interaction in a Markov decision process is shown in Fig. 7.

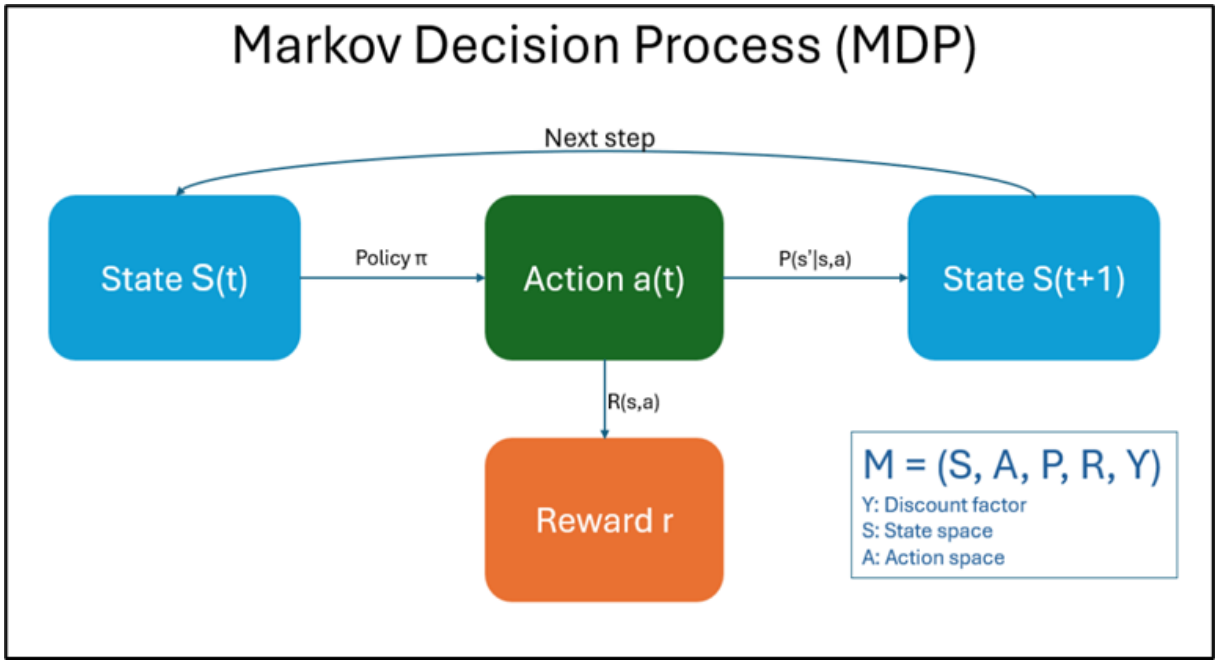

Figure 7: Markov Decision Process framework

The locomotion planning of the quadruped robot follows a deterministic policy. Accordingly, the state-value function $V^{\pi}(s)$ under a given policy π represents the expected cumulative reward starting from state s, and is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots \mid s_0 = s\right] \tag{17}$$

Given the presence of an action, equation (17) can be reformulated into the action-value function when a deterministic policy is adopted. Here, the action is determined as $a = \pi(s)$ based on the state s, resulting in:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots \mid s_0 = s, a_0 = a\right] \tag{18}$$

To derive the optimal policy, we aim to maximize both the state-value and action-value functions. By selecting the maximum expected return, we obtain the optimal state-value function V^* and the optimal action-value function Q^*, which are formalized through the Bellman optimality equations, as shown in equation (19).

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) = \max_{a}\left\{R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^*(s')\right\} \\ Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} P_{sa}(s') \max_{a'} Q^*(s',a') \end{aligned} \tag{19}$$

Where, s' is the next state, a' is the action taken in the next state, $P_{sa}(s')$ is the probability of taking action a to transit to state s' from state s, and is also known as the state transition probability. The strategy adopted in calculating the Bellman optimality equation is the optimal strategy π^*, as shown in equation.20.

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) = \arg\max_{a \in \mathcal{A}}\left\{R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^*(s')\right\} \tag{20}$$

In the research of quadruped robot locomotion planning problem in dynamic environments, the Markov property refers to that the mobile robot selects the optimal action by sensing the state of the surroundings and information of its own position and posture. Based on the optimal action, it gets the feedback return value by sensing the changed environment and its own state, and then adjusts the next action, and continues to cycle until a complete optimal strategy is formed.

2.4 TD3 Algorithms

2.4.1 TD3 Background

Twin Delayed Deep Deterministic Policy Gradient (TD3) is a reinforcement learning algorithm that belongs to the Actor–Critic family with a deterministic policy. The actor network uses the target policy $\mu(s|\theta^\mu$ to predict the actions for a given state, while the critic network estimates the action-values function Q, which is the total discounted reward of an action applied to the given state. In other words, the Q value gives an insight for the quality of the action applied. It was developed to address the shortcomings of DDPG (Deep Deterministic Policy Gradient), such as overestimation bias and instability during training.

TD3 is particularly effective in environments with continuous action spaces, low noise, and high precision requirements, making it suitable for industrial robotic control tasks.

The objective of TD3 is to find a deterministic policy $\mu(s \mid \theta^\mu)$ such that the expected cumulative reward over a finite horizon is maximized:

$$J(\theta^\mu) = \mathbb{E}_{s_0 \sim p(s)} \left[\sum_{k=0}^{\infty} \gamma^k r(s_t, \mu(s_t)) \right] \tag{2.28}$$

2.4.2 Structure of TD3

The structure of TD3 is presented in algorithm 1. TD3 uses three main components:

- **Actor:** policy $\mu_\theta(s) : \mathcal{S} \to \mathcal{A}$
- **Twin Critic:** Estimating the action-value function $Q^\mu(s, a)$ to reduce bias
- **Target Networks:** μ', Q'_1, Q'_2, which are delayed copies of the main network to stabilize training

The update process includes:

Update Critic:

$$y = r_t + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \pi_\phi(s_{t+1})) \tag{22}$$

where r_t is the reward at time t, γ is the discount factor, and π_ϕ is a deterministic policy with parameters ϕ, which maximizes the expected return. For each Critic Q_θ, the mean squared error loss function is used:

$$L(\theta) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \left[\left(Q_\theta(s_t, a_t) - y_t \right)^2 \right] \tag{23}$$

Update Actor: The objective of the Actor is to maximize the value estimated by Critic 1:

$$L(\theta) = -\mathbb{E}_{s_t}[Q_\theta(s_t, \pi_\phi(s_t))] \tag{24}$$

Gradient according to the Deterministic Policy Gradient Theorem:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s_t \sim D} \left[\nabla_a Q_1(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s) \right] \tag{25}$$

Where:

- $\nabla_\phi J(\phi)$: Gradient of the objective function (expected return) with respect to the policy parameters ϕ
- $\mathbb{E}_{s_t \sim D}$: The state s is sampled from the replay buffer (or from the state distribution induced by running the policy)
- $\nabla_a Q_1(s, a)|_{a=\pi_\phi(s)}$: The gradient of the critic with respect to the action, evaluated at $a = \pi_\phi(s)$
- $\nabla_\phi \pi_\phi(s)$: The gradient of the policy (Actor network) with respect to its parameters ϕ

Using Polyak averaging (also known as "soft update"):

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{26}$$

Where $\tau \in [0.001, 0.01]$ is the soft update coefficient. Updating the target networks helps stabilize training and prevents sudden changes.

