RL-MPC for Autonomous Greenhouse Control

Combining RL and MPC to maximise the economic benefit of a greenhouse

Systems and Control Masters Thesis

Murray Harraway

Delft University of Technology

TU Delft

RL-MPC for Autonomous Greenhouse Control

Combining RL and MPC to maximise the economic benefit of a greenhouse

by

Murray Harraway

Student Name	Student Number
Murray Harraway	5827973

Supervisor:	Tamas Keviczky
Daily Supervisor:	R.D. McAllister
Project Duration:	November 2023–August 2024
Faculty:	Delft Center for Systems and Control, Delft

TU Delft

Preface

A preface...

Murray Harraway
Delft, July 2024

Abstract

- Background
- Aim
- Method
- Results
- Conclusion

Contents

Nomenclature

If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

Abbreviations

Abbreviation	Definition
ISA	International Standard Atmosphere
...	

Symbols

Symbol	Definition	Unit
V	Velocity	[m/s]
...		
ρ	Density	[kg/m^3]
...		

1

Introduction

The world population is set to increase to a staggering 10 billion people in the year 2050 [1], substantially increasing food demand. Currently, 800 million people are chronically hungry, with 2 billion people suffering from micronutrient deficiencies [2]. The situation is compounded by the anticipated rise in food demand, which is expected to increase from 30% to 62% between 2010 and 2050, resulting in 30% of the population being at risk of hunger [3]. Therefore, there is a pressing need to enhance food production by at least 70% [4]. Despite significant investments in increasing food productivity, major difficulties such as food losses, waste, and climate change continue to persist[2]. The agriculture space has drastically increased to meet these food demands [5]; however, this increase means the sector accounts for almost 15% of the world's energy and more than 70% of water consumption [4]. Although greenhouses have been extensively used to combat these problems and have been shown to reduce the environmental burden compared to typical open-land production [6], they still consume about ten times more energy than traditional farming [4] due to the operating costs of a greenhouse. Moreover, growers are under increasing pressure to adopt more effective growing methods with the soaring operating energy costs associated with greenhouses and a global trend indicating an increase in gas and electricity prices [7]. As a result, agreement policies have been signed to reduce the $C0_2$ emissions of these greenhouses to an acceptable level [8].

Smart greenhouses are designed to enhance crop yield per hectare using climate-controlled environments [9]. These smart greenhouses are essential in combating the degrading effects of climate change on crop quality and yield. However, efficiently maintaining such an environment requires advanced control methods, especially for economic profit. These control methods must be able to adjust factors such as temperature, humidity, lighting, and C02 levels to accommodate ideal conditions for crop growth [10] whilst keeping energy costs at a minimum. Growing crops in a controlled environment can ensure the extension of their growing season and protection from outside temperature and weather changes. The advent of smart and advanced greenhouses necessitates skilled labour for operation, yet there is a scarcity of qualified personnel [11]. Coupled with the escalating labour costs, the move to autonomous greenhouses is attractive.

It is common to use controllers, such as PID, to control actuators, adjusting conditions based on set points manually specified by the grower [12]. Although these techniques exist, often called automatic greenhouses, growing crops still relies heavily on the grower's expertise. Due to the numerous factors that affect crop growth, determining the optimal set-points becomes highly complex. The complexity is further intensified by the fact that the development of a plant is highly influenced by the control inputs taken days or even weeks in advance. Therefore, it is crucial to choose control inputs strategically in order to maximise future rewards such as economic profit. Control strategies such as reinforcement learning (RL) and model predictive control (MPC) can be implemented [12] to achieve this. Both strategies provide optimal control to pursue the same goal, optimising a reward/cost function. This cost function gives an indication of the quality of a particular action or decision, and in this thesis,

it determines the extent to which an action is favourable or unfavourable in relation to its economic benefit. Both control schemes have advantages and disadvantages. However, there is a notable similarity between the two, suggesting that combining them could lead to a more efficient solution for autonomous greenhouse control.

Although RL and MPC are both used for optimal control, RL focuses on learning from interactions with an environment to maximise long-term rewards. At the same time, MPC leverages model-based predictions and optimisation to determine optimal control actions over a finite time horizon. Several methods seek to integrate the two control schemes, shedding light on the strengths and weaknesses of the resulting controller concerning its specific application. There are two main methods for combining RL and MPC: using MPC as the function approximator for the RL agent, or modifying the MPC's objective function and terminal constraints to incorporate RL knowledge. The latter approach allows the learned knowledge from RL to guide and improve the performance of the MPC controller, potentially leading to more effective and adaptive control strategies. This approach will be investigated in this thesis.

1.1. Problem Statement

The ability of RL to learn from interactions with a highly complex environment, even in the presence of uncertainty, leads to its ability to learn a policy that aims to be optimal within that particular environment. However, the quality of control is strongly influenced by the training of the RL algorithm. Moreover, RL does not directly impose state constraints. While it is possible to indirectly incorporate these constraints in the reward function with penalty functions, such an approach does not ensure that the optimal policy obtained will always adhere to these constraints. Moreover, the value function obtained using RL is only an approximation. This becomes an issue when the problem at hand is safety critical. Finally, RL faces a limitation in online flexibility due to its reliance on a feed-forward pass for policy evaluation.

In contrast to the policy obtained using RL, the quality of the policy obtained using MPC is subject to the accuracy of the prediction model, which is often simplified to reduce the computational burden, especially with non-convex dynamics, which may lead to sub-optimal control. Nevertheless, MPC is recognised for its easier implementation, superior constraint handling, and sample-efficiency as compared to RL. However, model mismatch and unforeseen uncertainties in forecasted disturbances can lead to a significant deterioration in the MPC's performance.

Arguably, the most crucial characteristic of both control strategies lies in their respective prediction horizon. Both controllers use future information to determine the optimal control actions. MPC achieves this by employing explicit optimisation over a finite prediction horizon, while RL explores and interacts with its environment to optimise for immediate and long-term rewards. Hence, a shortcoming of MPC is the finite prediction horizon. This limitation becomes even more pronounced when dealing with sparse rewards and slow system dynamics. In such cases, actions taken at the current time step may only yield rewards past the prediction horizon, causing the MPC controller to be myopic. It is possible to counteract this drawback with an extended prediction horizon, but to the detriment of simplicity and computational efficiency of the controller. Moreover, with no terminal constraint or cost function, there are no closed-loop performance guarantees for an MPC that optimises for economic benefit (EMPC). Importantly, two main methods exist to assure closed-loop performance: to use a sufficiently large horizon or the application of an appropriate terminal constraint and terminal cost function.

While MPC's prediction horizon is finite, RL uses a discounted infinite prediction horizon, allowing the RL agent to weigh the benefit of future rewards against current actions. The exploration present in RL will enable it to discover patterns and optimal policies that a typical MPC might not be able to achieve, particularly in environments characterised by non-linear dynamics. A synergistic approach to combining the two control strategies would be to have an MPC controller optimise a short prediction horizon while propagating future information provided by RL. This integration of the two controllers is further justified in the context of greenhouse dynamics, where actions executed at the current time step may result in rewards that manifest over the long term. The terminal cost function in the MPC formulation, which must encapsulate information beyond the prediction horizon, underscores the clear connection with the value function obtained through RL to supply the information required

for achieving the desired system performance. Notably, this may also be viewed as unrolling the value function and performing an N-step look-ahead minimisation on the resulting equation [13]. This is a common application of improving on the policy that generated the value function by value iterations. Nevertheless, incorporating the learned value function (typically a neural network) in the MPC's formulation introduces additional complexity to the problem due to its highly non-linear nature. Consequently, a naive implementation of this control strategy could have severe detrimental effects. Alternatively, the RL policy can also provide the MPC with a terminal region or constraint to guide it towards a control policy that is closer to the optimal solution more effectively. Ultimately, the myopic nature of MPC for optimising economic benefit can be counteracted by providing knowledge of the future, through a cost function and terminal constraints as provided my RL. This approach enables the use of a short prediction horizon to meet computational requirements, while avoiding myopic decision-making. Therefore, in the development of the RL-MPC algorithm, for greenhouse control, the following research questions will be answered:

- *How does the economic performance of the RL-MPC algorithm compare to the standalone RL and MPC algorithms?*
  - *In a deterministic environment*
  - *In a stochastic environment*
- *What modifications and/or approximations can be employed to reduce the computational time of the RL-MPC algorithm?*

1.2. Thesis Contribution

Answering the above questions leads to several contributions in the field. Notably, among the existing works, there is a scarcity of algorithms that independently train a RL agent and subsequently employ MPC for the N-step look-ahead minimization. Specifically, none of the algorithms identified in the related literature use MPC for the specified N-step look-ahead minimisation on the pre-trained value function during online play, particularly in scenarios involving continuous state and action spaces. Lastly, it is worth noting that all the proposed RL-MPC algorithms in recent and related developments are applied in the context of set-point regulation or trajectory tracking and are not explicitly geared to maximising economic benefits. Therefore, the main contribution of this thesis will involve developing and implementing a framework that incorporates a learned value function and terminal constraints provided by RL into a economic non-linear model predictive controller (ENMPC) for a continuous state and action space and making such an algorithm generate on-time control actions. A greenhouse is a suitable environment and system for developing such an algorithm due to its non-linear dynamics, continuous state and action spaces and slow dynamics that result in sparse rewards. Importantly, the objective of a greenhouse is to maximise profits from the cultivation and sale of crops. In the context of greenhouse operations, it is noteworthy that the primary aim is to maximise profits. This goal typically does not entail tracking specific setpoints for crop growth.

1.3. Recent and Related Developments

Various literature explores the implementation of RL and MPC such as [14]–[18] whereas [13], [15], [19], [20] examine the theoretical background of such a controller. Most notably, the works in Arroyo, Manna, Spiessens, *et al.* [14], Sikchi, Zhou, and Held [18], Bertsekas [19], and Lin, Sun, Xia, *et al.* [20] are the most similar to what is proposed in this thesis, whereby an RL agent is trained, and the resulting learned value function is unrolled with the Bellman equation. The optimal action is computed using an N-step look-ahead minimisation on the unrolled equation during online play. Works from [14], [15], [20] are the only ones incorporating an MPC for the N-step look-ahead minimisation. These authors propose that the reinforcement learning process could be assisted with MPC and show the implementation of such an algorithm, but doing this might impact the agents exploratory nature. Nonetheless, in all cases, the various RL-MPC algorithms have shown to outperform its RL counterpart and in most cases, its MPC counterpart to. Furthermore, a concise explanation will be provided to differentiate this thesis from the concept of employing MPC as a function approximator, which is a common application of a RL-MPC algorithm. This aims to clarify what the thesis does not focus on. As discussed in section 2.2,

the DQN is usually implemented as a neural network and is parameterized by a θ. The approximated value of a state s, parameterised by a weight vector θ is then given as $\hat{v}(s, \theta) = v^*(s)$ where $\hat{v}$ [17]. However, it is possible to use an MPC scheme instead of a neural network to facilitate parametrisation for approximating the value function and policy. This is the goal of using MPC as a function approximator for RL. This thesis does not explore this concept. Instead, it explores the design and implementation of an (E)NMPC algorithm that uses the value function learned by RL in its optimal control problem formulation to propagate information beyond the prediction horizon.

1.4. Thesis Outline

This thesis begins by discussing the background knowledge in Chapter 2. This chapter presents the greenhouse model that will be used and the rationale behind its selection. It also outlines the optimisation objective of the RL, MPC, and combined RL-MPC controllers. In addition, this chapter will present the concepts of RL, MPC, and RL-MPC. Chapter 3 explores establishing and training the RL agent in two environments: one without uncertainty (the nominal case) and one with uncertainty (the stochastic case). The chapter also examines the performance of the resulting agent in its respective environments. Moreover, this chapter explores the process of training accurate value functions using a fixed RL policy and discusses the obtained outcomes. Chapter 4 entails the formulation of the optimal control problem for the MPC and discusses the performance of the MPC in an environment with and without uncertainty. Chapter 5 examines the various implementations of the RL-MPC controller and evaluates their performance on the nominal case. Chapter 6 applies the best RL-MPC implementations from Chapter 5 to a stochastic environment and discusses the findings.Chapter 7, explores how the RL-MPC implementation can be optimised for computational efficiency and emphasises the significance of these optimisations. The thesis concludes with a presentation of the conclusion and future research in Chapter 8, and a draft research paper provided in Appendix C.

2

Background

This chapter provides an overview of the relevant background information to this thesis. This chapter centres on the chosen greenhouse model and the rationale behind its selection. It examines the principles of RL and MPC, and finally provides an overview of combining these algorithms.

2.1. Greenhouse Model

The accuracy and complexity of a model directly impacts the quality of the control policy. The dynamics of a greenhouse can be characterised by various modelling approaches depending on its application. Methods such as computational fluid dynamics (CFD) are used to develop numerical models of the indoor climate of a greenhouse using partial differentiable equations. In recent years, CFD simulations have become increasingly more accurate in simulating indoor greenhouse climates [21]; however, these models are highly complex and require significant computational resources in simulation and optimisation tasks. Their complexity makes them intractable for mathematical solvers such as MPC, and they present substantial challenges to learning model-free methods, resulting in a demanding and laborious training process [22]. Simplified models, such as mechanistic models, of greenhouse dynamics are developed by assuming homogeneity in the greenhouse climate and assumptions about CO_2 and humidity are made to yield a model conducive to control applications, as discussed in Jansen [22] and Lopez-Cruz, Fitz-Rodríguez, Raquel, *et al.* [23]. These mechanistic process models are often derived from first principles and use ordinary differential of difference equations to represent the system dynamics. While these mechanistic models sacrifice some accuracy in representing the system, they significantly reduce computational demands, often making them tractable for mathematical solvers. In contrast, data-driven models such as in Gong, Yu, Jiang, *et al.* [24] and Maestrini, Mimić, Van Oort, *et al.* [25] have also been used to create a black box model of the greenhouse dynamics. Although such models may yield accurate results, they do so only in the environment from where the data originates. Mechanistic process models often generalise better to different environments and situations. While such black box models could be used for RL, they could pose challenges for mathematical solvers due to their potential complexity and unknown mathematical makeup.

Although accurate models may seem appealing, they are often complex and entail a high-dimensional state space. Both factors adversely affect model-free RL algorithms and MPC, making such models impractical for developing optimal control policies. This arises because, in the case of MPC, finding a tractable solution in the given time constraints may prove challenging. This can results in failure to meet timing requirements and constraints, or find no solution. Similarly, in model-free RL algorithms, using highly accurate models could destabilise the learning procedure, making it a more complex and time-consuming task to learn a policy [26], [27]. Therefore, for optimal control, it is desirable to use a low-dimensional simplified model whereby the most critical dynamics of the system are still captured. Moreover, the control law obtained from the simplified model may be adjusted and applied to the more accurate models to validate them. However, if the control law is unsatisfactory, the simplified model must be adapted, and a new control policy obtained. Often, this leads to an iterative cycle of updating the model and validating the control policy until a satisfactory performance of the high-accurate model

is obtained [28]. Since this thesis focuses on the impact and implementation of merging RL and MPC for optimal control, designing and developing a model is outside the scope. Therefore, a validated simplified mechanistic model of a greenhouse with lettuce crops will be selected. Due to the relative simplicity of lettuce crop models, they find utility alongside a greenhouse model that assumes an indoor homogeneous climate. A model that encapsulates the fundamental dynamics of the greenhouse and crops is detailed in Henten [29] and has been explicitly developed for optimal control applications. More importantly, the propsed model in Henten [29] has been validated in Van Henten [30], and has been extensively used in obtaining optimal control policies such as in Morcego, Yin, Boersma, *et al.* [9], Lubbers [17], Jansen [22], Van Straten and Henten [31], and Ghoumari, Tantau, and Serrano [32]. Therefore, the choice was made to use the model suggested in Henten [29] for this thesis.

2.1.1. Model Description

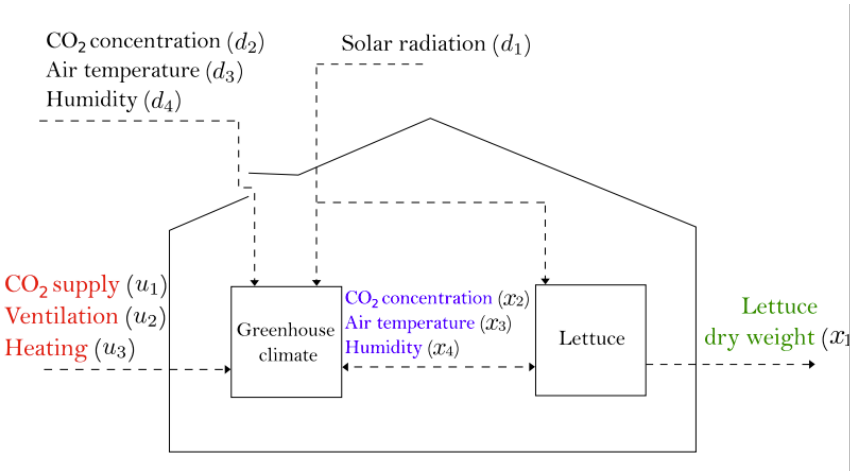

Figure 2.1: Graphical representation of greenhouse crop production [29]

Figure 2.1 displays a graphical representation of the greenhouse model from the works of Henten [29]. The control inputs are highlighted in red, the states of the indoor climate in blue, the state of the crop in green and the external disturbances in black. The states of the system, control inputs, disturbances and outputs are described in Equation 2.1.

$$\begin{aligned} x(k) &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T \\ u(k) &= \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T \\ d(k) &= \begin{bmatrix} d_1 & d_2 & d_3 & d_4 \end{bmatrix}^T \\ y(k) &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T \end{aligned} \tag{2.1}$$

The state of the system includes crop dry mass, indoor $C0_2$ density, temperature and the absolute humidity. Crop dry mass is used since it is a more reliable and accurate measure of the plant's biomass. Unlike fresh weight, which varies with the plant's water content and is influenced by weather conditions, crop dry mass provides a more stable and consistent metric. The control vector encapsulates the $C0_2$ injection and ventilation rate and the heating supply. Furthermore, the disturbances from the weather include the incoming solar radiation, outside $C0_2$ density, temperature and the outside humidity. The measured output is essentially the same as the state of the system; however, the units of the indoor $C0_2$ density (y_2) and relative humidity (y_4) differ in that they report in units used in standard measurement sensors. The non-linear continuous-time dynamics is represented by Equation 2.2

$$\begin{aligned} \dot{x} &= f(x(t), u(t), d(k), p) \\ y(k) &= g(x(k), u(k), d(k), p) \end{aligned} \tag{2.2}$$

where $p = \mu_p$ and μ_p is the value of all the parameters used in the model and displayed in Table 2.2, x is the model state, y is the measurement outputs and d is the weather disturbance. The next section examines the model's state equations in more depth.

2.1.2. Model State Equations

State Equations The model described in [29] is fully described by the crop dry weight, temperature, humidity and $C0_2$ levels of the indoor climate. All variables and their units are given in Table 2.1 and model constants in Table 2.2. These states are modelled using a set of ordinary differentiable equations given below:

$$\frac{dx_1}{dt} = c_{\alpha\beta}\phi_{phot,c}(t) - \phi_{resp,d}(t) \tag{2.3}$$

$$\frac{dx_2}{dt} = \frac{1}{c_{cap,c}}(-\phi_{phot,c}(t) + \phi_{resp,c}(t) + u_{C0_2}(t)\cdot 10^{-6} - \phi_{vent,c}(t)) \tag{2.4}$$

$$\frac{dx_3}{dt} = \frac{1}{c_{cap,q}}(u_q(t) - Q_{vent,q}(t) + Q_{Io,q}(t)) \tag{2.5}$$

$$\frac{dx_4}{dt} = \frac{1}{c_{cap,h}}(\phi_{transp,h}(t) - \phi_{vent,h}(t)) \tag{2.6}$$

where parameter values,units and descriptions can be found in Table 2.2. The process equations, denoted as $\phi(\cdot)$ are described in Table 2.1 and are represented by the following set of equations:

$$\phi_{phot,c}(t) = (1 - e^{-C_{LAI,d}x_d(t)})\frac{c_{Io}^{phot} d_{Io}(t)\cdot\phi(t)}{c_{Io}^{phot} d_{Io}(t) + \phi(t)} \tag{2.7}$$

$$\phi(t) = (-c_{C0_2,1}^{phot} x_T(t)^2 + c_{C0_2,2}^{phot} x_T(t) - c_{C0_2,3}^{phot})(x_{C0_2}(t) - c^{phot}) \tag{2.8}$$

$$\phi_{resp,d}(t) = c_{resp,d}\cdot\phi_{resp}(t) \tag{2.9}$$

$$\phi_{resp}(t) = x_d(t)\cdot c_{Q_{10},resp}^{(x_T(t)-25)/10} \tag{2.10}$$

$$\phi_{resp,c}(t) = c_{resp,C0_2}\cdot\phi_{resp}(t) \tag{2.11}$$

$$\phi_{vent,c}(t) = (u_v(t)\cdot 10^{-3} + c_{leak})(x_{C0_2}(t) - d_{C0_2}(t)) \tag{2.12}$$

$$Q_{vent,q}(t) = (c_{cap,v}u_v(t) + c_{go})(x_T(t) - d_T(t)) \tag{2.13}$$

$$Q_{I_o,q}(t) = c_{og}^{rad} d_{I_o}(t) \tag{2.14}$$

$$\phi_{transp,h}(t) = c_{ca}^{evap}(1 - e^{-C_{LAI,d}x_d(t)})\cdot\left(\frac{c_{H_2O,1}^{sat}}{c_R(x_T(t) + c_T)} e^{\frac{c_{H_2O,2}x_T(t)}{x_T(t)+c_{H_2O,3}}} - x_h(t)\right) \tag{2.15}$$

$$\phi_{vent,h}(t) = (u_v(t)\cdot 10^{-3} + c_{leak})(x_h(t) - d_h(t)) \tag{2.16}$$

Output Measurement Equations The output measurements for the indoor humidity and $C0_2$ levels are reported in different units via the function $g_{C0_2}(\cdot)$ and $g_h\cdot$ respectively. The output measurements for crop dry mass and indoor temperature correspond to the state variables, as illustrated below:

$$\begin{aligned} y_1(t) &= x_1(t) \\ y_2(t) &= g_{C0_2}(x_3(t), x_2(t)) \\ y_3(t) &= x_3(t) \\ y_4 &= g_h(x_3(t), x_4(t)) \end{aligned} \tag{2.17}$$

$$g_{C0_2}(z_T(t), z_{C0_2}(t)) = 10^3 \cdot \frac{R(z_T(t) + c_T)}{PM_{C0_2}} \cdot z_{C0_2}(t) \tag{2.18}$$

$$g_h(z_T(t), z_h(t)) = \frac{R(z_T(t) + c_T)}{c^{sat}_{H_20,4} \cdot \exp(\frac{c^{sat}_{H_20,5} z_T(t)}{z_T(t) + c^{sat}_{H_20,6}})} \cdot z_h(t) \tag{2.19}$$

where $R, c_T, P, M_{C0_2}, c_{H_20,4}, c_{H_20,5}, c_{H_20,6}$ are constants and their descriptions and units given in Table 2.2. The greenhouse model was discretized in time to facilitate the use of MPC and RL. The fourth-order Runge Kutta method was used and results in the discrete model below:

$$\begin{aligned} x(k+1) &= f(x(k), u(k), d(k), p) \\ y(k) &= g(x(k), p) \end{aligned} \tag{2.20}$$

[33] recommends discretising the van Henten model using a time-step between 15 minutes and 1 hour. Therefore, a time interval of 30 minutes was selected for this study. The initial interval of 30 minutes was initially set at 15 minutes. However, this decision was revised due to the excessive computational burden it imposed on the MPC solver, with minimal benefits. Increasing the time step enables an exponential speedup of the simulation of the growing period, while causing only minor deterioration in performance. Consequently, a larger quantity of training episodes could be used to train the RL.

