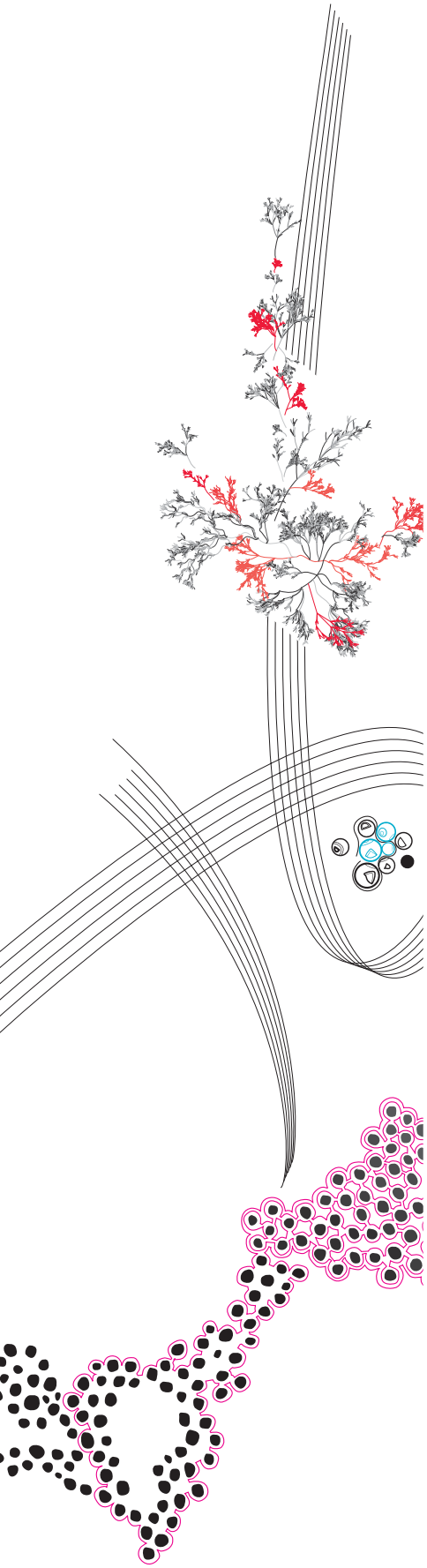
MSc Systems and Control
Final Project



Reinforcement learning control of unicycle robot

Shahab Farmehinifarahani

Supervisor:
Prof.dr.ing. B. Rosic
Dr.ir. R.G.K.M. Aarts

April 8, 2024

Applied Mechanics and Data Analysis (AMDA) group,
Faculty of Engineering Technology (ET),
University of Twente.

UNIVERSITY OF TWENTE.

Preface

It is with great pleasure that I present this thesis, which represents dedicated research and exploration into the field of reinforcement learning-based control. This journey has been both challenging and rewarding, filled with moments of discovery, growth, and insight. The motivation behind this work stemmed from a deep curiosity and passion for machine learning, coupled with a desire to contribute meaningfully to the existing body of knowledge in this domain.

Throughout the process, I have been fortunate to receive guidance, support, and encouragement from my supervisors, Prof.Dr.Ing. B. Rosic and Dr.Ir. R.G.K.M. Aarts, who have played pivotal roles in shaping this endeavor. Their guidance has been instrumental in steering this project towards fruition, and I want to express my gratitude for their invaluable mentorship and wisdom.

Additionally, I extend my thanks to my friends and family members who have offered their encouragement, understanding, and support throughout this journey.

As with any scholarly pursuit, this thesis is not without its limitations and areas for further exploration. It is my hope that the findings and insights presented herein will serve as a springboard for future research endeavours and contribute to the ongoing discourse within the field of reinforcement learning-based control.

Summary

Unicycle robots, distinguished by their single-wheel design, offer a significant advantage in manoeuvrability due to their compact footprint. Maintaining stability for these robots presents a unique challenge. While the driving wheel directly controls longitudinal stability, achieving lateral stability requires a more intricate approach. The Moment Exchange Unicycle Robot (MEUR) addresses this challenge with an innovative design. It utilises a motor-powered reaction wheel to generate a counteracting force, ensuring lateral stability.

This thesis delves into the application of reinforcement learning for achieving self-balancing control in the MEUR. Unlike traditional methods that rely on pre-defined models or hand-crafted rules, RL agents learn autonomously through repeated interactions with their environment. This "trial-and-error" learning process allows them to tackle complex and unpredictable control challenges. The ability to learn from experience and adapt to changing environments makes RL a compelling choice for various robotics applications, and the MEUR serves as a compelling case study in this regard.

Through this research, the effectiveness of two prominent algorithms, namely Deep Q-learning (DQN) and Advantage Actor-Critic (A2C), is showcased in attaining stability across both roll and pitch angles of the MEUR. Building upon this foundation, the research further investigates the ability of these algorithms to control the MEUR's movement in the longitudinal direction (forward and backward) while maintaining stability. Finally, the ability of these RL-based controllers to deal with noise and uncertainties is discussed.

Contents

Chapter 1

Introduction

Unicycles, characterised by a single wheel demanding delicate balance, have long fascinated riders for their unique blend of simplicity and challenge (Figure 1.1a). As technology advances, unicycle design has evolved, giving rise to unicycle robots that mimic the agility and balance of human riders (Figure 1.1b). These machines utilise sensors, complex algorithms, and intricate control systems to maintain balance and navigate environments. Compared to two-wheeled robots, unicycle robots require more intricate control due to the need for stabilisation in both lateral and longitudinal directions. However, their advantage lies in a smaller footprint, which offers superior manoeuvrability, especially in narrow environments. Their lightweight construction enhances nimbleness, enabling fast and precise movement. Such characteristics are practical in various sectors, such as food delivery, maintenance of machines in factory halls, or inspection of pipelines.

(A) Pedal powered unicycle [1]

(B) Unicycle robots [2]

FIGURE 1.1: Examples of unicycles

Unicycle robots have to be designed carefully in order to optimally perform the previously described tasks. For example, they should proficiently track trajectories while maintaining stability. Furthermore, the unicycle should adapt to varying payloads that could be

imposed during its use or to the changes in both the lateral and longitudinal slopes of the road during navigation.

Among unicycle robots that meet the aforementioned requirements, the Moment Exchange Unicycle Robot (MEUR) stands out with its unique design (Figure 1.2). It comprises a body, a reaction wheel powered by a motor to maintain lateral stability by generating a counteracting force, and a wheel driven by a separate motor to move the body and ensure longitudinal stability. Equipped with only two motors, the robot lacks a direct force to alter its yaw direction. Although the mechanical design of this robot is promising, the challenging task is to maintain and precisely control the robot's balance and movement.

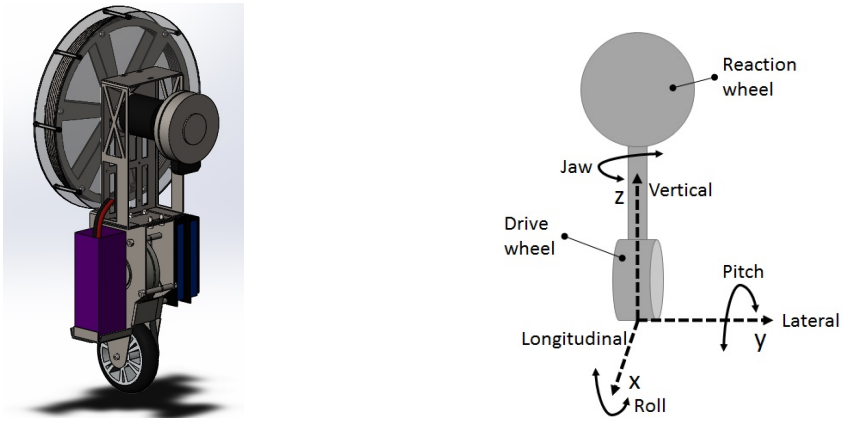


FIGURE 1.2: The Moment Exchange Unicycle Robot (MEUR) [3]

Various approaches can be employed in designing controllers for the MEUR, typically falling into three main categories: traditional linear control techniques, nonlinear control strategies, and more recently, reinforcement learning-based approaches.

Linear control techniques offer a simplified approach by linearizing the MEUR's dynamic equations around its upright configuration. This essentially treats the system as a collection of independent subsystems, making it easier to derive the dynamic equations and design the controller. This linearized system modeling approach has been successfully employed in various studies. Specifically, in [4], sliding mode control and feedback linearization are used to preserve balance during forward movement over a predetermined distance. In [5], the same method is used for the roll to minimize switching-function chattering. Yet, for pitch control, a Linear Quadratic Regulator (LQR) controller is implemented, steering the unicycle robot toward following the desired velocity trajectory. In [6], a fuzzy-sliding mode controller is adopted to uphold balance during forward motion across ramp and ladder displacement profiles. This study considers the impact of coupled terms within the pitch and roll axes as disturbances. Furthermore, in pursuit of real-time speed control, a strategy employing both sliding-mode control and a nonzero set-point linear quadratic regulator (LQR) is introduced in [7]. This hybrid approach is designed to ensure stability and sustain the desired speed tracking performance. The validation process encompasses more intricate speed profiles, including step velocity and trapezoidal velocity. In [8], a PID controller is employed to regulate the rotation motors

situated on the reaction wheel and wheel. This control scheme is informed by pitch and roll feedback from IMU sensors and rotary encoders. This research demonstrates the unicycle's ability to regain stability when starting from an initially unstable state or when encountering external disturbances. The outcomes illustrate the robot's capability to recover, displaying a recuperation span of ± 23 degrees in the pitch angle and ± 3.5 degrees in the roll angle. Additionally, both pitch and roll exhibit a steady-state error of merely 0.1 degrees, affirming the effectiveness of the control mechanism. Conclusively, in [9], a control method is presented that utilizes the interconnectedness of motions in the yaw, pitch, and roll directions to achieve yaw angle control. The paper demonstrates that yaw angle manipulation can be achieved by modulating pitch and roll postures. The proposed control strategy effectively guides the unicycle robot's yaw angle towards zero from any initial orientation. However, it does not delve into the trajectory tracking problem.

Taking into account the interdependence of nonlinear system dynamics introduces an added layer of complexity to controller design. In this context, [10] tackles the challenge by deriving nonlinear equations and subsequently implementing integral LQR and integral sliding mode controllers.

The aforementioned studies have predominantly employed classical control techniques to address unicycle balancing. Nonetheless, in recent times, the adoption of reinforcement learning (RL) for learning control policies has gained momentum. Compared to traditional control techniques that rely on meticulously crafted mathematical models of the environment, RL offers a refreshing alternative for complex control problems. This is particularly valuable in real-world scenarios where creating a perfect model can be difficult or even impossible. Instead of relying on pre-defined models, RL agents learn through a process of interacting with their environment. By exploring and exploiting different actions, they discover effective control strategies without needing a complete understanding of the system's underlying dynamics [11]. This data-driven approach allows RL to be highly adaptable in dynamic environments. Unlike traditional methods that require re-training for every change, RL agents continuously update their control policies based on new experiences. This adaptability extends to unforeseen situations, as RL agents can learn from unexpected consequences and adjust their behaviour to maintain stability. These strengths, coupled with the ability to optimise complex behaviours in high-dimensional state spaces, make RL a compelling choice for a wide range of robotics applications in the ever-changing world around us.

In [12], the policy gradient method is explored to achieve balance in MEUR, substantiating its outcomes through simulation. According to the paper findings, the controller demonstrates satisfactory performance in the lateral balancing of the MEUR. However, oscillatory behavior is observed in longitudinal balancing. Notably, the controller's performance under conditions where the MEUR is in motion on sloped terrain or when carrying additional loads is not investigated.

Similar to unicycle robots, bicycle robots are inherently unstable in the lateral direction, necessitating sophisticated control mechanisms to maintain balance and navigate efficiently. The conventional approaches to balancing bicycle robots can be divided into two primary categories. The first category solely utilizes steering and velocity as inputs, while the second category employs auxiliary balancing mechanisms, such as reaction wheels, to

achieve lateral balance. This approach is analogous to the balancing strategy employed in MEUR. In [13] and [14], a controller for bicycles utilizing the DDPG (Deep Deterministic Policy Gradient) algorithm is introduced. In these papers, it is demonstrated that the bicycle can achieve stable balance and navigate to designated locations. In addition, in [15], an innovative approach is presented integrating online serial-parallel RL with conventional control techniques. This unique strategy enables path tracking and balance control for a reaction wheel bicycle robot on curved pavements.

While existing research has demonstrated promising advancements in enhancing MEUR maneuverability, balancing, and trajectory tracking, the full potential of RL in transforming the MEUR into an efficient delivery drone remains largely unexplored, calling for further investigation. Specifically, several critical aspects warrant further exploration. For instance, ensuring stability in both lateral and longitudinal directions while tracking trajectories and effectively handling noise and external disturbances has not been adequately investigated. To address these limitations and fully explore the potential of RL in developing a controller for the MEUR across diverse scenarios, this study aims to answer the following research questions:

- Can the MEUR be effectively controlled with RL methods to maintain both lateral and longitudinal stability across a range of initial conditions?
- Can the MEUR navigate a specific distance in either forward or backward directions while consistently maintaining stability?
- Can the RL-based controller effectively handle measurement noise, load disturbances, or external disturbances without compromising the MEUR's stability and maneuverability?
- Does controlling the MEUR with RL methods make it suitable for performing delivery tasks in real-world scenarios?

Guided by research questions, this research evaluates the performance of RL as a control strategy for unicycle robots. The practicality of this approach is tested in a simulated environment. This study aims to provide valuable insights into controlling unicycle robots, especially through the innovative use of RL techniques.

This thesis delves into a detailed investigation through five chapters. The theoretical foundation is established in Chapter 2, which details the mathematical model of the MEUR and provides an overview of RL-based control strategies. Chapter 3 delves into the methodology, specifically focusing on the implementation of two prominent RL algorithms: Deep Q-network and Advantage Actor-Critic. This chapter also details the design of the reward functions used for training the RL agents and implementation methods. Chapter 4 presents the results obtained from applying these algorithms to control the MEUR, showcasing their effectiveness in achieving stability. Finally, Chapter 5 concludes the thesis with a summary of the findings, along with recommendations for future research directions.

Chapter 2

Theoretical Background

The subject of this thesis delves into the application of RL to design an effective controller for the MEUR, building upon a previous student project at the University of Twente [16, 17]. The primary objective is to stabilize the MEUR around its upright position and prevent it from falling while in motion. The robot employs its drive wheel for longitudinal stabilization and a substantial reaction wheel for lateral stabilization, following the axis definitions presented in Figure 1.2. Accurately controlling the unicycle requires real-time measurements of its motion parameters. The rotations of the reaction wheel and drive wheel are readily determined using the encoders integrated with the motors, while three analog gyroscopes and a triaxial accelerometer provide precise measurements of the unicycle's pitch, roll, and yaw angles with adequate processing. In [3], the sensor setup and the algorithms used to extract these vital parameters from the raw sensor data are detailed, ensuring accurate and reliable information for controller operation.

In the context of RL control for a unicycle robot, the mathematical model plays a dual role. Firstly, it provides a simulated environment for the RL agent to interact with. Secondly, the model's state variables and control inputs become the core elements of the RL agent's learning process. The next section details the mathematical model of the MEUR, focusing on its state variables, which represent the system's current state, and the control inputs, which the RL agent can manipulate.

2.1 Mathematical Model

To simplify the dynamic analysis, the unicycle robot is modeled as a compound object consisting of two sub-systems: the reaction wheel pendulum and the inverted pendulum. The reaction wheel pendulum comprises the reaction wheel and the robot body (Figure 2.1), while the inverted pendulum consists of the robot body and the driving wheel (Figure 2.2). The terms and parameter values depicted in Figures 2.1 and 2.2 are further elucidated in Table 2.1.

Given the assumption that the unicycle remains in proximity to its upright position and experiences only minor angles, the unicycle's lateral and longitudinal dynamics can be treated as decoupled systems [3, 7, 10]. By further assuming rigid bodies, no slippage, and

frictionless operation, the mathematical models become simpler. The non-ideal behavior resulting from these assumptions can be treated as perturbations to the system and addressed in an appropriate manner.