Algorithm 1 TD3 (Twin Delayed Deep Deterministic Policy Gradient)

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target network weights $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
    Select action with exploration noise $\mathbf{a}_t \sim \pi_\phi(\mathbf{s}_t) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$
    Observe reward r_t and new state $\mathbf{s}_{t+1}$
    Store transition tuple $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in $\mathcal{B}$
    Sample mini-batch of N transitions $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ from $\mathcal{B}$
    $\tilde{\mathbf{a}} \leftarrow \pi_{\phi'}(\mathbf{s}_{t+1}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r_t + \gamma \min_{i=1,2} Q_{\theta'_i}(\mathbf{s}_{t+1}, \tilde{\mathbf{a}})$
    Update critics: $\theta_i \leftarrow \arg\min_{\theta_i} \frac{1}{N} \sum (y - Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t))^2$ for $i \in \{1, 2\}$
    if $t \mod d = 0$ **then**
        Update ϕ by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_a Q_{\theta_1}(\mathbf{s}_t, \mathbf{a}_t)|_{\mathbf{a}=\pi_\phi(\mathbf{s}_t)} \nabla_\phi \pi_\phi(\mathbf{s}_t)$$

        Update target networks:

$$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i \quad \text{for } i \in \{1, 2\}$$

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$

    end if
end for

2.4.3 Core Concepts of TD3

TD3 introduces three main enhancements over DDPG to improve performance [13]. Firstly, it utilizes a clipped double Q-learning strategy, replacing the conventional Q-learning approach used in DDPG. Secondly, TD3 incorporates action noise to address overfitting issues related to sharp value function peaks, which are common in deterministic policies. This action noise also contributes to smoother target policy behavior. At each timestep, the two Q-value networks, $(Q_{\theta_1}, Q_{\theta_2})$, are updated toward the minimum target value of the actions selected by the target policy, as described in Equation (21).

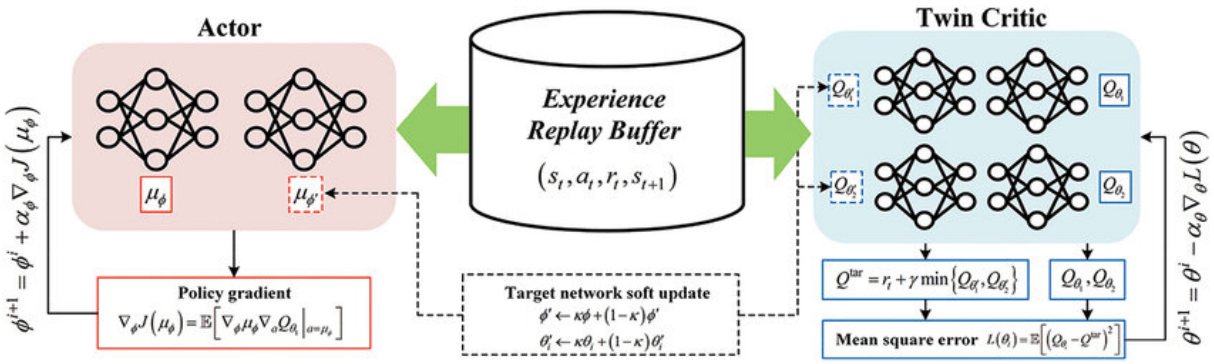

Figure 8: Workflow of TD3 algorithm

$$y = \mathbf{r}_t + \gamma \min_{i=1,2} Q_{\theta_i}\left(\mathbf{s}_{t+1}, \pi_{\phi'}\left(\mathbf{s}_{t+1}\right) + \varepsilon\right). \tag{21}$$

where $\mathbf{r}_t$ is the reward at time t, γ is the discount factor and π_ϕ is a deterministic policy, with parameters ϕ, which maximizes the expected return. ε is the clipped Gaussian action noise added and is defined by equation.(22)

$$\varepsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{22}$$

The third feature of TD3 is to delay the policy updates by a fixed number of updates to the critic. This is done to suppress the value estimate variance caused by the accumulated TD error. Parameters ϕ are updated according to the deterministic policy gradient shown in equation.(23):

$$\nabla_\phi J(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\nabla_a Q_\theta(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)\right] \tag{23}$$

where Q_θ is the action-value function defined in Equation (23).

3 TD3 Training Setup

Table 2: Hyperparameters and Their Functions

Hyperparameter	**Value**	**Function**
batch_size	256	Size of minibatch used for training
Discount factor γ	0.9995	Discount factor for future reward
tau	0.005	Polyak averaging for soft updates of target networks
Policy_noise	0.2	Gaussian noise added to target actions
Noise_clip	0.1	Clipped noise to prevent overlarge action noise
Policy_freq	2	Frequency of actor update
Actor_hidden_sizes	600-400	Number of neurons in the two hidden layers of actor networks
Critic_hidden_sizes	600-400	Number of neurons in the two hidden layers of critic networks
Activation_function	ReLU + Tanh	ReLU used for hidden layers, Tanh used for actor output
Optimizer	Adam	Optimizer for both actor and critic
Learning rate	1e-4	Used Adam's default setting

3.1 Training Environment

3.1.1 Virtual Model

This thesis uses PyBullet as a VM simulation engine. The VM under the engine is a high-fidelity model with complex physical attributes of a quadruped robot, which can simulate the behavior of the robot in real environment by modifying the corresponding settings.

To create the Virtual model of robot in PyBullet environment, the simulation involves the following basic steps:

– **Building the 3D model of the robot**: The robot model is constructed from mechanical elements (blocks, rotating axes, joints) using CAD (Computer-Aided Design) models or available simulation tools.

– **Configuring joints and forces**: The robot's joints must be accurately configured to simulate movements of the legs, head, and body. Each joint can have parameters such as rotation angle, damping force, and actuation force, which affect the robot's dynamics.

– **Running the simulation and observing results**: Once the model is configured, control algorithms are applied to make the robot perform tasks such as walking, climbing, or navigating rough terrain. The simulation results can be analyzed to optimize performance and evaluate the robot's capabilities in a virtual environment.

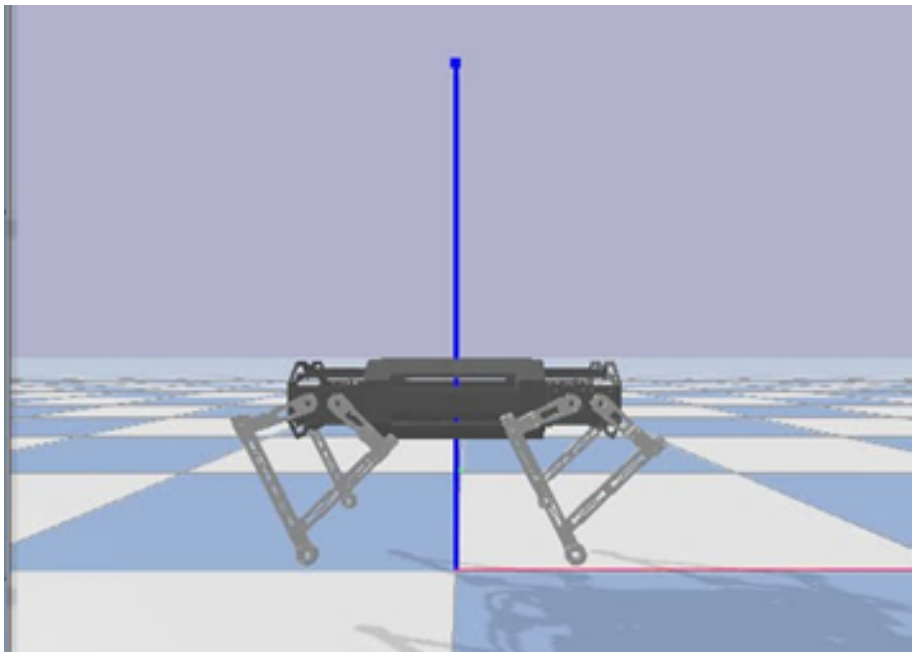

Figure 9: QTbot Virtual Model

3.1.2 Experiment Environment

In this project, I train Spot Dog to walk normally on a flat surface, on slope terrain $(5°, 9°)$. The terrain map is generated in PyBullet environment as shown in Figure.10 and Figure.11.