Category	Symbol	Description	units
State Variables	x_1	crop dry matter	$kg \cdot m^{-2}$
	x_2	Indoor $C0_2$ density	$kg \cdot m^{-3}$
	x_3	Indoor air temperature	$°C$
	x_4	Indoor absolute humidity content	$kg \cdot m^{-3}$
Control Inputs	u_1	$C0_2$ injection rate	$mg \cdot m^{-2} \cdot s^{-1}$
	u_2	Ventilation rate	$mm \cdot s^{-1}$
	u_3	Heating supply	$W \cdot m^{-2}$
Disturbance	d_1	Outside irradiation	$W \cdot m^{-2}$
	d_2	Outdoor $C0_2$ density	$kg \cdot m^{-3}$
	d_3	Outside ambient air temperature	$°C$
	d_4	Outside absolute humidity content	$kg \cdot m^{-3}$
Outputs	y_1	Lettuce dry weight	$kg \cdot m^{-2}$
	y_2	Indoor $C0_2$ concentration	ppm
	y_3	Indoor air temperature	$°C$
	y_4	Indoor relative humidity content	%
Processes	$\phi_{phot,c}$	Gross canopy rate	$kg \cdot m^{-2} \cdot s^{-1}$
	$\phi_{resp,d}$	Respired dry matter from maintenance respiration of the crop	$kg \cdot m^{-2} \cdot s^{-1}$
	$\phi_{resp,c}$	Crop $C0_2$ respiration rate	$kg \cdot m^{-2} \cdot s^{-1}$
	$\phi_{vent,c}$	Mass exchange of $C0_2$ between indoor and outdoor climate	$kg \cdot m^{-2} \cdot s^{-1}$
	$Q_{vent,c}$	Heat energy exchange between indoor and outdoor climate	$W \cdot m^{-2}$
	$Q_{I_o,q}$	Incoming heat energy from outside irradiation	$W \cdot m^{-2}$
	$\phi_{transp,h}$	Canopy transpiration rate	$kg \cdot m^{-2} \cdot s^{-1}$
	$\phi_{vent,h}$	Mass exchange of water vapour between indoor and outdoor climate	$kg \cdot m^{-2} \cdot s^{-1}$

Table 2.1: Model Variables

Symbol	Description	Value	units
$c_{\alpha\beta}$	Yield factor	0.544	-
$c_{cap,c}$	Volumetric $C0_2$ capacity of indoor air	4.1	$m^3[\text{air}] \cdot m^{-2}[\text{gh}]$
$c_{cap,q}$	Effective heat capacity of indoor air	30000	$J \cdot m^{-2}[\text{gh}] \cdot^{\circ} C$
$c_{cap,h}$	Volumetric humidity capacity of indoor air	4.1	$m^3[\text{air}] \cdot m^{-2}[\text{gh}]$
$c_{LAI,d}$	Effective canopy surface	53	$m^{-2}[\text{L}] \cdot kg^{-1}[\text{dw}]$
$c_{I_o}^{phot}$	Light use efficiency	$3.55 \cdot 10^{-9}$	$kg[C0_2] \cdot J^{-1}$
$c_{C0_2,1}^{phot}$	Influences temperature gross canopy photosynthesis	$5.11 \cdot 10^{-6}$	$m \cdot s^{-1} \cdot^{\circ} C^{-2}$
$c_{C0_2,2}^{phot}$	Influences temperature gross canopy photosynthesis	$2.30 \cdot 10^{-4}$	$m \cdot s^{-1} \cdot^{\circ} C^{-1}$
$c_{C0_2,3}^{phot}$	Influences temperature gross canopy photosynthesis	$6.29 \cdot 10^{-4}$	$m \cdot s^{-1}$
c^{phot}	Carbon dioxide compensation point	$5.2 \cdot 10^{-5}$	$kg[C0_2] \cdot m^{-3}[\text{air}]$
$c_{resp,d}$	Respiration rate of the dry crop matter	$2.65 \cdot 10^{-7}$	s^{-1}
$c_{Q_{10,resp}}$	Maintenance respiration factor	2	-
$c_{resp,c}$	$C0_2$ release rate factor from respiration	$4.87 \cdot 10^{-7}$	s^{-1}
c_{leak}	Greenhouse cover ventilation leakage	$7.5 \cdot 10^{-6}$	$m \cdot s^{-1}$
$c_{cap,v}$	Heat capacity of indoor temperature per volume	1290	$J \cdot m^{-3}[\text{gh}] \cdot^{\circ} C^{-1}$
c^{go}	Heat transmission through cover factor	6.1	$W \cdot m^{-2}[\text{gh}] \cdot^{\circ} C^{-1}$
c_{og}^{rad}	Solar heat load coefficient	0.2	-
c_{ca}^{evap}	Vapour mass transfer factor between leaf and air	$3.6 \cdot 10^{-3}$	$m \cdot s^{-1}$
$c_{H_20,1}^{sat}$	Influences water vapour saturation point	9348	$J \cdot m^{-3}$
$c_{H_20,2}^{sat}$	Influences water vapour saturation point	17.4	-
$c_{H_20,3}^{sat}$	Influences water vapour saturation point	239	$^{\circ}C$
$c_{H_20,4}^{sat}$	Influences water vapour saturation point	610.48	Pa
$c_{H_20,5}^{sat}$	Influences water vapour saturation point	17.2694	-
$c_{H_20,6}^{sat}$	Influences water vapour saturation point	238.3	$^{\circ}C$
c_R	Gas constant	8314	$J \cdot K^{-1} \cdot \text{kmol}^{-1}$
c_T	For conversion between Kelvin and Celsius	273.15	K
R	Molar gas constant	8.3144598	$J \cdot \text{mol}^{-1} \cdot K^{-1}$
P	Pressure at 1 atmospheric pressure	101325	Pa
M_{C0_2}	Molar mass of $C0_2$	0.0441	$kg \cdot \text{mol}^{-1}$

Table 2.2: Model Constants

2.1.3. Model uncertainty

In reality, uncertainty is present in all aspects of the model. In the case of the greenhouse environment, uncertainty may arise in the weather predictions, there may be measurement noise on the outputs and the control inputs may not be exact. Additionally, the simplified model leads to a model mismatch, which may result in significant prediction errors. As was done in [17], [34], the source of uncertainty in the stochastic environment is modelled as parametric uncertainty, which aims to capture all the uncertainty in the greenhouse environment. Moreover, in the objective function (discussed further in subsection 2.1.4), the pricing of energy and the lettuce may also be stochastic. However it is assumed to be fixed, along with the weather predictions. This thesis does not focus on accurately modelling the uncertainty in a greenhouse crop environment. However, it is essential to observe the impact of uncertainty on the policy generated for each algorithm. This will help determine the performance of these controllers in a more realistic scenario. Thus, the parameters of the environment are represented as uncertain with well-defined probabilistic properties. It is assumed that the uncertain parameters, $\hat{p}$, follow the uniform probabilistic distribution:

$$\hat{p} \sim U(\mu_p, \delta_p) \tag{2.21}$$

where $\mu_p = p$ is the mean value of the parameters (shown in Table 2.2) and δ_p are expressed as a percentage where the upper and lower bounds of the distribution is expressed as $p(1 - \delta_p)$ and $p(1 + \delta_p)$. It must be noted that since all parameters are perturbed, including parameters of the output measurement equations, output noise is also introduced. The uniform distribution was chosen for its

higher level of aggression compared to the normal distribution. While it may not accurately represent the uncertainty at hand, it is useful for evaluating the potential variability and risks associated with the various policies, especially at higher degrees of uncertainty. Introducing this uncertainty results in the following system dynamics:

$$\begin{aligned} \hat{x}(k+1) &= f(\hat{x}(k), u(k), d(k), \hat{p}(k)) \\ \hat{y}(k) &= g(\hat{x}(k), \hat{p}(k)) \end{aligned} \tag{2.22}$$

where the states and output measurements of the system are uncertain and $\hat{p}(k)$ denotes the uncertain parameters and is resampled at every time step.

2.1.4. Optimisation Goal

Maximising the economic profit of the greenhouse is highly desirable. This concept involves maximising the lettuce's size while minimising the resources used throughout its entire growth period. The optimisation goal can be modeled and is referred to by the economic profit indicator (EPI), as demonstrated below:

$$EPI = \sum_{k=ts}^{tf} (c_{p_1} \cdot u_1(k) + c_{p_2} \cdot u_3(k)) \cdot \Delta t - c_{p_3} \cdot y_1(tf) \tag{2.23}$$

where t_s denotes the starting time, t_f the end time, Δt the time interval,c_{p1}, c_{p2} the pricing coefficients of injecting $C0_2$, and heating, respectively and c_{p3} as the price of lettuce. These pricing factors are shown in **??**. It is important to highlight that ventilation, such as opening windows, does not incur any costs and, therefore, does not have a pricing factor. Although it is also possible to optimise tf, i.e. when is the best time to sell the lettuce, it is decided to keep the growing period fixed. The growing period for lettuce typically falls within 30 to 50 days. Therefore, a fixed growing period of 40 days was selected. As a result, over 40 days (1 episode or 1 complete simulation), there is a total of:

$$k_{total} = \frac{40 \frac{days}{growing\ period} \cdot 24 \frac{hrs}{day} \cdot 60 \frac{min}{hr}}{30 \frac{min}{timestep}} = 1920 \frac{time\ steps}{growing\ period}$$

Moreover, this optimisation goal is subject to constraints where the control constraints are:

$$\begin{aligned} u_{min} &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \\ u_{max} &= \begin{pmatrix} 1.2 & 7.5 & 150 \end{pmatrix}^T \\ \delta u_{max} &= \frac{1}{10} u_{max} \end{aligned} \tag{2.24}$$

and the state constraints are defined as:

$$\begin{aligned} y_{min} &= \begin{pmatrix} 0 & 500 & 10 & 0 \end{pmatrix}^T \\ y_{max} &= \begin{pmatrix} \infty & 1600 & 20 & 100 \end{pmatrix}^T \\ \delta_u &= \frac{1}{10} u_{max} \end{aligned} \tag{2.25}$$

The values in question were obtained from typical horticultural practices. In many papers, such as Boersma, Sun, and van Mourik [34], the lower and upper boundaries of the indoor temperature are time-varying to aid MPC in making better decisions. It was decided to keep the lower and upper bounds of the temperature constant to give RL and MPC as much freedom for optimizing for an economic benefit.

Symbol	Value	Units
c_{p_1}	$1.906 \cdot 10^{-7} \times 10^{-4}$	$s€ \cdot mg^{-1}$
c_{p_2}	3.558×10^{-8}	$€ \cdot J^{-1}$
c_{p_3}	22.285	$€ \cdot kg^{-1}$

Table 2.3: Pricing factors

The pricing factors accounts for the required conversions from mg to kg and joules to kWh for the $C02$ injection and heating, respectively. Furthermore, c_{p_3} incorporates the factor of harvestable fresh weight, which quantifies the amount of fresh relative to dry weight. Finally, the pricing of the $C0_2$ in $€/kg$, the price of heating in $€/kWh$ and the pricing of lettuce in $€/kg$ is also incorporated as shown in Equation 2.26

$$\begin{aligned} c_{p1} &= C0_{2_{price}} = \frac{0.1906}{10^6} \\ c_{p2} &= \text{Heating}_{price} = \frac{0.1281}{3.6 \cdot 10^6} \\ c_{p3} &= (1 - c_\tau) \cdot c_{fw} \cdot Let_{price} = (1 - 0.07) \cdot 22.5 \cdot 1.065 \end{aligned} \tag{2.26}$$

The prices for $C0_{2_{price}}$, Heating_{price}, and Let_{price} were sourced from [35], while the factors for harvestable fresh weight, c_τ and c_{fw}, were obtained from [29]. While these factors may not accurately reflect the present economic conditions, it is important to keep them constant when comparing the MPC, RL, and RL-MPC algorithms. This ensures a meaningful comparison between algorithms, which aligns with one of the objectives of this thesis. As shown in Equation 2.23, this optimisation goal makes the reward incredibly sparse, making it difficult to optimise directly with both MPC and RL. Consequently, both algorithms use a stage cost received at every time interval, whereby the total sum of these stage costs throughout the growing period corresponds to the optimisation objective in Equation 2.23. Therefore the optimisation goal that RL, MPC and RL-MPC aims to optimise is:

$$\min_{u_1,u_3} \sum_{k=ts}^{tf} (c_{p_1} \cdot u_1(k) + c_{p_2} \cdot u_3(k)) \cdot \Delta t - c_{p_3} \cdot (y_1(k) - y_1(k-1)) \tag{2.27}$$

Section 3.1 and Section 4.1 further discuss the implementation of Equation 2.27 for RL and MPC, respectively.

2.2. Reinforcement Learning

With the surge in AI development, the demand for optimal greenhouse control has brought RL into the spotlight. Although AI algorithms are already used in farming and horticulture, most applications are centred on detecting and protecting plant and crop quality, such as plant stress, pests and disease and weed detection [36]. Moreover, the prediction of crop yield and growth has also been a focus area for AI owing to its unparalleled non-linear approximation capabilities [24]. Additionally, recent advances in RL have displayed its ability to outperform humans in decision-making when presented with sizeable complex problem spaces [37]. Because of RL's ability to evaluate optimal control actions for large state spaces, it has shown great success, such as in Google DeepMind's chess engine, AlphaZero, creating one of the most powerful chess engines in the world. Furthermore, reinforcement learning has demonstrated exceptional proficiency in handling uncertainty and disturbances [38]. This makes it an ideal control strategy for greenhouse management, particularly considering the uncertainties inherent in weather conditions and crop growth.

Recently, research has focused on optimising crop growth and yield in greenhouse control, with studies such as [22], [39]–[44] employing reinforcement learning to address this challenge. This momentum has been fueled by initiatives like the Autonomous Greenhouse Challenge hosted by Wageningen University and Research, first held in 2018, in which the challenge was designed to push the limits of AI

in fresh food production. In this competition, teams vie against each other to employ AI to optimise cucumber crops in a climate-controlled greenhouse. Specifically, deep reinforcement learning policies have been applied and measured against each other, as well as current control strategies and expert growers, showing promising results and, in some cases, outperforming expert growers [35], [40]. As the years have progressed, more teams have succeeded in outperforming expert growers [35], [36]. Such achievements using reinforcement learning in a real-world application have displayed its potential to achieve complete autonomous greenhouse control.

2.2.1. The RL problem

Reinforcement Learning consists of four main components that define its nature: a policy, a reward signal, a value function and a representation of the environment [45]. To further conceptualize the idea of RL, there exists an agent,environment and a reward signal that outlines the goal of the optimization problem (Figure 2.2).This agent explores and takes actions in the environment where at time k follows a policy π_k and the agent receives a reward R_k from the environment based on the action taken and the state of the agent. This policy is defined as a control law that maps the agent's states to probabilities of selecting each possible action that an agent can take [45]. The only objective of the agent is to maximize the cumulative reward of the agent, therefore the agent learns to make decisions and take actions that lead to favourable outcomes, as measured by the received rewards in the environment. Finally, the value function is defined as the expected future rewards based on the current state of the agent. Therefore the reward received is a measure of the immediate desire for an state, and the value function indicates the long-term desirability of a state. [45].

This interaction between agent and environment in Reinforcement Learning is modeled as a Markov Decision Process (MDP). This mathematical framework formalizes the decision process that satisfies the condition whereby future states depend only on the current state and action taken. The agent-environment interaction is as shown in Figure 2.2 and depicts the agents taking actions A_t in the environment and receiving reward R_{t+1} and transitioning to the next state S_{t+1} with probability P at every discrete time step k.

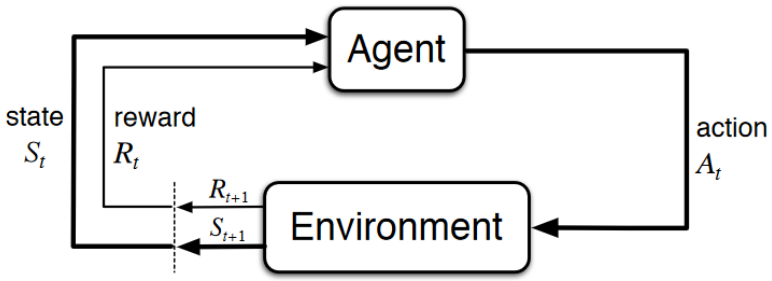

Figure 2.2: Agent-Environment Interface [45]

Furthermore the MDP problem can be characterized as a tuple of states, actions, rewards and a state transition function. Such that:

- states: ($s \in S$) where S is the finite state space.
- Actions ($a \in A$) where A is the finite action space
- Transition probability:$P(s_{t+1}, r|s_t, a_t)$ where $f : X \times U \rightarrow X$ is the state transition function
- Rewards: $r : S \times A \rightarrow \mathbb{R}$ where $r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a]$ is the reward function.

Furthermore, the agent follows a deterministic policy π that establishes a mapping from states to actions, $\pi : S \rightarrow A$. The optimal policy seeks to maximize the total expected return (cumulative reward) of the agent where the return, G_t, is defined as some specific function of the reward sequence [45] as shown in

Equation 2.28.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T = \sum_{k=0}^{T} R_{t+k+1} \tag{2.28}$$

However this is only relevant for scenarios where there is a time horizon and/or an idea of a final time step T such as a task with some defined end point or events that are episodic in nature. This does not however extend to the greenhouse control problem as it does not necessarily follow naturally identifiable episodes, and therefore the notion of an infinite time horizon with discounted returns is introduced. Where the return function now becomes:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{2.29}$$

and the discount factor $\gamma \in [0, 1]$. Although the return (**??**) is a sum of an infinite number of terms, it is still finite if the reward is nonzero and constant if $\gamma < 1$ [45]. As the discount factor, γ, approaches one, the more future rewards are considered. In contrast, values approaching zero render the agent "myopic", focusing solely on maximizing immediate rewards. The discount factor, γ, commonly fall within the range of 0.9 to 0.995 [35]. This is an important concept since it can be shown that if the rewards are bounded (i.e. $\in R$) and a discount value of $\gamma < 1$ is used, then a stable optimal policy can be found [13]. There follows that a value function exists, under a specific policy π of a state s, denoted as v_π and defined as the expected return starting in state s and following the policy π thereafter. This value function is depicted below in Equation 2.30.

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \forall s \in S \tag{2.30}$$

Given the absence of transition dynamics in a standard reinforcement learning (RL) problem, this can be extended for a state-action pair value function as in Equation 2.31 whereby Q_π denotes the expected return starting from state s and taking action a, under the policy π [39].

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, At = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, At = a \right] \tag{2.31}$$

Important to note, that from **??** and Equation 2.30, a value function under a generic stationary policy π satisfies the Bellman equation as shown in Equation 2.32 [13], [46].

$$V_\pi(s) = \mathbb{E} \left[r(s, a) + \gamma V_\pi(s') \right] = \mathbb{E} \left[r(s, \pi(s)) + \gamma V_\pi(f(x, \pi(s))) \right] \tag{2.32}$$

Moreover, the value function under the optimal policy, π^* can be represented as shown in Equation 2.33 and for the state-action value function in Equation 2.34, where the relation between the value function and state-action value function is shown in Equation 2.35 and Equation 2.36.

$$V^*(s) = \max_a \mathbb{E} \left[r(s, a) + \gamma V^*(f(x, a)) \right] \tag{2.33}$$

$$Q^*(s, a) = \mathbb{E} \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \tag{2.34}$$

$$V^*(s) = \max_{a \in A} Q^*(s, a) \tag{2.35}$$

$$Q^*(s, a) = r(s, a) + \gamma V^*(s') \tag{2.36}$$

Finally, if the optimal state-action value function (also known as Q-value) is known, then the optimal policy, π^* can be found by Equation 2.37 [47], whereby the optimal policy greedily selects an action that maximizes the Q-value function. Various methods exist to approximate this Q-value function, and it is this goal that reinforcement seeks to achieve. However it also possible to directly optimize the policy, π to find the optimal policy π^*.

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a), \forall s \in S \tag{2.37}$$

2.2.2. Q learning

These family of algorithms learn an approximating to the Q value function (Equation 2.34). Moreover, these algorithms are "off-policy" algorithms whereby each update can use data collected at any point during training. Q learning offers a deterministic policy, whereby in each state, the action is selected based on Equation 2.37. In order to find the optimal Q value function, the update rule in Equation 2.38 is used in combination with a Q-table (a table that holds all possible combinations of states and actions). This iterative update rule aims to approximate the bellman equation in Equation 2.34. Once the Q-function converges to the optimal Q-function, the optimal policy can be determined [48].

$$Q^{new}(s, a) = (1 - \alpha) \underbrace{Q(s, a)}_{\text{old value}} + \alpha \overbrace{(R_{t+1} + \gamma \max_{a'}(Q(s', a'))}^{\text{Learned value}} \tag{2.38}$$

where the temporal difference error is

$$TD_{error} = (R_{t+1} + \gamma \max_{a'}(Q(s', a')) - Q(s, a) \tag{2.39}$$

Where TD error equals zero when optimality has been achieved (Equation 2.34). However, this is only feasible for small problems, where the state and action space are small and discrete. In order to accommodate a continuous state space, the Deep-Q Network (DQN) was developed. In this approach, a neural network is utilized to estimate the Q-function, enabling the learning of a deterministic policy from data with high-dimensional features [49]. However, DQN still outputs a discrete policy and therefore does not accommodate a continuous action space [49]. Nevertheless, recent advancements have extended DQN's applicability to continuous action spaces.

2.2.3. Policy Optimization

Policy optimization offers a more direct approach in learning the optimal policy as apposed to Q-learning. Such algorithms represent a policy as $\pi_\theta(a|s)$, whereby $\pi_\theta(a|s)$ is a stochastic policy that gives the probabilities of choosing action a given state s. The vector θ parameterize the policy, and it is these parameters that are optimized, by gradient ascent on the performance objective $J(\pi_\theta)$ such as an Equation 2.40.

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t) \tag{2.40}$$

Here, $\nabla \hat{J}(\theta_t)$ represents a stochastic estimate, where its expected value approximates the gradient of the performance objective function with respect to the policy's parameters, θ [45]. Such algorithms are called "on-policy" algorithms, which mean that each update is performed from data that was collected from the most recent version of the policy. All RL algorithms that follow this update rule on the policy are considered policy gradient/optimization methods, whether or not they also learn an approximation to a value function. However, such algorithms are often called actor-critic methods [45]. Policy Optimization algorithms are also naturally able to handle continuous state and action spaces.

2.2.4. Actor-Critic

Perhaps a more interesting class of RL algorithms combines both q-learning and policy optimization. Policy optimization methods directly optimize the policy, therefore they tend to be more stable and reliable in contrast to Q-learning and are better for continuous and stochastic environments [45]. However, because Q-learning learns "off-policy", it has a substantially higher sample efficiency than that of policy optimizations methods [45]. Actor-critic methods often incorporate the strengths from both policy gradient methods and q-learning, in order to achieve stable and fast learning. The critic learns a value function and the actor learns a policy. The critic provides feedback on how good an action was and uses this to update the policy. In this way, the actor can learn from both its own experience and the critic's feedback Moreover, actor-critic algorithms are also suitable in tackling continuous state and action spaces [45]. It is worth mentioning that these algorithms can include both the critic and actor in the MPC's framework to enhance its performance in an economic optimisation setting. A commonly used actor-critic algorithm is Soft-Actor-Critic which will be the selected Rl algorithm in this thesis. Numerous actor-critic algorithms exist and for a more in depth explanation on the various algorithms and the decision on why SAC was selected for this thesis see section A.1 and section A.2 respectively.

2.2.5. SAC

Soft-Actor-Critic (SAC) is an actor critic method that incorporates an entropy term in its reward function to address the trade-off between exploration and exploitation. This results in the agent learning a stochastic policy,$\pi_\theta(\cdot|s)$, parametrized by θ, where the entropy term is a measure of randomness in the policy. Consequently, the agent aims to optimise a balance between the total expected reward and entropy, ultimately seeking to maximise rewards while maintaining a policy that is as random as possible. The optimal policy is rewritten as:

$$\pi_\theta^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H(\pi_\theta(\cdot|s))\right)\right] \tag{2.41}$$

where $H(\cdot)$ is the entropy term and is denoted asand $\alpha > 0$ is called the trade-off coefficient that determines the balance between exploration and exploitation. The entropy of a random variable x with a probability density function P is calculated as:

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)] \tag{2.42}$$

The value function of a policy π is changed to include the entropy term and is denoted as:

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H(\pi_\theta(\cdot|s))\right) \,\middle|\, s_0 = s\right] \tag{2.43}$$

and the Q-value function as:

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi_\theta(\cdot|s)) \,\middle|\, s_0 = s, a_0 = a\right] \tag{2.44}$$

Thus the value function is connected to the Q-value function through:

$$V_\pi(s) = \mathbb{E}\left[Q_\pi(s, a)\right] + \alpha H(\pi_\theta(\cdot|s)) \tag{2.45}$$

The recursive bellman equation of the Q-function of a policy π, parametrized by ϕ becomes:

$$Q_\phi^{\pi_\theta}(s, a) = \mathbb{E}\left[R(s, a, s') + \gamma(Q_\phi^{\pi_\theta}(s', \bar{a}') - \alpha \log(\pi_\theta(\bar{a}'|s'))\right], \quad \bar{a}' \sim \pi_\theta(\cdot|s') \tag{2.46}$$

where s, s', a represent the state,next state and action taken respectively and is sampled from the replay buffer. $\bar{a}'$ denotes the action to take in state s' and is sampled using the current policy $\pi_\theta(\cdot|s')$. The

replay buffer., denoted as $\mathcal{D}$, is used to store the tuple (s, a, r, s') as sampled from the environment based on the agents policy and transitions. A mini batch is uniformly sampled from this replay buffer in order to train the Q-network.