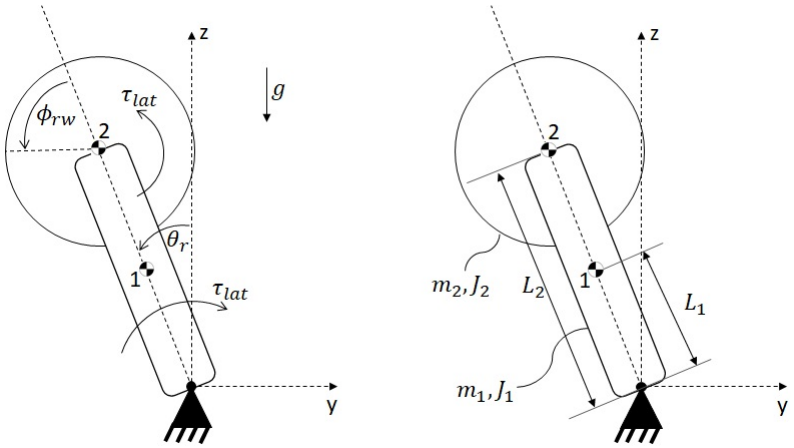


FIGURE 2.1: Reaction wheel pendulum: The lateral model [3]

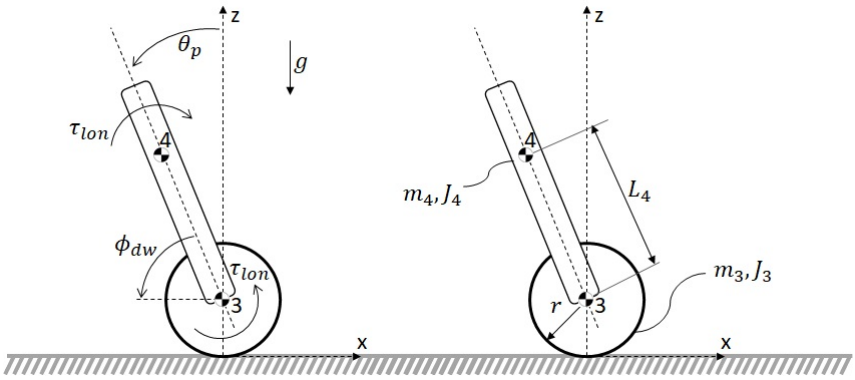


FIGURE 2.2: Inverted pendulum: The longitudinal model [3]

Symbol	Description	Value
m_1	Drive wheel mass	5.5 kg
m_2	Reaction wheel mass	1.5 kg
m_3	Frame mass	0.115 kg
J_1	Lateral frame inertia	0.086 kgm^2
J_2	Reaction wheel inertia	0.017 kgm^2
J_3	Drive wheel inertia	0.0001 kgm^2
J_4	Longitudinal frame inertia	0.122 kgm^2
L_1	Height of the center of mass (without reaction wheel)	0.18 m
L_2	Reaction wheel height	0.316 m
L_4	Height of the center of mass(with reaction wheel)	0.21 m
r	Drive wheel radius	0.05 m

TABLE 2.1: Values and descriptions of parameters [3]

2.1.1 Dynamic model for roll axis

In this section, the dynamic equations governing the unicycle robot's motion are derived using the Lagrange formulation written as:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i, \tag{2.1}$$

where L is the Lagrangian, which is the difference between the kinetic energy T and the potential energy V of the system. q_i are the generalized coordinates, which are variables that describe the configuration of the system. $\dot{q}_i$ are the generalized velocities, which are derivatives of q_i with respect to time, and Q_i are the generalized forces, which are the forces acting on the system. By substituting $T - V$ instead of L, the equation can be written in the following form:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} - \frac{d}{dt}\left(\frac{\partial V}{\partial \dot{q}_i}\right) + \frac{\partial V}{\partial q_i} = Q_i. \tag{2.2}$$

Considering Figure 2.1, the independent coordinates can be written as:

$$q_i = \{\theta_r, \phi_{rw}\}, \tag{2.3}$$

where θ_r is the absolute roll angle and ϕ_{rw} is the relative reaction wheel angle. Moreover, the generalized force is as follows:

$$Q_i = \{0, \tau_{lat}\}. \tag{2.4}$$

The kinetic energy of the unicycle robot consists of two components: translational kinetic energy and rotational kinetic energy. As a result, the total kinetic energy T of the robot is expressed as:

$$T = \frac{1}{2}(m_1 L_1^2 + J_1)\dot{\theta_r}^2 + \frac{1}{2}m_2 L_2^2 \dot{\theta_r}^2 + \frac{1}{2}J_2(\dot{\theta_r}^2 + \dot{\phi}_{rw}^2 + 2\dot{\theta}_r\dot{\phi}_{rw}), \tag{2.5}$$

where $\dot{\theta}_r$ and $\dot{\phi}_{rw}$ are roll angular velocity and reaction wheel angular velocity, respectively. The potential energy V of the unicycle robot is associated with the gravitational force acting on the robot's center of mass. It is given by:

$$V = m_1 g L_1 \cos(\theta_r) + m_2 g L_2 \cos(\theta_r). \tag{2.6}$$

The independent coordinates can be written as:

$$q_i = \{\theta_r, \phi_{rw}\}. \tag{2.7}$$

And the generalized force is as follows:

$$Q_i = \{0, \tau_{lat}\}. \tag{2.8}$$

By calculating the partial derivatives and performing substitutions, the Lagrange equation can be rewritten to derive the equations of motion as follows:

$$\begin{bmatrix} m_1L_1^2 + J_1 + m_2L_2^2 + J_2 & J_2 \\ J_2 & J_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_r \\ \ddot{\phi}_{rw} \end{bmatrix} + \begin{bmatrix} -(m_1L_1 + m_2L_2)g \ \sin(\theta_r) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_{lat} \end{bmatrix}. \quad (2.9)$$

To eliminate coupling in the equations of motion, one can rewrite the bottom line of equation 2.9 and then substitute it into the equations of motion.

$$J_2\ddot{\theta}_r + J_2\ddot{\phi}_{rw} = \tau_{lat} \rightarrow \ddot{\phi}_{rw} = \frac{\tau_{lat} - J_2\ddot{\theta}_r}{J_2} = \frac{1}{J_2}\tau_{lat} - \ddot{\theta}_r \quad (2.10)$$

$$(m_1L_1^2 + J_1 + m_2L_2^2 + J_2)\ddot{\theta}_r + J_2(\frac{\tau_{lat}}{J_2} - \ddot{\theta}_r) - (m_1L_1 + m_2L_2)g \ \sin(\theta_r) = 0 \quad (2.11)$$

By neglecting higher-order terms and considering that $\sin(\theta_r) \approx \theta_r$ for small angels, the linearized dynamic equation around the upright position is given by:

$$\ddot{\theta}_r = -\frac{1}{m_1L_1^2 + J_1 + m_2L_2^2}\tau_{lat} + \frac{(m_1L_1 + m_2L_2)g}{m_1L_1^2 + J_1 + m_2L_2^2} \ \theta_r, \quad (2.12)$$

$$\ddot{\phi}_{rw} = (\frac{1}{m_1L_1^2 + J_1 + m_2L_2^2} + \frac{1}{J_2})\tau_{lat} - \frac{(m_1L_1 + m_2L_2)g}{m_1L_1^2 + J_1 + m_2L_2^2} \ \theta_r. \quad (2.13)$$

Finally, the state space equation for roll axis dynamics is given by:

$$\begin{bmatrix} \dot{\theta}_r \\ \ddot{\theta}_r \\ \dot{\phi}_{rw} \\ \ddot{\phi}_{rw} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(m_1L_1+m_2L_2)g}{m_1L_1^2+J_1+m_2L_2^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{(m_1L_1+m_2L_2)g}{m_1L_1^2+J_1+m_2L_2^2} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_r \\ \dot{\theta}_r \\ \phi_{rw} \\ \dot{\phi}_{rw} \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{m_1L_1^2+J_1+m_2L_2^2} \\ 0 \\ \frac{1}{m_1L_1^2+J_1+m_2L_2^2} + \frac{1}{J_2} \end{bmatrix} \tau_{lat} \quad (2.14)$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_r \\ \dot{\theta}_r \\ \phi_{rw} \\ \dot{\phi}_{rw} \end{bmatrix}. \quad (2.15)$$

The specific goals of the unicycle robot's lateral control system are as follows:

- Maintain stability: The roll angle (θ_r) should be maintained within a range of $[-\theta_{r,max}, \theta_{r,max}]$, where $\theta_{r,max}$ is a predefined threshold value.

- Minimize roll angular velocity: The roll angular velocity ($\dot{\theta}_r$) should be kept as low as possible to prevent excessive rotational motion and enhance stability, ideally within a range of $[-\dot{\theta}_{r,max}, \dot{\theta}_{r,max}]$, where $\dot{\theta}_{r,max}$ is a predefined threshold value.
- Constrain reaction wheel velocity: The absolute value of reaction wheel velocity ($|\dot{\phi}_{rw}|$) should not exceed a specified threshold value of $\dot{\phi}_{rw,max}$, which is determined by safety considerations.

It is also worth mentioning that the motor torque (τ_{lat}) is the input to the system and is constrained to a specific range due to motor limitations and specifications.

2.1.2 Dynamic model for pitch axis

The same approach is used to write equations of motion for pitch angle. Considering Figure 2.2, the independent coordinates can be written as:

$$q_i = \{\theta_p, \phi_{dw}\}, \tag{2.16}$$

where θ_p is the absolute angle while ϕ_{dw} is the relative angle. Also, the generalized force is as follows:

$$Q_i = \{0, \tau_{lon}\}. \tag{2.17}$$

The total kinetic energy T of the robot is expressed as:

$$\begin{aligned} T = &\frac{1}{2}(m_3 r^2 + J_3)(\dot{\phi}_{dw} + \dot{\theta}_p)^2 \\ &+ \frac{1}{2} m_4 (r^2(\dot{\phi}_{dw} + \dot{\theta}_p)^2 + L_4^2 \dot{\theta}_p^2 + 2rL_4(\dot{\phi}_{dw} + \dot{\theta}_p)\dot{\theta}_p \ \cos(\theta_p)) + \frac{1}{2} J_4 \dot{\theta}_p^2. \end{aligned} \tag{2.18}$$

And the potential energy V is given by:

$$V = m_3 gr + m_4 gr + m_4 g L_4 \ \cos(\theta_p). \tag{2.19}$$

By calculating the partial derivatives and performing substitutions, the Lagrange equation can be reformulated to derive the equations of motion as follows:

$$\begin{aligned} &\begin{bmatrix} J_{eq,lon} & (J_{eq,lon} + m_4 r L_4 \ \cos(\theta_p)) \\ (J_{eq,lon} + m_4 r L_4 \ \cos(\theta_p)) & (J_{eq,lon} + m_4 L_4^2 + J_4 + 2 m_4 r L_4 \ \cos(\theta_p)) \end{bmatrix} \begin{bmatrix} \ddot{\phi}_{dw} \\ \ddot{\theta}_p \end{bmatrix} \\ &\qquad + \begin{bmatrix} -m_4 r L_4 \dot{\theta}_p^2 \ \sin(\theta_p) \\ -m_4 r L_4 \dot{\theta}_p^2 \ \sin(\theta_p) - m_4 g L_4 \ \sin(\theta_p) \end{bmatrix} = \begin{bmatrix} \tau_{lon} \\ 0 \end{bmatrix}. \end{aligned} \tag{2.20}$$

where:

$$J_{eq,lon} = m_3 r^2 + m_4 r^2 + J_3. \tag{2.21}$$

To eliminate coupling in the equations of motion, one can rewrite the top line of equation 2.20 and then substitute it into the equations of motion.

$$\ddot{\phi}_{dw} = \frac{1}{J_{eq,lon}}\tau_{lon} - \frac{J_{eq,lon} + m_4 r L_4 \ \cos(\theta_p)}{J_{eq,lon}}\ddot{\theta}_p + \frac{m_4 r L_4 \dot{\theta}_p^2 \ \sin(\theta_p)}{J_{eq,lon}}. \tag{2.22}$$

$$\begin{aligned}&\frac{J_{eq,lon} + m_4 r L_4 \cos(\theta_p)}{J_{eq,lon}}\tau_{lon} - \frac{(J_{eq,lon} + m_4 r L_4 \cos(\theta_p))^2}{J_{eq,lon}}\ddot{\theta}_p + \frac{J_{eq,lon} + (m_4 r L_4 \dot{\theta}_p)^2 \cos(\theta_p)\sin(\theta_p)}{J_{eq,lon}} \\ &+ (J_{eq,lon} + m_4 L_4^2 + J_4 + 2 m_4 r L_4 \cos(\theta_p))\ddot{\theta}_p - m_4 L_4 (r\dot{\theta}_p^2 \sin(\theta_p) + g\sin(\theta_p)) = 0.\end{aligned} \tag{2.23}$$

The linearized dynamic equation around the upright position for pitch axis dynamics, in state space equation form, is derived by neglecting higher-order terms and approximating $\cos(\theta_p) \approx 1$ and $\sin(\theta_p) \approx \theta$ for small angels.

$$\begin{bmatrix} \dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\phi}_{dw} \\ \ddot{\phi}_{dw} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{m_4 g L_4}{J_{eq,lon2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{(J_{eq,lon} + m_4 r L_4) m_4 g L_4}{J_{eq,lon} J_{eq,lon2}} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_p \\ \dot{\theta}_p \\ \phi_{dw} \\ \dot{\phi}_{dw} \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{J_{eq,lon} + m_4 r L_4}{J_{eq,lon} J_{eq,lon2}} \\ 0 \\ \frac{(J_{eq,lon} + m_4 r L_4)^2}{J_{eq,lon}^2 J_{eq,lon2}} + \frac{1}{J_{eq,lon}} \end{bmatrix} \tau_{lon} \tag{2.24}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_p \\ \dot{\theta}_p \\ \phi_{dw} \\ \dot{\phi}_{dw} \end{bmatrix} \tag{2.25}$$

The primary objective of the longitudinal control system is to maintain the stability and maneuverability of the unicycle robot by achieving the following goals:

- Maintain Pitch Stability: The pitch angle should be maintained within a specified range of $[-\theta_{p,max}, \theta_{p,max}]$, where $\theta_{p,max}$ is a predefined threshold value. This ensures that the unicycle robot remains upright and stable during its operation.
- Minimize Pitch Angular Velocity: The pitch angular velocity ($\dot{\theta}_p$) should be kept as low as possible, ideally within a range of $[-\dot{\theta}_{p,max}, \dot{\theta}_{p,max}]$, where $\dot{\theta}_{p,max}$ is a predefined threshold value. This minimizes unnecessary pitching motion and enhances stability.
- Control Driving Wheel Angle: The driving wheel angle (ϕ_{dw}), which is related to the unicycle displacement, should be maintained at a specified value to keep the unicycle robot following a specific trajectory.
- Constrain Driving Wheel Angular Velocity: The driving wheel angular velocity ($\dot{\phi}_{dw}$) should be maintained within a specified range of $[-\dot{\phi}_{dw,max}, \dot{\phi}_{dw,max}]$ where $\dot{\phi}_{dw,max}$ is a predefined threshold value.

It is also worth mentioning that the motor torque (τ_{lon}) is the input to the system and is constrained to a specific range due to motor limitations and specifications.

2.2 RL-based control

The previous section provided a detailed insight into the mathematical model of the MEUR, elucidating its dynamics and control parameters. While traditional control methods can be effective under controlled conditions, real-world robotics applications often encounter uncertainties and disturbances. Balancing a unicycle robot using pre-defined control strategies becomes particularly challenging in such scenarios. Furthermore, traditional control techniques rely heavily on meticulously crafted mathematical models of the environment. However, for complex control problems, developing a perfect model can be difficult or even impossible. Additionally, traditional methods necessitate re-training for every environmental change.

The emergence of RL has revolutionized traditional control methods in robotics. Unlike conventional approaches that rely on pre-defined models or hand-crafted rules, RL agents autonomously acquire knowledge from repeated trial-and-error interactions with an environment, enabling them to tackle complex and unpredictable challenges [18]. This ability to learn from experience and adapt to changing environments makes RL a compelling choice for robotics applications, where environments can be dynamic and unpredictable. In this part, the theoretical foundation is established by providing a short introduction to RL and its fundamental concepts.