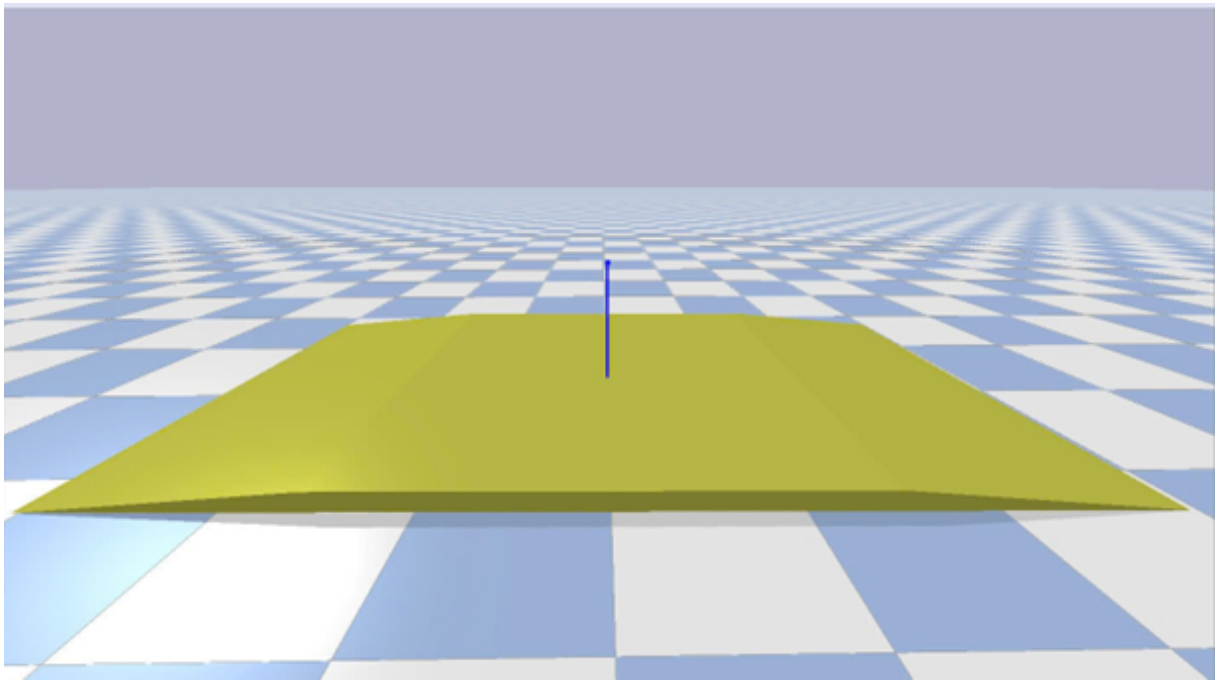

Figure 10: 5° slope

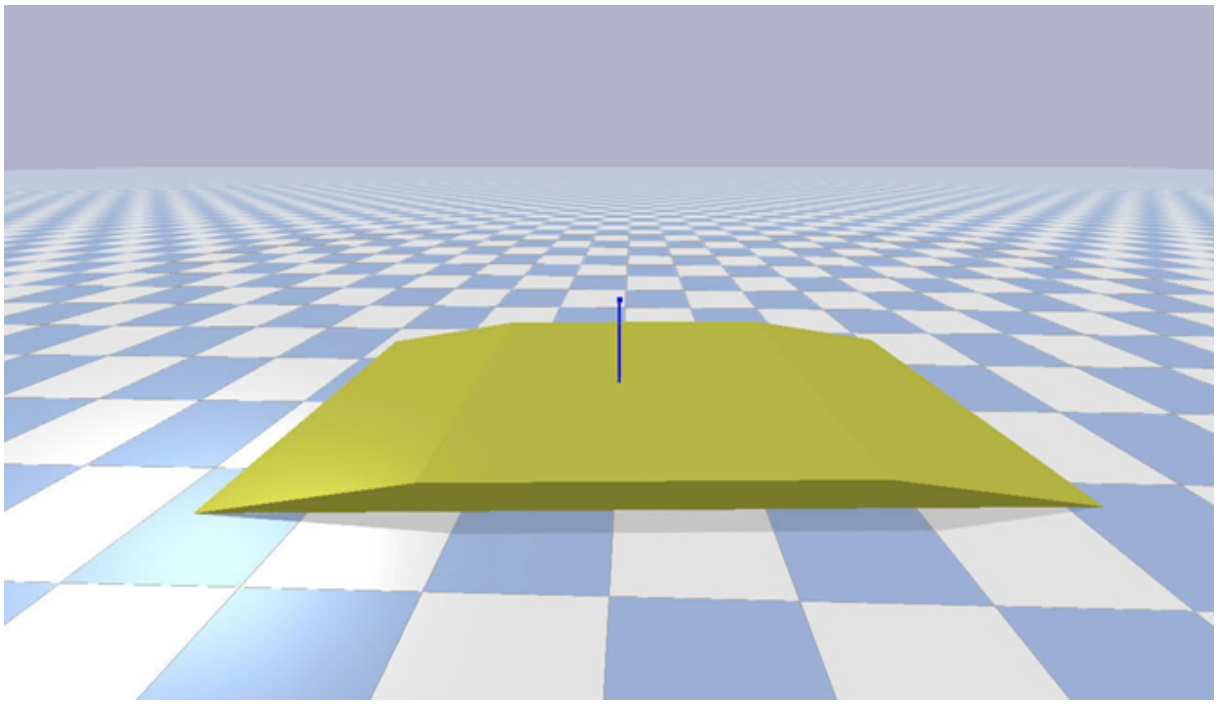

Figure 11: 9° slope

3.2 Observation space

The quadruped robot's observation space includes its base orientations and terrain slope parameters, which are essential to determine its state and make action decisions. Equipped with an IMU sensor, the robot measures base angles (roll, pitch, yaw) and terrain slope angles (roll, pitch) to operate effectively on various terrains. Overall, the observation space is defined as an 11-dimensional vector:

$$s_t = [\Phi_{t-2}, \Phi_{t-1}, \Phi_t, \gamma_t, \sigma_t] \tag{24}$$

where $\Phi_t \in \mathbb{R}^3$ represents the base orientations of the robot, γ_t and σ_t denote the roll and pitch angles of the sloped terrain at time step t. Observations of the robot's base orientations at time steps $t-2$ and $t-1$ are also crucial to improving the robot's performance.

3.3 Action Space

The robot's action space is a 12-dimensional vector that adjusts each leg's semi-elliptical trajectory for flexible and stable movement. It includes step-length parameters that affect speed and stability, as well as deviations along the x and z axes to enable direction changes and terrain adaptation.

$$a_t = [l_t, x_t, z_t] \tag{25}$$

here, $l_t, x_t, z_t \in \mathbb{R}^4$, where:

- l_t represents the step lengths for each of the four legs, which directly influence the forward motion and walking speed of the robot.
- x_t denotes the lateral deviations along the x-axis of the semi-elliptical trajectory, allowing for sideward adjustments or turning maneuvers.
- z_t encodes the vertical deviations along the z-axis, contributing to elevation changes that help the robot step over obstacles or adapt to uneven terrain.

These action components are carefully coordinated by the reinforcement learning policy to produce agile and energy-efficient walking patterns. The semi-elliptical trajectory generator, which interprets and converts these high-level action parameters into leg joint commands, plays a crucial role in translating the abstract policy output into precise foot trajectories. This approach allows for both robust stability and real-time adaptability in challenging environments.

3.4 Reward Function

The reward function used in this work is meticulously designed to quantify and guide the robot's behavior by evaluating its performance in terms of both movement efficiency and postural stability. These two dimensions are critical for successful locomotion, especially in unstructured or uneven terrains. In order to effectively shape the learning process, the reward function incorporates a combination of penalization terms for deviations in body orientation and a positive incentive for forward progression. Specifically, our reward function is formulated as follows:

$$r = G_{w_1}(e_R) + G_{w_2}(e_P) + G_{w_3}(e_Y) + G_{w_4}(e_H) + W\Delta x \tag{26}$$

In this expression:

- $e_R, e_P, e_Y, e_H \in R$ represent the instantaneous deviations in roll, pitch, yaw, and height (or elevation) of the robot's base body from a stable reference configuration.

- Each of the terms $G_{w_i}(e) = \exp(-w_i e^2)$ represents a Gaussian-shaped penalty function defined as:

  $$G_{w_i}(e) = \exp(-w_i e^2)$$

  with $w_i > 0$ controlling the sharpness of the penalty curve. This ensures that larger deviations are penalized more harshly, while small deviations are tolerated to promote smooth control.

- Δx is the forward distance traveled along the x-axis during the current time

step.