Similarly to DDPG and TD3, SAC uses a clipped double-Q trick with soft updating to learn its Q-value function. The structure of the SAC includes two parametrized Q-functions, Q_{ϕ_1}, Q_{ϕ_2}, known as the critics, and one parametrized policy function π_θ, known as the actor, where ϕ_1, ϕ_2, θ refer to the parameters of its respective neural network. Additionally, each critic has its own target critic with parameters $\phi_{targ,1}$ and $\phi_{targ,2}$ respectively. These target parameters are initialized by copying ϕ_1, ϕ_2. Unlike in other actor critic algorithms such as mentioned above that train a deterministic policy, the stochastic policy function trained in SAC outputs two vectors describing the mean μ_θ and the (natural) log standard deviation $\log(\sigma_\theta)$ to determine the policy distribution. However, instead of sampling $\tilde{a}$ directly from this distribution, a reparameterization trick is used where:

$$\tilde{a}_\theta(s, \xi) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \quad \xi \sim \mathcal{N}(0, I) \tag{2.47}$$

where ξ is the noise vector and is sampled from a Gaussian distribution and the output squashed between $[-1, 1]$ with the tanh function. The greedy deterministic policy (used after training) is then:

$$\pi_\theta(s) = \tanh(\mu_\theta) \tag{2.48}$$

The reparameterization facilitates the learning procedure. The loss function for the Q-networks are defined as:

$$L(\phi_i, \mathcal{D}) = \mathbb{E}\left[\left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2\right] \tag{2.49}$$

where the target is shared between the two Q-networks and is given by:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{j=1,2} Q_{\phi_{targ,j}}(s', \tilde{a}') - \alpha \log(\pi_\theta(\tilde{a}'|s'))\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s') \tag{2.50}$$

The loss function is subsequently minimized by using stochastic gradient descent or a variant of it. The target Q-networks are soft updated by polyak averaging:

$$\phi_{targ,i} \leftarrow \eta\phi_i + (1 - \eta)\phi_{targ,i}, \quad i = 1, 2 \tag{2.51}$$

The policy aims to maximize the expected return and thus the policy loss function becomes:

$$\max_\theta \mathbb{E}\left[\min_{j=1,2} Q_{\phi_{targ,j}}(s', \tilde{a}_\theta(s, \xi)) - \alpha \log(\pi_\theta(\tilde{a}_\theta(s, \xi)|s)\right] \tag{2.52}$$

2.3. MPC

MPC demonstrates exceptional proficiency in managing constrained Multiple Input, Multiple Output (MIMO) systems, making it highly effective in applications like greenhouse control. Moreover, MPC's ability to handle non-linear dynamics and its optimality has contributed to its widespread adoption and has become the standard approach for implementing constrained multivariable control in the process industry [50]. Nearly every application introduces constraints, whether it be on the state of the system or the control inputs. Actuators, constrained by physical and safety limits such as temperature or pressure control, makes it necessary for controllers to handle such constraints. While the online computational time for such controllers may be expensive, for slow dynamics such as a greenhouse, sufficient computational time is available to accommodate this. Furthermore, the ongoing trend of faster computation mitigates this concern, making it less of a problem.

Although many control schemes exist in controlling greenhouses, it has been shown that an MPC controller outperforms an adaptive PID controller for greenhouse air control in [32]. Even though the control of a greenhouse is inherently non-linear, non-linear Model Predictive Control schemes have demonstrated effectiveness in handling this complexity, as evidenced in [51], [52] specifically for temperature control in a greenhouse. Furthermore, [53] shows insight of the performance of MPC when optimized for energy savings of a greenhouse, displaying impressive results over set-point tracking controllers. Further, [34] successfully implements a robust MPC to control the economic benefit of a greenhouse with parametric uncertainty, displaying its potential in greenhouse control. Moreover, as confirmed by [53], MPC stands out as the most effective control approach in smart greenhouses for energy savings. The study concludes that MPC is particularly advantageous when the system's dynamics can be reasonable approximated by a model and are sufficiently slow relative to the required time needed to perform the optimization.

Additionally, MPC has been shown to have a strong theoretical foundation. Researches have developed rigorous methods in analysing performance and stability on a MPC under certain conditions, including the optimality and robustness of the implemented controller. Indeed, if the system dynamics are known, it is possible to design an MPC to fulfill specific design requirements [54]. Nevertheless, while a solid mathematical framework exists for set-point tracking MPC, the same level of assurance cannot be provided for economic MPCs (EMPC), which aim to optimise for economic benefits.

2.3.1. The General MPC problem

MPC is an advanced method of control that uses a model of the system to predict the behaviour of the system over a finite horizon, known as the prediction horizon. The MPC controller aims to solve the infinite-horizon optimal control problem (OCP) as a series of finite-horizon problems in order to make the problem computationally tractable [15]. At each discrete time interval, the MPC solves the optimisation problem over this prediction horizon to compute the optimal inputs that minimize a given cost function, subject to constraints such as the system dynamics, initial state and state and control constraints. Each optimisation yields a sequence on control actions to take over this prediction horizon . The first control input is applied to the system, and at the subsequent time step, the system is sampled again and the entire process is recalculated providing an updated control sequence. While this approach is not optimal, it is an approximation of the infinite time horizon problem and has been shown to yield excellent results in practice as discussed above. The choice of prediction horizon length is often a trade-off between computational efficiency and the desired control performance.

While MPC can be used for both continuous and discrete time system dynamics, continuous time system must be discretized to render the optimal control problem (OCP) tractable. This is necessary since a continuous time system gives rise to an infinite-dimensional problem. Therefore the discrete case will be considered as illustrated in Equation 2.20. It is possible that the state may be unknown (as in Equation 2.22), and therefore a state estimation is required, however we will further assume that the state is fully measurable or perfectly estimated, i.e. $\hat{x}(k) = x(k)$ and therefore $\hat{p} = p$. When considering a finite-time horizon, N_p, the OCP problem can be formulated as in Equation 2.53.

$$
\begin{aligned}
&\min_{x(k_0:N_p+1),u(k_0:N_p)} J(x(k),u(k)) = \\
&\min_{x(k_0:N_p+1),u(k_0:N_p)} \sum_{k=k_0}^{N_p-1} l(x(k),u(k)) + V_f(x(N_p)) \\
\text{s.t.}\quad & x(k_0) = x_{k_0} && \forall k = k_0,\dots,N_p, \\
& x(k+1) = f(x(k),u(k),d,p) && \forall k = k_0,\dots,N_p, \\
& y(k) = g(x(k),u(k),d,p) && \forall k = k_0,\dots,N_p, \\
& x_{min} \le x(k) \le x_{max} && \forall k = k_0,\dots,N_p, \\
& u_{min} \le u(k) \le u_{max} && \forall k = k_0,\dots,N_p,
\end{aligned}
\tag{2.53}
$$

where $x(k_0 : N_P + 1)$ denotes a sequence of states, i.e. $[x(k_0), x(k_0 + 1), \dots, x(N_P + 1)]^T$ and may be denoted as $\mathbf{x}$ and similarly for the control actions $\mathbf{u}$ and known disturbances $\mathbf{d}$. The cost function $J(\cdot)$

may be explained by a stage cost $l(\cdot)$, where $l : \mathbb{X} \times \mathbb{U} \rightarrow \mathbb{R}$, which is the cost incurred at every time step k based on the current state and input applied and a terminal cost, $V_f(\cdot)$, which accounts for the cost associated with the state at the end of the prediction horizon. [34], [50]. The state and control constraints as shown. In the case of a setpoint tracking MPC, the stage cost and terminal cost function would commonly be represented as a quadratic term that penalizes the deviation between the state and the desired setpoint. Additionally, a setpoint tracking MPC would include a terminal constraint to guarantee performance and stability. This terminal constraint is typically set to the desired setpoint. However in economic optimisation, selecting an appropriate terminal constraint and/or cost function is difficult.
It is noted that the more accurate the model, the better the prediction and thus the better the actions. In the case of greenhouse control, if the model accurately captured all crop and greenhouse phenomena, state constraints would not be needed, since these constraints would be naturally incorporated into the dynamics. However, in practice, such dynamics are often still captured by explicit constraints. Moreover, an increasing complexity in the prediction model often leads to an increase in computational time and power which may exceed the time-scale of the system. This can make control actions impractical or infeasible within the required time frame [54]. Greenhouse environments are known to be complex and highly non-linear in nature. Hence, for control purposes, it is necessary to have simplified models that accurately predict system dynamics such as the model presented by van Henten [29] and as discussed in section 2.1 for greenhouse control.

Moreover, uncertainties in the model may greatly impact the performance of the MPC and the relation between a deterministic model and an uncertain model is explored in [34] and shown that even when a robust MPC strategy is used a 20% in relative parametric uncertainty decreased the crop yield by 11%.

2.3.2. Economic Model Predictive Control (EMPC)

The primary objective of an EMPC is to maximise economic benefit, whereby the economic benefit could refer to the profitability, efficiency and/or sustainability of the controlled process [55]. Typical processes that maximise for economic benefit would be in chemical and greenhouses processes with the objective of optimising profitability. In both processes it is difficult to define a steady state to which the MPC must control the system to. In the case of a greenhouse the notion of a steady state is nonsensical, since it is desirable for the crop to grow with as little energy consimption. In the case of chemical processes it has become common practice to rely on a separate layer of control called real-time optimisation (RTO) to determine the steady state. This layer ensures that the current steady-state brings the greatest economic benefit. Subsequently, MPC is employed to steer the process towards this optimized steady state. However, this separation in control information is not necessarily desirable and/or optimal in regards to maximizing economic benefit. Thus, EMPC is introduced, wherein the economic objective is directly incorporated as the objective function of the optimal control problem that the MPC aims to minimize [56]. Here, the stage cost $l(x, u)$ encapsulates an economic model of the process, and in this thesis the MPC aims to optimise Equation 2.23.

By adopting EMPC, it becomes feasible that dynamic, transient or time varying operation of the process could lead to higher economic benefit as compared to controlling the process to an optimal (time-varying) steady-state. As stated in [56], incorporating the economic model directly into the cost function does not assume the same structure of the stage costs of a traditional set-point tracking MPC and has therefore no stability guarantees. Furthermore, since the EMPC optimizes the process economics over a finite time horizon,it does not guarantee optimal closed-loop performance over an extended period. In general no closed-loop performance guarantees under EMPC exist[55].

[56] further showed in the case of a chemical process, despite the increase in economic benefits, the EMPC resulted in unstable plant dynamics which is highly undesirable for such a process. While a greenhouse may not experience this, it is still possible to encounter undesirable transient effects. For example, during the initial stages of the growth phase, the EMPC may not find it advantageous to expend energy due to the minimal growth, which could potentially cause irreparable harm to the plant. Nevertheless, this stage could arguably be considered the most crucial phase of the crop cycle in order to achieve optimal economic gain, as maximising crop growth in the early stages may have cumulative effects throughout the entire growth period.

In order to achieve performance and stability, a terminal constraint in the optimal control problem is

introduced [57]. This terminal constraint must be selected carefully to achieve the desired performance and stability [56]. However, [57], introduces a terminal cost function with a terminal region constraint and demonstrates its superiority over a terminal constraint in terms of performance while maintaining system stability. Although, it is possible to develop a terminal cost/constrain to achieve desired closed-loop performance and stability, authors of [56], [57] and [55] acknowledge the difficulties in constructing such an appropriate terminal cost/constrain and a corresponding terminal region. Limited works provide theoretical analysis in order to construct an appropriate terminal cost in the absence of stability constraints. Most often, simulations are performed to design cost functions suitable for the specific application. Importantly, two main methodologies exist to assure closed-loop performance: to use a sufficiently large prediction horizon or the application of an appropriate terminal region and cost function [55]. Stability of the system's dynamics is not necessarily a concern, due to the nature of the greenhouse control system.

3

Reinforcement Learning Setup

This chapter provides an overview of developing the RL agent used in creating the RL-MPC algorithm. This chapter describes the environment in which the RL agent is trained and the agents performance in a deterministic and stochastic environment. Finally, the training of a value function for a fixed policy is also investigated.

3.1. Environment Description

This section describes the environment for the RL agent to learn an optimal policy. The environment is built on the Greenhouse model described in section 2.1; it outlines important features to train an RL agent successfully. This includes the observation space available for the agent to make decisions on, the action space available to the agent, the reward function, and finally, the weather data used for the training period.

Observation Space The observation space of the agent must be carefully selected to achieve desirable results. Providing too little information may degrade performance; however, giving the agent too much information about the state of the environment may introduce unwanted noise, making it difficult to infer an optimal policy. Typically, the state of dry weight of the lettuce crop would not be available for an expert grower to make decisions as it is difficult to measure without disrupting the crop's life cycle. However, various methods exist for predicting the state of the crop dry mass, such as a non-linear Kalman observer and other machine learning techniques [24]. However, it is assumed the dry mass may be measured and is available to the agent. Other states of the greenhouse, such as the temperature, C02 and humidity levels, are easily measured and form part of the observation space. The current weather conditions are also made available to the agent to make better decisions. As shown in section 2.1, since the control input depends on the previous control input (i.e., it may only deviate a maximum of 10% from the previous input), it is important to provide the previous control action to the agent. Lastly, the agent is considered time-aware, so the current time step is also provided. Although not necessary, it enables the agent to learn a non-stationary policy. Considering that the growing period is 40 days (as discussed in subsection 2.1.4), the problem has a fixed episodic length, where a growing period can be considered an episode. The agent can leverage the current time to make better decisions by incorporating time awareness. As discussed in Equation 2.27 and shown in Equation 3.5, the optimisation goal includes maximising the growth difference between time steps, so knowledge of the previous dry mass and the growth experienced in the previous time step may be beneficial to learning an optimal policy. However, the three following environment state tuples $s(k)$ at time k have been separately tested to represent the observation returned to the agent:

$$s(k) = (y_1(k), y_2(k), y_3(k), y_4(k), u(k-1), k, d(k)) \tag{3.1}$$

$$\begin{aligned} s(k) &= (\Delta y_1(k), y_2(k), y_3(k), y_4(k), u(k-1), k, d(k)) \\ \Delta y_1(k) &= y_1(k) - y_1(k-1) \end{aligned} \tag{3.2}$$

$$s(k) = (y_1(k-1), y_1(k), y_2(k), y_3(k), y_4(k), u(k-1), k, d(k)) \tag{3.3}$$

It is noted that Equation 3.3 is expected to perform better than Equation 3.1 and Equation 3.2 since more information is provided regarding the Markov decision processes of the model and the reward received, thereby allowing the agent to potentially infer the system dynamics more accurately. However, Equation 3.1 is the simplest and, therefore, facilitates learning a value function and easier integration in the RL-MPC algorithm, as discussed in section 3.6 and section 5.1, respectively. It was found that no substantial benefit was gained in providing the knowledge of the previous drymass state of the crop; hence, for simplicity, Equation 3.1 was used in this thesis

Action Space The continuous action space, denoted as $\mathcal{A}$, is defined as $\mathcal{A} = [-1, 1]^3$, where $\mathcal{A} \subseteq \mathbb{R}^3$. To ensure that the current control input, $u(k)$ satisfies the constraints outlined in Equation 2.24, the agent's action, denoted as $a(k)$, where $a \in \mathcal{A}$, is regarded as a modification to the control input. Consequently, the current control input can be determined as follows:

$$u(k) = \max(u_{\min}, \min(u(k-1) + a(k) \cdot \delta u_{\max}, u_{\max}))$$

where $\delta u_{max}(k)$, u_{min}, u_{max} are defined in subsection 2.1.4

Initial Conditions Initial conditions were kept constant for every episode for the stochastic and deterministic case and shown in Equation 3.4.

$$\begin{aligned} x(0) &= \begin{bmatrix} 0.0035 & 0.001 & 15 & 0.008 \end{bmatrix}^T \\ y(0) &= g(x(0)) \\ u(0) &= \begin{bmatrix} 0 & 0 & 50 \end{bmatrix}^T \end{aligned} \tag{3.4}$$

Reward Function The reward function is modelled after the optimisation goal, as defined in subsection 2.1.4 , and represents the same optimisation goal defined for the MPC OCP. Although the van Henten model sufficiently describes the dynamics of lettuce growth in a climate-controlled environment, it does not do so over the entire state space. Therefore, state constraints are imposed to ensure states operate within reasonable limits to provide realistic conditions. As stated in section 2.2, state constraints cannot be directly imposed but can be indirectly incorporated by a penalty function in the reward function. It is common practice to impose a linear penalty function for state violations when learning a policy with RL. Therefore, the resulting reward function becomes:

$$\begin{aligned} R(k) = &-(c_{p_1} \cdot (u_1(k)) + c_{p_2} \cdot (u_3(k))) + c_{p_3} \cdot (y_1(k) - y_1(k-1)) \\ &- (P_{c02} \cdot (y_2(k)) + P_T \cdot (y_3(k)) + P_H \cdot (y_4(k))) \end{aligned} \tag{3.5}$$

where $c_{p_1}, c_{p_2}, c_{p_3}$ are defined in subsection 2.1.4 and correspond to the pricing of the lettuce and control inputs and the penalty terms P_{c02}, P_T, P_H are defined as follows:

$$
\begin{aligned}
P_{\mathrm{CO2}} &= \begin{cases} c_{p_{\mathrm{CO2}}} \cdot (y_2(k) - y_2^{\max}) & \text{if } y_2(k) > y_2^{\max}, \\ c_{p_{\mathrm{CO2}}} \cdot (y_2^{\min} - y_2(k)) & \text{if } y_2(k) < y_2^{\min}, \\ 0 & \text{otherwise} \end{cases} \\
P_T &= \begin{cases} c_{p_{T_{ub}}} \cdot (y_3(k) - y_3^{\max}) & \text{if } y_3(k) > y_3^{\max}, \\ c_{p_{T_{lb}}} \cdot (y_3^{\min} - y_3(k)) & \text{if } y_3(k) < y_3^{\min}, \\ 0 & \text{otherwise} \end{cases} \\
P_H &= \begin{cases} c_{p_H} \cdot (y_4(k) - y_4^{\max}) & \text{if } y_4(k) > y_4^{\max}, \\ c_{p_H} \cdot (y_4^{\min} - y_4(k)) & \text{if } y_4(k) < y_4^{\min}, \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{3.6}
$$

The penalty constants $c_{p_{CO_2}}, c_{p_{T_{ub}}}, c_{p_{T_{lb}}}, c_{p_H}$ were found empirically in Jansen [22] to effectively account for deviations from desired states and their impact on the economic benefit. It should be noted that the upper bound of the temperature imposes stricter penalties for violations compared to the lower bounds due to the absence of active cooling in the system. Thus, during periods of increased temperature throughout the day, the agent needs to make appropriate decisions. The penalty constants and their respective units are displayed in Table 3.1. The selection of minimum and maximum temperatures was based on the typical operating ranges for lettuce crops and safe levels of CO_2 for brief human operation. Note that $c_{p_{CO_2}}$ is multiplied by 10^{-3} since the units used for $C0_2$ density in Jansen [22] is reported in $10^3 \cdot ppm$.

parameter	value	units
$c_{p_{\mathrm{CO2}}}$	$\frac{10^{-3}}{20}$	€$\cdot(ppm \cdot m^2)^{-1}$
$c_{p_{T_{ub}}}$	$\frac{1}{200}$	€$\cdot(C^\circ \cdot m^2)^{-1}$
$c_{p_{T_{lb}}}$	$\frac{1}{300}$	€$\cdot(C^\circ \cdot m^2)^{-1}$
c_{p_H}	$\frac{1}{50}$	€$\cdot(RH_{\%} \cdot m^2)^{-1}$
y_2^{max}	1600	ppm
y_2^{min}	500	ppm
y_3^{max}	20	C°
y_3^{min}	10	C°
y_4^{max}	100	$RH_{\%}$
y_4^{min}	0	$RH_{\%}$

Table 3.1: Penalty Constants

3.2. Experimental Setup

The observations returned from the environment were normalised by a running mean and variance to facilitate the learning process of RL. The actor and the critic were trained with these normalised observations. The VecNormalizeWrapper in Stable-Baselines3 is responsible for updating the mean and variance for every observation received from the environment. The observation is normalised as per Equation 3.7 and clipped between $[-10, 10]$.

$$
obs_{norm} = \frac{(obs - \mu_{obs})}{\sqrt{\sigma_{obs}^2 + 1 \cdot 10^{-8}}} \tag{3.7}
$$

where obs is the unnormalised observation as given in Equation 3.1 and μ_{obs} and σ_{obs}^2 represent the running mean and variance of the agents observation (Equation 3.1), respectively. The value $1 \cdot 10^{-8}$ is included to prevent division by zero. Although the VecNormalizeWrapper also facilitates this process, it is necessary to replicate this step when incorporating it into the RL-MPC algorithm. Finally, a seed value of 4 was used to generate random numbers to ensure reproducibility. This seed value was used to initialise neural network weights and select actions for exploration.

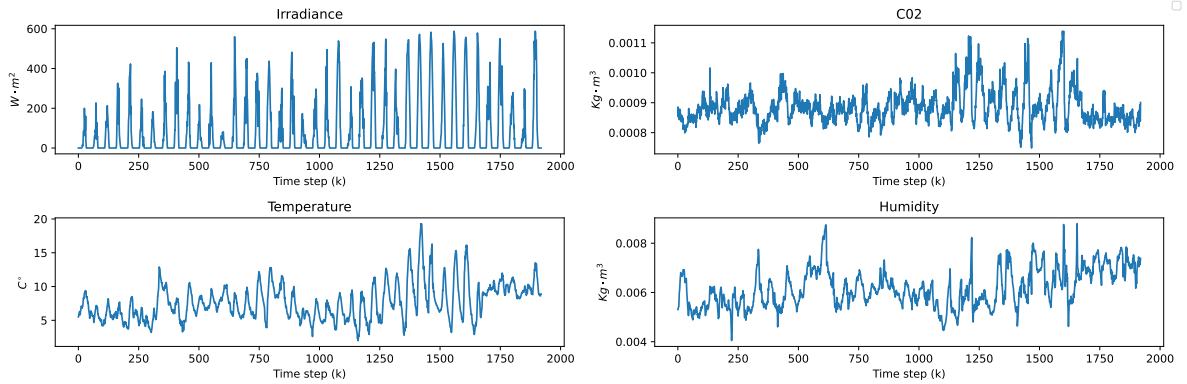

Figure 3.1: Weather Data

Weather Data The weather data used in training was obtained from the Venlow Greenhouse in Bleiswijk from January 30–March 11, 2014. The weather data was resampled from its original 5-minute interval to a 30-minute interval following the timestep of the environment. The weather data remains consistent throughout the episodes, irrespective of whether the training and evaluation is conducted under deterministic or stochastic conditions since it is not important to have the agent generalize well across various weather patterns in this thesis. Therefore, the validation data is the same as the training data, i.e. the same weather data is used in training as well as evaluating the agents performance. In practice, it may be necessary to evaluate the agent on unseen weather data to determine how well it is able to generalize to weather patterns. However, provided that all algorithms/controllers use identical weather data, it can be considered a suitable basis for comparing algorithms if the agent learns a suitable policy for this specific weather pattern.

Deterministic and Stochastic Case When learning a policy in stochastic and deterministic environments, the key distinction lies in the evolution of the state of the greenhouse. In the stochastic case, this evolution follows the principles outlined in Equation 2.21. In the stochastic case, three levels of uncertainty were tested, namely $\delta_p = 5\%$,$\delta_p = 10\%$ and $\delta_p = 20\%$. Although these uncertainty levels might be considered extreme, it was desirable to learn a policy for each of them to compare to MPC and RL-MPC under identical conditions. The optimal configuration that produce the most favourable outcomes in the deterministic scenario will be employed in the stochastic scenario, and there will be no reevaluation of hyperparameters. While the stochastic environment represents a scenario closer to real-life conditions, the deterministic case offers a nominal measure of the RL agent's performance and the resulting RL-MPC algorithm when combined with MPC. The nominal case serves as an upperbound on the performance and allows for easier development and comparison between the various controllers.

Performance Metrics The primary performance metric for evaluating RL agents is the final cumulative reward obtained over the 40-day growing period, as demonstrated in Equation 3.5. The performance metric under consideration is conceptually equivalent to the EPI (Equation 2.23) minus the summed temperature, C02 and humidity violations. This is a natural selection of the final performance metric as it directly corresponds to what the agent and the MPC and RL-MPC controllers are optimising. Other metrics include the EPI, total growth, total C02 usage, total heating, computational time to compute control input, temperature and C02 violations. It is difficult to compare these lesser performance metrics across policies since they all form part of the reward function (except the computational time) and are not directly optimised. Therefore, comparisons would be meaningless, and only observations can be made. However, the computational time to compute the optimal control action is an important metric, particularly when combining RL with MPC. Finally, when comparing the controllers in a stochastic environment, the variance of the final cumulative reward is also examined. Although variance is not optimised, it serves as a measure of robustness of the resulting agent in a stochastic environment.

3.3. Hyper-parameter Tuning

Hyper-parameter tuning is frequently laborious and requires exhaustive exploration to determine the best configuration to maximise the cumulative reward of the agent. Therefore, the final hyper-parameters are posted in Table 3.2 and were found empirically. Refer to **appendix A** for a more comprehensive analysis of the hyper-parameters obtained. The discount factor and activation function are not reported here, since they are further discussed in subsection 3.4.1 and subsection 3.4.2 respectively. The effect of these two parameters is important to consider when integrating the value function of the learned agent with MPC. The defaults provided by SB3 are used for hyper-parameters that are not reported.

Training episodes	100
Warm-up episodes	9
Hidden layers	2
Neurons per hidden layer	128
Batch size	1024
Learning rate	$5 \cdot 10^{-3}$
Buffer size	100000

Table 3.2: Hyper-parameters

Activation Function The importance of the activation function lies in whether the resulting activation allows the output of the neural network to be differentiable with respect to the inputs. For instance, the ReLu activation function is commonly used due to its simplicity and superior convergence, unlike activation functions like tanh and sigmoid, which also encounter the vanishing gradient issue. Although ReLu is differentiable concerning the weight of the neural network, it is not differentiable to the inputs of the neural network. This is important to note since the trained value function must be differentiable with respect to its inputs if it is to be used as a cost function in the MPC formulation. Hence, the tanh function may be used instead, as it is a frequently used activation function that is differentiable concerning the inputs.