In RL-based control (Figure 2.3), an agent, acting as a decision-maker or controller, interacts with an environment, which in the context of balancing the MEUR, is not a physical environment but the mathematical model established in the previous section. At each time step, the agent perceives the current state of the environment (s_t), which serves as input to a policy function. This function is a mapping from states to actions (a_t) that the agent should take in those states.

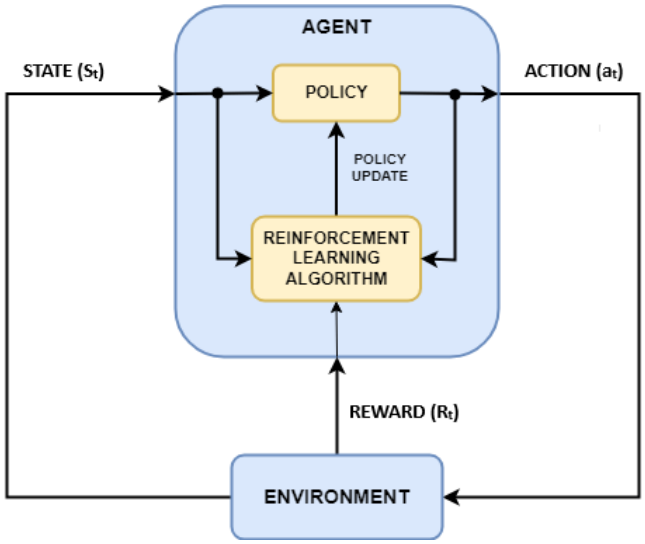


FIGURE 2.3: Reinforcement learning-based control [19]

For the lateral model, the states include the roll angle, roll angular velocity, reaction wheel angle, and reaction wheel angular velocity, with the lateral torque serving as the

corresponding action. Similarly, in the longitudinal model, the states encompass the pitch angle, pitch angular velocity, driving wheel angle, and driving wheel angular velocity, with the action being the longitudinal torque.

$$\vec{s}_{longitudinal} = (\theta_p, \dot{\theta}_p, \phi_{dw}, \dot{\phi}_{dw}) \tag{2.26}$$

$$\vec{s}_{lateral} = (\theta_r, \dot{\theta}_r, \phi_{rw}, \dot{\phi}_{rw}) \tag{2.27}$$

$$a_{longiitudinal} = \tau_{lon} \tag{2.28}$$

$$a_{lateral} = \tau_{lat} \tag{2.29}$$

There are two main types of policy functions: deterministic and stochastic. A deterministic policy function always maps a particular action to a given state:

$$a_t = \pi(s_t). \tag{2.30}$$

A stochastic policy function, on the other hand, maps the current state to a probability distribution over actions:

$$\pi(a|s) = P(a_t = a|s_t = s), \tag{2.31}$$

where P is the probability of taking that action in that state. The agent then selects an action according to the policy function and performs it in the environment, whether the policy function is stochastic or deterministic. The action causes the environment to transition to a new state (s_{t+1}) and deliver a numerical reward (R_{t+1}) to the agent. The reward is a scalar signal that reflects the desirability of the agent's chosen action. Designing an effective reward function is no easy feat. It needs to be informative, guiding the agent towards the ultimate goal without getting lost in the immediate consequences of each action. Reward functions need to strike a balance, considering both immediate rewards and long-term objectives, to avoid suboptimal strategies. For example, in this study, the unicycle receives rewards for keeping the pitch and roll angles within a specific range and incurs penalties for deviating. This process shapes agent decision-making towards stabilisation.

The objective of the agent is to maximize the cumulative reward it receives throughout its interactions with the environment. This objective is formalized through the concept of the expected discounted return function, represented as J:

$$J = \sum_{t=0}^{\infty} \gamma^t R_t. \tag{2.32}$$

where t is the time step, R_t is the reward received at time step t, and γ is the discount factor,a value between 0 and 1 that determines how much weight is given to future

rewards. A higher γ value places a greater emphasis on future rewards, encouraging the agent to consider long-term goals. Conversely, a lower γ value prioritizes immediate rewards, potentially leading to suboptimal behavior.

Apart from agent, environment, reward, and the policy function, additional sub-elements of reinforcement learning can be defined as:

- State Value Function: The state value function, denoted as $V_\pi(s)$, represents the expected cumulative reward obtained from starting in state s and following a particular policy. It captures the desirability of a particular state regardless of the agent's subsequent actions:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]. \tag{2.33}$$

- Action Value Function: The action value function, denoted as $Q_\pi(s, a)$, represents the expected cumulative reward obtained from starting in state s and taking action a, following a particular policy. It captures the desirability of a particular state-action pair, considering the potential for future rewards:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \tag{2.34}$$

The total reward earned from any state can be understood as the sum of two components: the immediate reward received after taking an action and the discounted reward expected from the resulting state, assuming the same policy is followed thereafter. This fundamental relationship underpins reinforcement learning and is captured by the Bellman Equation. As a result, the state value function and action value function can be rewritten as follows:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(s_{t+1}) \right], \tag{2.35}$$

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})]. \tag{2.36}$$

RL-based control algorithms can be broadly categorized into two main approaches: model-based and model-free. Model-based RL agents use the environment's dynamics, enabling them to calculate future states and plan actions accordingly. While this approach offers greater foresight, it requires significant computational resources and may not be feasible for complex environments when even writing a parametric model is difficult or if the model changes over time. In contrast, model-free RL agents directly learn from interactions with the environment. They develop policies that map states to actions based on observed rewards and penalties. This approach is more scalable and can handle complex or unknown environments that make it suitable for controlling the MEUR.

Model-free RL algorithms are further categorized into three main sub-methods (Figure 2.4): policy-based, value-based, and actor-critic methods. These methods differ in their approach to learning the optimal policy, each offering unique advantages and trade-offs.

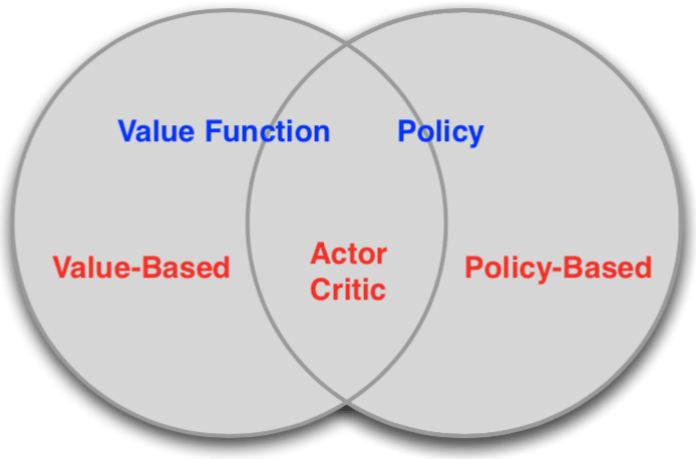


FIGURE 2.4: Reinforcement learning-based control methods [20]

Policy-based methods focus on directly optimizing the policy function, aiming to improve the agent's overall behavior without explicitly representing value functions. They learn to map states to actions, gradually adjusting the policy based on the observed rewards and penalties. This approach is relatively straightforward and computationally efficient, making it suitable for real-time control applications. However, policy-based methods can be sensitive to local optima, requiring careful exploration of the policy space to find the optimal policy [21].

Value-based methods focus on estimating the value functions, representing the desirability of states or actions. They learn the state value function ($V(s)$) or the action value function ($Q(s, a)$), which indicate the expected cumulative reward obtained from starting in a particular state or taking a specific action. These value functions guide the agent in making informed decisions, enabling it to select actions that lead to higher expected rewards.

Actor-critic methods combine the strengths of policy-based and value-based approaches, leveraging the advantages of both [22]. They employ two separate estimators: an actor that learns the policy function and a critic that estimates the value functions. The critic provides feedback to the actor, guiding it towards better policies that lead to higher expected rewards. This interplay between the actor and critic enables actor-critic methods to learn both optimal policies and value functions, making them more versatile and robust than either approach alone.

After explaining the core principles of policy-based, value-based, and actor-critic methods, this project endeavors to utilize two distinct reinforcement learning techniques to control the maneuverability and stability of the MEUR. Specifically, the Deep Q-Network (DQN) method, which falls under the value-based paradigm, and the Advantage Actor-Critic (A2C) method, belonging to the actor-critic framework, are selected.

Chapter 3

Methods

Building upon the theoretical foundation and mathematical model of the unicycle in Chapter 2, this chapter delves into the specific RL methods chosen to control this dynamic system. The reward functions designed to guide the unicycle to maintain stability in an upright position are dissected as well. Lastly, the implementation strategies to utilise the employed RL methods are discussed.

3.1 Deep Q- Networks

DQN is a reinforcement learning algorithm that combines Q-learning with deep neural networks to learn optimal policies in complex environments [23]. Q-learning is a value-based RL algorithm that estimates the expected cumulative reward for each state-action pair (Equations 2.34 and 2.36). Deep neural networks provide a powerful function approximation mechanism for learning these Q-values in high-dimensional state spaces. To elucidate this algorithm, it is imperative to explore the Q-learning method and deep neural networks in depth.

Q-learning is a value-based, model-free RL algorithm that has been successfully applied to a wide range of RL problems, including game playing, robotics, and control systems [24]. The Q-learning algorithm updates the Q-value of a state-action pair based on the Bellman equation and an observed transition. The update rule is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)]. \tag{3.1}$$

where s' is the next state, $max_{a'}Q(s',a')$ is the maximum Q-value over all possible actions in state s', and α is the learning rate, which determines how much to update the Q-value based on the observed transition. The Q-learning algorithm maintains a table containing all Q-values, which is updated at each time step according to the Q-learning updating rule. Through iterative interaction with the environment and subsequent feedback, the algorithm refines these Q-values until they converge to their optimal values.

At each step, the agent employs the ϵ-greedy policy to select the current action based on the current state, striking a balance between exploration and exploitation. Initially,

lacking experience, exploration dominates as the agent prioritizes discovering the environment's dynamics. This proactive phase gradually transitions towards exploitation as the agent accumulates knowledge, enabling it to confidently leverage the learned Q-values for optimal action selection. This dynamic balance prevents premature convergence to suboptimal solutions (local minima) that could arise from solely exploiting limited initial knowledge.

To achieve this adaptive exploration-exploitation trade-off, a decaying threshold (ϵ) is employed. With each episode, the probability of exploration gradually decreases, promoting exploitation over time. In this study, exponential decay is used to decrease the threshold. At each step, a uniformly distributed random number is generated between 0 and 1. If this value falls below the current threshold, the agent ventures into exploration by selecting a random action. Otherwise, it prioritizes exploitation, selecting the action associated with the highest known Q-value and maximizing immediate reward based on its acquired knowledge. In algorithm 1, the Q-learning is represented in pseudocode [21].

Algorithm 1 Q learning algorithm

```
Initialize the Q table
for each episode do
    Initialze s_0
    for every time step do
        Choose an action a using the ϵ-greedy policy
        Obtain the reward R and the next state s' from the environment
        Choose an action a' from s' by maximizing Q(s', a') over argument a'
        Update Q(s, a) based on Equation 3.1.
        s ← s' ; a ← a'
    end for
end for
```

The Q-learning algorithm excels in scenarios with finite state and action spaces. However, it faces significant limitations in high-dimensional state spaces, where storing and manipulating the entire state space becomes impractical due to the sheer number of possible states. Additionally, in continuous environments, where states can vary continuously rather than being discrete, it is even more challenging to visit all states [21]. To overcome this constraint, function approximation techniques are employed to replace the Q-table with a Q-function. Function approximation techniques allow RL agents to approximate high-dimensional continuous Q-value functions using analytical, parameterized functions. A common function approximation techniques used in RL is Linear Function Approximation (LFA). This approach involves representing value functions as linear combinations of basis functions, such as basis vectors or radial basis functions. This simplified representation facilitates efficient learning and reduces the computational burden associated with fully representing the value function [21].

Despite the simplicity and effectiveness of LFA, it often falls short in handling the complex, nonlinear relationships inherent in many RL environments. In these scenarios, neural networks (NN), particularly deep neural networks, have emerged as powerful function approximators capable of modeling intricate nonlinear relationships between states

and actions [25]. Their ability to capture complex dependencies has led to remarkable performance in various RL applications. These advantages made NNs a better choice for function approximation in this project.

NNs are a class of machine learning algorithms inspired by the structure and function of the human brain. They are composed of interconnected nodes called neurons, which process and transmit information through a network of connections. NNs learn by adjusting the weights and biases of their connections, which control the flow of information through the network. This process is known as training. During training, NNs are presented with a set of training data, which consists of input-output pairs. The NN's goal is to learn a mapping from inputs to outputs, such that it can accurately predict the output for new, unseen inputs.

NNs are composed of interconnected nodes called neurons (Figure 3.1), which process and transmit information through a network of connections. Each neuron receives inputs from its connected neurons, applies an activation function to the weighted sum of these inputs, and produces an output.

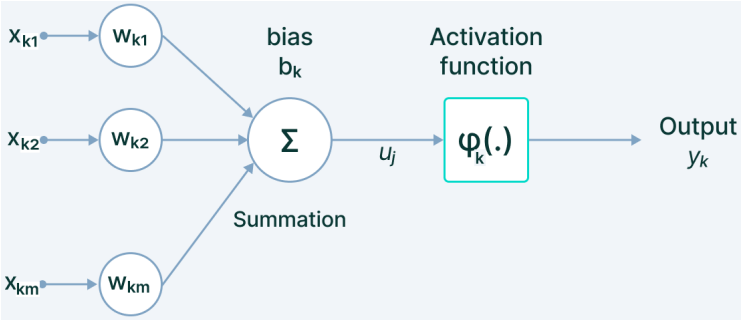


FIGURE 3.1: Neuron in NN [26]

The bias is a constant value that is added to the weighted sum of inputs before applying the activation function. The bias helps to shift the neuron's output, allowing it to model a wider range of relationships. Activation functions are mathematical operations that are applied to the outputs of neurons. They introduce non-linearity into the network, which is essential for learning complex patterns. There are various types of activation functions, each with its own characteristics. Some common activation functions include sigmoid, tanh, and ReLU (rectified linear unit) [27]. The neuron can be described in mathematical terms as:

$$y_k = \varphi\left(\sum_{i=1}^{m} w_{ki}x_{ki} + b_k\right). \tag{3.2}$$

The basic structure of an NN can be represented as a sequence of layers (Figure 3.2), where each layer consists of a set of interconnected neurons. The first layer is called the input layer, and it receives the input data. The last layer is called the output layer, and it produces the predicted output. In between the input and output layers are hidden layers, which perform intermediate computations.

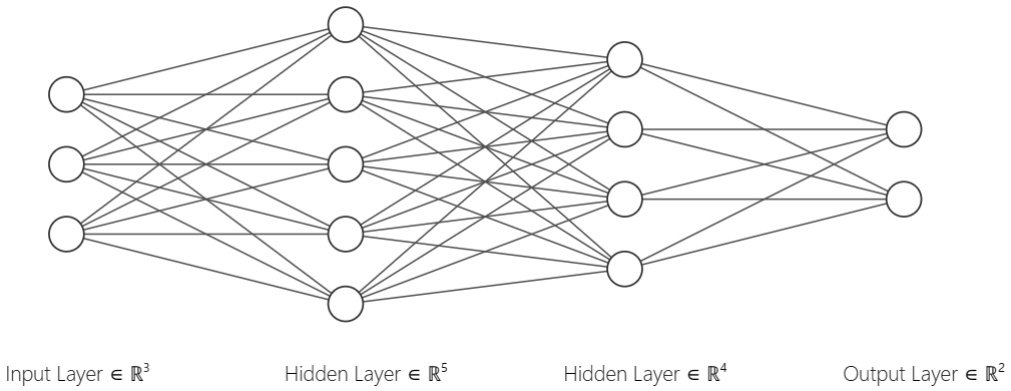


FIGURE 3.2: Multi-layer NN

The specific mathematical representation of each layer will depend on the type of activation function used and the number of neurons in each layer.