- W is a scalar weight that modulates the importance of translational progress relative to stability and orientation maintenance.

The inclusion of Gaussian-based penalties enables the reward function to softly enforce stability, making the learning process smooth and effective. It balances competing objectives: minimizing orientation and height deviations to maintain stability, while incentivizing the robot to make meaningful forward progress.

By optimizing this reward signal, the agent learns policies that exhibit both agile locomotion and robust stabilization. The structure of the reward is particularly well-suited to reinforcement learning in dynamic environments, where both precision and adaptability are required for successful navigation.

3.5 Actor-Critic Network

After setting up the training environment of QTbot, TD3 algorithm is implemented to train model. The Hyperparameters are shown in Table 2.

The actor neural network consists of two hidden layers with 600 and 400 hidden neurons respectively. It outputs continuous actions using a scaled tanh activation, and its parameters are optimized using the ADAM algorithm with a learning rate of 0.0001.

The critic neural network also contains two hidden layers of the same size (600 and 400 neurons). It is used to estimate the Q-values for the current state–action pairs. Similar to the actor, the critic's parameters are trained using the ADAM optimizer with the same learning rate.

A replay buffer containing 1,000,000 transitions is used to store past experiences. During training, samples are randomly drawn from this buffer to break correlations between sequential data and to improve training stability. The buffer size was chosen to ensure sufficient diversity in training data and to prevent overfitting.

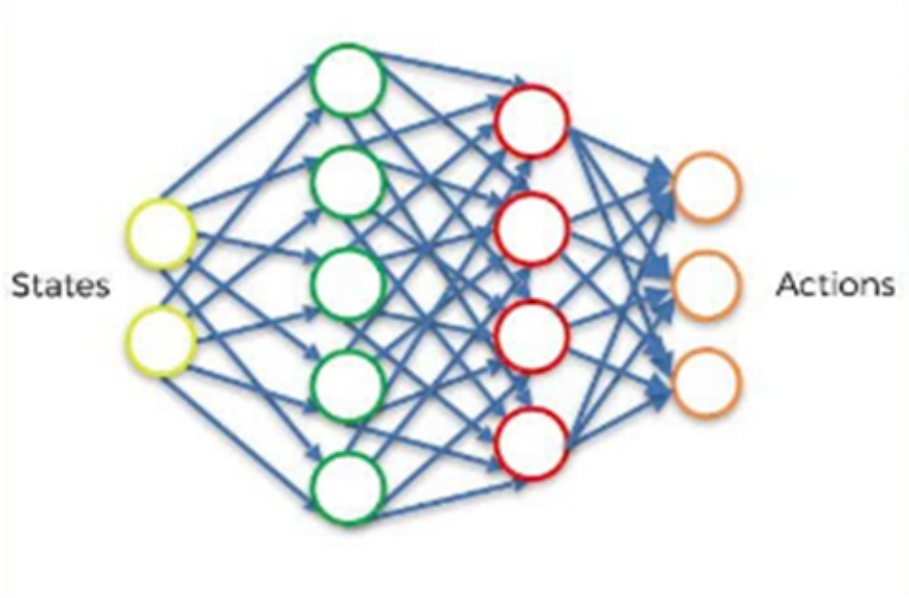

Figure 12: Actor network

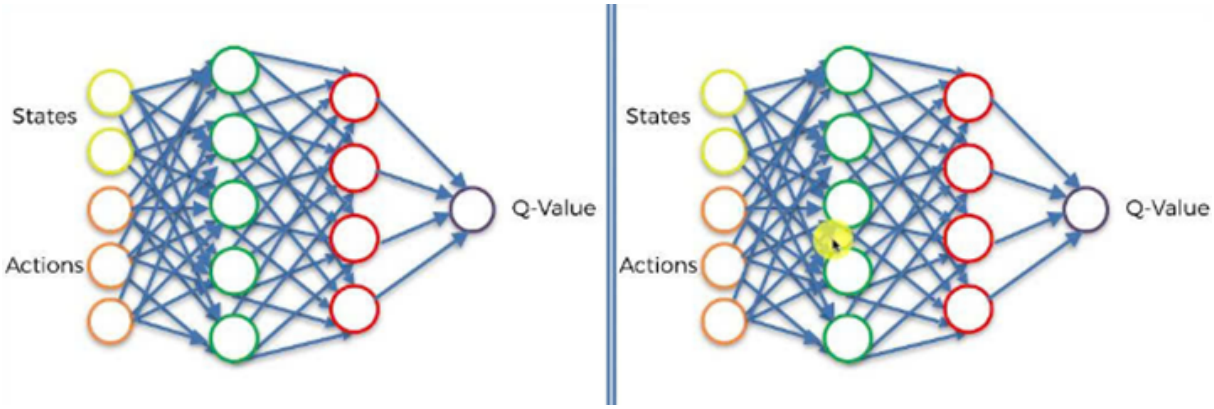

Figure 13: Double Critic network

4 System Design

The virtual-real interaction of the quadruped robot consist of three mains subsystem: Virtual Model (VM), Physical model(PE), and control system. The functions of each system and the connection between them are shown in Fig.14.

4.1 Physical Model

4.1.1 Components

1. **Nvidia Jetson Nano:**

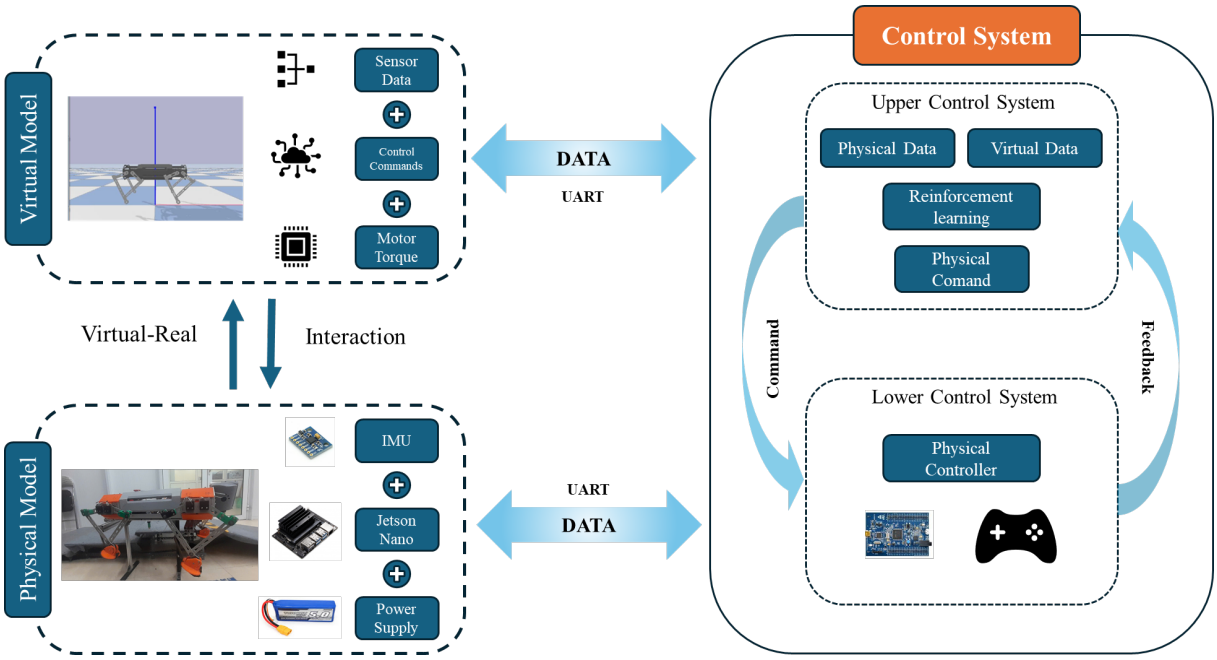

Figure 14: Virtual-Real Interaction Framework

The Jetson Nano is a small, powerful computer developed by NVIDIA for embedded AI applications. It features a quad-core ARM Cortex-A57 CPU and a 128-core Maxwell GPU, making it ideal for real-time image processing, computer vision, and deep learning tasks. With support for popular AI frameworks like TensorFlow, PyTorch, and OpenCV, the Jetson Nano enables developers to build and deploy intelligent robots, drones, and IoT devices efficiently. Its compact size, low power consumption, and affordable price make it a popular choice for both education and prototyping in AI and robotics.