Discount Factor Another consideration is the discount factor, denoted as γ. Since the problem is episodic and the cumulative rewards are bounded, it is possible to have a $\gamma = 1$. By setting the discount factor γ to 1, the agent can consider the entire prediction horizon when deciding its actions. Additionally, the value function obtained during training would contain information about the entire prediction horizon, which is highly desirable, especially for optimising economic benefits. Having $\gamma < 1$ will shorten the agent's 'prediction horizon', and the resulting value function may not provide significant benefits for the MPC when integrated. However, it may stabilise the learning and yield a better policy than when $\gamma = 1$. Results about the activation function and discount factor are shown and discussed in subsection 3.4.3.

3.4. Deterministic Results

This section presents and examines the outcomes of modifying the discount factor and the activation function. Finally, a discussion of the chosen policies and identifying the desirable characteristics that can be integrated with the MPC is performed. While several hyperparameters can impact the performance of the resulting RL policy, this section will focus on the discount factor and activation function. These two hyperparameters are considered necessary aspects of the RL policy, particularly when it is desired to integrate it with MPC.

3.4.1. Discount Factor

Typically, the discount factor lies between 0.9 and 0.99. Economic optimisation should have this discount factor as close to 1 as possible. Therefore, four discount factors were tested, namely 1, 0.99, 0.95 and 0.9. Going any lower may yield a more stable learning procedure at the cost of a more myopic policy. It should be noted that the ReLu activation function is used for the following results about investigating the discount factor. The value functions trained on their respective discount factor are denoted $V_{\pi|\gamma}(s)$. The ReLu activation function since learning proved to be more stable than tanh, and these results serve to investigate the effect of the discount factor.

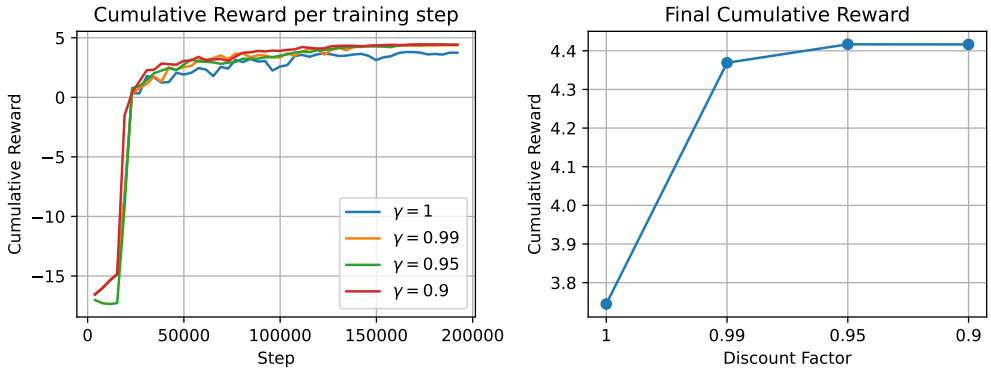

Figure 3.2: Cumulative reward vs discount factor (γ)

Performance The cumulative reward achieved over the training period and the final cumulative reward achieved for each different discount factor is depicted in Figure 3.2. The cumulative reward per training step is an example of the learning process, while the final cumulative reward depicts the performance of the RL system after training. As seen in Figure 3.2, it is clear that $\gamma = 1$ does not perform as well as lower discount factors. It is noted that the degradation in performance comes from the increase in problem complexity when the discount factor is 1. Hence, it becomes more difficult to find an optimal policy, requiring a different set of hyper-parameters and potentially a significantly larger number of training episodes. However, the policy generated with this discount factor will provide a value function that holds information across the entire time horizon, which is desirable. Conversely, the learning procedure is much more stable when lower discount values are used, and thus, a higher final cumulative reward is achieved. Given the desirability of using the value function in the MPC formulation, it is necessary to analyse the value function for each tested discount factor.

Value Function It is difficult to determine whether the critic has converged. Given that the critic serves as a q-value function approximator in the SAC algorithm, the actor must identify the optimal action for a given state to compute the value of that state using the critic. Therefore, convergence of the value function is also dependent upon the actor policy. Although the training curves indicate that the critic has converged, it has only converged in alignment with the actor's policy. It is necessary to visualise and compare the actual (computed by Equation 2.29) and the predicted value (given by the learned value function) of a given visited state during the 40-day simulation to assess whether the critic (and actor) have achieved convergence in predicting the value function. In addition, Equation 3.8 may be used as a visual aid to compare the accuracy of the learned value function to a value function that encompasses the entire prediction horizon (a non-discounted value function, i.e. $\gamma = 1$).

$$\begin{aligned} V_{\pi|\gamma=1}(s_k) &= r_k + V_{\pi|\gamma=1}(s_{k+1}) \\ \therefore V_{\pi|\gamma=1}(s_{k-1}) &= r_{k-1} + V_{\pi|\gamma=1}(s_k) \\ \therefore V_{\pi|\gamma=1}(s_{k-2}) &= r_{k-2} + r_{k-1} + V_{\pi|\gamma=1}(s_k) \\ \therefore V_{\pi|\gamma=1}(s_0) &= \sum_{i=0}^{k-1} r_i + V_{\pi|\gamma=1}(s_k) \end{aligned} \tag{3.8}$$

During each time step, the initial state's value is computed by the cumulative rewards received up to that point and the approximation of the current state's value using the value function. If a value function can approximate the non-discounted value of a state accurately, then Equation 3.8 should yield the same result for every time step, resulting in a horizontal line, $y = V_{\pi|\gamma=1}(s_0)$. It is noted, that only a value function trained on $\gamma = 1$ would be able to achieve this; however, it serves as an important visual aid to show the accuracy of the trained value functions.

For each discount factor, the predicted value of each state and the cumulative rewards obtained thus far were plotted and evaluated for the entire 40-day period. For the deterministic case, the realisations of state trajectories and rewards received do not differ between simulations. Hence, the exact value of each state may be calculated.

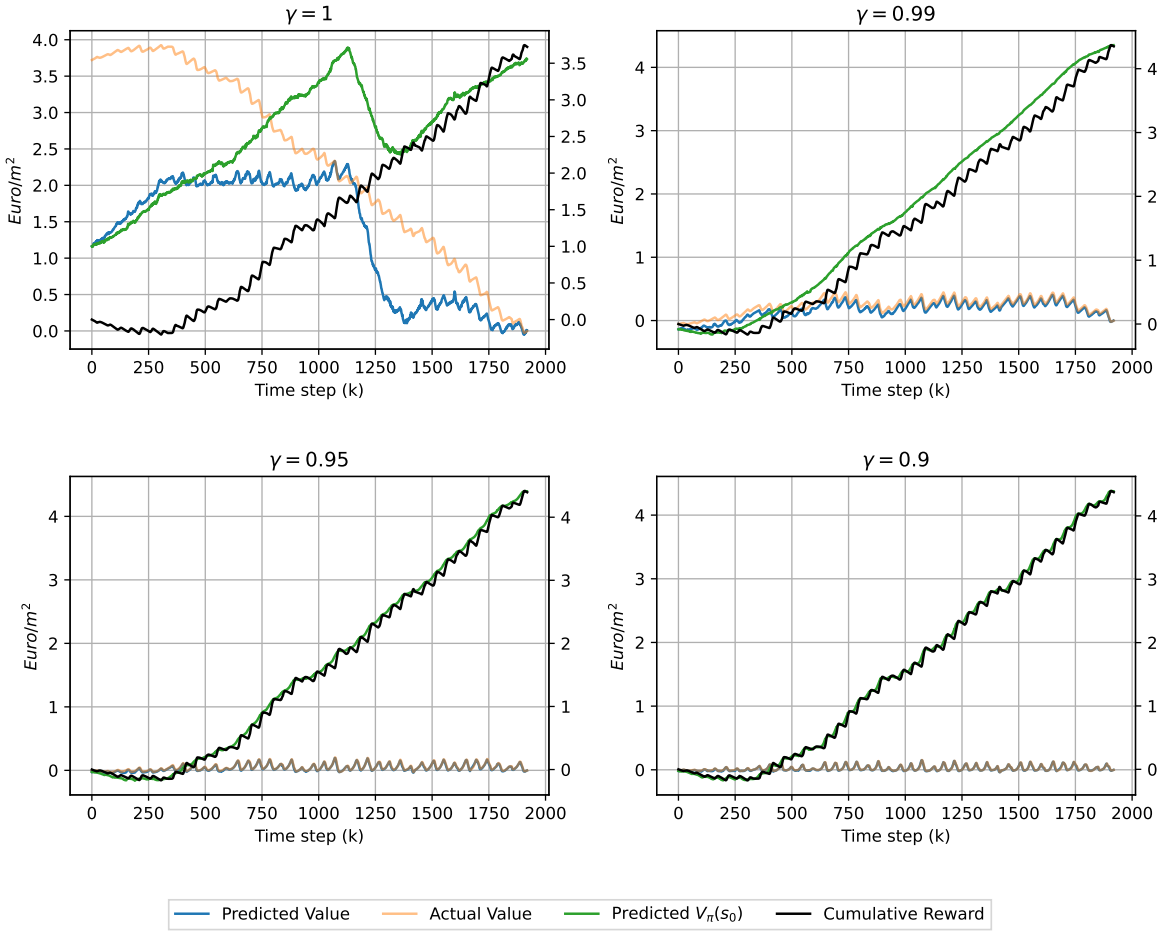

Figure 3.3: Predicted value vs actual value vs discount factor

Figure 3.3 illustrates the comparison between each state's predicted and actual value for each discount factor. In addition, the cumulative reward at each time step is also logged. Naturally, the cumulative reward differs slightly since the policy generated is also dependent on the value function obtained. It is acknowledged that the trajectory of the actual value for each state (for $\gamma = 1$) is a horizontally reflected trajectory of the cumulative reward trajectory and can be seen in Figure 3.3. Naturally, this is not the case for discount factors lower than one since the value only embeds knowledge of future rewards to be received over a shorter horizon. This is further seen by the predicted $V_\pi(s_0)$, whereby the value functions do not produce a horizontal $V_\pi(s_0)$ curve, indicating their inadequacy in predicting the true non-discounted value function. Not even $V_\pi(s|\gamma = 1)$ predicts this true value function accurately, shown by the clear non-horizontal predicted $V_\pi(s_0)$ curve. It is also noted that the lower the discount factor, the more accurate the predicted discounted-value of a state becomes, as seen by the overlapping 'predicted' and 'actual value' curves. This may be due to decreased problem complexity and more accurate approximations. When $\gamma = 0.9$, the critic is capable of accurately predicting the actual value of a state. However, it falls short of accurately representing the true value of a state over the entire time horizon, given its nature of discounting future rewards. The same can be said for all discount factors lower than one. It was important to train an actor and critic with a $\gamma = 1$ because it was believed that the trained critic could still provide a reasonable estimation of the value of a state throughout the entire simulation despite having a worse-performing policy. When examining Figure 3.3, it becomes

evident that this assumption is incorrect. During the investigation into suitable hyper-parameters for the learning agent, it was observed that when $\gamma = 1$, the trained critic struggled to estimate the value of a state in all cases accurately. Figure 3.3 is just one such realisation.

3.4.2. Activation Function

It is also important for the trained critic to use differentiable activation functions to ensure differentiability. This is done so that it may be used in the MPC framework. However, an activation function, such as the commonly used tanh activation, may or may not yield desirable results in maximising cumulative reward. This section compares the performance of a learned agent with tanh activation functions against and agents with ReLu activation functions for a $\gamma = 1$ and $\gamma = 0.95$, with the ReLu acting as the baseline performance.

Discount Factor	Performance		SpeedUp (%)
	ReLU	tanh	
1	3.72	3.44	-7.53
0.95	4.27	4.17	-2.34

Table 3.3: Effect of different activation functions (ReLU and tanh) on performance and speed-up at varying discount factors.

Table 3.3 displays the effect of the tanh activation on the agent's final performance. It is clear that ReLu outperforms the tanh activation function in terms of total cumulative reward. This situation presents a dilemma. It is desirable to have an effective reinforcement learning policy in which the critic has differentiability. It seems there is a decline in performance by incorporating the concept of differentiability into the critic. While additional investigation into the hyperparameters may be necessary to reject this notion, the focus of this thesis does not lie in developing the best possible RL-generated policy. Rather, chapter 3 aims to create a policy and an accurate critic that is competitive with MPC to produce a potentially better policy when merged. However, these results suggest that special care should be taken when merging RL with MPC.

3.4.3. Final Results and Conclusion

From the findings, the best policy produced by RL is with the hyperparameters shown in Table 3.2, with a $\gamma = 0.95$ and with ReLu activation functions. While the best-performing policy is desired, this configuration does not produce adequate conditions for its critic to be used in the MPC formulation; namely, the value function does not hold information across the entire prediction horizon and is not differentiable. An agent learned with a $\gamma = 1$ and tanh activation function, results in a worse performing policy and a critic that struggles to predict the value of states accurately. With that being said, a critic that has undergone training with a discount factor of $\gamma = 0.95$ is able to accurately predict the discounted-value of a state and may still possess sufficient knowledge regarding future rewards to be advantageous for MPC. Additionally, a critic learned with a a $\gamma = 1$, albeit bad approximations (as seen by the non-horizontal line in Figure 3.3), may also provide enough information about future rewards. However, the following section examines how a value function may be trained on a fixed policy. All other configurations are unnecessary if an accurate differentiable value function can be trained on the best-performing policy. However, a time-series analysis of the state and input trajectories was performed using the agent configuration shown in Table 3.4.

Agent	γ	Activation Function	Final Performance
1	0.95	ReLU	4.27
2	0.95	Tanh	4.18
3	1	Tanh	3.44

Table 3.4: Performance of selected agents with different activation functions and discount factors

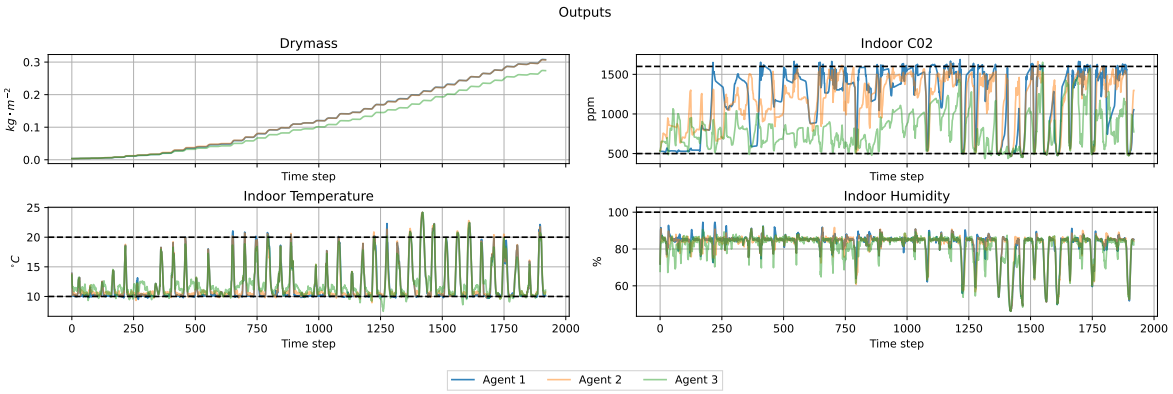

Figure 3.4: Time series of system outputs

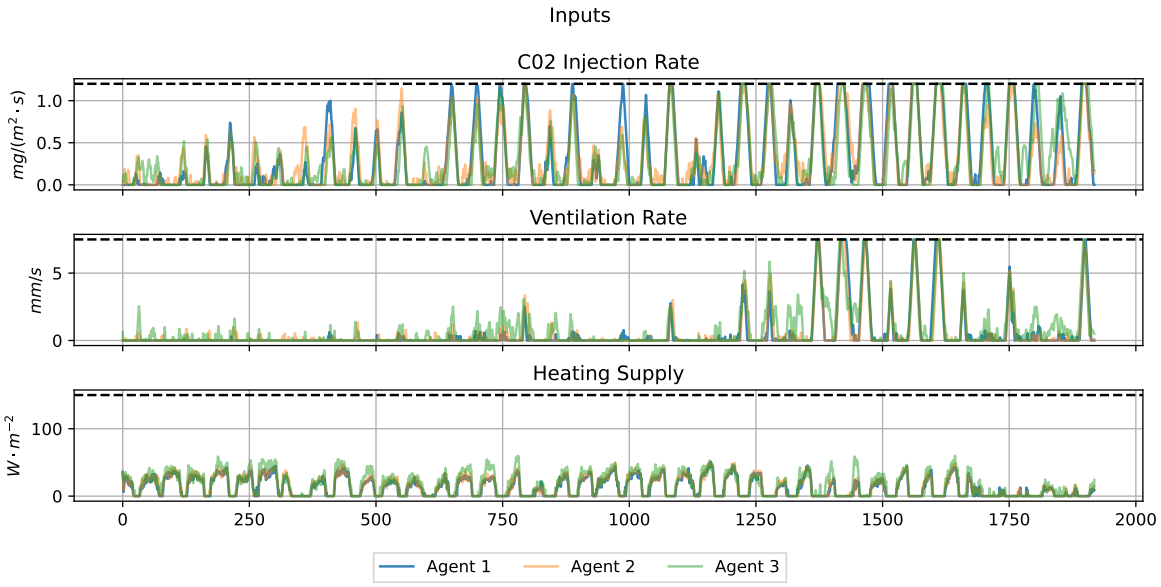

Figure 3.5: Time series of system outputs

The time series plot of the system outputs and inputs across the 40 days for every agent, as selected in Table 3.4, is shown in Figure 3.4. These time series plots are similar to those reported in Morcego, Yin, Boersma, *et al.* [9] and Jansen [22]. Direct comparisons are not possible since Morcego, Yin, Boersma, *et al.* [9] does not specify the weather data range used and the reward function differs from what is used in this thesis. Furthermore, Morcego, Yin, Boersma, *et al.* [9] includes additional constraints on temperature levels during the day to encourage heating by solar radiation during the day and the heating system at night. This paper did not include these constraints because the thesis aimed to give RL more independence in optimising EPI while minimising dangerous constraint violations for plant and/or human operation. Furthermore, Jansen [22] also reports similar results; however, a direct comparison is not possible since hyper-parameters and reward functions differ. However, it is noted that results, time series and cumulative rewards are similar enough to give confidence in the correct training of the RL agent.

Metric	Agent 1	Agent 2	Agent 3
EPI	4.964	4.807	3.727
Total Growth	0.304	0.303	0.270
Total C02 Usage	1.057	1.0318	1.029
Total Heating	12.5462	13.661	16.381
Computational Time	0.000216	0.00024	0.00023
Temp Violations	110.007	119.2	138.93
C02 Violations	3311.43	1046.47	1972.61
Final Performance	4.27	4.173	3.031

Table 3.5: Performance Metrics of agents

Other performance metrics are reported for completeness in Table 3.5. The computational time needed to compute the optimal control action takes $\approx 0.2ms$. This is expected since a simple inference of the actor network is required to calculate the optimal action. Other metrics, as reported in Table 3.5, may be useful when designing a controller where the total heating and $C0_2$ may provide additional insight. However, the RL-MPC controller developed in this thesis focuses on the economic optimisation of a system and is only concerned about optimisation goal.

3.5. Stochastic Results

In training the stochastic RL agent, the same hyper-parameter configuration was used to produce the best nominal agent, namely the same configuration as Agent 1 (Table 3.4). Although this produces a critic that is problematic for integration with MPC since it uses ReLu activation functions and does not provide accurate prediction of the value, this is solved in section 3.6. Three stochastic agents were trained on a different level of uncertainty as specified in section 3.1 with the uncertainty model according to Equation 2.21. Performance metrics are reported, and a comparison is made with the nominal model (Agent 1 from Table 3.4). Performance metrics are evaluated by repeating the 40-day simulation period 30 times and taking the average and variance of the cumulative rewards over the complete time horizon. Each agent is assigned a name based on the degree of uncertainty on which they received training. For example, an agent that has undergone training in a stochastic environment with a $\delta_p = 20\%$ is referred to as 'Agent 0.2'. The nominal agent named 'Agent 1' from Table 3.4 from is also referred to as the 'nominal agent'. Each agent is compared to the other stochastic agents in an environment with every level of uncertainty

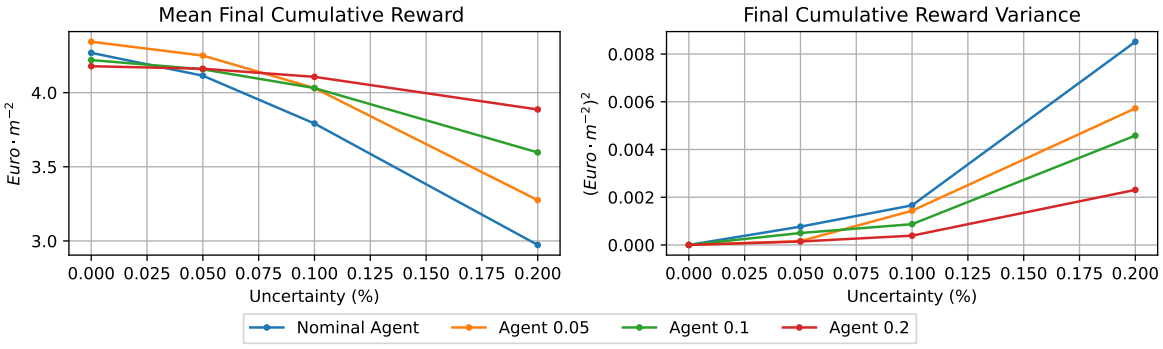

Figure 3.6: Stochastic RL policy performances

The final mean cumulative reward and variance of each agent under different uncertainty levels are presented in 3.6. As expected, as more uncertainty is injected into the environment, the mean cumulative reward decreases and the variance increases. There is a noticeable trend where agents trained with higher levels of uncertainty tend to have a slower decline in performance as uncertainty increases. However, they may not perform as well at lower levels of uncertainty compared to agents trained with lower levels of uncertainty. Similarly, agents trained on higher uncertainty levels maintain a lower variance in their final cumulative reward as uncertainty increases, compared to agents trained on data

with lower uncertainty. Although, in practice, the uncertainty is not known, if a more reliable policy is desired, training a reinforcement learning agent on a higher level of uncertainty than expected may achieve the desired outcomes. Also, in general, the results also suggest that an agent trained on a level of uncertainty does not necessarily produce the best policy for that level of uncertainty. It seems that, in the nominal case ($\delta_p = 0\%$), the agent trained on a 5% uncertainty level (Agent 0.05) outperforms the nominal agent. Moreover, Agent 0.2 seems to achieve a higher average cumulative reward than Agent 0.1 when tested on a 10% uncertain model. It is anticipated that agents who are trained and tested based on their respective uncertainty would exhibit superior performance. However, this disparity in performance can be attributed to the agent's increased exploration due to the added noise.

3.5.1. Conclusion

It is widely recognised that RL can address uncertainty by appropriate training methods, as evidenced by the findings presented in Figure 3.6. The incorporation of these stochastic policies into the MPC framework will be examined to determine whether an RL policy learned from stochastic data can transfer its characteristics to the RL-MPC framework. Finally, each agent, specifically the nominal agent, Agents 0.05, 0.1 and 0.2, are employed in its corresponding stochastic environment. Although Agent 0.05 may outperform the nominal agent under nominal conditions, each stochastic environment uses its corresponding agent. Moreover, the value function as trained by the RL algorithm is problematic. This is because a ReLu activation function with a discount factor of 0.95 has been used. The ReLu activation function makes the critic non-differentiable and the discount factor does not allow the critic to hold information across the entire growing period. Although the tanh activation function can be used, a new set of hyper-parameters are required in order to achieve the same performance. In addition, a discount factor of 1 can be used to estimate the value of a state that includes information on future rewards throughout the entire growing period. However, this approach increases the complexity of the problem, leading to reduced performance of the RL agent and produces in inaccurate critic. Therefore, it was decided to separately train a critic with a discount factor of 1 on a fixed policy to stabilize learning.

3.6. Trained Value Function

Although training an agent using SAC produces an actor and a critic network, it was shown in subsection 3.4.3 that the produced critic had undesirable characteristics. This critic was trained based on a changing policy dependent on the critic itself; therefore, it is not surprising that the approximation to the value function was sub-optimal. However, this section aims to train a value function approximator with a fixed policy[1]. Therefore, a value function may be trained on the best policy obtained in subsection 3.4.3, namely Agent 1 (Table 3.4). Additionally, there is more freedom in choosing the architecture of the value function since it is now trained independently of the policy. Therefore, simpler models may be made and tested to approximate the value function. Specifically, upon inspection of Figure 3.4 and Figure 3.3, it is noticed that the cumulative reward at each time step primarily depends on the state of the crop's dry mass at time k, due to the similarity in the two curves. Therefore it may be possible to learn a value approximator solely based on the crop's dry mass and current time. It is noted that this value function is only accurate under the policy on which it was trained. Lastly, the tanh activation functions must be used to ensure differentiability. Two methods were employed to learn various value function approximators: the temporal difference learning method and the expected return method, each with their respective advantages.

3.6.1. Temporal Difference Learning

This method uses a similar technique by which SAC, DDPG and TD3 update their critic, which is most similar to DDPG. Two neural networks represent the value function, a current and a target network. The mean squared Bellman error is minimised between the target values (from the target network) and the current values (from the current network), as shown in Equation 3.10. Moreover, Polyak averaging is used to update the target networks. This method is very sample-efficient compared to the expected return learning. As a result, it will be easier to cover a wide range of states in the sampled state space. Consequently, the method is effective in learning a value function that can effectively generalise across the entire state space.