Training a NN involves adjusting its weights and biases to minimize the error between its predicted outputs and the desired outputs for training samples. The goal of training is to find a set of weights and biases that allow the network to accurately map inputs to outputs for unseen data. To quantify the performance of the network and guide the update of its weights and biases, a loss function is utilized. The loss function measures the difference between the predicted outputs of the network and the desired outputs. A common loss function is the mean squared error (MSE) [28], which is defined as the average squared difference between the predicted outputs and the desired outputs:

$$L_k = \frac{1}{n}\sum_{i=1}^{n}(y_{ki} - \hat{y}_{ki})^2 \tag{3.3}$$

where y_{ki} represents the desired output of the k^{th} output neuron for the i^{th} sample, $\hat{y}_{ki}$ denotes the predicted output of the k^{th} output neuron for the i^{th} sample, and n indicates the number of samples.

In DQN method, the agent uses a NN to approximate Q-values. The NN receives a state and an action as input and calculates the corresponding Q-values for that state-action pair as output. The input layer of the NN is dimensioned identically to the combined size of the state and action spaces, while the output layer is of size one. At time step t, the agent stores an experience that includes the current state of the environment, the selected action, the received reward, and the subsequent state of the environment in a memory buffer called replayed buffer. Once the memory buffer reaches its capacity, the agent selects a random minibatch of experiences from the replay memory and utilizes them to train the NN. It is important to note that training the NN with single samples would result in high variance for each sample and its corresponding gradients. This variance would prevent the network weights from converging effectively. Moreover, uniform random sampling is preferred over selecting consecutive experiences because consecutive experiences tend to be highly correlated, potentially leading to overfitting situations [29].

Finally, the desired Q-value for each experience is calculated based on the Q-learning updating rule (Eq 3.1).

To update the weights and biases of an NN during training, a backpropagation algorithm is used. It works by propagating the error from the output layer back to the input layer, adjusting the weights and biases along the way. Backpropagation is an iterative process, and it is repeated until the NN converges to a satisfactory solution. The backpropagation algorithm involves the following steps [30]:

- Calculate the error at the output layer: The error at the output layer is the difference between the desired outputs and the predicted outputs. This error is calculated using the loss function.
- Propagate the error backward: The error is propagated backward through the network, layer by layer.
- Update the weights and biases: At each layer, the error is utilized to adjust the weights and biases of the neurons in that layer to minimize the error. The specific update rule depends on the type of activation function used.

The process of backpropagation involves calculating the gradients of the loss function with respect to the weights and biases of the NN. These gradients are then used to update the weights and biases in the direction that minimizes the loss function. The gradient for a weight or bias can be calculated using the chain rule. Once the gradients have been calculated, the weights and biases can be updated using the gradient descent rule [31]:

$$w_{ki} \leftarrow w_{ki} - \alpha \cdot \frac{\partial L}{\partial w_{ki}}, \tag{3.4}$$

$$b_k \leftarrow b_k - \alpha \cdot \frac{\partial L}{\partial b_k}. \tag{3.5}$$

where α is the learning rate, which controls the size of the updates to the weights and biases. The process of backpropagation is repeated until the network converges to a satisfactory solution, meaning the loss function is minimized.

An additional issue with this approach arises from the frequent updates to the Q network's weights during each training time step. While this enhances the prediction accuracy of the Q values, it also alters the direction of the predicted Target Q values, causing fluctuations after each update. This scenario resembles pursuing a moving target.

To mitigate this challenge, a second network, known as the target network, can be introduced. Unlike the primary network, the target network remains unchanged during training time steps, ensuring stability in the target Q values. After a predetermined number of training time steps, the learned weights from the Q network are transferred to the target network. Utilizing a target network has been demonstrated to yield more stable training outcomes [29].

To sum up, the pseudocode of the DQN algorithm can be explained as algorithm 2 [32].

Algorithm 2 DQN algorithm

Initialize the Q network ($Q_{\boldsymbol{\theta}}$) and the target network ($Q_{\boldsymbol{\theta'}}$) with the same random weights ($\boldsymbol{\theta'} = \boldsymbol{\theta}$).
Initialize replayed memory D
for each episode **do**
    Initialze $\mathbf{s_0}$
    for every time step **do**
        Choose an action a using the ϵ-greedy policy
        Obtain the reward R and the next state $\mathbf{s}'$ from the environment
        Store $(\mathbf{s}, a, R, \mathbf{s}')$ in replayed memory D
        $\mathbf{s} \leftarrow \mathbf{s}'$; $a \leftarrow a'$
    end for
    for every training step **do**
        Sample a minibatch D_s randomly from D
        Choose an action a' from $\mathbf{s}'$ by maximizing $Q_{\boldsymbol{\theta'}}(\mathbf{s}', a')$ over argument a' for each experience $(\mathbf{s}, a, R, \mathbf{s}')$
        Update $Q_{\boldsymbol{\theta'}}(\mathbf{s}, a)$ based on Equation 3.1
        Train the Q network ($Q_{\boldsymbol{\theta}}$) to minimize $(Q_{\boldsymbol{\theta'}}(\mathbf{s}, a) - Q_{\boldsymbol{\theta}}(\mathbf{s}, a))$
    end for
    for predetermined number of training time steps **do**
        Update the target network with the trained value network: $\boldsymbol{\theta'} \leftarrow \boldsymbol{\theta}$
    end for
end for

In the longitudinal model, the input layer consists of five neurons, incorporating states and longitudinal torque. The lateral model employs four neurons in its input layer, encompassing states (excluding reaction wheel angle) and lateral torque.

$$\text{Longitudinal model inputs} = (\theta_p, \dot{\theta}_p, \phi_{dw}, \dot{\phi}_{dw}, \tau_{lon}). \tag{3.6}$$

$$\text{Lateral model inputs} = (\theta_r, \dot{\theta}_r, \dot{\phi}_{rw}, \tau_{lat}). \tag{3.7}$$

It is important to note a key distinction between the longitudinal and lateral models. While the angle of the rotation wheel isn't crucial for controlling the lateral model, the driving wheel angle plays a vital role in positioning the unicycle in a predetermined stance in the longitudinal model and should be considered as an input feature.

The specific configuration of hidden layers and neurons in each layer may vary between the two models, as detailed in the results chapter on hyperparameter optimization. However, in both models, the output layer consists of a single neuron representing the Q value.

3.2 Actor-Critic

Policy learning forms the cornerstone of reinforcement learning algorithms, where an agent aims to maximize its long-term reward by learning a policy to select actions in different states. Policy-based methods offer unique advantages compared to other reinforcement learning approaches. Firstly, they directly learn the probability of choosing each action in any state, granting fine-grained control over the agent's behavior. This ability also allows them to naturally manage the exploration-exploitation trade-off by adjusting action probabilities. Additionally, policy-based methods seamlessly handle continuous action spaces, unlike greedy methods and value-based methods which often struggle in such scenarios [21]. The policy function is typically represented by a set of parameters denoted by θ. In this study, these parameters govern a NN that maps states to actions probability. The ultimate goal is to adjust these parameters to maximize the expected discounted reward accumulated over an entire trajectory ($R(\tau)$). Mathematically, this translates to:

$$\text{Maximize:} \quad J(\theta) = \mathbb{E}_\pi[R(\tau)]. \tag{3.8}$$

In essence, finding the optimal parameters (θ_*) that maximize the objective function (J) solves the reinforcement learning problem. Similar to many optimization tasks in machine learning, a prominent approach for tackling this challenge is gradient ascent (or descent, depending on the objective) [31]. This iterative method is called policy gradient, which guides towards better parameters by taking small steps in the direction with the steepest increase (ascent) or decrease (descent) in J. The updating rule in policy gradient method can be written as follows:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t). \tag{3.9}$$

A crucial hurdle arises in computing the gradient of the objective function. To handle that, the gradient of the objective function can be reformulated as follows [21]:

$$\begin{aligned} \nabla \mathbb{E}_\pi[R(\tau)] =& \nabla \int \pi(\tau) R(\tau) d\tau = \int \nabla \pi(\tau) R(\tau) d\tau \\ &= \int \pi(\tau) \nabla \log \pi(\tau) R(\tau) d\tau = \mathbb{E}_\pi[R(\tau) \nabla \log \pi(\tau)]. \end{aligned} \tag{3.10}$$

Also, the policy function can be expanded to [21]:

$$\pi_\theta(\tau) = P(s_0) \Pi_{t=1}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}, R_{t+1}|s_t, a_t). \tag{3.11}$$

Understanding the computation requires a closer look at its components.

- Initial Distribution ($P(s_0)$): This term represents the starting point of the agent, captured by the probability distribution P over the initial state s_0.

- Action Product (Product Rule): We leverage the Markov property, which assumes each action's probability depends only on the current state, not the entire history. This allows us to use the product rule of probability. Essentially, we multiply the probabilities of taking each action over the trajectory length T, reflecting the sequence of actions chosen based on the policy (π_θ) and the environment's dynamics.

By taking the log, the expected reward can be written as follows:

$$\begin{aligned} &\log \pi_\theta(\tau) = \log P(s_0) + \Sigma_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \Sigma_{t=1}^{T} \log P(s_{t+1}, R_{t+1}|s_t, a_t) \\ &\Rightarrow \nabla \log \pi_\theta(\tau) = \Sigma_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t) \\ &\Rightarrow \nabla \mathbb{E}_{\pi_\theta}[R(\tau)] = \mathbb{E}_{\pi_\theta}\left[R(\tau)\left(\Sigma_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)\right)\right]. \end{aligned} \tag{3.12}$$

This outcome is known as the policy gradient theorem [21]. It establishes a key relationship: the derivative of the expected reward with respect to the policy parameters aligns with the expected product of individual rewards and the gradient of the policy's log-probability.

The total reward obtained throughout a trajectory ($R(\tau)$) can be replaced by the Q value. So equation 3.12 can be written as follows:

$$\nabla_\theta J(\theta) = \nabla \mathbb{E}_{\pi_\theta}[R(\tau)] = \mathbb{E}_{\pi_\theta}\left[Q(s_t, a_t)\left(\Sigma_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)\right)\right]. \tag{3.13}$$

To further enhance the algorithm's performance, additional refinements can be made. Consider an action-value function Q, whose scalar value is assumed to have an offset and operate within a specific range. Subtracting this average offset would reduce the variance while keeping the estimated gradient unchanged [33]. This can be achieved by subtracting the state-value function V from Q, resulting in the advantage function A:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{3.14}$$

Substituting $A(s, a)$ into equation 3.12 instead of $r(\tau)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[A(s_t, a_t)\left(\Sigma_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)\right)\right] \tag{3.15}$$

results in a method known as Advantage Actor-Critic (A2C) [33]. A2C represents a hybrid architecture that combines both value-based and policy-based methods, thereby aiding in stabilizing training by minimizing variance. Within A2C, an actor dictates the behavior of our agent, employing a policy-based approach, while a critic evaluates the efficacy of the action undertaken, employing a value-based method.

Moreover, to deal with uncertainties, disturbances, and noises, a stochastic policy function is implemented. In contrast with deterministic policy functions, where a single action is always prescribed for each state, in stochastic policy functions, the probability distribution of taking action a in state s is calculated. Since the action space is continuous

in this study, the probability becomes a probability density function. To represent this stochastic policy function, a Gaussian probability distribution [34] is used.

$$\pi(a|\mu,\sigma) \equiv \mathcal{N}(a|\mu,\sigma) \tag{3.16}$$

where, μ represents the mean and σ represents the standard deviation, defining the action's central tendency and dispersion, respectively.

To sum up, the Advantage Actor-Critic algorithm can be explained using the following steps:

Algorithm 3 Advantage actor-critic algorithm

Initialize critic networks (value network $V(\mathbf{s})$ and target network $V'(\mathbf{s})$) and actor network $\pi(\mathbf{s})$
Initialize replayed memory D
for each episode **do**
    Initialze s_0
    for every time step **do**
        Choose an action a using the current policy $\pi(\mathbf{s})$
        Obtain the reward R and the next state $\mathbf{s}'$ from the environment
        Store $(\mathbf{s}, a, R, \mathbf{s}')$ in replayed memory D
        $\mathbf{s} \leftarrow \mathbf{s}'$; $a \leftarrow a'$
    end for
    for every training step **do**
        Sample a minibatch D_s randomly from D
        Calculate the $V^*(\mathbf{s}) = R + \gamma V'(\mathbf{s}')$
        Train the critic network (value network $V(\mathbf{s})$) to minimize $\delta_t = V^*(\mathbf{s}) - V(\mathbf{s})$
        Train the actor network to minimize $L = -log(\pi(a|\mu(\mathbf{s_t}), \sigma(\mathbf{s_t}))).\delta_t$
    end for
    for predetermined number of training time steps **do**
        Update the target network with the trained value network
    end for
end for

This study employs the advantage actor-critic algorithm separately for both the lateral and longitudinal models. Neural networks are used to approximate both the value and policy functions. The value neural network has only one output, denoted as V, whereas the policy neural network has two outputs: the mean (μ) and standard deviation (σ). The probability of selecting an action is subsequently calculated with

$$\pi(a|\mu,\sigma) = \frac{1}{\text{constant}} e^{(a-\mu)^2/\sigma^2}. \tag{3.17}$$

The choice of input parameters for the policy and value neural networks differs slightly between the longitudinal and lateral models. Both models rely on the state vector as

input. However, the lateral model can disregard the angle of the rotation wheel, as it is non-essential for controlling the roll angle. As a result, this distinction results in a slightly different set of inputs for each neural network, as detailed in the following equations:

$$\text{Longitudinal model inputs} = (\theta_p, \dot{\theta}_p, \phi_{dw}, \dot{\phi}_{dw}), \tag{3.18}$$

$$\text{Lateral model inputs} = (\theta_r, \dot{\theta}_r, \dot{\phi}_{rw}). \tag{3.19}$$

It is worth mentioning that the actual number of hidden layers and neurons in each layer may differ. This is discussed in detail in the chapter 4 through hyperparameter optimization.

3.3 Reward function

At the core of successful reinforcement learning, the reward function serves as the guiding force for the agent amid numerous potential actions. For the unicycle, maintaining its balance in both lateral and longitudinal directions and navigating towards a target location require meticulously crafted reward mechanisms for both the lateral and longitudinal models.

In the longitudinal model, the reward function is designed to incentivize actions that ensure the pitch angle and velocity remain within safe bounds, thus preventing the unicycle from toppling over. The ultimate goal is to guide the unicycle towards a destination point while maintaining longitudinal stability. Rewards are structured to encourage forward or backward movements that steer the unicycle closer to the target, while any deviations from the destination point or signs of impending falls result in corresponding penalties. This delicate balance between maintaining balance and progressing towards the goal shapes the agent's learning process. To achieve this, the longitudinal model's reward function is composed of three distinct components, each contributing to the overall learning objective.

- Pitch angle penalty (P_{pa}): This component penalizes the absolute value of the pitch angle, compelling the unicycle to maintain a near-zero pitch for stability.
- Velocity Penalty (P_{pv}): The squared values of both pitch and wheel angular velocities are penalized in this component. By discouraging sudden changes in pitch and wheel angles, this term fosters smoother control actions by the agent.
- Distance Reward (R_d): Incentivizing the agent's proximity to the destination point (ϕ_{dp}), this component calculates the remaining distance from the target. As the agent approaches the target, the reward escalates, fostering convergence.

$$\begin{cases} P_{pa} = -|\theta_p| \\ P_{pv} = -(\dot{\theta}_p^2 + \dot{\phi}_{dw}^2) \\ R_d \ = 1 - |\phi_{dp} - \phi_{dw}| \end{cases}$$

The state variables are normalized before calculating the reward components. The normalization factors are adjusted based on the provided ranges for each variable. The final reward (R_T) combines these individual components.

$$R_T = w_{pa}P_{pa} + w_{pv}P_{pv} + w_dR_d \tag{3.20}$$

For the lateral model, the agent strives to maintain the roll angle (θ_r) within a specific range while minimising the roll angular velocity ($\dot{\theta}_r$) and the reaction wheel velocity ($\dot{\phi}_{rw}$). Actions that keep the unicycle gracefully upright and minimise unnecessary energy expenditure should be rewarded. However, deviations from the desired state should be penalised, reminding the agent to adjust its actions to regain balance. Unlike the longitudinal model, there is no component for distance reward in the lateral model.

$$\begin{cases} P_{ra} = -|\theta_r| \\ P_{rv} = -(\dot{\theta}_r^2 + \dot{\phi}_{rw}^2) \end{cases}$$

Once more, the state variables are normalized before calculating the reward components, and then the final reward function combines these individual components. The reward function corresponding to the lateral model is represented by:

$$R_T = w_{ra}P_{ra} + w_{rv}P_{rv}. \tag{3.21}$$

3.4 Implementation

Following the exploration of theoretical underpinnings and reward function design, this section now transitions into the implementation of chosen RL methods to control the unicycle's lateral and longitudinal dynamics.