In this project, Jetson Nano is the bridge between the Virtual Model and the Physical model close together. The trained policy will be directly deployed to Jetson Nano to control the physical model.

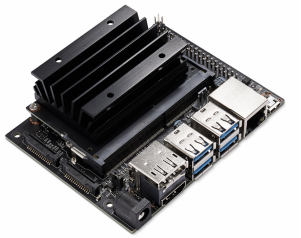

Figure 15: Jetson Nano

2. **IrobotLab Smart Driver:**

   Smart driver is the microcontroller driver developed by Irobot Lab. It features an STM32F103 chip and many IC circuit to generate PWM signal for the motor. This driver is an embedded PD control using angle as an input and PWM signal as the output.

Figure 16: Irobotlab Smart Driver

3. **Potentiometer:**

   The Potentiometer is used as an encoder to read the rotation angle of robot's joints. The joint rotation angle q_r is read by calculating the ADC value as follows:

$$q_r = 360^\circ \frac{R}{R_{max}} \tag{27}$$

   where R is the resistance ADC value and R_{max} is the potentiometer maximum resistance.

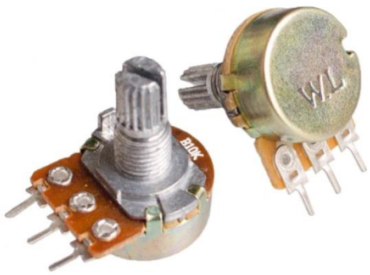

Figure 17: Potentiometer WH148

4. **IMU MPU6050 Module:**

The GY-521 6DOF IMU MPU6050 sensor is used to measure six parameters: three-axis angular rotation (Gyroscope) and three-axis directional acceleration (Accelerometer).

In this project, the IMU sensor MPU6050 is employed to measure the roll, pitch, and yaw angles of the Spot Dog robot, which are used as part of the observation space for reinforcement learning (RL) training. A Kalman Filter is applied to the raw sensor data to reduce the influence of measurement noise. However, the detailed implementation of the filtering process is beyond the scope of this report and is therefore not elaborated here.

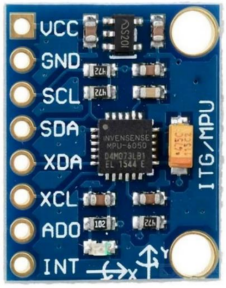

Figure 18: IMU MPU6050 compass IC

4.1.2 Communication

Table 3: Model Parameters

link	**Length**(mm)	**Mass**(kg)
Body	$L = 600$ $W = 400$	7
upper hip(l_1)	110	0.039
upper knee(l_2)	110	0.039
lower hip(l_1')	200	0.053
lower knee(l_2')	250	0.203

The experiment is tested on Spot Dog, developed by the IrobotLab team, HaUI. This is an 8-DoF robot with a five-bar mechanism leg, which can carry high loads. The detail parameters are in table.3:

The main communication technique used to transmit data between modules of the physical model is UART (Universal Asynchronous Receiver/Transmitter). The

angular signals are sent from the main board Jetson Nano to each driver. From the driver, it generate PWM to control the motor. The encoder reads the motor current angle and send an ADC signal to STM32F1, which then transmits to computer. The model's communication workflow is shown in Fig.19. The position and orientation of the body is read by IMU sensor and transmits back to main board and transmit to computer.

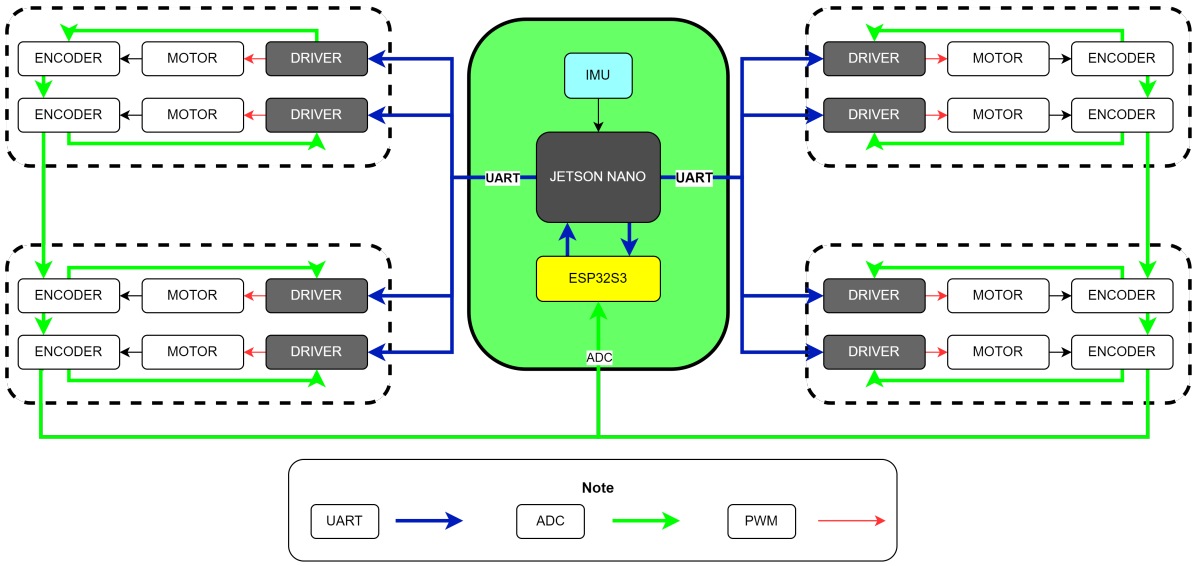

Figure 19: PM Communication Schematic

4.2 Virtual - Real Interaction

In this project, the interaction between the virtual and real environments plays a critical role in validating and enhancing the performance of the trained control policy. Once the reinforcement learning model is trained in simulation, the resulting policy is deployed onto the physical quadruped robot. This deployment allows the robot to execute complex locomotion behaviors learned virtually in a real-world setting.

To enable synchronous operation between the virtual and real models, we adopt a **Virtual-Real Interaction (VRI)** framework, where actions taken by the real robot are mirrored in a virtual simulation environment in real time. This two-way communication ensures that the simulation reflects the actual physical state of the

robot, enabling close monitoring, debugging, and further fine-tuning of the control strategy.

At the core of this interaction is the **Jetson Nano** board, which acts as a computational and communication bridge. It performs three key functions:

- **Policy Inference**: The Jetson Nano runs the trained policy locally and outputs control commands to the robot's actuators.
- **Sensor Data Acquisition**: It collects real-time sensor feedback (e.g., joint angles, IMU data, foot contact states) from the physical robot.
- **Virtual Synchronization**: The same control inputs and feedback data are streamed to the virtual model to ensure that the simulation environment reflects the real-time motion and condition of the hardware.

This synchronized execution not only allows for real-time visualization of the robot's performance in simulation but also provides a platform for identifying discrepancies between the virtual and physical behaviors. By aligning both environments, the system can detect failures, adapt to real-world uncertainties, and ultimately improve the robustness of the locomotion policy.

This virtual-real interaction loop is essential for achieving a reliable sim-to-real transfer and provides a foundation for future developments such as digital twins, remote diagnostics, and online learning.

5 Result & Analysis

5.1 Deep Reinforcement Learning Training results

All trainings were carried out with a mid-range desktop computer equipped with 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz, Nvidia GeForce GTX 1650 GPU, and 15GB RAM running Ubuntu 22.04.