[1]This fixed policy may come from any control lay, however the RL policy is used due to its computational efficiency in determining control actions, enabling large amounts of data points and/or trajectories to be obtained

Obtaining Data The nominal agent (Agent 1 from Table 3.4) was used to acquire the data. A similar methodology for data acquisition as described in Lin, Sun, Xia, *et al.* [20] was employed. Along the nominal trajectory, q ($q \in \mathbb{N}_{>0}$) internal states, x and inputs u, were uniformly sampled from $\hat{\mathbb{X}}^4$ (Equation 3.9) and $\mathbb{U}^3$ at time k respectively. So that at time k, a set denoted as $\hat{S}_k = \{\hat{s}_{k_1}, \hat{s}_{k_2}, ..., \hat{s}_{k_q}\}$, where $\hat{s}_{k_i}$ is constructed using Equation 3.1 from the sampled states and inputs. Each element in $\hat{S}_k$, denoted $\hat{s}_{k_i}$, is taken separately as an initial state and evolved one step in time with the RL agent, receiving a reward $\hat{r}_{k_i}$ and a Boolean d indicating whether a terminal state has been reached. $\hat{s}_{k_i}$ and $\hat{s}_{k+1_i}$ are both normalised as per Equation 3.7 and stored in a transition tuple along with the received reward and d, denoted as $(\hat{s}_{k_i}, \hat{s}_{k+1_i}, \hat{s}_{k_i}, d)$. The environment is then set back to the actual state s_k and evolved for one time step, and the process repeats itself, until the 40 days are over. The transition tuple of the actual system is also stored. The quality of sampled internal states and inputs is important to ensure a value function approximator generalises well across the state space. From the time series plots Figure 3.4 and Figure 3.5, it can be seen that not the entire state space needs to be sampled, especially for the dry mass state, x_1. The control actions were sampled across the entire set $\mathbb{U}^3$ as shown in Equation 2.24. States x_2, x_3, x_4 were sampled with a range slightly larger than their respective minimum and maximum constraints (Equation 2.25) range. This decision was made as it was deemed unnecessary to sample states that significantly violate constraints. Finally, x_1 was sampled around the nominal x_1 trajectory such that the sampled state space $\hat{\mathbb{X}}^4$ is defined as:

$$\begin{aligned} \hat{\mathbb{X}}^4 = \{(x_1, x_2, x_3, x_4) \mid\ & x_1 \in [\hat{x}_{1\,\min}(x_{1_k}), \hat{x}_{1\,\max}(x_{1_k})], \\ & x_2 \in [\hat{x}_{2\,\min}, \hat{x}_{2\,\max}], \\ & x_3 \in [\hat{x}_{3\,\min}, \hat{x}_{3\,\max}], \\ & x_4 \in [\hat{x}_{4\,\min}, \hat{x}_{4\,\max}]\} \end{aligned} \tag{3.9}$$

where the bounds are specified in Table 3.6 and were found empirically.

Parameter	value	unit
$\hat{x}_{1\,\min}(x_{1_k})$	$x_{1_k} \cdot (1 - 0.8) - 0.01$	$kg \cdot m^{-2}$
$\hat{x}_{1\,\max}(x_{1_k})$	$x_{1_k} \cdot (1 + 0.7) + 0.01$	$kg \cdot m^{-2}$
$\hat{x}_{2\,\min}$	$g_2^{-1}(\hat{x_{3_k}}, 400)$	ppm
$\hat{x}_{2\,\max}$	$g_2^{-1}(\hat{x_{3_k}}, 1800)$	ppm
$\hat{x}_{3\,\min}$	7	C°
$\hat{x}_{3\,\max}$	30	C°
$\hat{x}_{4\,\min}$	$g_3^{-1}(\hat{x_{3_k}}, 50)$	$RH_{\%}$
$\hat{x}_{4\,\max}$	$g_3^{-1}(\hat{x_{3_k}}, 100)$	$RH_{\%}$

Table 3.6: Sample State Space bounds

With only 10 additional samples every time step ($q = 10$), the desired state space can be adequately covered.

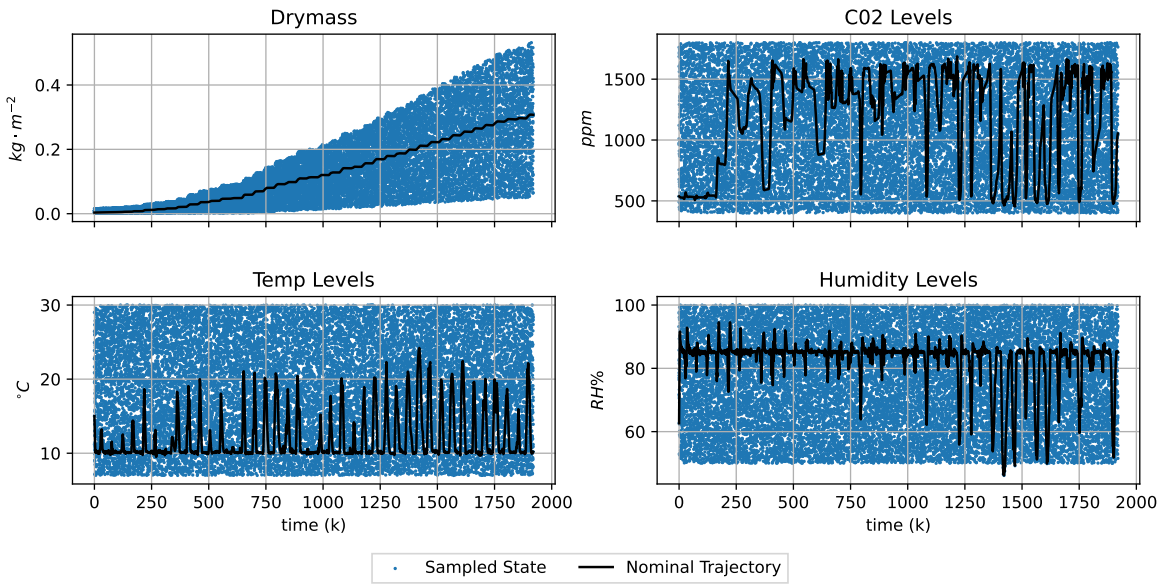

Figure 3.7: Sampled States for Temporal Difference

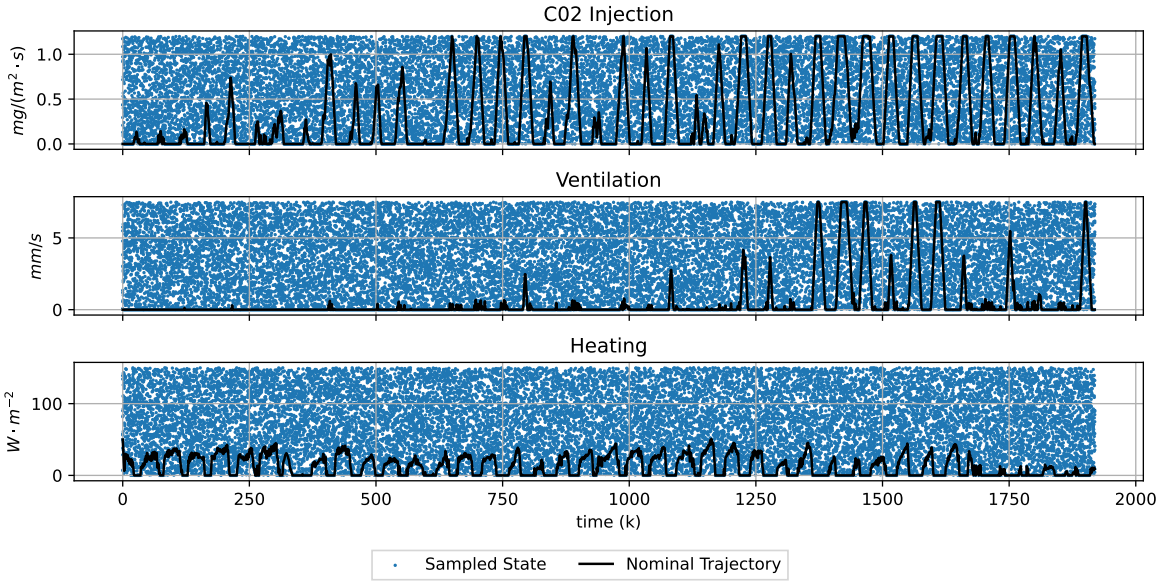

Figure 3.8: Sampled Inputs for Temporal Difference

Figure 3.7 and Figure 3.8 display the states and inputs sampled during a 40-day period, respectively, with an additional ten samples taken at each time step, resulting in a total of 21120 data points or transition tuples.

Training Once data is generated, it is split into a validation and training dataset with a 20% and 80% split to ensure that the function approximator does not overfit the seen data. Transition tuples are sampled from the training set, and the following loss function is minimised:

$$L(\phi, \mathcal{D}) = V_\phi(s_k) - (r_k + (1-d)V_{\phi_{targ}}(s_{k+1})) \tag{3.10}$$

where ϕ and ϕ_{targ} are the current and target weights of the respective function approximators, and $\mathcal{D}$

is the training data set. The Adam optimiser minimises Equation 3.10 over a batch size $\mathcal{B}$. The target weight ϕ_{targ} are updated every learning iteration by Polyak averaging ϕ by:

$$\phi_{targ} \leftarrow (1-\rho)\phi_{targ} + \rho\phi \tag{3.11}$$

where ρ represents the Polyak coefficient, a hyperparameter that needs to be tuned. Despite its widespread usage, it was discovered that the learning of the value function was consistently unstable regardless of the chosen hyperparameters. It failed to learn an appropriate value function, but further research can be conducted to make it functional. However, stable learning was not attained in our work.

3.6.2. Expected Return Learning

This method includes obtaining the expected return of each state visited from a simulated trajectory under a fixed policy and using them as targets for that state. Compared to the temporal difference learning method, this approach offers the benefit of considerably more stable training. Unlike the TD method, the targets remain unchanged while the weights of the function approximator are updated. However, this learning method is much less sample efficient, requiring significantly more data to generalise across the state space. Many trajectories are simulated until termination, and the return must be calculated for each state visited. More importantly, only the starting states of the trajectory are sampled, which makes it harder to obtain the same data spread as the TD method.

Obtaining Data A greater number of starting points must be sampled to obtain a spread comparable to the TD method; however, a significantly larger amount of data is needed because the trajectory must be run through to the end of the simulation. However, targets are calculated for the initial state, and each state encountered along the trajectory by using Equation 2.29.. Given the inferior sample efficiency of this method, it is important to carefully choose the initial points to ensure that the learned value function can effectively generalise across the state space that the agent is likely to encounter during its simulation. A similar approach to subsection 3.6.1 was used. However, all states and inputs were uniformly sampled around a region of the nominal trajectory at time k and not only the dry mass, y_1. Therefore, initial states and inputs were sampled from $\hat{\mathbb{X}}^4$ and $\hat{\mathbb{U}}^3$ and the initial time step k is uniformly sampled across the entire time horizon as shown in Equation 3.12.

$$\begin{aligned}
\hat{\mathbb{X}}^4 = \{(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4) \mid\ & \hat{x}_1 \in [\hat{x}_{1\,\min}(x_{1_k}), \hat{x}_{1\,\max}(x_{1_k})], \\
& \hat{x}_2 \in [\hat{x}_{2\,\min}(x_{2_k}), \hat{x}_{2\,\max}(x_{2_k})], \\
& \hat{x}_3 \in [\hat{x}_{3\,\min}(x_{3_k}), \hat{x}_{3\,\max}(x_{3_k})], \\
& \hat{x}_4 \in [\hat{x}_{4\,\min}(x_{4_k}), \hat{x}_{4\,\max}(x_{4_k})]\} \\
\hat{\mathbb{U}}^3 = \{(\hat{u}_1, \hat{u}_2, \hat{u}_3) \mid\ & \hat{u}_1 \in [\hat{u}_{1\,\min}(u_{1_k}), \hat{u}_{1\,\max}(u_{1_k})], \\
& \hat{u}_2 \in [\hat{u}_{2\,\min}(u_{2_k}), \hat{u}_{2\,\max}(u_{2_k})], \\
& \hat{u}_3 \in [\hat{u}_{3\,\min}(u_{3_k}), \hat{u}_{3\,\max}(u_{3_k})]\} \\
& k \sim U(0, 1919)
\end{aligned} \tag{3.12}$$

where the minimum and maximum limits are calculated as per Equation 3.13

$$\begin{aligned}
\hat{z}_{min} &= z_k \cdot (1-\sigma) \\
\hat{z}_{max} &= z_k \cdot (1+\sigma)
\end{aligned} \tag{3.13}$$

which represent the minimum and maximum range of the sample state space for a specific state and input where z_k represents the nominal trajectory, σ denotes the desired spread of sampled initial states, which is expressed as a percentage. In doing this, initial states maybe uniformly sampled around/near the nominal trajectory. As can be seen from Figure 3.5 and Figure 3.4, it can be observed that the performance of policies can vary significantly with minimal changes in the state and input trajectories. Thus, sampling the trajectories in this manner can be expected to cover enough of the state space to

capture all feasible trajectories. As was done for the temporal difference learning, the fixed policy was generated from the nominal agent. Given that the computation of a control action requires a time of $0.2ms$, it is possible to sample a large number of trajectories to achieve appropriate coverage of state and input spaces. In the case of stochastic conditions, the same state may yield a different return; therefore, if a state has been visited more than once, then the mean of the return is used as training data.

Training Once trajectories are sampled, for each state observed/visited, the total return is calculated, and the tuple (s_k, TR) is stored in a dataset. The dataset is then divided into an 80:20 ratio, with 80% of the data used for training and 20% used for validation. A neural network as a function approximator is now trained with inputs as the state, s_k, and labels as the total return,TR and the loss function in Equation 3.14 is minimised with the Adam optimiser:

$$L(\phi, \mathcal{D}) = V_\phi(s_k) - \mathbb{E}(G_t(s_k)) \tag{3.14}$$

where V_ϕ is the function approximator with weights ϕ and TR is the total return of state s_k. Hyper-parameters include the structure of the neural network, learning rate, and batch size.

Experimental Setup It was decided to train four value functions based on different architects and/or states used as inputs to investigate the effect of the value function in the MPC framework. These models are listed in Table 3.7, along with their distinctive network architecture. All models were trained on 200 epochs with a learning rate of $1 \cdot 10^{-3}$ and batch size of 1024.

Name	Observation Space	Hidden Layers	Neurons per Layer
V_{ϕ_1}	Equation 3.1	2	128
V_{ϕ_2}	Equation 3.1	2	32
V_{ϕ_3}	Equation 3.1	1	128
V_{ϕ_4}	$(y_1(k), k)$	2	128

Table 3.7: Value Functions

Each value function was trained on the nominal agent. Additionally, value functions with the same architecture as V_{ϕ_4} was trained on each stochastic policy, namely Agents 0.05, 0.1' and 0.2. One thousand trajectories were simulated and sampled from these agents, resulting in nearly one million data points consisting of states and their corresponding total return. Finally, the initial state and inputs were sampled with a spread of $\sigma = 0.5$ to ensure adequate coverage of the state and input spaces. The architects were chosen based on the principle that each subsequent architecture model becomes less complex, with V_{ϕ_1} serving as the initial baseline architecture. This is done to investigate the effect of these value functions in the RL-MPC framework.

Performance metrics include the squared error between the predicted and the actual total returns, as shown in Equation 3.14. Moreover, the accuracy of the resulting value function across the simulation period will be visualised by using Equation 3.8.

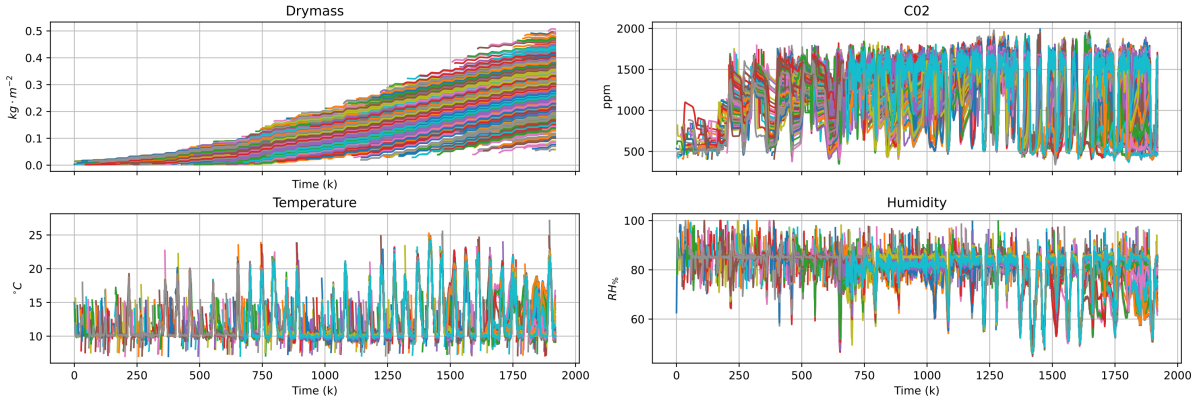

Figure 3.9: Sampled states

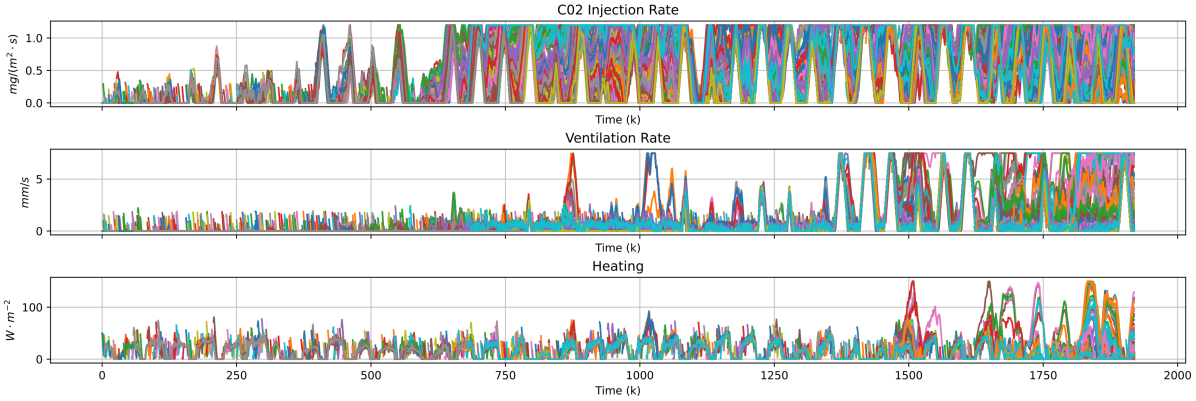

Figure 3.10: Sampled inputs from nominal conditions

Figure 3.9 and Figure 3.10 are the results of all 1000 trajectories sampled from the nominal agent. The figures reveal that the sampled trajectories exhibit less coverage of the state and input spaces than the temporal difference learning, Figure 3.7 and Figure 3.8. Nevertheless, a sufficient level of coverage is achieved. Following the completion of training and validation, a small number of additional trajectories will be sampled to verify the accuracy of the prediction model.

3.6.3. Results

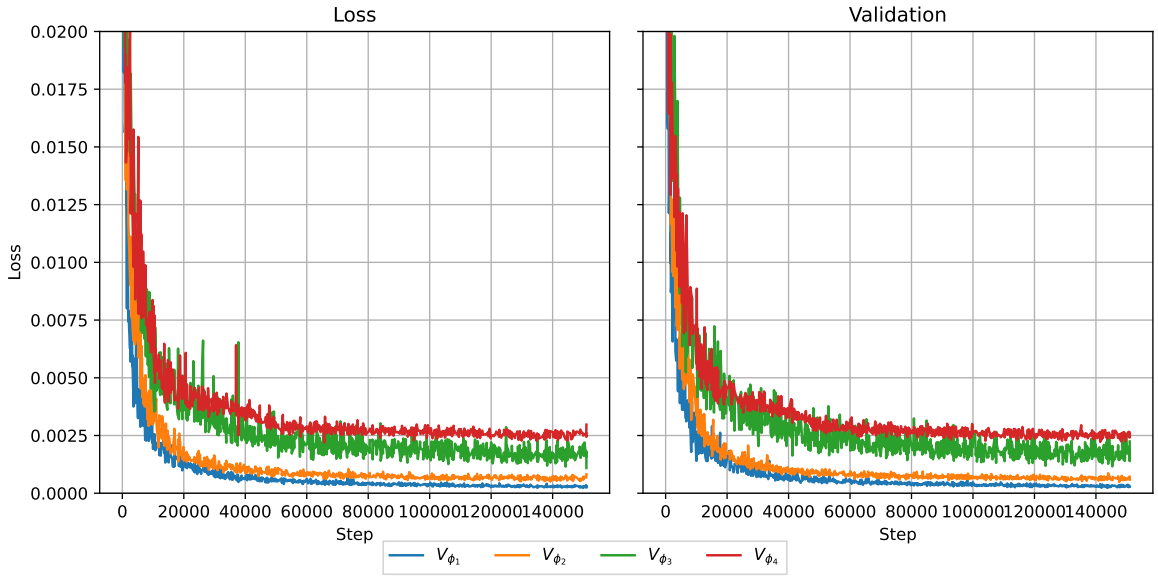

Figure 3.11: Performance Curves, trained on the nominal Agent

Figure 3.11 displays the loss curves of all four models trained on data generated by the nominal agent. As expected, the baseline model (V_{ϕ_1}) achieves the highest accuracy compared to the simpler models. This can be attributed to its more complex structure and using the full observation returned by the agent, with each subsequent simpler model exhibiting lower accuracy. Nevertheless, when using the reduced observation space model, denoted as V_{ϕ_4}, the model demonstrates a high level of accuracy, as evidenced by a mean squared error of less than 0.5% between the actual and predicted values. The high level of accuracy indicates that the value of a state is primarily influenced by the current time and the condition of the crop's dry mass. In addition, when incorporating these value functions into the RL-MPC framework, the optimiser would only need to optimise over two variables for V_{ϕ_4}, as opposed to 13 variables for the other value functions.

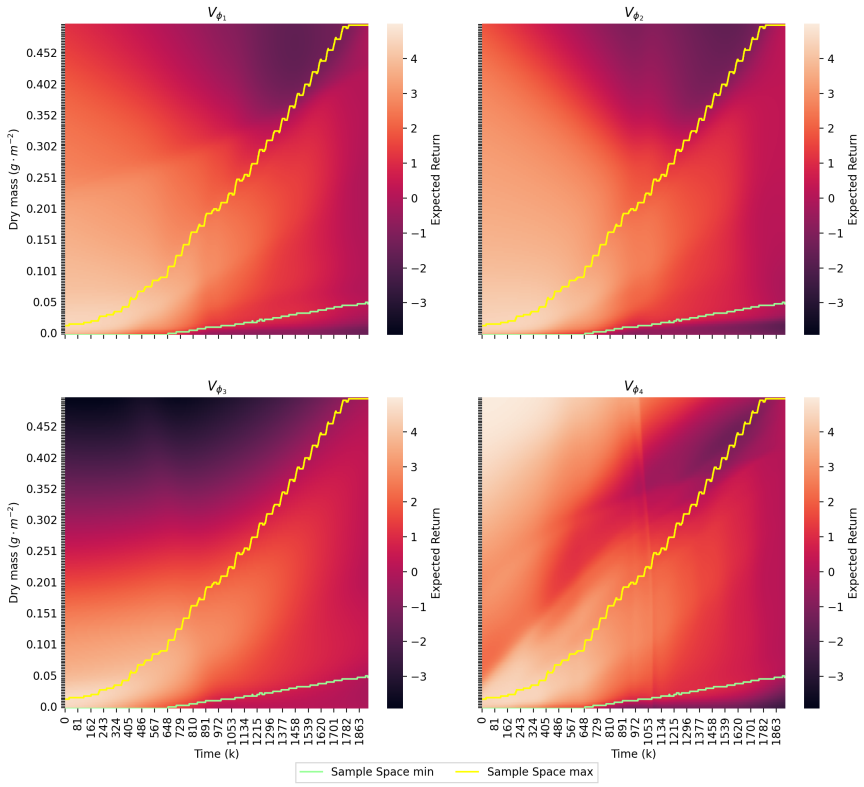

Figure 3.12: Value vs Drymass vs time

Figure 3.12 gives a visual representation of the value of a state given the dry mass and time. It is noted that although V_{ϕ_1}, V_{ϕ_2} and V_{ϕ_3} require an observation space as given in Equation 3.1 to determine its value; since the value is mostly dependent on the dry mass state and time, the other states in the observation space were fixed and only the time and drymass was varied to produce the heatmaps as shown in Figure 3.12. The lower and upper limits in Figure 3.12 indicate the range within which the value function approximator can be deemed reliable. This range corresponds to the portion of the sample space from which the dry mass was sampled for training, i.e. this corresponds to the maximum and minimum dry mass trajectory as seen in Figure 3.9.

The intuitiveness of Figure 3.12 stems from the fact that the highest expected return is observed at the beginning of the growing period, which can be attributed to its longer growing duration. Moreover, at a given time, having a higher dry mass leads to greater expected return. This behaviour is seen across all four models within the training bounds. It is important to note that the greenhouse model limits the dry mass to a maximum of $400g \cdot m^{-2}$. Consequently, there will be minimal expected return for dry masses near this value. Although the nominal agent will make efforts to cultivate the plant by supplying carbon dioxide and heat, it will not result in any growth hence the lower returns in Figure 3.12. It is noted that the reward function is determined by the difference in growth. Consequently, the return is calculated based on the difference in growth starting from the initial state. It is also noted that Figure 3.12 suggests that the value function is smooth concerning the dry mass and time.