Python is chosen as the preferred programming language due to its versatility and rich ecosystem of libraries tailored for scientific computing and machine learning tasks [35]. The construction and training of NNs responsible for approximating Q-values and policy functions are accomplished using the TensorFlow library [36].

The Adam optimizer is employed for training NNs. This algorithm facilitates first-order gradient-based optimization of stochastic objective functions by employing adaptive estimates of lower-order moments. It is easy to implement, computationally efficient, and requires minimal memory [37]. Additionally, the MAE loss function is employed to measure the discrepancy between the network's predictions and the actual values, providing feedback for the optimizer to fine-tune the network weights.

To discern optimal hyperparameters such as learning rates and network architectures that significantly influence RL algorithm performance, GridSearchCV from the scikit-learn library is employed [38]. This tool systematically evaluates various hyperparameter combinations, facilitating the selection of configurations yielding the best results.

The Runge-Kutta 4th order (RK4) method was utilized to solve the core mathematical model governing the unicycle's lateral and longitudinal movements. This numerical integration technique is known for its high accuracy and stability, minimising errors that can accumulate during integration and ensuring reliable results [39]. This is crucial for simulation purposes, as inaccurate state information can impact controller performance. In this study, the RK4 implementation utilises a time step of 0.1 seconds. This selection balances computational efficiency with the need to capture the dynamics of the system with sufficient detail.

Finally, the golden section optimization method is employed for the ϵ-greedy policy. This optimization technique is efficient and requires only a few function evaluations to converge to the optimal solution [40]. This is crucial for real-time applications, where computational efficiency is paramount. In this implementation, the golden section method utilizes a stopping criterion of 0.05 for the difference between the upper and lower bounds. This stopping criteria ensures a balance between efficient computation and achieving a sufficiently accurate solution.

By meticulously implementing these elements and leveraging Python libraries, successful translation of theoretical concepts into practical solutions is achieved.

Chapter 4

Results

In this chapter, the results are presented, and the outcomes derived from the application of two distinct reinforcement learning methods—DQN and A2C—on both the lateral and longitudinal dynamics models of the unicycle are discussed. The analysis extends beyond simply presenting the outcomes. An exploration of initial conditions and various neural network hyperparameters is undertaken to elucidate their impact on the efficacy and stability of the learned policies, offering insights into the strengths and limitations of the employed methods.

4.1 DQN for the longitudinal model

In this section, results for implementing the DQN algorithm for controlling the unicycle's longitudinal dynamics are provided. However, the success of RL algorithms and the performance of the resulting agent heavily depend on carefully chosen hyperparameters. These parameters, unlike the model's learnable weights, are set manually before training begins. To enhance the DQN's performance, hyperparameter optimization is conducted to understand their influence on the agent's ability to achieve the control objective. To ensure comparability of results, all networks are trained under similar conditions, with only the parameter of interest varied. Following 100 training steps, the trained agents are deployed to control the unicycle under the same initial conditions, enabling the calculation of the cumulative reward for this specific scenario.

The number of hidden layers and the number of neurons in each layer in the neural network play a crucial role in its ability to learn complex relationships between the unicycle's state and the optimal control actions. With more hidden layers and more neurons, the network possesses greater representational power and can potentially capture more intricate features within the state space. This leads to improved performance in complex control tasks where numerous factors influence the optimal action selection. A very low number of hidden layers leads to underfitting, where the network is incapable of capturing the necessary complexity within the state space. Table 4.1 indicates that two hidden layers with 100 neurons each effectively learn the Q values. Decreasing the number of neurons to 50 results in an unstable agent, whereas increasing it yields comparable performance. Implementing only one hidden layer with 100 neurons fails to produce a

stable controller. Furthermore, increasing the number of hidden layers does not improve performance.

NN structure	[50,50]	[100,100]	[200,200]	[100,100,100]	[200,200,200]
Cumulative reward	957	3478	3416	3213	3146

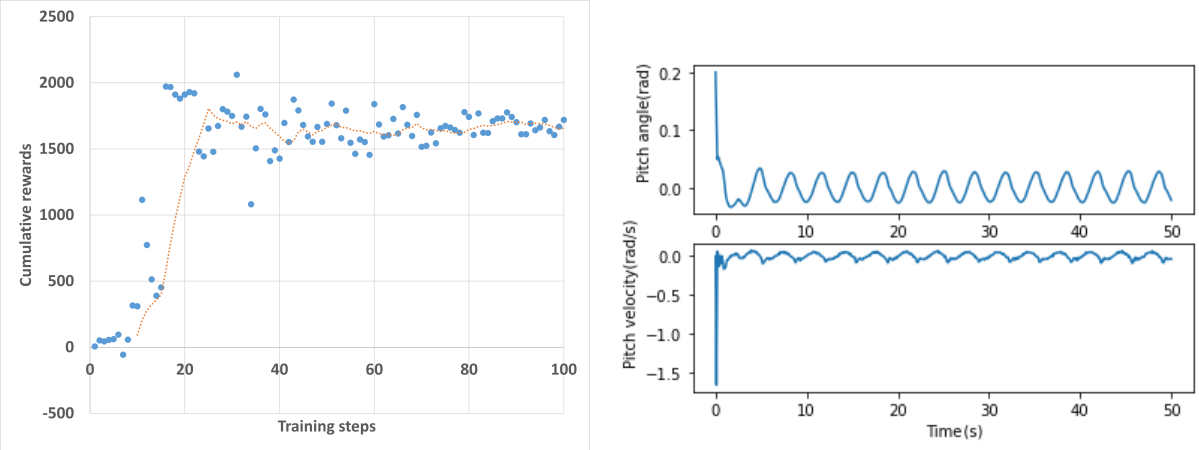
TABLE 4.1: Cumulative rewards for different NN structures

Since the problem is a regression-like task and the output layer should predict the desired Q value of the state-action pair, choosing a linear activation function is inevitable. Otherwise, using non-linear activation functions such as sigmoid or tanh would distort the output range and limit the representational capacity of the network. For the hidden layers, three different activation functions are employed, as illustrated in Table 4.2. The agent's overall performance remains consistent across all activation functions, with cumulative rewards falling within a similar range.

Hidden layer activation function	relu	elu	selu
Cumulative reward	3479	3468	3477

TABLE 4.2: Cumulative rewards for different activation functions

Decreasing the exploration decay rate emphasizes exploitation and accelerates convergence toward previously learned policies, resulting in premature convergence on suboptimal solutions, as observed in Figure 4.1. Increasing the exploration decay rate prolongs the exploration phase, but, as depicted in Table 4.3, it does not enhance the training performance.



(A) Cumulative rewards

(B) Pitch angle (θ_p) and velocity ($\dot{\theta}_p$)

FIGURE 4.1: Cumulative rewards and pitch angle for exploration decay rate equal to 0.95

Exploration decay rate	0.95	0.96	0.97	0.98	0.99
Cumulative reward	2139	2771	3470	3240	3367

TABLE 4.3: Cumulative reward for different values of the exploration decay rate

Table 4.4 shows cumulative rewards for different values of the learning rate. Increasing the learning rate induces instability and oscillations during training. The network constantly bounces back and forth between suboptimal policies. Decreasing the learning rate significantly slows down the learning process, requiring a longer training period to reach a desirable level of performance.

Learning rate	0.1	0.3	0.5	0.7	0.9
Cumulative reward	3480	3488	3325	2553	2974

TABLE 4.4: Cumulative reward for different values of the learning rate

Higher discount factors emphasise the importance of future rewards, encouraging the agent to plan for long-term goals. Lower discount factors place greater weight on immediate rewards, prompting the agent to focus on short-term gains. As it can be seen in Table 4.5, excessively low discount factors lead to myopic behaviour, where the agent only considers the immediate consequences of its actions and fails to plan for the long term.

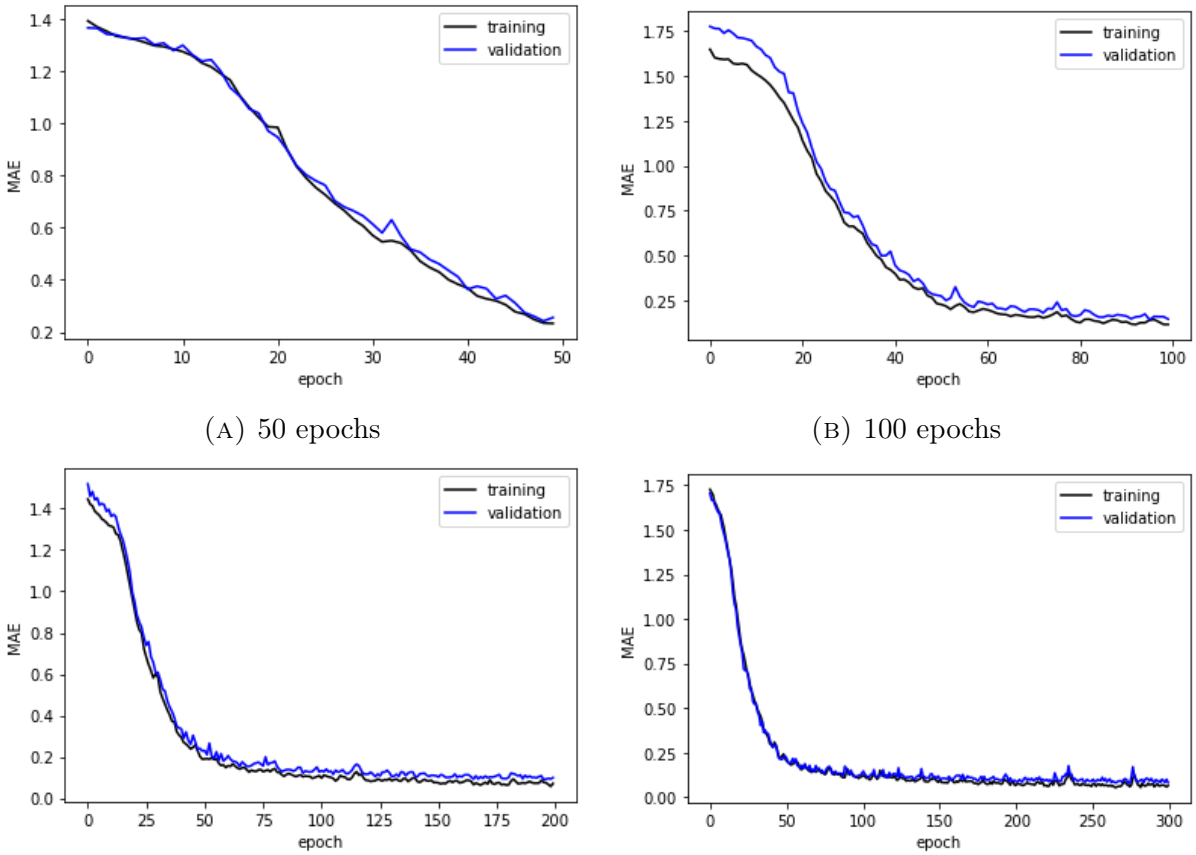
Discount factor	0.6	0.7	0.8	0.9	0.95
Cumulative reward	2006	2395	3073	3449	3344

TABLE 4.5: Cumulative reward for different values of the discount factor

The initial values assigned to a neural network's weights significantly impact the training process. Improper initialization can lead to vanishing or exploding gradients, hindering the network's ability to learn effectively [41]. Selecting the right initialization method is crucial for efficient learning and achieving optimal performance. During the hyperparameter optimization process, various initialization methods were explored, including 'uniform', 'lecun uniform', 'normal', 'zero', 'glorot normal', 'glorot uniform', 'he normal', and 'he uniform' [42]. Notably, 'lecun uniform', 'glorot uniform', and 'he uniform' consistently yielded the highest cumulative rewards, with all three performing very similarly.

Training with larger batches can lead to faster updates due to the efficient use of computational resources. This can be beneficial for reducing overall training time, especially when dealing with large datasets. However, larger batches also introduce higher variance in the gradient estimates used for weight updates. This can lead to unstable training and potentially hinder convergence to the optimal policy. Smaller batches result in more frequent but smaller updates, leading to slower convergence compared to larger batches. In this study, the agent undergoes training for various batch sizes, including 32, 64, 128, and 256. Despite variations in training duration, the controller's performance remains nearly consistent across all batch sizes.

Increasing the number of epochs provides the network with more opportunities to learn from the entire dataset, potentially enhancing performance, particularly for complex tasks requiring extensive training to capture underlying dynamics. However, excessive epochs can lead to overfitting, where the network memorizes specific training data patterns instead of generalizing well to unseen scenarios. Conversely, lower epoch numbers offer quicker training, especially for simpler tasks where the network can swiftly grasp essential control strategies. Yet, too few epochs may result in underfitting, where the network fails to effectively learn from the data, leading to suboptimal performance. In this study, the agent undergoes training with varying epoch numbers: 50, 100, 200, and 300. The training results remain consistent across all epochs, with no evidence of overfitting observed. During training, a validation split rate of 0.3 is employed, utilizing 2000 training data points for each training step. As illustrated in Figure 4.2, the Mean Absolute Error (MAE) exhibits minimal reduction beyond 200 epochs.



(A) 50 epochs

(B) 100 epochs

(C) 200 epochs

(D) 300 epochs

FIGURE 4.2: Training results for different epoch numbers

To ensure optimal performance of the RL agent, a thorough exploration of hyperparameters was conducted. Building on the insights gained from this analysis, the most effective hyperparameters were selected to guide the subsequent implementation phase. Detailed specifications regarding these network parameters can be found in Table 4.6.

Number of hidden layers	2	Exploration rate	1
Number of neurons in each layer	100	Exploration decay rate	0.97
The hidden layer's activation function	relu	Minimum exploration rate	0.01
Learning rate	0.3	Discount factor	0.9
Number of experiences to train NN	2000	Batch size	128
NN's weight initialization method	Xavier [43]	Epoch number	100

TABLE 4.6: Parameters values for the DQN algorithm implemented on the longitudinal model

Figure 4.3 depicts cumulative rewards over training steps and represents the performance of the training process for the agent trained according to the specifications detailed in Table 4.6. The graph showcases how the cumulative reward obtained by the reinforcement learning agent evolves over successive episodes of training. A rising averaged trend (red dotted line) in cumulative reward indicates that the agent is learning and improving its performance over time.

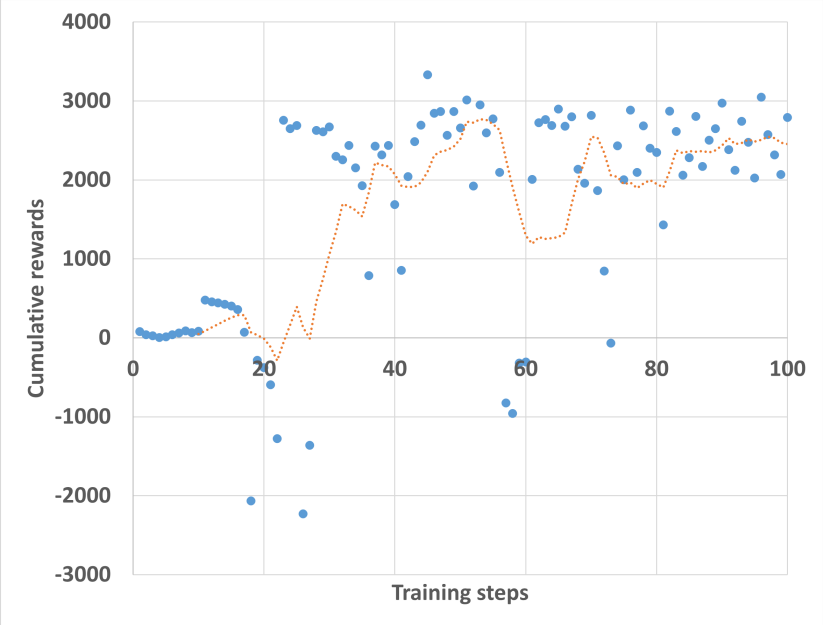


FIGURE 4.3: Cumulative rewards of the DQN algorithm for the longitudinal model

The performance of the trained agent in controlling the unicycle after 100 training steps is depicted in Figures 4.4 to 4.6. The initial condition of the unicycle is outlianed below:

$$(\theta_p, \dot{\theta}_p, \phi_{dw}, \dot{\phi}_{dw}) = (0.2, 0, 5, 0).$$

The setpoint for ϕ_{dw} is zero.