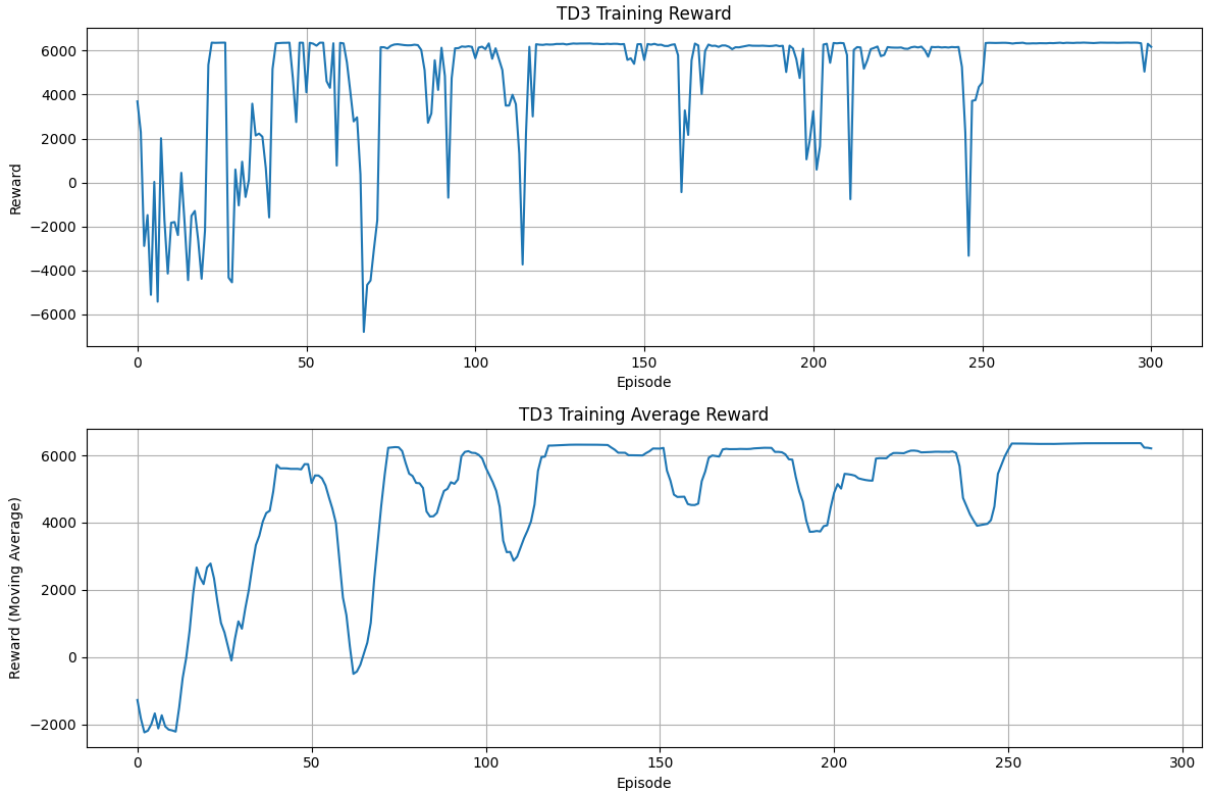

Figure 20: TD3 training reward on flat surface

The training performance of the quadruped robot on a flat terrain (0-degree slope) using the TD3 algorithm demonstrates a relatively rapid and stable learning process. As shown in the reward and moving average plots, the model initially exhibits high variance in reward values during early episodes, which is expected due to exploration. However, by around episode 100, the rewards begin to converge, and the moving average stabilizes at approximately 6000. This plateau indicates that the agent has successfully learned an effective policy for locomotion on flat terrain. Occasional drops in reward observed throughout training are likely

attributable to stochastic exploration noise or temporary policy degradation, but they do not significantly hinder the overall convergence. Overall, the results suggest the TD3 algorithm is effective in enabling the robot to achieve stable and efficient locomotion on level ground.

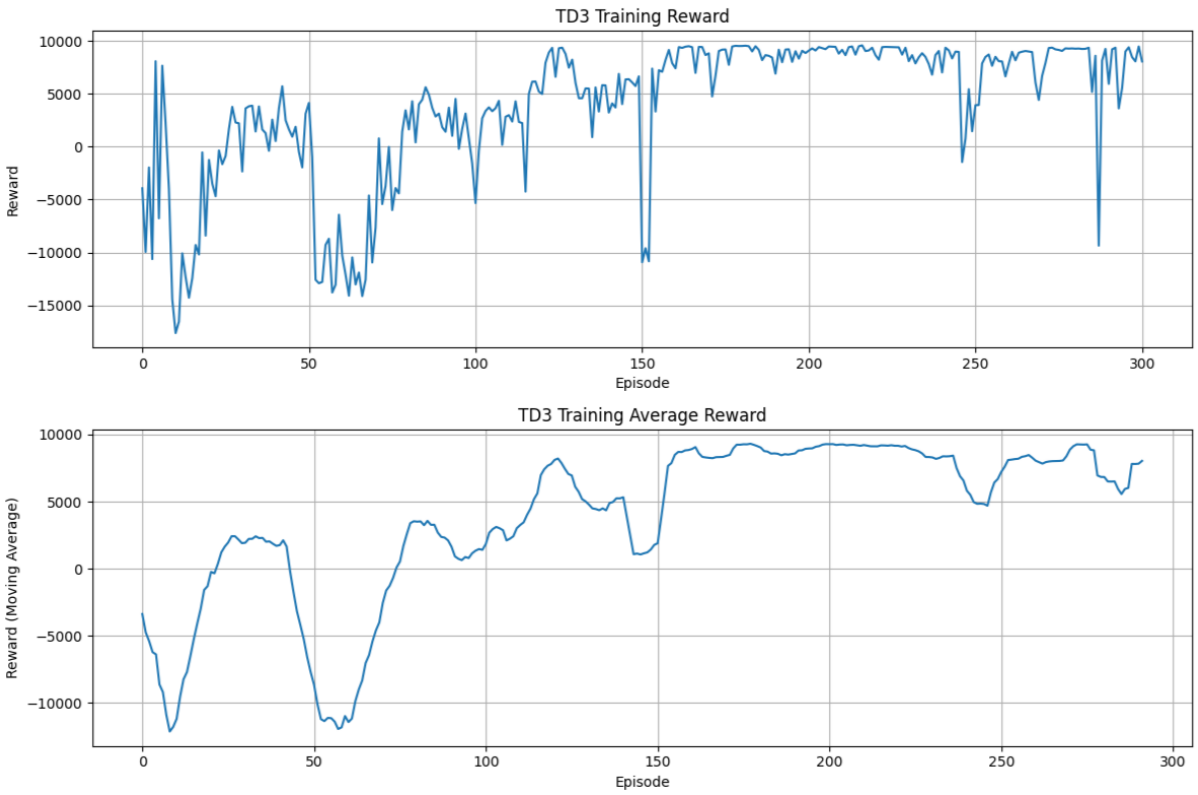

Figure 21: Reward after training Spot Dog walk on 5° slope

The training performance of the quadruped robot walking on a 5-degree slope using the TD3 algorithm is illustrated through Figure.21. In the initial episodes, the reward fluctuates significantly, indicating unstable learning and exploration behavior. However, as training progresses, both the instantaneous and moving average rewards exhibit a clear upward trend, stabilizing around episode 150. This suggests that the agent gradually learned an effective policy for navigating the sloped terrain. Despite occasional drops in reward, possibly due to exploration noise or environmental variations, the model consistently achieves high rewards in later stages, demonstrating successful convergence and adaptation to the 5-degree inclined surface.