Explain these graphs better

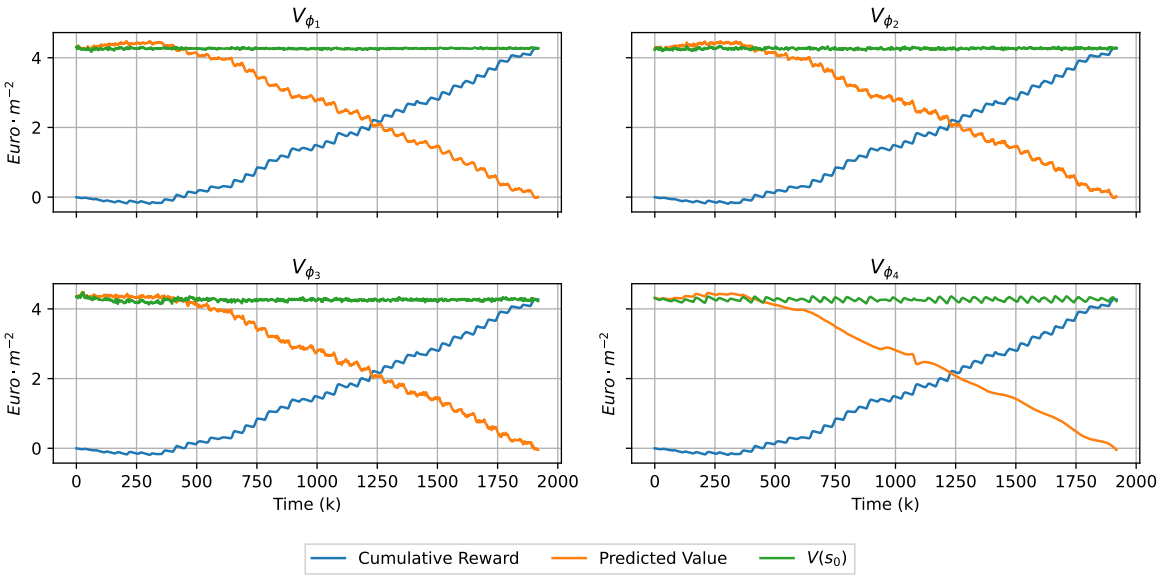

Figure 3.13: Value predictions - Entire Time Horizon

Figure 3.13 displays a time series plot of the cumulative rewards plotted against the predicted value at each time step over the entire simulation period. This plot is similar to that shown in Figure 3.3. Additionally, it includes the calculated $V_\pi(s_0)$ at each time step using Equation 3.8 as a visual indicator of the accuracy of the value function. As demonstrated in Figure 3.13 in conjunction with Figure 3.11, the predicted values show a high level of accuracy. This is evident from the nearly perfect horizontal line at $V_\pi(s_0)$ that spans across the prediction horizon and substantially outperforms the prediction accuracy of the trained value functions in Figure 3.3. However, what is interesting and cannot be seen in Figure 3.12 but in Figure 3.13 are the fluctuations present in each prediction. Naturally, V_{ϕ_4} is not able to make a precise prediction of the value since it only gets the time and dry mass as inputs, however its prediction is much smoother than all the others. While V_{ϕ_1},V_{ϕ_2},V_{ϕ_3} may be more accurate, more fluctuations are present. This observation again suggests that the primary factors determining the value of a state are its dry mass and time. At the same time, the minor fluctuations in rewards and expected returns are influenced by other factors that have minimal impact over the entire time.

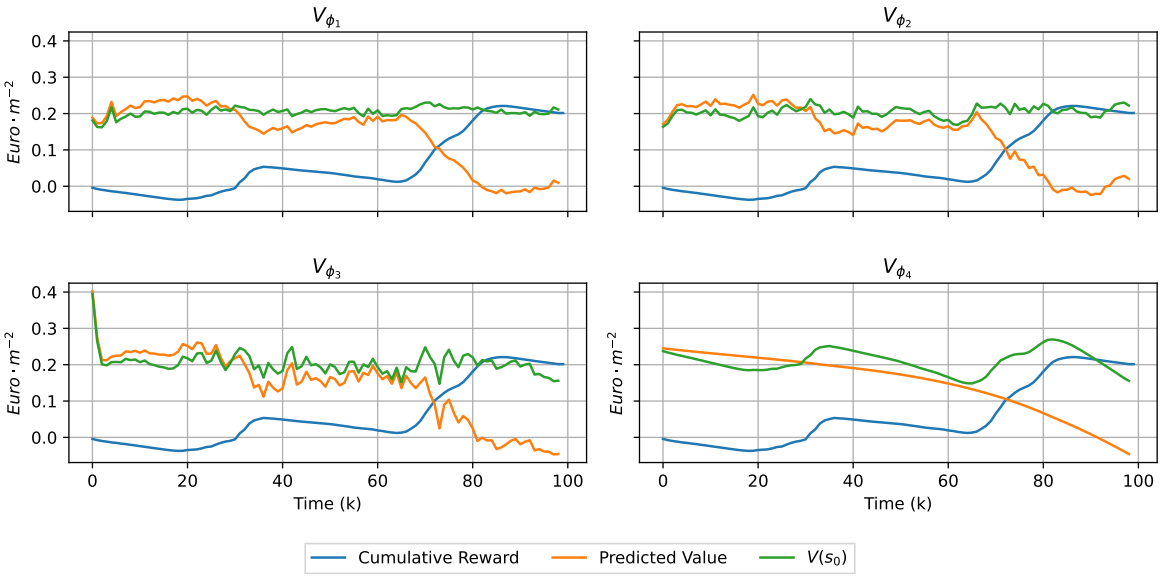

Figure 3.14: Value predictions - 2 Days

Figure 3.14 shows a time series of the cumulative reward and predicted values over two days.

Remark 1 *It is evident that, while $V_{\phi_1}, V_{\phi_2}, V_{\phi_3}$ can produce more precise estimations (as demonstrated by the proximity of their prediction line $V_\pi s_0$ to the actual value), V_{ϕ_4} remains significantly smoother. Moreover, V_{ϕ_3} is not as smooth as it may have seemed in Figure 3.12 and displays the highest level of fluctuations across the four models trained. Although this is one realisation of a trajectory, this behaviour was observed in all simulated trajectories. This behaviour should be noted since it is important for integrating it with MPC.*

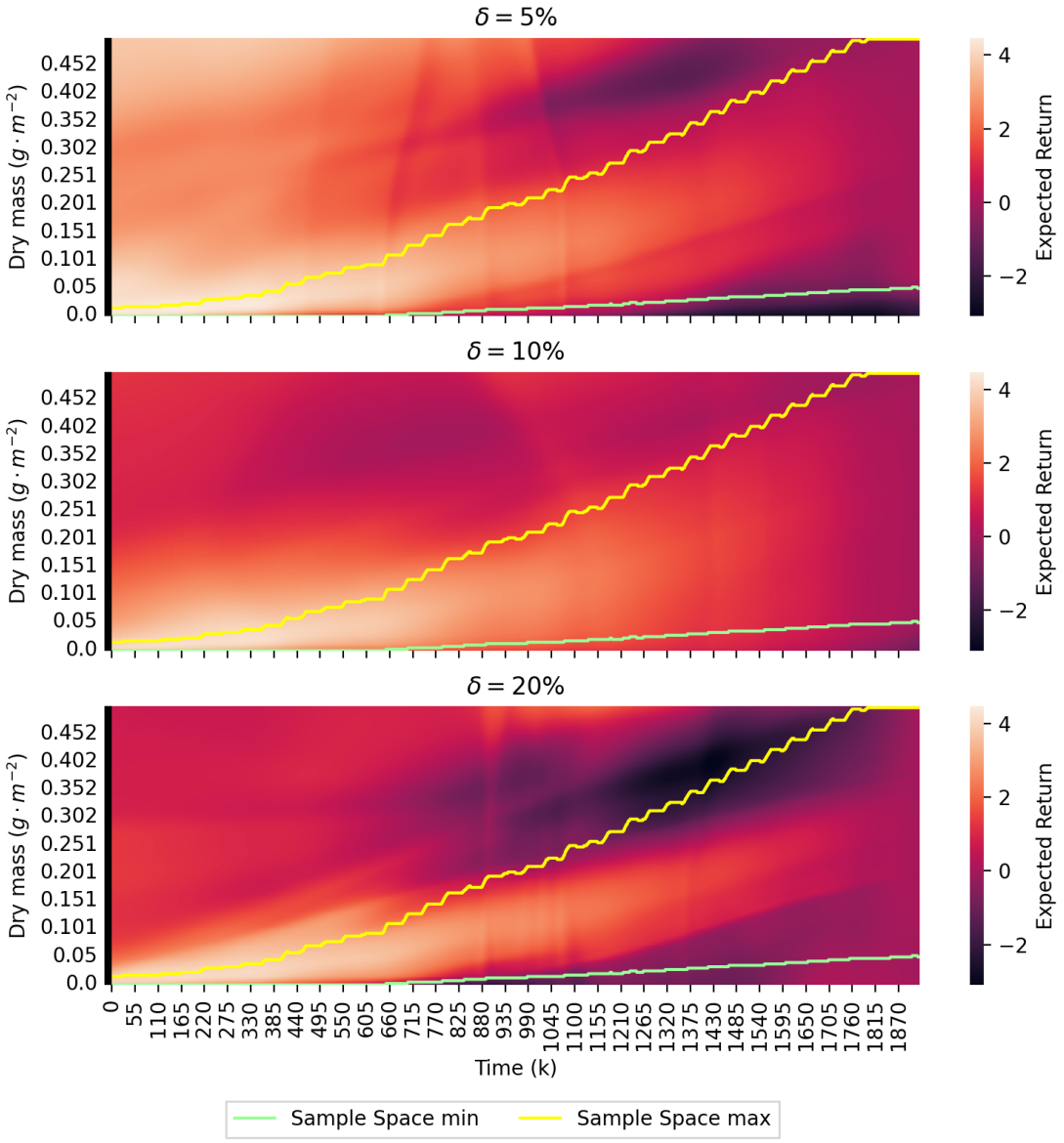

Figure 3.15: Drymass vs Time vs Value - Stochastic

Explain the 20% graph and why there seems like there is noise

Finally, a value function was trained with its corresponding agent for each level of uncertainty in the environment. Figure 3.15 displays the same heatmap as Figure 3.12, yielding similar behaviour compared to the nominal conditions. However, it appears to be more coarse, particularly beyond the range of the training data. Specifically, it seems that, for high uncertainties i.e. 20%), the trained value function exhibits a higher degree of non-linearity than the others trained on lower levels of uncertainty. This may indicate that more sample trajectories are required for further training; however, it still seems to generalise well in the region of the nominal trajectory.

3.7. Conclusion

The objective of this chapter was to develop a policy that is competitive with MPC and a value function that can accurately approximate the expected return given the state of the environment, both for the nominal and stochastic environment. It was discovered that a policy trained with a discount factor of 1 resulted in a critic that provided inaccurate value estimations and a policy that performed worse than an agent trained with lower discount factors. While reducing the discount factor leads to improved policies and a more precise critic, the critics fail to provide information about the entire growth period. Furthermore, the critic must be differentiable in order to integrate the critic with MPC. This requires the use of the tanh activation function. However, using this activation function leads to a less effective policy than using a non-differentiable activation function like ReLu.

Given these obstacles, it became evident that additional measures were necessary. It was decided to train a separate critic/value-function on the best RL policy obtained. Therefore, lower discount factors and non-differentiable activation functions may be used in the policy training. The best policy found and trained on the nominal data was denoted as Agent 1 and later referred to the nominal agent, with parameters shown in Table 3.4. Using the same hyper parameters as Agent 1, three different agents were learned based on each level of uncertainty in the greenhouse model, namely Agents 0.05, 0.1 and 0.2.

Empirical evidence demonstrated that the stochastic agents were more robust to noise than the nominal agent. Agents trained with greater levels of uncertainty exhibited policies with reduced variance across different levels of uncertainty. While the overall performance decreased as uncertainty levels increased in the environment, the decline became more gradual when trained on higher levels of uncertainty.

After generating sufficient policies for the nominal and stochastic environments, it was necessary to train an appropriate critic/value function approximator. Four different model architectures were used to train the critics, and each model architecture became progressively less complex. Every agent, whether in the nominal or stochastic case, had a critic trained to evaluate its policy under its particular level of uncertainty.

Results showed that accurate value function approximators could be trained, provided enough data was sampled. However, although they were accurate, upon closer inspection the predictions displayed small fluctuations, which could be problematic later when integrating the function appropriators into the RL-MPC framework. However, the simplest model architecture, trained on only the dry mass state and time, provided very smooth predictions, albeit not as accurate as the more complex models. This model architecture was used to train value function approximators for the stochastic agents. These agents and corresponding value functions established the foundation for incorporating RL with MPC into the RL-MPC framework.

4

Model Predictive Control Setup

This chapter presents the setup and creation of the MPC controller and investigates the performance of the resulting controller in the greenhouse environment. Moreover, the effect of the prediction horizon is investigated in nominal and stochastic conditions. The works in this chapter are similar to what is presented in Boersma, Sun, and van Mourik [34] and Morcego, Yin, Boersma, *et al.* [9]. However, direct comparisons are not possible.Boersma, Sun, and van Mourik [34] develops a robust sample-based controller, where this controller optimises over a number of possible trajectories (due to the parametric uncertainty). While this may produce a policy that will result in lower variance and a higher mean final cumulative reward than a conventional MPC, this chapter will focus on constructing a conventional MPC. This is done to examine the effects of uncertainty on this controller and later asses whether the integration of RL can mitigate these effects. In Morcego, Yin, Boersma, *et al.* [9], such an MPC controller is developed. However, it does not provide specific weather data. Finally, the optimisation goal used in this thesis differs from that in both papers. However, a qualitative comparison may be done.

4.1. Greenhouse MPC problem formulation

It is important that the optimisation goal is equivalent to compare the performance of the MPC to RL and the RL-MPC controller directly. As discussed in subsection 2.1.4, it is desired to optimise the economic benefit of the greenhouse environment. Similarly to section 3.1, the optimisation goal of the MPC is done to ensure that the sum of stage costs is equal to the actual economic benefit of the system. Therefore, the following optimisation goal is solved at every time step:

$$\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) \tag{4.1a}$$

$$\text{s.t.} \quad x(k+1) = f(x(k), u(k), d(k), p), \tag{4.1b}$$

$$y(k) = g(x(k+1), p), \tag{4.1c}$$

$$-\delta u_{max} \leq u(k) - u(k-1) \leq \delta u_{max}, \tag{4.1d}$$

$$u_{\min} \leq u(k) \leq u_{\max}, \tag{4.1e}$$

$$x(k_0) = x_{k_0}. \tag{4.1f}$$

$$x(k_0) = x_{k_0}. \tag{4.1g}$$

This MPC OCP is similar to that discussed in subsection 2.3.1, but it does not include a terminal cost or constraint/region. Furthermore, the constraints are aligned with that of the greenhouse environment. To ensure that the optimisation goal is exactly the same as the RR reward function (section 3.1), the cost function $V(u(k), y(k))$ becomes:

$$
\begin{aligned}
l(u(k), y(k)) = -c_{p_3}(y(k) - y(k-1)) + c_{p_1}u_1 + c_{p_2}u_3 + \sum_{i=1}^{6} s_i(k) \\
\text{where} \quad s_i(k) \geq 0, \\
s_1(k) \geq c_{p_{C02}} \cdot (y_2^{min} - y_2(k)), \\
s_2(k) \geq c_{p_{C02}} \cdot (y_2(k) + y_2^{max}), \\
s_3(k) \geq c_{p_{T_{lb}}} \cdot (y_3^{min} - y_3(k)), \\
s_4(k) \geq c_{p_{T_{ub}}} \cdot (y_3(k) + y_3^{max}), \\
s_5(k) \geq c_{p_H} \cdot (y_4^{min} - y_4(k)), \\
s_6(k) \geq c_{p_H} \cdot (y_4(k) + y_4^{max}),
\end{aligned}
\tag{4.2}
$$

The slack variables are introduced to accommodate the linear penalties on the outputs as in equation Equation 4.1, resulting in an optimisation problem equivalent to that of RL. The penalty constants $c_{p_{C02}}, c_{p_{T_{lb}}}, c_{p_{T_{ub}}}, c_{p_H}$ are the same as those used in the RL problem formulation and given in section 3.1. While MPC can impose hard constraints on the states of the system, which is one of its advantages over RL, it was decided that the same penalty on state violations must be given for a direct comparison between algorithms. Lastly, the policy generated by the MPC is denoted $\kappa(x, u, d, p)$ where the optimal control action to take at time k is

$$
u_k^* = \kappa(x_k, u_{k-1}^*, d_k, p) \tag{4.3}
$$

Simulation Setup Furthermore, the experimental setup is identical to that used in section 3.2 except for normalising observations. The performance metrics are the sum of the stage costs during the simulation period, equivalent to the EPI minus the sum of the state violations. The performance metric is calculated for the stochastic environment by taking the average of the sum of stage costs over 30 simulation periods and considering the variance of the resulting stage costs. In the case of stochasticity, the MPC algorithm still solves the optimisation problem defined by equation Equation 4.1 using the nominal system parameters. However, it is during the system evolution that the uncertainty in parameters is considered. Therefore, a deterministic prediction model is used but the evolution of the state and the output measurements are both uncertain.

Finally, to increase the computational time and feasibility of the MPC solver, the solver is warm-started with the previous solution to reduce the number of iterations required to reach the optimal solution [1]. The computational time of computing the optimal control actions will also be examined for each prediction horizon. These performance metrics are given for all prediction horizons. Finally, the MPC framework is built using the open-source software Casadi **Andersson2018** and the non-linear solver IPOPT [58] in python.

4.2. Deterministic Results

Simulations are conducted for every prediction horizon, N_p.The prediction horizons to be tested will include time intervals of 1, 2, 3, 4, 5 and 6 hours. Results include the final cumulative reward and the average time required to solve Equation 4.1 for each prediction horizon (averaged across the growing period).

[1]The solution to the OCP is heavily reliant on initial guesses due to the non-linearity nature of the problem, lagrangian multipliers from the previous solutions were also reused to mitigate instability in the solver. These instabilities are especially prevalent in longer prediction horizons.

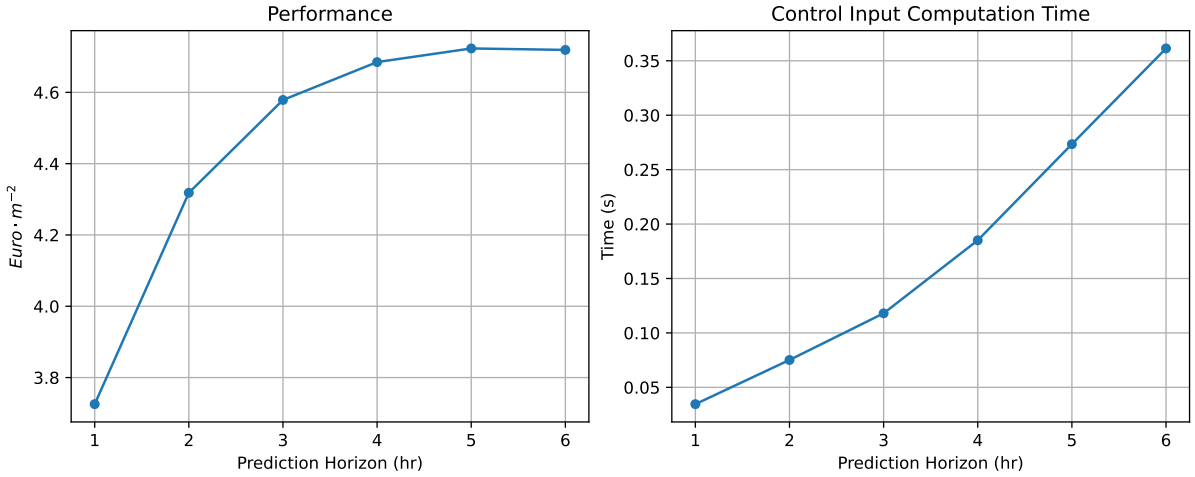

Figure 4.1: The final cumulative reward achieved using nominal MPC and the average computation time for each prediction horizon

Figure 4.1 exhibits the performance and the computational time of the MPC for each prediction horizon. It can be seen that performance increases with an increase in prediction horizon up to 6 hours. However, this increase in performance is not guaranteed for an economic model predictive controller as stated in Ellis, Durand, and Christofides [55] and Amrit, Rawlings, and Angeli [57], without a sufficiently long time horizon or an appropriate terminal cost function or constraints. Although not entirely clear in Figure 4.1, a prediction horizon of 6 hours produces a slightly lower-performing policy than a prediction horizon of 5 hours. Whether this is due to instabilities in the solver or the nature of the economic optimisation problem remains uncertain.
As expected, the computational time in computing the control action increases with increased complexity. It is noted that the RL algorithm can calculate the control action in approximately 0.2 milliseconds, while the fastest MPC controller, with a prediction horizon of 1 hour, takes approximately 35 milliseconds, making it 2 order of magnitudes slower. While this outcome is not surprising, it effectively demonstrates the computational demand of MPC, specifically for a highly non-linear model. Nevertheless, setting up and implementing the MPC controller was considerably easier than RL. RL demanded extensive time investment in fine-tuning hyperparameters. Despite all the fine-tuning in RL, MPC still outperforms the RL agent; this performance difference is analysed in chapter 5.

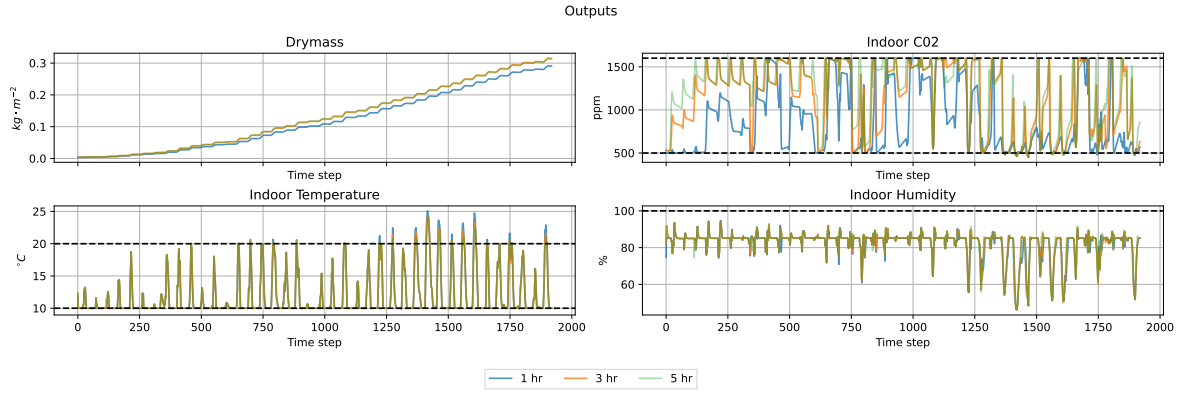

Figure 4.2: MPC 1hr and 5hr Time series of greenhouse outputs

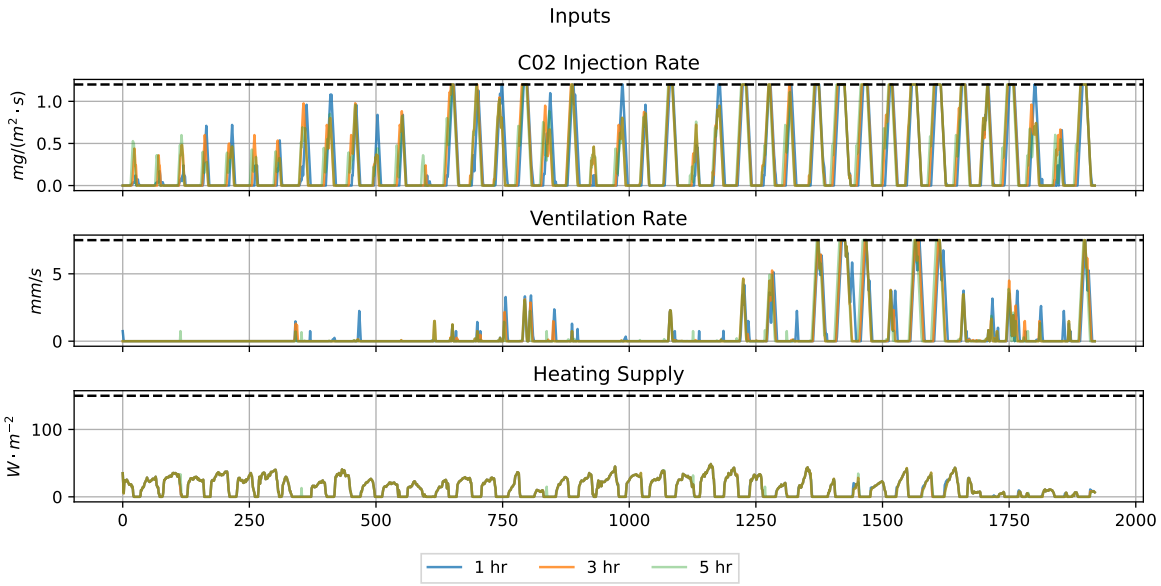

Figure 4.3: MPC 1hr and 5hr time series of controller inputs

Figure 4.2 and Figure 4.3 display the trajectories of the states and inputs of the best-performing MPC controller (5-hour prediction horizon) and the worst-performing (1-hour). These are similar to the RL agents Figure 3.4 and Figure 3.5, respectively. It is noted that, in both cases (RL and MPC), the better-performing policies rapidly raise the indoor C02 levels at the beginning of the growing period by reducing ventilation and increasing C02 injection. However, the trajectories of other states and inputs, particularly the indoor temperature, humidity and heating, appear unchanged. Either the temperature does not play a vital role in plant growth, or the prediction horizon is not long enough to see its effect on initial growth stages. However, it is clear that the MPC controller uses heating at night and irradiance during the day to ensure that temperature constraints are met. These results are very similar to Morcego, Yin, Boersma, *et al.* [9] and Boersma, Sun, and van Mourik [34], although difficult to compare quantitatively due to the difference in the optimisation goal and MPC problem formulation.