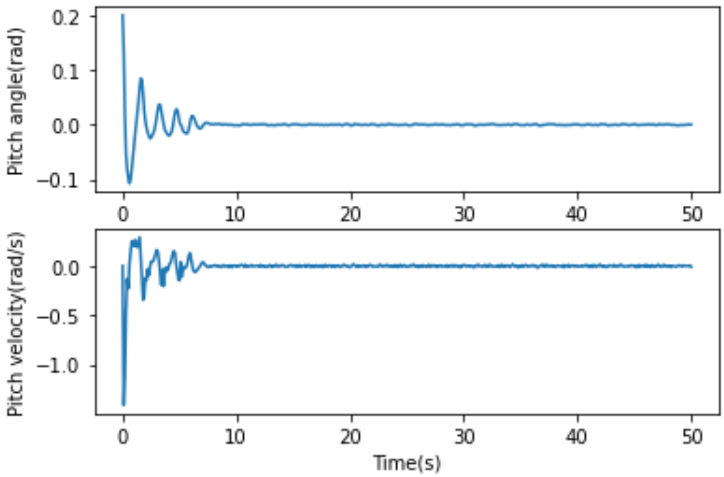


FIGURE 4.4: Pitch angle (θ_p) and pitch angular velocity ($\dot{\theta}_p$)

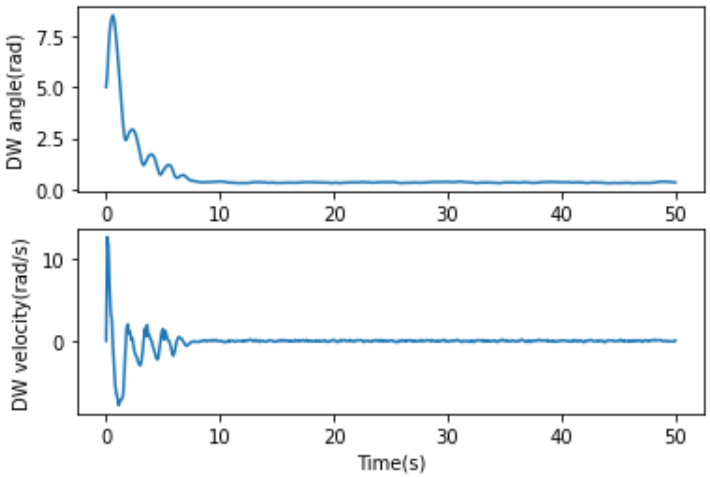


FIGURE 4.5: Driving wheel angle (ϕ_{dw}) and driving wheel angular velocity ($\dot{\phi}_{dw}$)

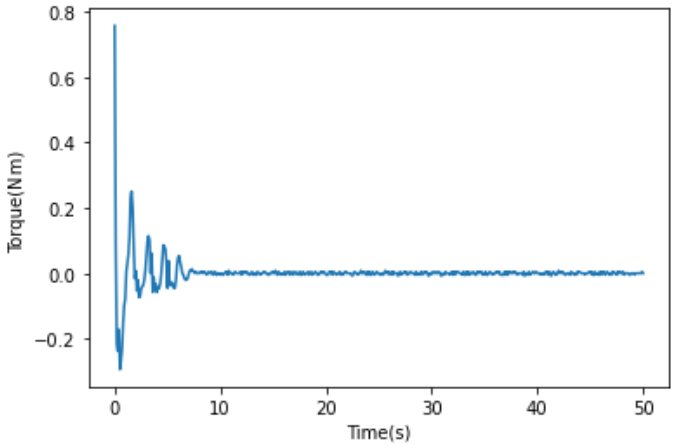


FIGURE 4.6: Longitudinal torque

The pitch angle initially deviates from zero and gradually approaches zero over time, indicating effective regulation by the DQN controller towards the desired setpoint. Despite some fluctuations around zero, the overall trend suggests successful control. Concurrently, the pitch angular velocity demonstrates a decreasing trend, converging towards zero, further affirming the controller's effectiveness in stabilizing the pitch angle, as a stable pitch angle necessitates a zero pitch angular velocity. Observations show a transition period of approximately 7 seconds, during which the pitch angle oscillations occur, but dampen quickly.

The agent possesses the ability to control the unicycle's driving wheel as well. Initially, the driving wheel angle increases owing to the application of high positive torque. This action is intended to alter the sign of the pitch angle, facilitating backward movement toward the destination while keeping the pitch angle within a safe range. Following this initial transition phase, characterized by fluctuations and oscillations in both the driving wheel angle and angular velocity, the agent successfully stabilizes the unicycle around the destination point.

While the provided figures indicate the DQN algorithm's ability to control the unicycle's driving wheel and achieve a degree of stability, it is important to acknowledge the potential for reaching suboptimal policies. This means the agent might not always find the action sequence leading to the perfect "zero" driving wheel angle, but instead converge on a policy that stabilizes the unicycle at a fixed, non-zero offset from the desired setpoint. Figure 4.7 shows the driving wheel angle and velocity for the same controller only with different initial conditions.

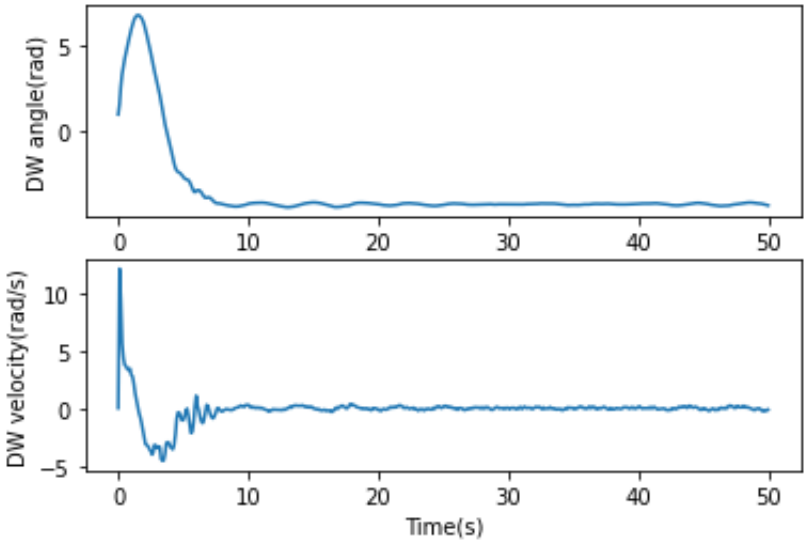


FIGURE 4.7: Suboptimal condition for the driving wheel angle (ϕ_{dw})

This suboptimal convergence can be attributed to several factors inherent to the control task and the learning process. The unicycle control environment exhibits complex dynamics. Exploring the full state space exhaustively is almost impossible, even with a very low decay rate for the exploration rate. This means there is always a chance of missing the optimal policy and converging to a suboptimal one that still achieves some

level of control. Moreover, the use of randomly sampled experiences to train the neural network introduces stochasticity. While crucial for exploration, it can also lead the agent to prioritize trajectories that achieve "good enough" performance rather than consistently leading towards the optimal solution.

The trend observed in Figure 4.3 also illustrates a notable aspect of the agent's behavior during training. At times, the agent appears to converge to the global maximum or a higher local maximum of the reward function. However, despite reaching these optimal or near-optimal states, the agent's exploration strategy introduces variability into its actions. As a result, the agent may deviate from these promising states and explore alternative actions or trajectories. While exploration is crucial for discovering new strategies and refining the agent's policy, it also introduces the possibility of converging to suboptimal states or exploring less favourable regions of the state space.

One approach to addressing the suboptimal convergence is to train a large population of agents. By evaluating their performance after training, the agent that achieves the best results, potentially the one with the highest cumlative reward can be selected for deployment. This approach leverages the inherent exploration aspect of DQN by training a diverse set of controllers and then selecting the "best" one based on a pre-defined performance metric.

Another effective strategy involves implementing a memory system to store the agent with the highest cumulative reward observed during the training process. Following the completion of training, this top-performing agent is preserved and can be deployed in place of the most recent version. During training, there is a risk that the agent's performance may decline compared to earlier stages due to exploration. Additionally, the exploration decay rate may cause the agent to become trapped in a suboptimal situation. By saving the best-performing agent, a fallback option is provided, ensuring that the agent demonstrating the best control performance during training is utilized.

4.2 DQN for the lateral model

Building upon the successful application of DQN to the unicycle's longitudinal dynamics, this section explores its effectiveness in controlling the lateral dynamics. Similar to the previous part, hyperparameter optimization is conducted to enhance the DQN's performance and understand its influence on the agent's ability to achieve the control objective. To ensure fair and comparable results, all neural networks are trained under identical conditions except for the specific hyperparameter being investigated. Following 100 training steps, the trained agents are then deployed to control the unicycle under the exact same initial conditions.

Table 4.7 illustrates cumulative rewards across varying numbers of hidden layers and neurons per layer. Like the longitudinal model, it demonstrates that two hidden layers with 100 neurons each effectively capture Q values. Reducing the neuron count to 50 leads to underfitting, while increasing it shows similar performance. Singularly employing one hidden layer with 100 neurons fails to stabilize the controller. Moreover, augmenting the number of hidden layers does not enhance performance.

NN structure	[50,50]	[100,100]	[200,200]	[100,100,100]	[100]]
Cumulative reward	1596	2943	2811	2909	774

TABLE 4.7: Cumulative rewards for different NN structures

Table 4.8 displays the utilization of three distinct activation functions for the hidden layers. The agent's performance marginally improves with the relu activation function.

Hidden layer activation function	relu	elu	selu
Cumulative reward	2930	2794	2756

TABLE 4.8: Cumulative rewards for different activation functions

As depicted in Table 4.9, reducing the exploration decay rate results in suboptimal agent performance, while increasing it does not improve training outcomes.

Exploration decay rate	0.95	0.96	0.97	0.98	0.99
Cumulative reward	2581	2706	2967	2993	2982

TABLE 4.9: Cumulative reward for different values of the exploration decay rate

Analysis of the learning rate reveals that higher values result in oscillations between suboptimal policies during the initial learning phase. However, after 100 training steps, there is no notable difference in agent performance (see Table 4.10), with the average cumulative rewards for the last ten episodes falling within the same range.

Learning rate	0.1	0.2	0.3	0.4	0.5
Cumulative reward	3207	3499	3331	3156	3245

TABLE 4.10: Cumulative reward for different values of the learning rate

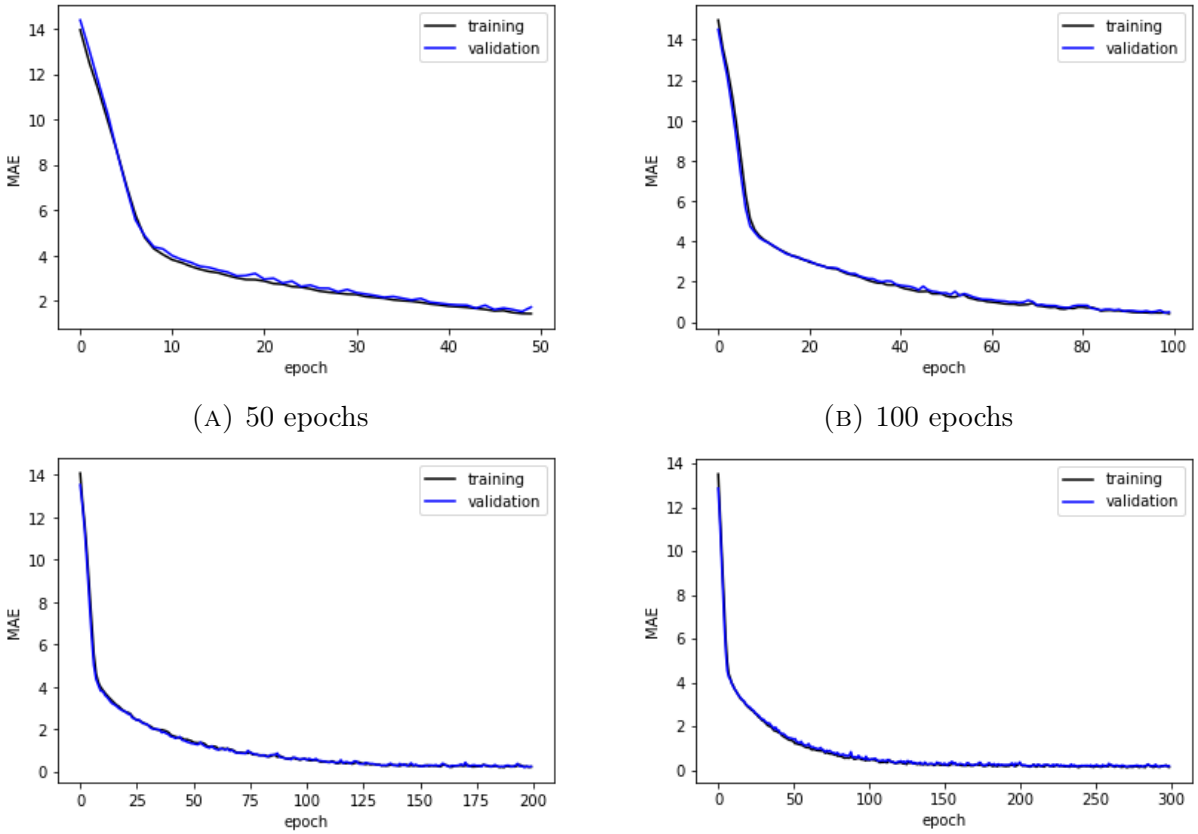
A higher discount factor generally yields better performance overall (see Table 4.11). This improvement may stem from the reward function's design. Conversely, a lower discount factor results in underperformance and, in some cases, instability, particularly with low learning rates.

Discount factor	0.25	0.5	0.8	0.9	0.95
Cumulative reward	Unstable	2761	2960	3012	2974

TABLE 4.11: Cumulative reward for different values of the discount factor

Like the longitudinal model, the agent is trained across different batch sizes, ranging from 32 to 256. The controller's performance remains relatively consistent across all batch sizes, with no significant impact observed on either controller performance or the

training process. Moreover, the agent is trained across different epoch numbers, ranging from 50 to 300. The training results remain consistent across all epochs, with no evidence of overfitting observed (Figure 4.8). Notably, the validation and training trends exhibit remarkable similarity, attributed to the ample training data where 70% is allocated for training and 30% for validation.



(A) 50 epochs

(B) 100 epochs

(C) 200 epochs

(D) 300 epochs

FIGURE 4.8: Training results for different epoch numbers

Hyperparameter optimisation analysis yielded valuable insights into how different hyperparameter configurations influenced the agent's effectiveness. Drawing upon these findings, the selection process for hyperparameters was meticulously guided to ensure the chosen configuration achieved the most effective control performance for the unicycle. Detailed specifications for these optimized network parameters are presented in 4.12.

Figure 4.9 illustrates the cumulative rewards obtained over training steps for the agent trained according to the specifications detailed in Table 4.12., offering insights into the performance of the training process. A rising trend, as indicated by the red dotted line, signifies that the agent is progressively learning and enhancing its performance throughout the training period.