The reward plots for training the robot to walk on a 9° slope using the TD3

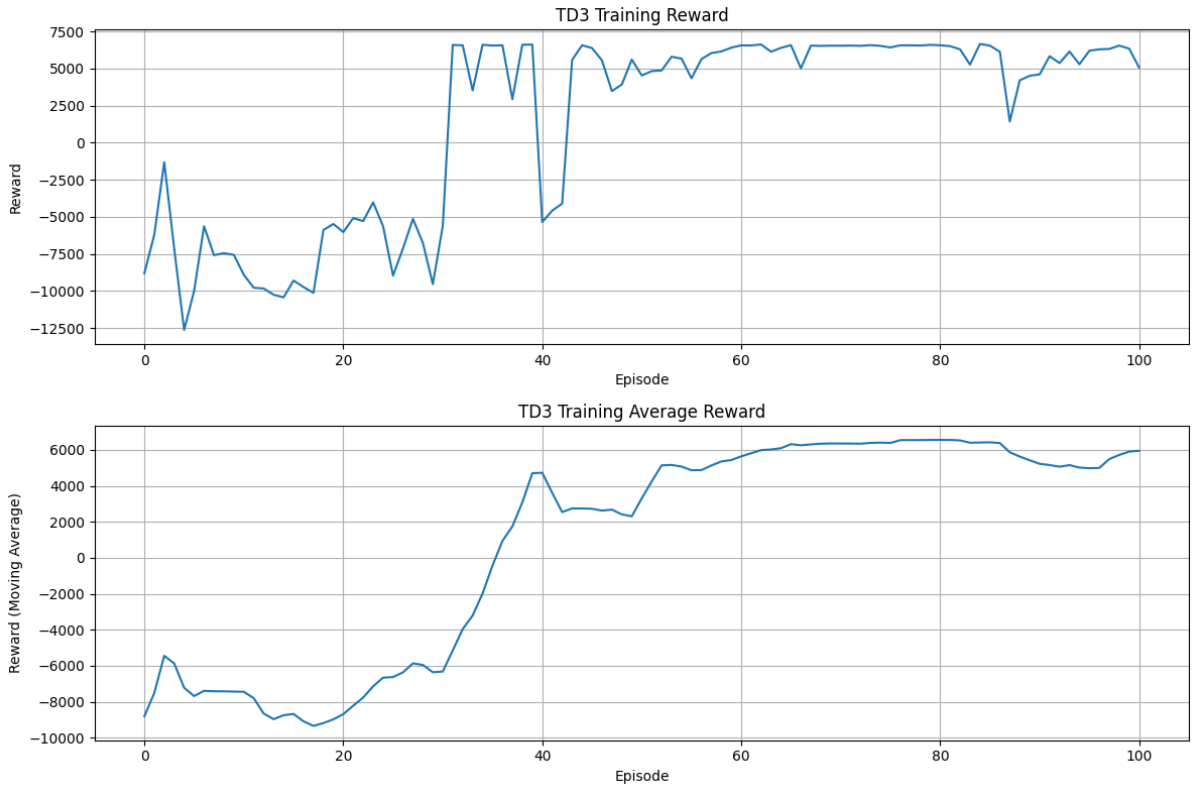

Figure 22: Reward after training Spot Dog walk on 9° slope.

algorithm indicate a significant learning progression over 100 episodes. In the top graph, which shows the total reward per episode, we observe high instability during the early training phase (episodes 0–40), with rewards fluctuating sharply between negative values (as low as −12,500) and occasional positive spikes. This reflects the robot's struggle to maintain balance and achieve forward locomotion on the inclined surface.

However, starting around episode 45, the reward consistently improves and stabilizes, indicating that the robot begins to learn an effective walking policy. In the bottom graph, which displays the moving average of the rewards, a clear upward trend is visible, with the average reward rising from around −9,000 to over 6,000. This consistent increase demonstrates that the TD3 policy gradually converged toward a successful locomotion strategy. Despite a slight drop in average performance around episodes 85–95, the policy quickly recovers, showing strong generalization and robustness in walking on a 9° incline.

The graph illustrates the roll and pitch angles of the robot during a single episode

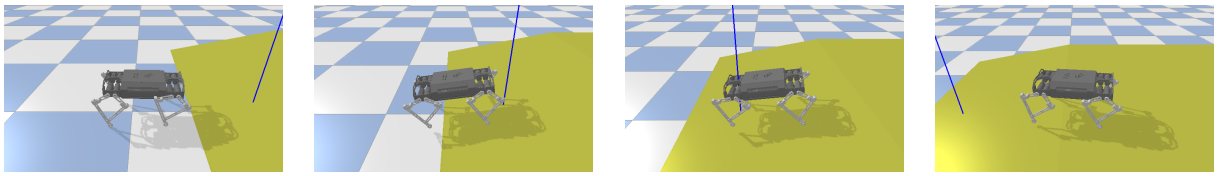

Figure 23: Model walking on a 5° slope after TD3 training.

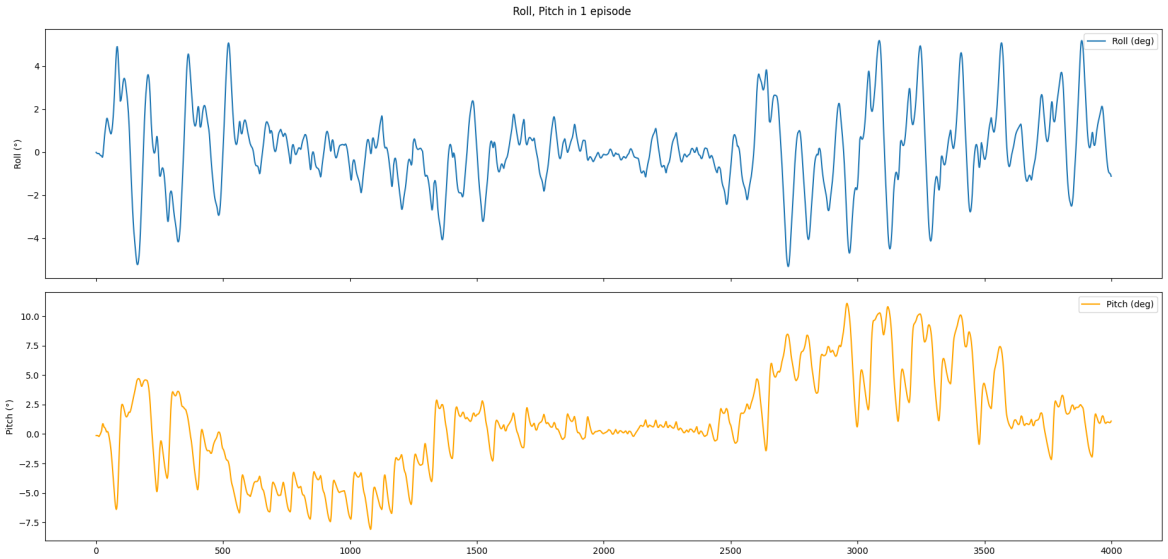

Figure 24: body roll-pitch angle

as it walks on a 5° slope, which includes three distinct phases: ascending the hill, walking on flat ground, and descending the hill. In the first phase (approximately steps 0–1300), the pitch angle increases steadily, peaking around 5° to 10°, indicating that the robot is tilting forward while climbing uphill. At the same time, the roll angle shows moderate oscillations between ±3°, reflecting the robot's lateral balance adjustments during ascent. In the second phase (around steps 1300–2600), both pitch and roll become more stable; the pitch fluctuates slightly around 0° to 2°, suggesting that the robot is walking on relatively level ground. The roll angle also remains close to 0°, indicating a stable lateral posture. In the final phase (steps 2600–end), the pitch angle first increases sharply and then decreases below zero (reaching down to approximately −8°), showing that the robot leans backward while descending. The roll angle exhibits larger oscillations, up to ±5°, suggesting that the robot exerts greater balancing effort due to the destabilizing effects of downhill movement. Overall, the robot maintains acceptable balance throughout

the episode, although the increased roll variability in the descent phase indicates potential areas for improvement in control strategy.

5.2 Digital Twin results

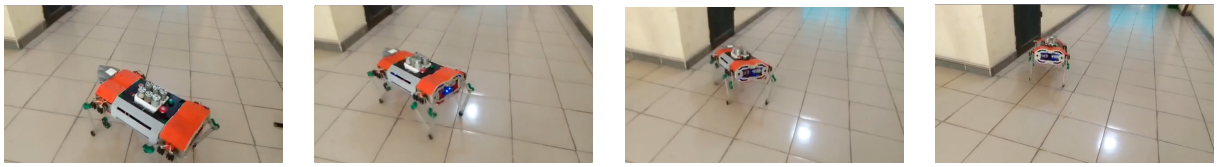

Figure 25: Normal walk test

After training in PyBullet, the trained policy was transferred to the physical quadruped robot. The policy was deployed on the Jetson Nano, where the generated actions were converted into joint angle control signals for the motors. The Jetson Nano monitored the robot's behavior through IMU sensor data transmitted via UART. The experimental results are shown in Figure.25. Based on Figure.25, the robot is capable of moving on a flat surface. However, it was observed that the robot can walk straight for a short distance before gradually veering to the right.