4.3. Stochastic Results

For each stochastic level, $\delta = 5\%$, $\delta = 10\%$, $\delta = 20\%$ were analysed with prediction horizons of 1, 3, and 5 hours. Additional prediction horizons were not considered due to limitations in simulation time.

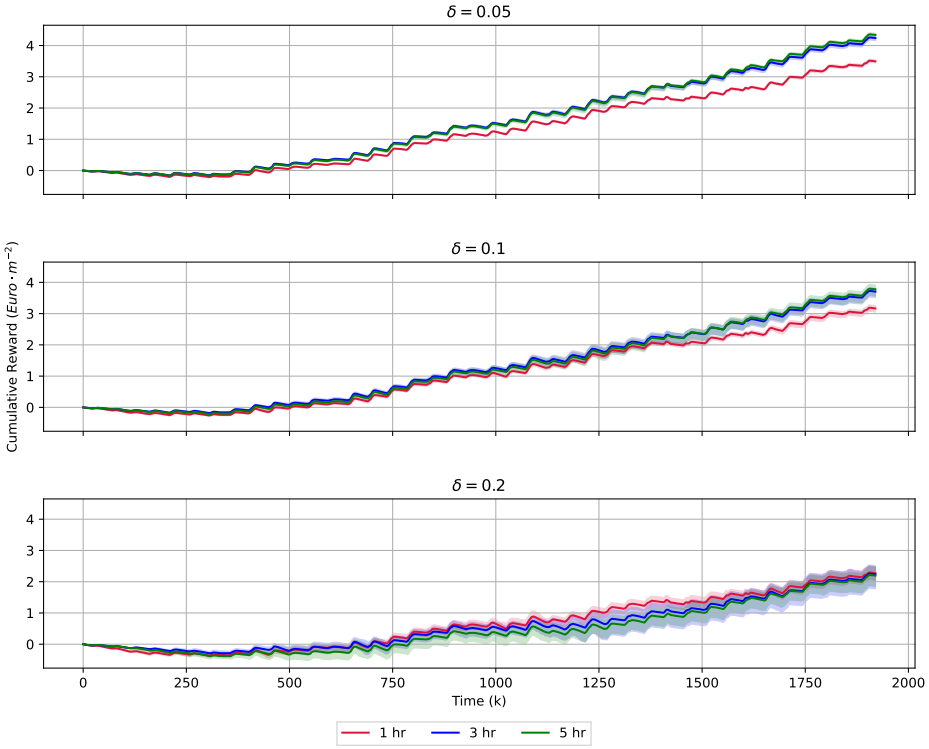

Figure 4.4: Time evolution of cumulative rewards. Displays the evolution of the cumulative reward over the growing period for each stochastic and prediction horizon level. Solid lines represent the mean cumulative reward trajectory with a range indicating the minimum and maximum trajectory recorded in the 30 runs.

Figure 4.4 shows the impact of introducing uncertainty and its effect on the cumulative reward obtained by the MPC controller at each time step. While a $\delta = 0.05$ does not noticeably impact performance, there is significant performance degradation for $\delta = 0.2$. Also notable is that a longer prediction still results in a higher mean cumulative reward. However, for high uncertainties, specifically $\delta = 0.2$, it seems that a longer prediction horizon has less of an impact on performance, and may even be detrimental in extreme cases of uncertainty.

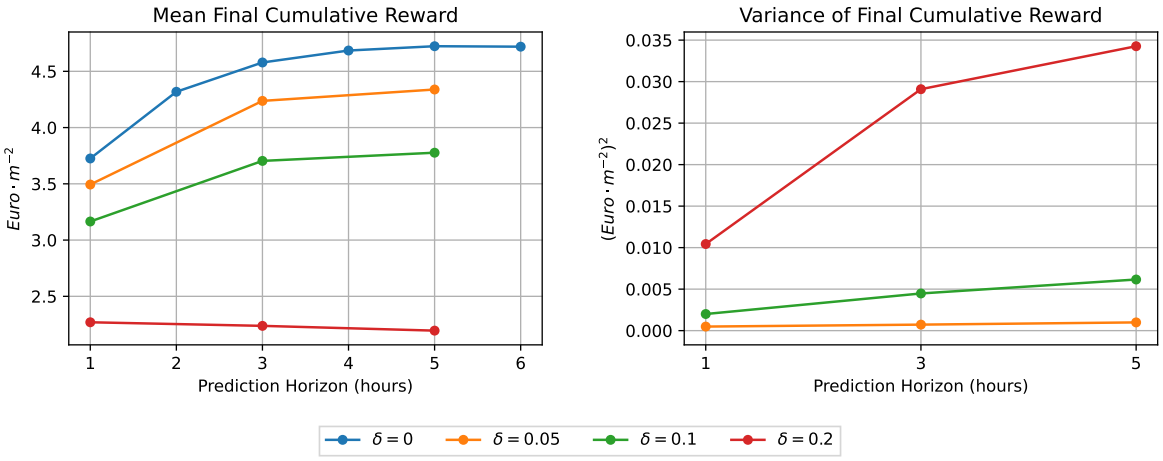

Figure 4.5: MPC performance in stochastic conditions

Figure 4.5 depicts the final mean and variance values of the cumulative reward for each prediction horizon and uncertainty level, including nominal conditions. Figure 4.5 may provide a better representation

of how stochasticity affects the final performance. Longer prediction horizons appear to have less performance increases and might become detrimental at very high uncertainty levels. Moreover, there is a clear adverse effect on variance as the prediction horizon is increased for all uncertainty levels. This can be attributed to longer prediction horizons becoming progressively less accurate due to the increasing uncertainty of the model parameters. It is acknowledged that a conventional MPC with no knowledge of the uncertainty present is not adept at handling it (although methods such as state estimation exist), and these results confirm this. Although it is possible to formulate a robust MPC, such as in Boersma, Sun, and van Mourik [34], it introduces a heavy computational burden on the controller. As previously mentioned, RL's decrease in performance due to uncertainty is not as drastic as MPC's as well as the variance in the final cumulative reward is also substantially lower. Thus, it is worth exploring whether the RL agent can assist the MPC in mitigating the adverse impacts of parametric uncertainty.

4.4. Conclusion

In conclusion, this chapter has presented the setup and implementation of the MPC framework whereby the EMPC OCP was designed to maximise the economic advantage of the greenhouse by aligning its optimisation objective with the optimisation goal specified in subsection 2.1.4. However, additional slack variables were introduced to align the optimisation problem with what the RL agent is optimising for more meaningful comparisons. The MPC controller's performance was thoroughly examined under nominal and stochastic conditions, focusing on various prediction horizons. The findings and comparisons drawn in this chapter shed light on the controller's efficacy and limitations in managing a greenhouse environment.

Under deterministic conditions, MPC exhibited varying performance across different prediction horizons. It demonstrated improved economic outcomes with longer prediction horizons, albeit with increased computational demands. However, increasing the prediction horizon does not guarantee an increase in performance without an adequate terminal cost function and/or terminal constraint/region. Notably, while MPC outperforms RL in final cumulative reward under deterministic settings, its computational overhead was significantly higher, highlighting a trade-off between computational efficiency and performance optimisation. The scalability of the computational time is considerably inferior to that of the RL agent. Therefore, the model is typically linearised or simplified in more complex systems, leading to a suboptimal policy. However, it is important to note that it seems that the MPC is not as myopic as original assumed. These short prediction horizons appear to have satisfactorily optimised the greenhouse environment's slow dynamics and sparse rewards.

Furthermore, stochastic simulations revealed MPC's vulnerability to uncertainty in the environment. Increasing uncertainty levels adversely affected MPC's performance, whereby increasing the prediction horizon may become harmful under extreme uncertainty, leading to compromised economic benefits and increased variability in the final cumulative reward. These results underscored MPC's limited robustness against stochastic influences compared to RL.

In conclusion, while MPC proved effective in deterministic scenarios with appropriate prediction horizons, its performance degradation under stochastic conditions necessitates exploring hybrid RL-MPC strategies to enhance its performance and robustness. Appropriate additions to its formulation must be made to guarantee performance for an EMPC, which the RL agent could potentially solve.

5

Deterministic RL-MPC

This chapter aims to construct the RL-MPC framework and examines various implementations to determine their effectiveness. The resulting controllers are evaluated by comparing them with the MPC and RL controllers developed in previous sections. The implementations serve as a means for the reader to understand the construction of the final RL-MPC algorithm, as each subsequent implementation builds on the previous one. In addition, this chapter will analyse the nominal case in which the parameters of the model and environment are known.

The smoothness of the value function curve is crucial because, when optimised, it can generate numerous local optima, which can disrupt the controller's performance.

5.1. Implementation

Although there are numerous implementations of RL-MPC, limited research focuses on maximising economic benefit specifically for continuous state and action spaces while training RL separately from MPC. As stated in Ellis, Durand, and Christofides [55] and Amrit, Rawlings, and Angeli [57], an EMPC without a terminal constraint and terminal cost function does not provide performance and stability guarantees. Specifically, Ellis, Durand, and Christofides [55] states that a terminal constraint is required to ensure closed-loop performance, while Amrit, Rawlings, and Angeli [57] extends this concept by proving that applying a terminal region constraint with an appropriate terminal cost function is required to guarantee closed-loop performance.Amrit, Rawlings, and Angeli [57] further claims that the terminal cost function with a terminal region is superior to the terminal constraint because it increases the size of the feasible set of initial conditions and may possibly improve the closed-loop performance. However, finding such suitable terminal constraints and cost functions proves to be very difficult. The objective of this thesis is to ascertain whether the RL agent is capable of providing this.
Furthermore, when considering the RL perspective in these implementations, it is important to note that the learned value function is merely an approximation. Consequently, when this value function is used in MPC, it is effectively unrolled, and value iterations are executed. This process can result in an improved policy compared to the original policy that generated the value function.

The integration of RL into MPC will increasingly involve more complex implementations to analyse the impact at each stage. Firstly, initial guesses from the actor will be examined. Subsequently, the RL agent will establish a terminal constraint. Following this, the RL agent will define and determine a terminal constraint region. The various value functions trained by the nominal agent (Table 3.7) will then be used as the terminal cost function, with and without the terminal region constraint. Lastly, a parallel problem will be presented to explore a slightly alternative application of the value function. The integration of the value function into the MPC's optimal control is facilitated by L4Casadi **<empty citation>**

5.1.1. RL-MPC problem formulations

Implementation RL-MPC 1 Implementation of RL-MPC 1 is identical to Equation 4.1 but includes initial guesses. However, instead of using the previous solution to the state and input trajectories as

initial guesses, the RL agent provides these initial guesses. Two sets of initial guesses will be tested and compared with one another. The solution to the OCP in Equation 4.1 at time k can be denoted as:

$$\begin{aligned} \mathbf{x}_{k|k} &= [x_{k|k}, x_{k_+1|k}, x_{k+2|k}, ..., x_{k+N_p|k}]^T \\ \mathbf{u}_{k|k} &= [u_{k|k}, u_{k+1|k}, ..., u_{k+N_p-1|k}]^T \end{aligned} \tag{5.1}$$

The two sets of initial guesses at the k step is denoted as:

$$\begin{aligned} \tilde{\mathbf{x}}_{k|k} &= [\tilde{x}_{k|k}, \tilde{x}_{k+1|k}, ..., \tilde{x}_{k+N_p|k}]^T \\ \tilde{\mathbf{u}}_{k|k} &= [\tilde{u}_{k|k}, \tilde{u}_{k+1|k}, ..., \tilde{u}_{k+N_p-1|k}]^T \end{aligned} \tag{5.2}$$

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= [\hat{x}_{k|k}, \hat{x}_{k+1|k}, ..., \hat{x}_{k+N_p|k}]^T \\ \hat{\mathbf{u}}_{k|k} &= [\hat{u}_{k|k}, \hat{u}_{k+1|k}, ..., \hat{u}_{k+N_p-1|k}]^T \end{aligned} \tag{5.3}$$

such that

$$\begin{aligned} \tilde{\mathbf{x}}_{k|k} &= [\mathbf{x}_{k|k-1}, f(x_{k-1+N_p|k-1}, \pi(x_{k-1+N_p|k-1}), d_{k+Np|k}, p)]^T \\ \tilde{\mathbf{u}}_{k|k} &= [\mathbf{u}_{k|k-1}, \pi(x_{k-1+N_p|k-1})]^T \end{aligned} \tag{5.4}$$

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= [x_{k|k}, f(x_{k|k}, \pi(x_{k|k}), d_{k|k}, p), ..., f(x_{k+N_p-1|k}, \pi(x_{k+N_p-1|k}), d_{k_1+Np-1|k}, p)]^T \\ \hat{\mathbf{u}}_{k|k} &= [\pi(x_{k|k}, \pi(x_{k+1|k}), ..., \pi(x_{k+Np-1|k})]^T \end{aligned} \tag{5.5}$$

Equation 5.4 takes the previous time steps solution, shifts it in time and uses the policy $\pi(\cdot)$, as provided by the actor, to calculate the optimal action and resulting state to take at the last time step. This method can be interpreted as extending the horizon. So, for every time step, the initial guesses of the sequence of actions and states are extended by one time step. Equation 5.5 unrolls the RL policy $\pi(\cdot)$ from the current state until the end of the prediction horizon. Consequently, the solutions obtained from the previous time, $\mathbf{x}_{k|k}$ and $\mathbf{u}_{k|k}$, step are disregarded, thus generating a new sequence without relying on the previous time step's solutions. It must be noted that, for the first time step, $k = 0$, initial guesses given by Equation 5.3 are used for both cases, therefore $\tilde{x}_{k|k} \leftarrow \hat{x}_{k|k}$ and $\tilde{u}_{k|k} \leftarrow \hat{u}_{k|k}$.

Based on prior analyses, generating these initial guesses is extremely fast. Furthermore, since it comes from a policy comparable to the MPC's, these initial guesses can be used for more than just initial guesses, but also to generate terminal constraints. The subsequent implementations explore the significance of these initial guesses, particularly the initial guesses mentioned in Equation 5.2.

Implementation RL-MPC 2 The second implementation incorporates a terminal constraint in addition to the initial guesses. Risbeck and Rawlings [59], Amrit, Rawlings, and Angeli [57], proves that an appropriate terminal constraint can result in superior performance. It is shown that, through this terminal constraint, for a time-varying system, performance can be guaranteed to be at least as good as a reference trajectory, $(\mathbf{x}_r, \mathbf{u}_r)$, that serves as a basis for a meaningful economic performance for the system. The terminal constraint should be imposed to keep the system close to this reference trajectory. Since the RL policy can be used as the optimal reference policy, it can be guaranteed that the resulting policy will be at least as good as the RL policy. The asymptotic average performance of the EMPC can be guaranteed to be no worse than the performance of a reference trajectory under the following assumptions from Amrit, Rawlings, and Angeli [57] and Risbeck and Rawlings [59]:

Assumption 1 (Properties of constraint sets) The set $\mathbb{Z}$ is compact, where $\mathbb{Z} \subseteq \mathbb{X} \times \mathbb{U}$

Assumption 2 (Continuity of cost and system) The functions $l(\cdot)$, $f(\cdot)$ are continuous on $\mathbb{Z}$. The terminal cost function $V_f(\cdot)$ is continuous on $\mathbb{X}_f$

Assumption 3 The reference trajectory $(\mathbf{x}_r, \mathbf{u}_r)$ satisfies $f(x_r(k), u_r(k), d(k), p) = x_r(k+1)$

Assumption 4 (Stability assumption) There exist a compact terminal region $\mathbb{X}_f \subseteq \mathbb{X}$, containing the point x_r in its interior, and control law $\kappa_f : \mathbb{X}_f \rightarrow \mathbb{U}$, such that the following holds

$$V_f(f(x, \kappa_f(x))) \leq V_f(x) - l(x, \kappa_f(x)) + l(x_r(t), u_r(t)) \quad \forall x \in \mathbb{X}_f \tag{5.6}$$

Assumptions 1, 2, and 3 hold that since all states and inputs are bounded, the stage cost, as defined in Equation 4.2, is continuous. For this implementation,$V_f(\cdot) \equiv 0$ since no terminal cost function is used. Therefore, assumption 3 holds if $x, \kappa_f(x)$ is constrained to x_r, u_r. Therefore, if the terminal state was constrained to the last initial guess of Equation 5.2, it would constrain it to the RL's reference policy. Therefore, the terminal constraint would be imposed on the last state of the solution such that:

$$\begin{aligned} x_{k+Np|k} &= \tilde{x}_{k+Np|k} \\ u_{k+Np-1|k} &= \tilde{u}_{k+Np-1|k} \end{aligned} \tag{5.7}$$

Nevertheless, this implementation is also evaluated for initial guesses provided by Equation 5.3. However, it is important to note that, because the reference trajectory depends on the current state and, therefore, changes at each time step, no meaningful comparison can be made to this changing reference trajectory. So, while it will perform at least as well as this changing reference trajectory, the performance on this changing trajectory is unknown.

Implementation RL-MPC 3 Implementation 3 builds upon implementation 2, in that instead of providing a terminal constraint, a terminal region as provided by the Equation 5.2 and Equation 5.3 is used. The terminal region is defined as:

$$\begin{aligned} (1 - \delta_T)\tilde{x}_{k+Np|k} \leq x_{k+Np|k} \leq (1 + \delta_T)\tilde{x}_{k+Np|k} \\ (1 - \delta_T)\tilde{u}_{k+Np-1|k} \leq u_{k+Np-1|k} \leq (1 + \delta_T)\tilde{u}_{k+Np-1|k} \end{aligned} \tag{5.8}$$

[57] suggests that this has the same performance guarantees as Implementation 2 under the same assumptions. However introducing a terminal region for the terminal state makes it difficult to meet assumption 3 as shown in Equation 5.6. However, [57] suggest that providing a terminal region may be more beneficial than a terminal constraint since more freedom is given to the EMPC. Finally, a terminal constraint and initial guesses will also be provided by Equation 5.3 to investigate performance. However, since unrolling from the current state does not result in following a fixed trajectory, no meaningful performance comparisons can be made to the RL agent (i.e. the reference trajectory).

Implementation RL-MPC 4 Implementation 4 only includes the value function learned in section 3.6 and initial guesses given in Equation 5.2. This implementation examines the effect of the value function on the performance of the resulting controller. The value function can be incorporated into Equation 4.1 by defining a cost function:

$$\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) - V_{\phi_i}(s'(k_0 + N_p)) \tag{5.9}$$

where V_{ϕ_i} represents the learned value function and $s'(k_0 + N_p)$ is the normalisation of $s(k_0 + N_p)$, as per Equation 3.7. For V_{ϕ_1},V_{ϕ_2} and V_{ϕ_3}, $s(\cdot)$ is observation as returned by the agent (Equation 3.1), while $s(\cdot)$ when using V_{ϕ_4} is $(y_{k_0+N_p}, k_0 + N_p)$ as discussed in section 3.6. This implementation aims to evaluate the impact of different neural network architectures, including a deep neural network (V_{ϕ_1}), a smaller deep neural network (V_{ϕ_2}), a shallow neural network (V_{ϕ_3}), and a reduced order deep neural network trained to learn the value function on only two system states (V_{ϕ_4}). According to approximate dynamic programming as stated in [19], this policy could be better than the policy that generated the value function. However, the performance is heavily dependent on the quality of the value function. If the approximate value function is inaccurate and the errors are significant and systematic, then unrolling this value function could lead to a worse policy. This can be considered a naive implementation

of RL-MPC. It can be conceptually viewed as either unrolling the value function and subsequently performing a minimisation or providing the MPC knowledge of the future, essentially extending its prediction horizon. Note that the value function is maximised by minimising the negative of the value function, since it represents total return and not cost.

Implementation RL-MPC 5 The implementation of RL-MPC 5 essentially combines RL-MPC 3 and RL-MPC 4. Amrit, Rawlings, and Angeli [57] states that finding an appropriate terminal cost function and corresponding terminal region proves non-trivial to satisfy assumption 3. This implementation is also claimed to be superior to the terminal point constraint of implementation RL-MPC 2 ([57]). Furthermore, RL-MPC 5 provides additional knowledge of the future rewards to be expected compared to RL-MPC 3; therefore, an increase in performance over RL-MPC 3 can be expected. Therefore, the resulting RL-MPC OCP is defined as:

$$\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) - V_{\phi_4}(s'(k_0+N_p)) \tag{5.10a}$$

$$\text{s.t.} \quad x(k+1) = f(x(k), u(k), d(k), p), \tag{5.10b}$$

$$y(k) = g(x(k+1), p), \tag{5.10c}$$

$$-\delta u \leq u(k) - u(k-1) \leq \delta u, \tag{5.10d}$$

$$u_{\min} \leq u(k) \leq u_{\max}, \tag{5.10e}$$

$$x(k_0) = x_{k_0}. \tag{5.10f}$$

$$\tilde{\mathbf{x}}_{k|k} = [\mathbf{x}_{k|k-1}, f(x_{k-1+N_p|k-1}, \pi(x_{k-1+N_p|k-1}), d_{k+Np|k}, p)]^T \tag{5.10g}$$

$$\tilde{\mathbf{u}}_{k|k} = [\mathbf{u}_{k|k-1}, \pi(x_{k-1+N_p|k-1})]^T \tag{5.10h}$$

$$(1-\delta_T)\tilde{x}_{k+Np|k} \leq x_{k+Np|k} \leq (1+\delta_T)\tilde{x}_{k+Np|k} \tag{5.10i}$$

$$(1-\delta_T)\tilde{u}_{k+Np-1|k} \leq u_{k+Np-1|k} \leq (1+\delta_T)\tilde{u}_{k+Np-1|k} \tag{5.10j}$$

$$\tag{5.10k}$$

This implementation was investigated to determine whether RL might provide an adequate terminal region and cost function to improve the MPC's performance and produce a high-performing EMPC.

Implementation 6 Implementation 6 is an alternative method of incorporating the value function. This implementation involves solving implementation 3; however, with Equation 5.2 and Equation 5.3 separately. Once solved, the terminal state of the two solution trajectories is compared by evaluating them with the value function. The solution trajectory with the terminal state that yields the most favourable outcome (given from the value function) is selected as the final solution, and the first control input of this solution is taken. It essentially selects the best policy. Although only two policies are compared, this method may warrant further research whereby multiple generated policies could be compared. The policies generated could originate from multiple RL agents, each providing their initial estimations and with corresponding terminal constraints. Although each problem could be solved in parallel to speed up computational speed, it was implemented sequentially in this thesis.

5.1.2. Initial RL and MPC performance

A review of the RL, $\pi(\cdot)$, and MPC,$\kappa(\cdot)$, policies are evaluated for the nominal conditions.

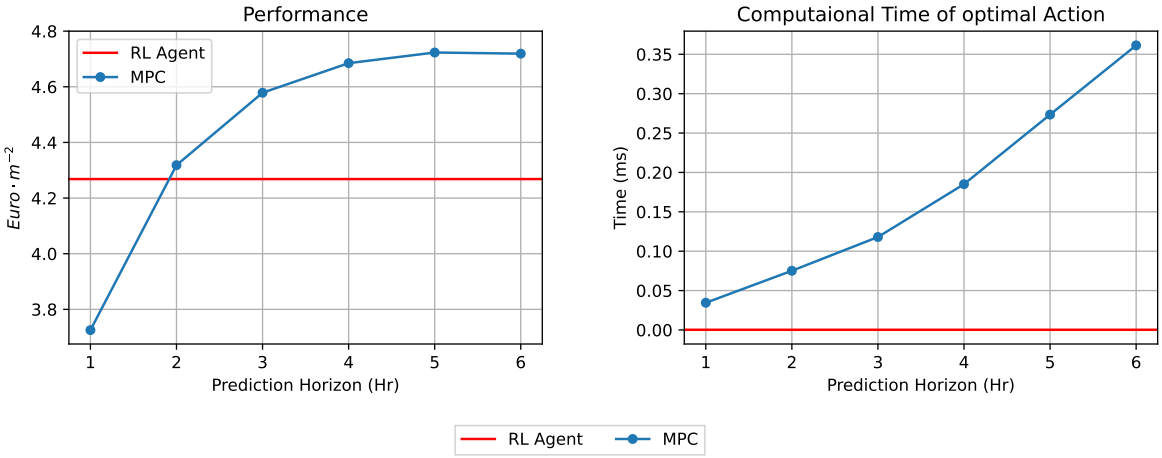

Figure 5.1: MPC vs RL in nominal conditions

Figure 5.1 demonstrates the performance of the MPC and RL agent in the nominal setting. It was previously thought that MPC would be myopic in this economic optimisation setting, however it achieves relatively high performance and outperforms RL. While the RL agent's performance is inferior to MPC for all prediction horizons (except 1 hour), it should not be assumed that this is the best RL policy available. RL can potentially generate an improved policy by conducting a more thorough hyper-parameter tuning. Additionally, with a discount factor ($\gamma = 0.95$) set at such a high value, RL should be capable of considering a longer time horizon beyond 6 hours. A more extensive hyper-parameter tuning may have to be done to achieve a policy that outperforms MPC. However, the RL agent is competitive and, as seen in Figure 5.1, can compute actions significantly faster. It is evident that the RL agent can be used to generate a reference trajectory for MPC with minimal increase in computational time. The performance of the resulting RL-MPC and its potential superiority will be analysed in the following sections and compared to the baseline performances, as depicted in Figure 5.1.