Number of hidden layers	2	Exploration rate	1
Number of neurons in each layer	100	Exploration decay rate	0.98
The hidden layer's activation function	relu	Minimum exploration rate	0.01
Learning rate	0.2	Discount factor	0.9
Number of experiences to train NN	2000	Batch size	128
NN's weight initialization method	Xavier	Epoch number	200

TABLE 4.12: Parameters values for the DQN algorithm implemented on the lateral model

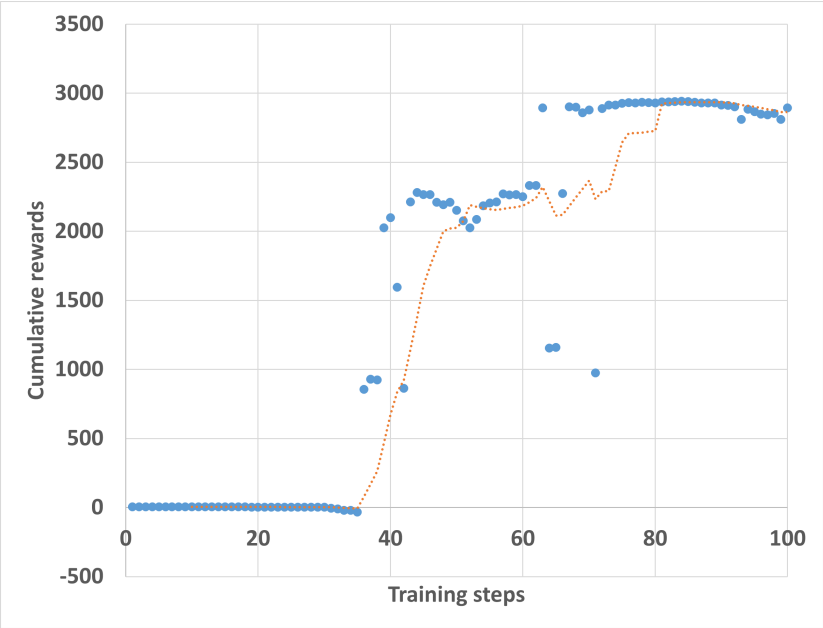


FIGURE 4.9: Cumulative rewards of the DQN algorithm for the lateral model

Figures 4.10 to 4.12 illustrate the effectiveness of the trained agent in controlling the unicycle following 100 training steps. The unicycle's initial condition is outlined as follows:

$$(\theta_r, \dot{\theta}_r, \phi_{rw}, \dot{\phi}_{rw}) = (0.2, 0, 0, 0).$$

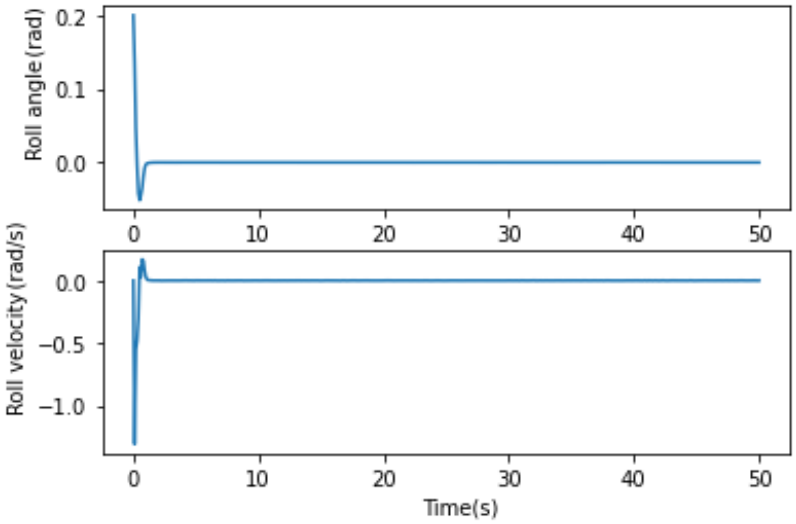


FIGURE 4.10: Roll angle (θ_r) and velocity ($\dot{\theta}_r$)

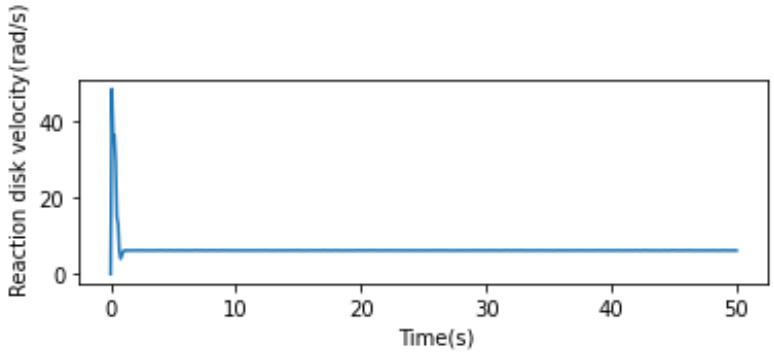


FIGURE 4.11: Reaction wheel velocity ($\dot{\phi}_{rw}$)

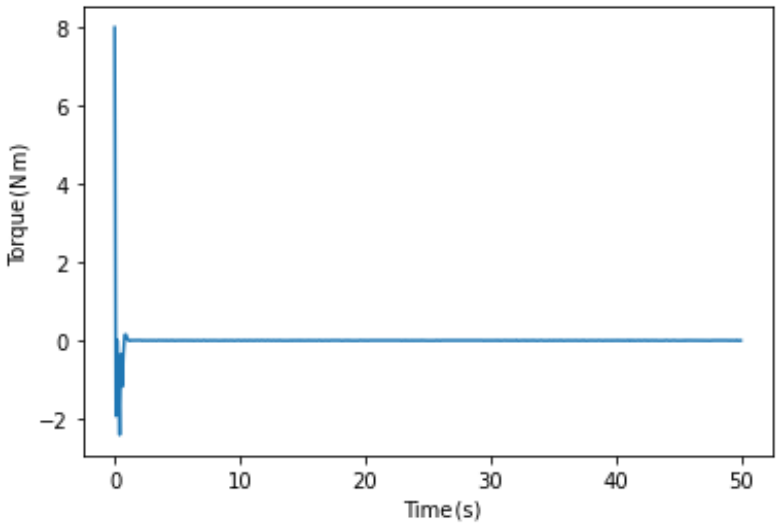


FIGURE 4.12: Lateral torque (τ_{lat})

The roll angle initially exhibits a deviation from zero, followed by a correction towards the desired setpoint (zero) over time. This indicates the DQN controller's ability to regulate the roll angle and stabilize the unicycle. The roll angular velocity plot complements the roll angle observations. It demonstrates a decreasing trend converging towards zero, mirroring the roll angle's approach to the setpoint. This reinforces the controller's effectiveness. A stable roll angle with minimal angular velocity signifies a balanced unicycle with minimal roll axis rotations.

Figure 4.11 initially depicts a notable increase in the reaction wheel velocity, indicating active correction of the unicycle's roll angle. Subsequently, a decrease in reaction wheel velocity follows, signifying successful roll angle correction by the controller, resulting in a reduction of the reaction wheel's angular velocity. However, unlike a perfect scenario where the velocity converges to zero, the plot shows a convergence to a non-zero steady-state value of approximately 5 rad/s. This is because in a complex environment, there are several suboptimal policies, and the agent might converge on one of them. This means the controller might not discover the absolute best course of action to achieve a perfectly zero roll angle and zero reaction wheel speed. Instead, it finds a policy that achieves a stable state with a slight residual roll angle and non-zero steady-state reaction wheel speed.

Overall, the observed behavior underscores the controller's agility in utilizing the reaction wheel to promptly address roll angle deviations.

4.3 A2C for the longitudinal model

In this section, the outcomes of applying the A2C algorithm to control the unicycle's longitudinal dynamics are delineated. Similar to the previous part, a meticulous hyperparameter optimisation process is undertaken to maximise the A2C agent's effectiveness. Tables 4.13 to 4.16 provide the comprehensive results obtained from this optimisation effort. A crucial aspect of this analysis involves ensuring fair and comparable results. To achieve this, all neural networks are trained under identical conditions, except for the specific hyperparameter being optimized. Following a consistent number of training steps, the trained agents are then deployed to control the unicycle under the exact same initial conditions.

NN structure	[50,50]	[100,100]	[200,200]	[100,100,100]	[100]]
Cumulative reward	521	3018	2867	2799	832

TABLE 4.13: Cumulative rewards for different NN structures

Hidden layer activation function	relu	elu	selu
Cumulative reward	2886	2804	2971

TABLE 4.14: Cumulative rewards for different activation functions

Learning rate	0.01	0.05	0.1	0.2	0.3
Cumulative reward	3123	3201	3078	3045	2996

TABLE 4.15: Cumulative reward for different values of the learning rate

Discount factor	0.5	0.6	0.7	0.8	0.9
Cumulative reward	1407	2439	2685	2832	2916

TABLE 4.16: Cumulative reward for different values of the discount factor

It is noteworthy that both the actor and critic neural networks within the A2C algorithm leverage a consistent network structure and activation function for hidden layers. Based on hyperparameter optimization, the neural networks employed for estimating both the policy function and value function consist of two hidden layers, each housing 100 neurons. Selu serves as the activation function for the hidden layers, whereas a linear activation function is used for the output layer. Notably, the policy neural network's output is bound to a maximum torque of 1.25 Nm for the longitudinal model.

The Adam Optimizer is employed to train neural networks, utilizing a learning rate of 0.05. With each training step, 1000 experiences are selected, and NNs undergo training with a batch size of 100 and an epoch number of 100. Additionally, the network weight initialization method adheres to the Glorot uniform methodology. The discount factor utilised is 0.9.

The figures 4.13 to 4.15 illustrate the performance of the trained agent in controlling the unicycle following 1000 training steps. The initial condition for the unicycle is

$$(\theta_p, \dot{\theta}_p, \phi_{dw}, \dot{\phi}_{dw}) = (-0.2, 0, 0, 0), \tag{4.1}$$

while the setpoint for the driving wheel is angle remains at zero.

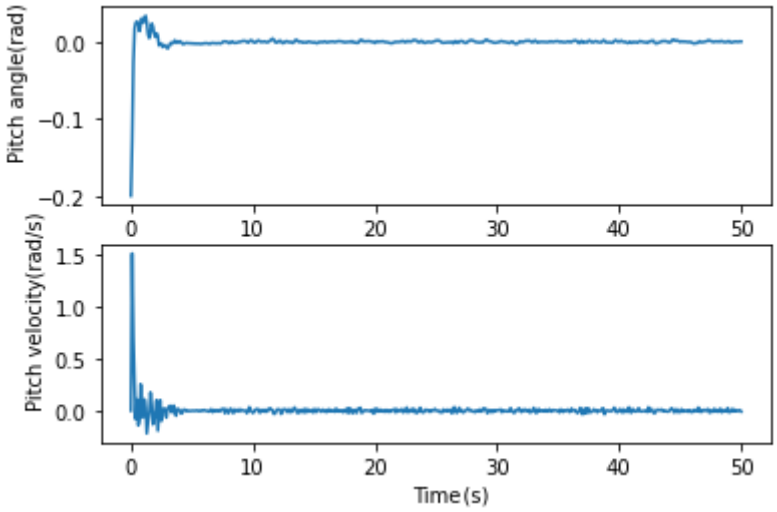


FIGURE 4.13: Pitch angle (θ_p) and velocity ($\dot{\theta}_p$)

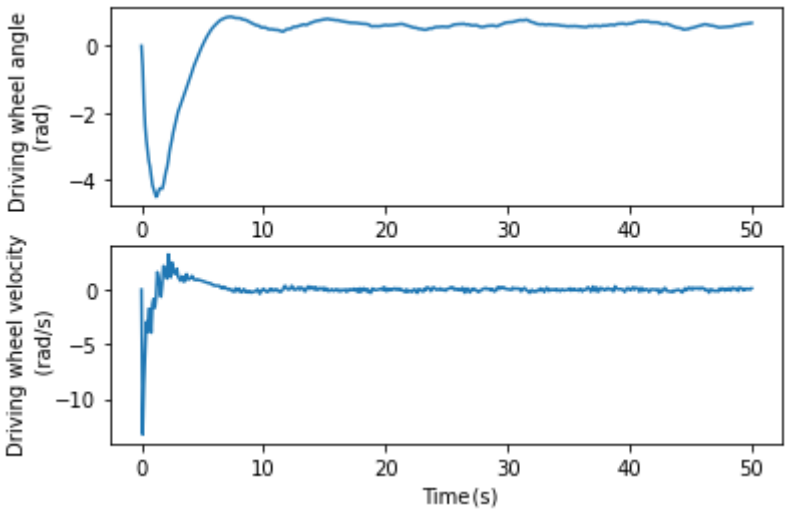


FIGURE 4.14: Driving wheel angle (ϕ_{dw}) and velocity ($\dot{\phi}_{dw}$)

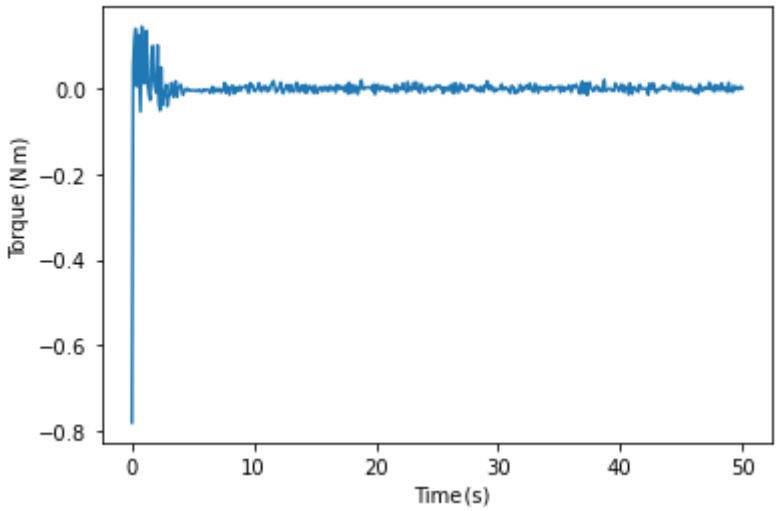


FIGURE 4.15: Longitudinal torque (τ_{lon})

The implemented A2C algorithm demonstrates promising results in controlling the unicycle's longitudinal dynamics, particularly in maintaining the pitch angle. The analysis reveals successful regulation of the pitch angle. Initial fluctuations are observed during the transition period as the controller establishes control. However, these fluctuations quickly dampen, and both the pitch angle and its velocity settle around zero. This signifies the controller's ability to effectively stabilize the unicycle's pitch angle.

The driving wheel angle also converges around zero. Notably, some minor oscillations persist around this equilibrium point. This behavior can potentially be attributed to the stochastic nature of the A2C algorithm. In A2C, the policy function, which dictates the controller's actions, has a certain level of inherent randomness. Even with a very small standard deviation, this stochasticity can introduce slight variations in the chosen actions, leading to the observed oscillations. While this stochasticity might cause slight

oscillations during normal operation, it plays a crucial role in the A2C algorithm's ability to handle unexpected disturbances and noise. By constantly exploring different actions due to stochasticity, the agent is better equipped to adapt to unforeseen changes in the environment, enhancing their overall resilience against external influences.

It is important to acknowledge the potential for suboptimal policy convergence in the A2C. This means that some agents within the training process might not discover the absolute best strategy for achieving a perfectly zero driving wheel angle. Due to the inherent randomness in the starting positions of each training episode (random initial conditions) and the use of randomly sampled past experiences for training the neural networks, the controller might sometimes converge on policies that achieve stability with a slight non-zero driving wheel angle offset.

Addressing suboptimal convergence during training can be mitigated through two complementary strategies. Similar to previous sections, training a large population of agents and evaluating their post-training performance allows for selecting the one with the highest cumulative reward, increasing the odds of finding a superior controller. Additionally, a memory mechanism can be implemented to track the agent achieving the peak cumulative reward observed throughout training. This "agent replay" approach ensures deployment of the agent that demonstrably showcased the best control capabilities during the training process, even if the final trained agent exhibits suboptimal performance.

4.4 A2C for the lateral model

Building upon the success of A2C in controlling longitudinal dynamics, this section explores its effectiveness in managing the unicycle's lateral dynamics. Tables 4.17 to 4.20 provide the comprehensive results obtained from the optimization effort.