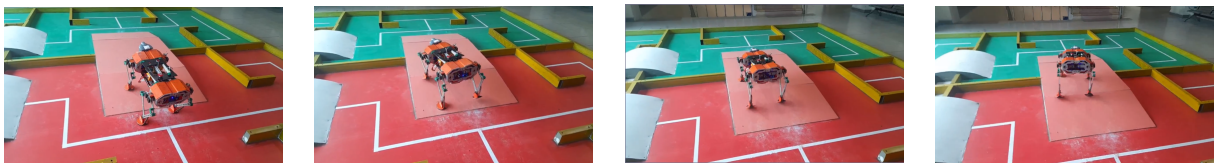

Figure 26: Spot Dog on 9° slope

Figure 26 illustrates the performance of the Spot Dog as it navigates a sloped surface with an inclination of 9°. In this experiment, the robot successfully ascended the slope within approximately 5 seconds, indicating the effectiveness of the trained policy when transferred to the physical environment. The results demonstrate that the robot is capable of maintaining balance and stability while adapting to changes in terrain elevation. This behavior suggests that the control policy, originally trained in simulation, has successfully generalized to real-world conditions, allowing the robot to compensate for gravitational effects and maintain coordinated leg movements during the climb. The ability to handle inclined surfaces is a critical

requirement for legged robots operating in unstructured or outdoor environments.

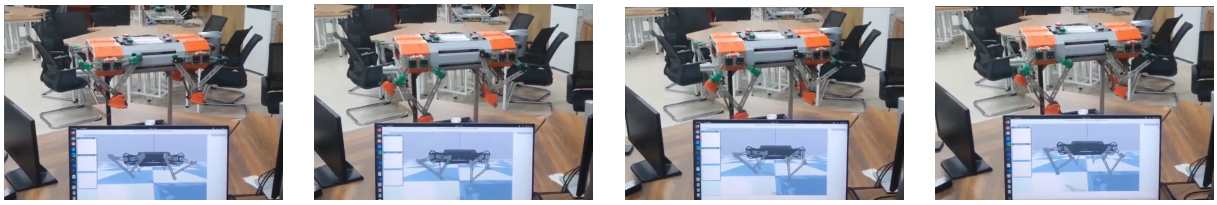

Figure 27: Virtual-Real parallel test

Figure 27 illustrates the Virtual-Real signal interaction when the robot is placed on the rack. From the figure, the angle signals of each motor are transmitted smoothly between the virtual and real models. However, due to a delay in receiving signals from the real model, the virtual model responds approximately 0.5 seconds slower.

Despite the promising results, several limitations of the study must be acknowledged:

1. Limitations in the physical robot's hardware have posed significant challenges to the successful deployment of the learned control policy. Issues such as mechanical imprecision, sensor inaccuracies, and integration constraints have prevented the policy from functioning optimally in the real-world setup.

2. Secondly, this project was primarily tested on a strictly controlled environment, which may not fully capture the complexity and variability of the real world complex terrains. As a result, the current locomotion policy and control system may not generalize well to more unpredictable environments. The absence of environmental variability during testing also limits the opportunity to evaluate the robot's robustness, adaptive capacity, and recovery mechanisms in the presence of external disturbances or terrain uncertainties.

3. In addition, the virtual model does not fully utilize the real world parameters. Therefore, this research still not close the reality gap between Virtual Model and Physical Model. Without full parameter utilization, the digital twin's role is reduced to a visualizer rather than a true co-pilot in the robot's operation.

6 Conclusion & Future Plan

Throughout this project, I successfully achieved the goal of designing, simulating, and controlling a quadruped robot, SpotDog, using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm for reinforcement learning. One of the primary accomplishments was the development of a complete robot model using the Unified Robot Description Format (URDF), which accurately represents the robot's mechanical structure, joint configuration, and dynamic properties. This URDF model served as the foundational blueprint for all subsequent simulations and control experiments.

In terms of locomotion, I implemented an elliptical gait generator capable of producing semi-elliptical foot trajectories. This enabled the robot to lift and place its legs in a smooth, natural manner, mimicking the movement of biological quadruped robots. The trajectory system was parameterized and directly controlled via the output of the learning policy, enabling adaptive adjustments to leg motion during training.

On the reinforcement learning side, the TD3 algorithm was fully implemented using the PyTorch deep learning framework. I constructed customized Actor and Critic networks tailored to the robot's continuous action space, and integrated key components, including twin Q-networks, a replay buffer for experience sampling, delayed actor updates, and target smoothing noise. These components were carefully tuned to handle the complexities of high-dimensional state-action spaces in a simulated robotics context.

A robust simulation environment was built in PyBullet, where the SpotDog robot was trained and tested across various terrains. I evaluated the robot's walking ability on flat surfaces and inclined planes with slopes of 0, 5, and 9 degrees. The learning outcomes showed that the robot could adapt its posture and gait to maintain balance and forward motion, even on sloped terrain. Furthermore, I logged and analyzed performance metrics, including reward progression, body orientation angles (roll, pitch, yaw), and joint stability.

The trained policy was successfully transferred from the simulation environment to the physical quadruped robot. Two testing scenarios were conducted to evaluate its performance: locomotion on a flat surface and traversal of a 5° inclined slope. The experimental results demonstrate that the digital policy exhibits effective control over the real robot, highlighting the feasibility and reliability of sim-to-real transfer in this application.

Future directions for this research include improving real-time feedback and accuracy expanding simulation capabilities to cover complex terrains and environments, and developing an intuitive monitoring interface for operators. Digital Twin-driven Reinforcement Learning will also be applied in predictive movements, maintenance scenarios and multi-robot cooperation systems. These developments will open a broad field of research in robotics.

References

[1] Shunyuan Li et al. "A hierarchical framework for quadruped locomotion based on reinforcement learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 9646–9653.

[2] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI]. URL: https://arxiv.org/abs/1802.09477.

[3] Jie Tan et al. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". In: *arXiv preprint arXiv:1804.10332* (2018).

[4] Ajay M Jain et al. "Hierarchical Reinforcement Learning for Quadruped Locomotion". In: *arXiv preprint arXiv:1905.08926* (2019).

[5] Xue Bin Yang et al. "Learning Vision-Guided Quadrupedal Locomotion End-to-End with Cross-Modal Transformers". In: *arXiv preprint arXiv:2107.03996* (2021).

[6] Hao Zhang et al. "MASQ: Multi-Agent Reinforcement Learning for Single Quadruped Robot Locomotion". In: *arXiv preprint arXiv:2408.13759* (2024).

[7] Lucas Smith et al. "Risk-Averse Quadrupedal Locomotion using Distributional Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 1234–1241.

[8] Yuzhu Sun et al. "Digital Twin-Driven Reinforcement Learning for Obstacle Avoidance in Robot Manipulators: A Self-Improving Online Training Framework". In: *arXiv preprint arXiv:2403.13090* (2024).

[9] Yixiong Du, Zhuang Liu, and Xuping Zhang. "Advancing Locomotion Control of a Quadruped Robot: Harnessing Hybrid Control Algorithms and Digital Twin Technology". In: *10th International Conference on Automation, Robotics, and Applications (ICARA)*. 2024, pp. 144–149.

[10] Amel Jaoua, Samar Masmoudi, and Elisa Negri. “Digital twin-based reinforcement learning framework: application to autonomous mobile robot dispatching”. In: *International Journal of Computer Integrated Manufacturing* 37.10-11 (2024), pp. 1335–1358.

[11] Nabeel Ahmad Khan Jadoon and Mongkol Ekpanyapong. “Quadruped Robot Simulation Using Deep Reinforcement Learning – A step towards locomotion policy”. In: *arXiv preprint arXiv:2502.16401* (2025).

[12] Hongnian Yu. “Modeling and control of hybrid machine systems—a five-bar mechanism case”. In: *International Journal of Automation and Computing* 3.3 (2006), pp. 235–243.

[13] Peng Li et al. “Path planning of mobile robot based on improved TD3 algorithm in dynamic environment”. In: *Heliyon* 10.11 (2024).