NN structure	[50,50]	[100,100]	[200,200]	[100,100,100]	[100]]
Cumulative reward	1782	2819	2848	2833	936

TABLE 4.17: Cumulative rewards for different NN structures

Hidden layer activation function	relu	elu	selu
Cumulative reward	2897	2955	3014

TABLE 4.18: Cumulative rewards for different activation functions

Learning rate	0.01	0.05	0.1	0.2	0.3
Cumulative reward	2453	3191	3188	3160	2940

TABLE 4.19: Cumulative reward for different values of the learning rate

Discount factor	0.5	0.6	0.7	0.8	0.9
Cumulative reward	1758	1941	2665	2900	2922

TABLE 4.20: Cumulative reward for different values of the discount factor

The outcomes derived from employing the A2C algorithm to govern the lateral dynamics of the unicycle are presented through figures 4.16 to 4.18. The policy neural network's output is capped at a maximum torque of 8 Nm for the lateral model. The initial condition is the same as Equation 4.1, but with the roll angle replacing the pitch angle.

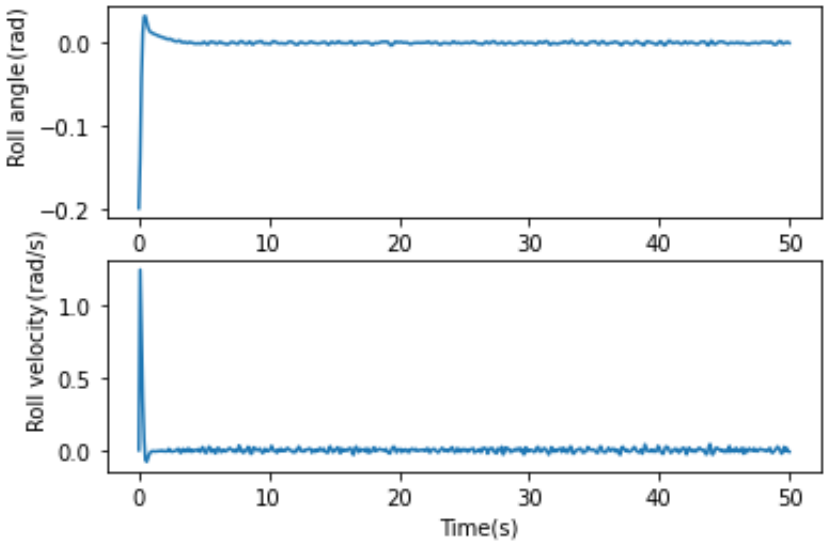


FIGURE 4.16: Roll angle (θ_r) and velocity ($\dot{\theta}_r$)

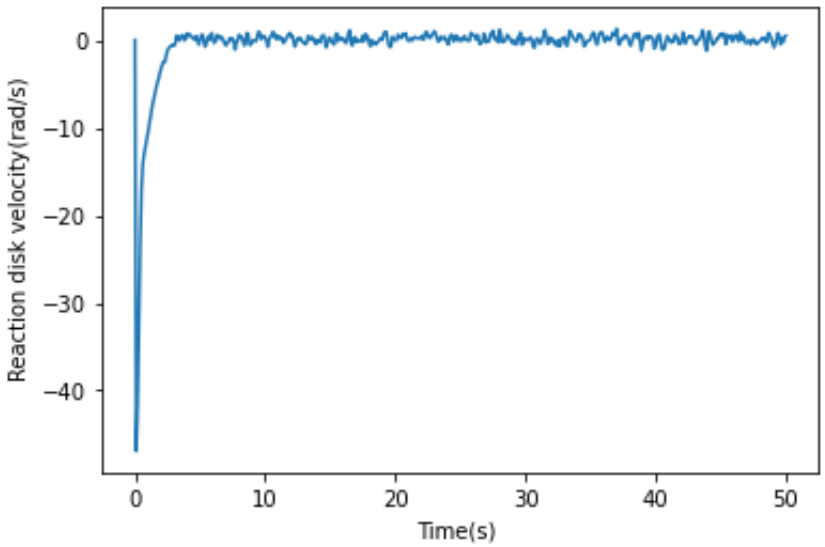


FIGURE 4.17: Reaction wheel velocity ($\dot{\phi}_{rw}$)

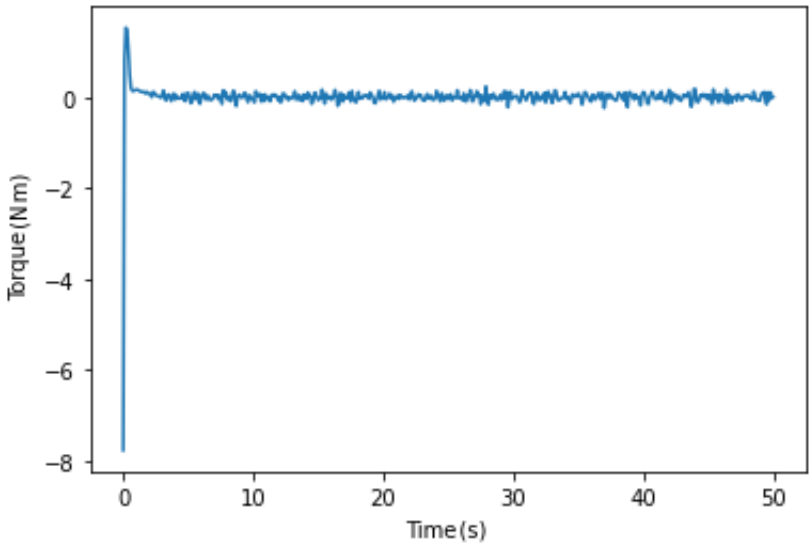


FIGURE 4.18: Lateral torque (τ_{lat})

The A2C algorithm demonstrates encouraging capabilities in managing the unicycle's lateral dynamics. Similar to the longitudinal control, the implemented A2C controller maintains the unicycle's lateral stability. As it is reflected in minimal deviations from the desired lateral position throughout the operation, the overall behavior suggests successful convergence. As observed in the longitudinal control, the stochastic nature of the A2C algorithm might introduce slight variations in the controller's actions. This can manifest in the form of minor fluctuations around the desired equilibrium point. The previously discussed potential for suboptimal convergence in A2C also applies to the lateral control task. During training, some agents might converge on policies that achieve stability, but with a slight offset from the ideal setpoint.

Chapter 5

Conclusion and recommendations

In pursuit of answering the research questions posed at the outset of this study, this chapter summarizes the key findings, limitations, and potential future directions of this thesis, which investigated the effectiveness of using RL methods for controlling a MEUR.

In addressing the effectiveness of RL-based methods to control the MEUR and maintain both lateral and longitudinal stability, this research demonstrates that both the DQN and the A2C algorithms can achieve stability in both roll and pitch angles in the simulated environment using a MEUR dynamic model. The analysis of the pitch and roll angle responses revealed successful regulation towards the desired setpoints of zero, indicating the controller's ability to maintain a stable upright position for the MEUR. For the driving wheel angle and reaction wheel velocity, while they converge to their setpoints, it is important to note the potential for suboptimal convergence, where the controller may not achieve perfect setpoints but stabilize the system at a slight offset from the desired setpoint. One approach to potentially mitigating suboptimal convergence is to train a large population of agents in parallel. By evaluating their performance after training, the agent that achieves the best results, such as the one with the most stable behavior and minimal deviations from desired setpoints, can be selected for deployment. Additionally, a memory mechanism can be implemented to track the agent achieving the peak cumulative reward observed throughout training. This approach ensures the deployment of the agent that demonstrably showcases the best control capabilities during the training process, even if the final trained agent exhibits suboptimal performance.

One of the key strengths of RL-based control methods lies in their adaptability. Unlike classical control methods, which hinge on pre-defined, system-specific equations, RL approaches can deal with different systems without extensive modifications. This is evident in the successful implementation of both DQN and A2C algorithms for controlling the unicycle's lateral and longitudinal dynamics. The adaptability stems from RL's core principle of learning through interaction with the environment. Moreover, this flexibility enables RL-based agents to seamlessly accommodate changes in dynamic environments. This continuous learning capability presents a notable contrast to classical methods, which often require extensive redesign when faced with significant alterations to the system.

Furthermore, investigation into the resilience of RL-based controllers to measurement

noise, load disturbances, and external perturbations provided valuable insights into their adaptive capabilities. The A2C method, with its stochastic nature, exhibits promising abilities to handle noises and disturbances. However, it is important to note that this study primarily focused on a simulated environment with minimal external influences. Future research endeavors should consider incorporating these factors into the simulation environment or conducting experiments with a physical MEUR prototype to gain a deeper understanding of the controller's resilience and potentially identify the need for additional adaptation mechanisms.

While this research demonstrates the initial promise of RL for MEUR control, significant challenges remain before widespread adoption in real-world delivery tasks. Unlike the controlled environment of simulations, real-world scenarios demand a more sophisticated MEUR capable of navigating dynamic environments and interacting safely with its surroundings. Successful navigation of a delivery MEUR hinges on its ability to perceive and respond to its surroundings. This necessitates the integration of additional sensors beyond those explored in this study. For example, LiDAR sensors combined with the camera would enable the MEUR to identify obstacles, locate delivery points, and interpret signage for proper navigation.

This study focuses primarily on maintaining longitudinal and lateral stability. Real-world deliveries often involve maneuvering through narrow corridors or navigating around obstacles. To achieve this level of agility, a yaw control mechanism and corresponding actuator are necessary. This would allow the MEUR to turn corners, adapt to changing delivery routes, and even perform more complex maneuvers. However, the possibility of falls or tip-overs during operation cannot be entirely eliminated. To ensure robust delivery capabilities, the MEUR would require a self-recovery mechanism. This could involve a combination of sensors to detect falls and actuators to right themselves autonomously, minimizing downtime and ensuring delivery completion. Last but not least, safety remains paramount when considering the real-world deployment of RL-controlled MEURs. Robust safety protocols are essential, including restricting operations to designated areas, emergency shut-off mechanisms, etc. Addressing these challenges presents promising avenues for future research, with the potential to advance the capabilities and reliability of RL-controlled MEURs for efficient and safe delivery operations.

In conclusion, this study represents a pivotal step towards unlocking the transformative potential of RL in the future of unicycle robots. While the challenges identified necessitate further research and development, the findings presented in this thesis lay the groundwork for the advancement of RL-based unicycle robots. By continuously improving RL algorithms, addressing limitations through innovative techniques, and ensuring the safe integration of such systems, a future where unicycle robots seamlessly integrate into our lives can be unlocked, performing complex tasks with efficiency.

Statement: During the preparation of this work, the author utilized ChatGPT to enhance readability and refine the language. Following the use of this tool, the author thoroughly reviewed and edited the content as necessary. The author takes full responsibility for the publication's content.

Bibliography

[1] Different styles of unicycling, August 2023. [Online; accessed 24-Aug-2023]. URL: https://www.unicycle.co.uk/faq/article/different-styles-of-unicycling.

[2] Kobi Shikar. Transwheel futuristic delivery robot in urban environment, August 2023. [Online; accessed 24-Aug-2023]. URL: https://www.tuvie.com/transwheel-futuristic-delivery-robot-in-urban-environment/?utm_content=cmp-true.

[3] J.F. de Vries. Redesign & implementation of a moment exchange unicycle robot. Master's thesis, University of Twente, October 2018. URL: http://essay.utwente.nl/76751/.

[4] Ming-Tzu Ho, Yusie Rizal, and Yi-Lung Chen. Balance control of a unicycle robot. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pages 1186–1191. IEEE, 2014.

[5] Shweda Mohan, JL Nandagopal, and S Amritha. Decoupled dynamic control of unicycle robot using integral linear quadratic regulator and sliding mode controller. *Procedia Technology*, 25:84–91, 2016.

[6] Yan Li, Jae-Oh Lee, and Jangmyung Lee. Attitude control of the unicycle robot using fuzzy-sliding mode control. In *Intelligent Robotics and Applications: 5th International Conference, ICIRA 2012, Montreal, QC, Canada, October 3-5, 2012, Proceedings, Part III 5*, pages 62–72. Springer, 2012.

[7] Seong I Han and Jang M Lee. Balancing and velocity control of a unicycle robot based on the dynamic model. *IEEE Transactions on Industrial Electronics*, 62(1):405–413, 2014.

[8] M Anfa'ur Rosyidi, Eko Henfri Binugroho, S Ekti Radin Charel, R Sanggar Dewanto, and Dadet Pramadihanto. Speed and balancing control for unicycle robot. In *2016 International Electronics Symposium (IES)*, pages 19–24. IEEE, 2016.

[9] S Majima, T Kasai, and T Kadohara. A design of a control method for changing yaw direction of an underacuatted unicycle robot. In *TENCON 2006-2006 IEEE Region 10 Conference*, pages 1–4. IEEE, 2006.

[10] Shweda Mohan, JL Nandagopal, and S Amritha. Coupled dynamic control of unicycle robot using integral linear quadratic regulator and sliding mode controller. *Materials Today: Proceedings*, 5(1):1447–1454, 2018.

[11] Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.

[12] Axel Ek. Neural network based control design for a unicycle system. Master's thesis, Linköping University, 2023.

[13] SeungYoon Choi, Tuyen P Le, Quang D Nguyen, Md Abu Layek, SeungGwan Lee, and TaeChoong Chung. Toward self-driving bicycles using state-of-the-art deep reinforcement learning algorithms. *Symmetry*, 11(2):290, 2019.

[14] Kanghui He, Chaoyang Dong, An Yan, Qingyuan Zheng, Bin Liang, and Qing Wang. Composite deep learning control for autonomous bicycles by using deep deterministic policy gradient. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 2766–2773. IEEE, 2020.

[15] Xianjin Zhu, Yang Deng, Xudong Zheng, Qingyuan Zheng, Zhang Chen, Bin Liang, and Yu Liu. Online series-parallel reinforcement-learning- based balancing control for reaction wheel bicycle robots on a curved pavement. *IEEE Access*, 11:66756–66766, 2023.

[16] E. Dannenberg. Redesign and control of a Moment Exchange Unicycle Robot 'MEUR'. Bachelor's thesis, University of Twente, 2017.

[17] A.A. Pleijsier R.C. Swiersema R. Giesen, T.H. Hoksbergen and K. Ywema. PD control of a self-balancing moment exchange unicycle robot based on multiple theoretical models. Bachelor's thesis, University of Twente, 2016.

[18] Zhong-Ping Jiang, Tao Bian, and Weinan Gao. Learning-based control: A tutorial and some recent results. *Foundations and Trends® in Systems and Control*, 8(3):176–284, 2020. URL: http://dx.doi.org/10.1561/2600000023, doi:10.1561/2600000023.

[19] What is reinforcement learning?, January 2024. [Online; accessed 08-Jan-2024]. URL: https://nl.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html.

[20] RL - Policy Gradient, 2018. [Online; accessed 24-March-2024]. URL: https://www.52coding.com.cn/2018/01/06/RL%20-%20Policy%20Gradient/.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 1998.

[22] Sergios Karagiannakos. The idea behind actor-critics and how a2c and a3c improve them, 2018. [Online; accessed 04-April-2024]. URL: https://theaisummer.com/Actor_critics/.

[23] 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.

[24] Jesse Clifton and Eric Laber. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7:279–301, 2020.

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[26] The essential guide to neural network architectures, January 2024. [Online; accessed 10-Jan-2024]. URL: https://evbn.org/the-essential-guide-to-neural-network-architectures-1677902860/.

[27] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[28] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[29] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[30] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[31] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[32] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

[33] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Yun-Nung Chen, and Kam-Fai Wong. Adversarial advantage actor-critic model for task-completion dialogue policy learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6149–6153. IEEE, 2018.

[34] Peter Ahrendt. The multivariate gaussian probability distribution. *Technical University of Denmark, Tech. Rep*, page 203, 2005.

[35] Python Software Foundation. Welcome to python.org. URL: https://www.python.org/.

[36] Tensorflow.org. Tensorflow. URL: https://www.tensorflow.org/.

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

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vin-

cent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[39] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[40] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[41] Katherine Li. Vanishing and exploding gradients in neural network models: Debugging, monitoring, and fixing, 2023. [Online; accessed 28-March-2024]. URL: https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing.

[42] Layer weight initializers, 2024. [Online; accessed 25-March-2024]. URL: https://keras.io/api/layers/initializers/.

[43] Andrew Ng. Xavier initialization and regularization, 2022. [Online; accessed 24-March-2024]. URL: https://cs230.stanford.edu/section/4/#xavier-initialization.