The RL-MPC was initially developed to aid MPC in its short sightedness by providing it information about the system past its prediction horizon. However MPC's performance can already be considered satisfactory. The deterministic RL-MPC framework is developed to determine whether information from RL can still help the MPC, specifically at shorter prediction horizons, in making better decisions. Therefore shorter prediction horizons may be used while keeping computational costs low and performance high. Once this has been determined, the RL-MPC framework will be tested in a stochastic environment to determine whether knowledge of the uncertainty present can be transferred to the RL-MPC from RL.

5.2. Results - RL-MPC 1

This implementation consists of passing in initial guesses for the MPC solver by unrolling the agent. Initial guess 1 refers to Equation 5.2 and Initial guess 2 refers to Equation 5.3.

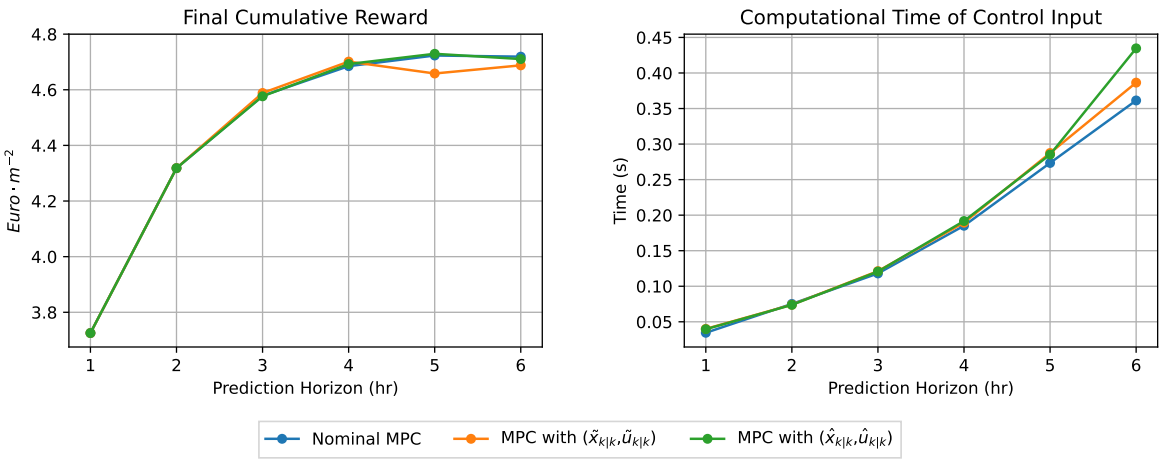

Figure 5.2: MPC vs MPC with initial guesses

Figure 5.2 presents the results of passing in initial guesses by extending the horizon with the actor and unrolling the actor, respectively. Equation 5.3 seems to have very little impact on the final cumulative reward; however, it noticeably increases the computational time of the control input. This could be because the initial guesses are derived from a less than optimal policy. Moreover, initial guesses by extending the prediction horizon from Equation 5.2 also have minimal impact on the final cumulative reward for shorter prediction horizons and seem to be slightly beneficial for a 3 and 4-hour prediction horizon. However, performance degrades significantly for a prediction horizon of 5 and 6 hours compared to the nominal MPC. Additionally, computational time also increases. It would be expected that computational time decreases for both initial guesses. A reason for the sub-optimal performance may be due to using the initial guesses of the Lagrangian multipliers found from the previous solution. The actor's initial guesses may differ significantly from the previous solution, rendering the Lagrangian multipliers' guesses nonsensical. While the actor may offer initial guesses, they are insufficient for improving or assisting the MPC, and simply using the previous solution as an initial guess may be more beneficial. However, the importance of having good initial guesses becomes more significant as the prediction horizon is extended and the problem complexity increases. Therefore, if a superior policy to the nominal MPC were to provide initial guesses instead of a suboptimal policy at a longer prediction horizon, it could result in a more favourable outcome.

5.3. Results - RL-MPC 2

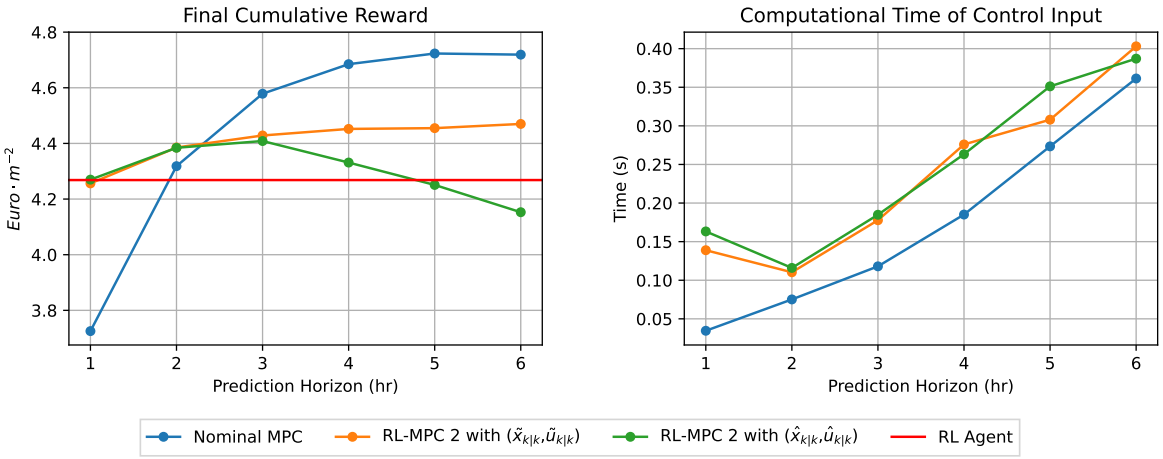

Figure 5.3: MPC with terminal constraints from agent

The approach described in implementation RL-MPC 2 was used t0 guarantee that the RL-MPC policy achieves a performance level equal to or better than the RL policy. The performance of the resulting policies is illustrated in Figure 5.3. It is evident that, for a terminal constraint and initial guesses as given by Equation 5.2, the resulting policy is at least as good as the RL's, even at lower prediction horizons where RL performs better than MPC. Nevertheless, the RL-MPC policy exhibits notably inferior performance to the MPC policy when the prediction horizon exceeds 3 hours. Beyond the 3-hour mark, the MPC policy consistently outperforms the RL and RL-MPC policies. While implementing RL-MPC may enhance the RL policy, it does not guarantee that the resulting policy will surpass the policy generated purely by MPC. If the RL policy is more competitive and surpasses the performance of the MPC, then implementing this RL-MPC implementation would be advantageous, as is the case for a prediction horizon of 1 and 2 hours. This marks a very important design choice. One can choose to either impose a terminal constraint to guarantee a certain level of performance or instead opt to extend the prediction horizon of the MPC until satisfactory performance is acquired. However, as mentioned, extending the horizon for the MPC controller does not guarantee better performance under economic optimisation. Therefore, a safer choice may be to implement the terminal point constraint provided by the RL agent.
The lack of performance guarantees for the terminal constraint and initial guesses, as indicated by equation Equation 5.3, is evident in the decrease in performance, which is even worse than RL, when longer prediction horizons are considered.
The computational time for the control input significantly increases. The terminal constraint may be excessively limiting the MPC due to the presence of a sub-optimal terminal constraint, particularly when dealing with longer prediction horizons.

5.4. Results - RL-MPC 3

This implementation aims to move away from the terminal constraint and allow the MPC more freedom by providing it with a terminal region constraint as outlined in Equation 5.8, with a chosen $\delta_T = 5\%$, allowing for a 10% deviation in the terminal state. As claimed in [57], this could prove to be more beneficial than the terminal constraint, provided that an appropriate cost function is supplied. For this implementation, the cost function supplied is effectively $V_f(\cdot) \equiv 0$.

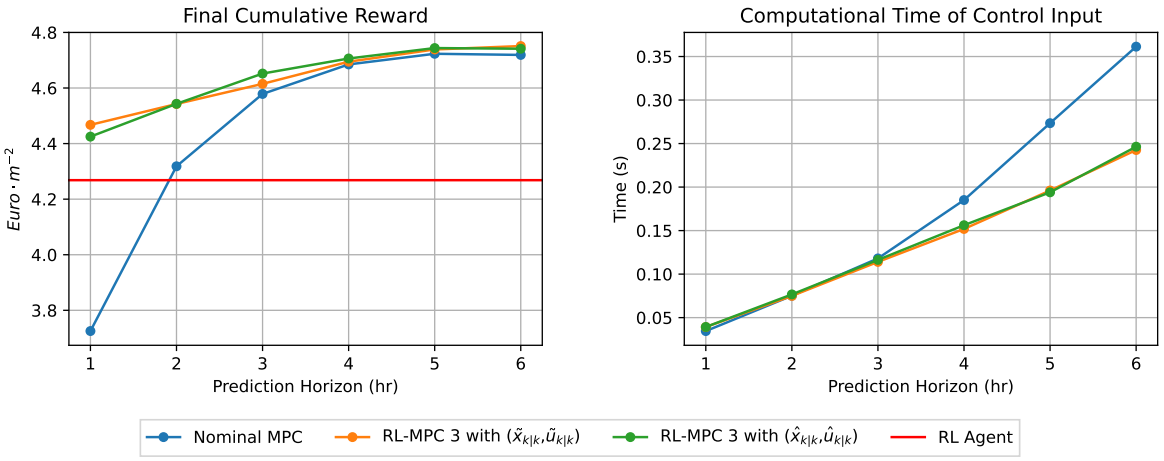

Figure 5.4: MPC with terminal region

The results from the implementation are depicted in Figure 5.4. Shortening the prediction time horizons results in a substantial increase in the overall cumulative reward compared to the standalone MPC and RL policies. Furthermore, the RL-MPC obtains a marginally superior final cumulative reward compared to the MPC even at longer prediction horizons with both terminal regions generated from each set of initial guesses. Lastly, it appears that this performance is increasing monotonically with an increase in prediction horizon. This could indicate that an appropriate terminal region has been found to guarantee performance and stability. However, longer prediction horizons would be required

to provide conclusive evidence.

Additionally, this terminal region also allowed for a lower computation time of the control inputs, with a more noticeable faster compute time at longer prediction horizons. This is likely because the terminal region is less limiting than the terminal constraint while guiding the MPC, resulting in reduced computational times. This is particularly noticeable at longer prediction horizons. It was noted that fewer instabilities in the solver were encountered at higher prediction horizons, which could be the reason for such a noticeable speedup. However, Figure 5.4 also indicates that the terminal region can be created using either a fixed reference trajectory ($(\tilde{\mathbf{x}}_{k|k}, \tilde{\mathbf{u}}_{k|k})$) or a changing one ($\hat{\mathbf{x}}_{k|k}, \hat{\mathbf{u}}_{k|k}$), to observe improvements in performance. Proposing that providing any guidance while still allowing freedom is advantageous for the EMPC.

It is noted that the resulting performance (increase in total cumulative reward and decrease in computational time) of the RL-MPC policy outperforms both standalone policies, even when the terminal region and initial guesses are supplied by a policy that performs substantially worse than the MPC controller. This underscores the necessity of giving certain future-oriented information to the EMPC to attain optimal economic advantage, performance, and stability.

5.5. Results - RL-MPC 4

This implementation investigates the effect of supplying the MPC with a cost function, specifically an approximate value function represented by a neural network, V_ϕ. Each value function, as listed in Table 3.7, is tested as a terminal cost function for the MPC.

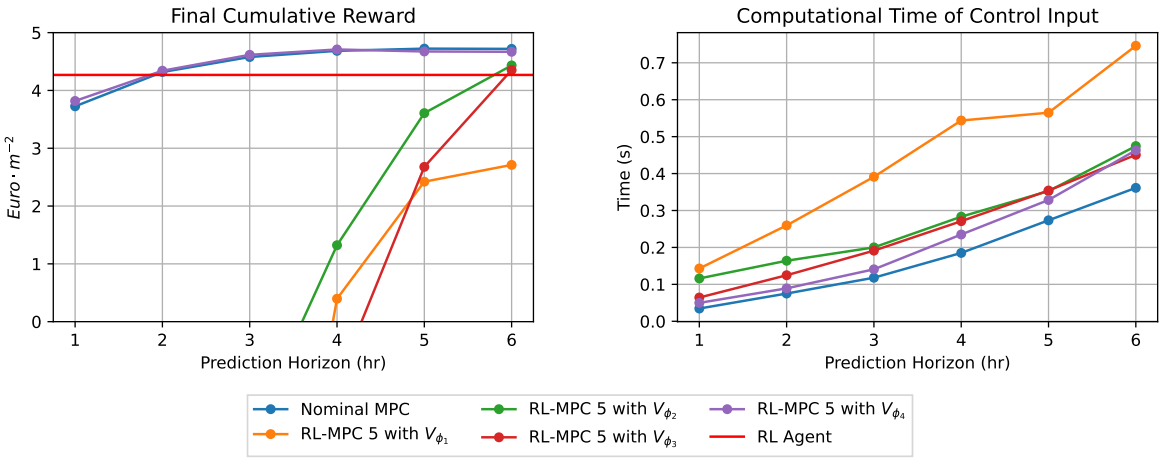

Figure 5.5: MPC with terminal cost function that optimises over all terminal states

Figure 5.5 is a naive implementation of merging RL and MPC. As can be seen, the outcomes are unsatisfactory regarding the overall cumulative reward and the computational time required for generating control inputs. Caution must be exercised when employing a neural network in an optimiser, as they are an extremely non-linear. As illustrated in Figure 5.5, the performance deteriorates further as the neural network becomes more complex. The neural networks exhibit highly non-convex behaviour that may cause the MPC optimiser to become easily trapped in local optima. The only value function that seems not to destabilize the optimiser is V_{ϕ_4} which is also the only value function that exhibits a smooth behaviour, see remark 1. The MPC's optimisation task is solely focused on maximising the drymass in this particular value function, with time being a constant parameter. However, the MPC must optimise across all possible model states and inputs for the other value functions. Although this leads to more accurate inference for the value of the specific environmental state, it also introduces additional dimensions and consequently a greater number of possibilities for becoming stuck in local optima that may not be useful.
It is natural to observe an increase in computational time in Figure 5.5. Once again, the highly non-convex

nature of the behaviour can disrupt the MPC optimiser and lead to unpredictable computational times. However, the simpler the neural network's architecture, the more reduced the computational time. Ultimately the results in Figure 5.5 highlight the significance of obtaining a high-quality value function.

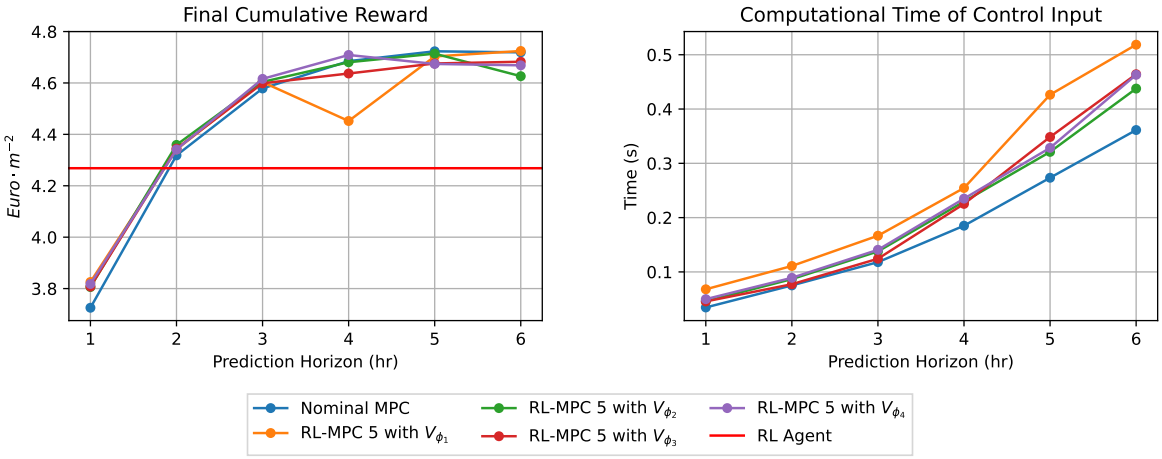

Figure 5.6: MPC with terminal cost function that optimises over time and terminal dry mass

With the exception of the dry mass, all decision variables were constrained at their initial guesses to reduce the non-linearity of the value functions. This action was taken due to the clear indication that the value of a state can be reasonably approximated by considering the time and the state of the drymass, as explained in section 3.6. Therefore, the optimisation process for V_{ϕ_1}, V_{ϕ_2}, and V_{ϕ_3}, which use a complete state observation (Equation 3.1) as inputs, was solely focused on optimising the dry mass input while keeping the other inputs fixed, similar to V_{ϕ_4}.

The performance of the resulting RL-MPC policy is demonstrated in Figure 5.6. Adding the value functions improves the final cumulative reward over MPC at shorter prediction horizons but degrades performance for longer prediction horizons (5 and 6 hours). Although the value functions have been simplified, increasing the prediction horizon still allows the MPC's optimisation to get trapped in local optima; this could be the reason for the observed performance degradation in Figure 5.6. Moreover, simplifying of the value function also resulted in more predictable times, with very little increase compared to the nominal MPC.

The performance of V_{ϕ_4} is superior to the others (at prediction horizons 1-4 hours) because it was trained exclusively on the drymass state and time, enabling it to make more accurate predictions of the value of a specific state using only these two inputs; therefore V_{ϕ_4} is used in the final implementation. These results suggest that a value function can increase performance over nominal MPC but not necessarily over the RL policy from which it was derived from. In addition, for the value function to be effective, it should have as few local optima as possible (i.e., it should have as low a degree of non-linearity as possible) while maintaining accuracy. Furthermore, it should be smooth and capable of generalising effectively across the state space to ensure reliable performance. Lastly, it is important to balance the performance gains with the increased computational time. Shown in Table 5.1, the increase in computational time far outweighs the marginal performance increase for shorter prediction horizons.

	1hr	2hr	3hr	4hr	5hr	6hr
Final Cumulative Reward Increase(%)	2.484	0.525	0.814	0.512	-1.041	-1.069
Computational Time Increase (%)	50.888	40.358	30.859	29.062	14.243	23.013

Table 5.1: RL-MPC $\tilde{V}_4$ performance comparison compared to MPC

5.6. Results - RL-MPC 5 and 6

Implementation of RL-MPC 5 aims to incorporate both a terminal region constraint and a terminal cost function. V_{ϕ_4} is used as a cost function with a terminal region generated by $(\tilde{\mathbf{x}}_{k|k}, \tilde{\mathbf{u}}_{k|k})$ initial guesses as in Equation 5.2, with a $\delta_T = 5\%$. Implementation RL-MPC 6 solves two problems, each with its initial guesses $((\tilde{\mathbf{x}}_{k|k}, \tilde{\mathbf{u}}_{k|k})$ and $(\hat{\mathbf{x}}_{k|k}, \hat{\mathbf{u}}_{k|k}))$ that are used to construct each of the problem's terminal regions. After solving the two problems, the better policy is determined by evaluating the terminal state in each solution using the value function.

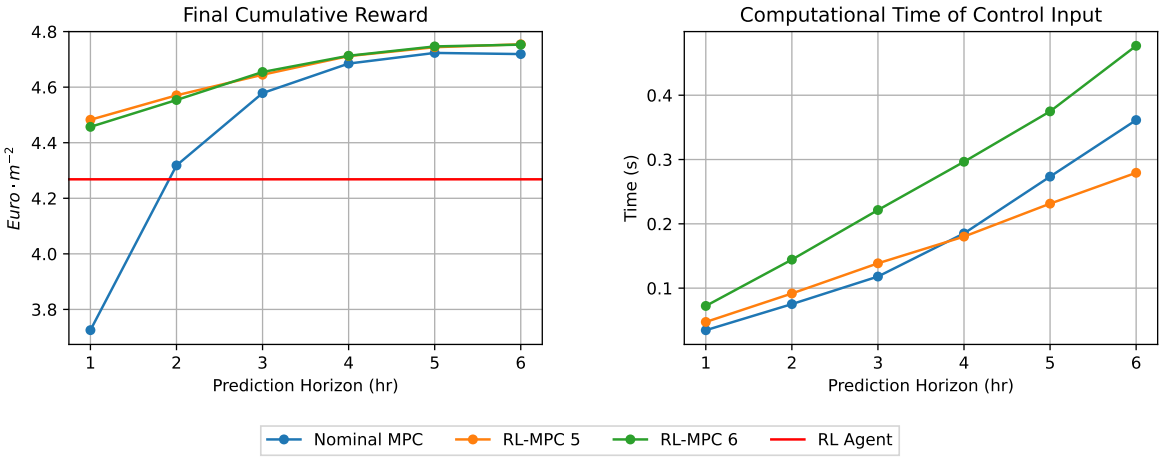

Figure 5.7: MPC vs RL-MPC with terminal region and cost function vs RL-MPC solving two separate problems and evaluating best policy with value function

As demonstrated in Figure 5.7, both implementations outperform the standalone MPC and RL policies across all MPC prediction horizons. The computational time of implementation RL-MPC 6 is notably higher because it solves two optimisation problems instead of just one. However, it is possible to solve the two problems in parallel, effectively reducing the computational time by half. Although the performance gains are substantial as compared to the standalone MPC, it is noted that the majority of this performance increase may be due to the terminal region constraint. Furthermore, despite including a neural network in the optimisation problem (Implementation RL-MPC 5), the computational time is reduced to a similar level as the standalone MPC when a terminal region constraint is present. Whether or not a cost function is necessary when a terminal region constraint has already improved performance so much is investigated in section 5.7.

5.7. Final Result and Conclusion

The three best implementations are compared, namely RL-MPC 3 (with $\tilde{x}_{k|k}$,$\tilde{u}_{k|k}$), RL-MPC 5 and 6.

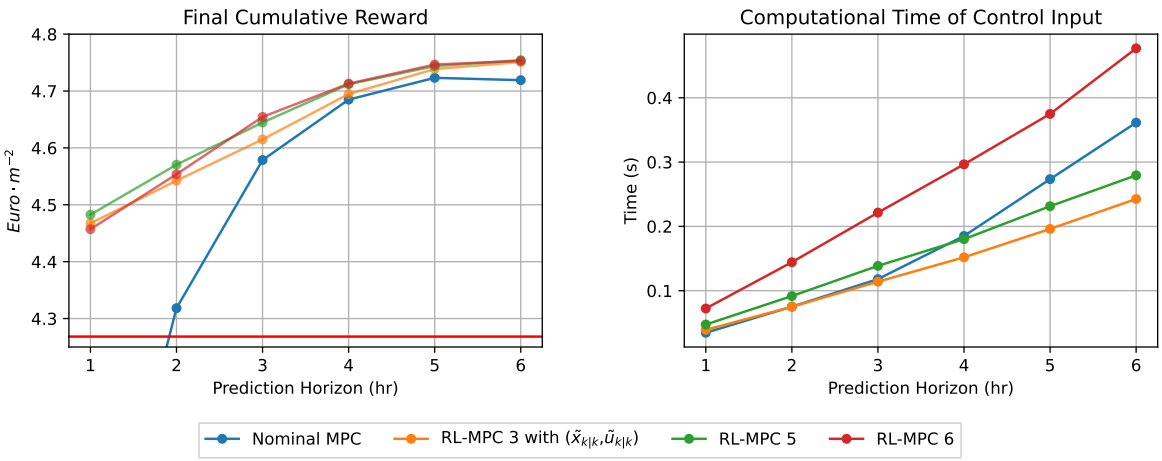

Figure 5.8: MPC final

The final results of the mentioned implementations are shown in 5.8. While including a terminal region is the primary factor in improving performance, incorporating the value function can additionally enhance performance, albeit with the drawback of increased computational time. This suggests that the value function and terminal region provided by RL satisfy assumption 3 to ensure the performance of an EMPC. However, formal proof would be needed to confirm this conclusively. Moreover, the increase in computational time is not that extreme when a neural network is implemented, especially when adding a terminal region to aid the MPC in finding solutions faster. While the computational time increase may appear insignificant, this system is relatively uncomplicated compared to others. If this approach is used in highly complex systems, this increase in computational time could lead to large enough delays in the control input to destabilise the system. Therefore, any method to reduce the computational time is highly beneficial. Moreover, it seems that there is very little benefit to adding a cost function when the advantages of adding a terminal constraint are already so beneficial.

Table 5.2: Performance Comparison: RL-MPC 3 and 5 vs. MPC and RL

	1 hr		2 hr		3 hr		4 hr		5 hr	
	Perf	Time	Perf	Time	Perf	Time	Perf	Time	Perf	Time
RL-MPC 3 (%)	19.91	10.91	5.191	2.38	0.79	-5.38	0.21	-8.47	0.328	-16.2
RL-MPC 5 (%)	20.32	24.26	5.84	18.26	1.44	44.37	0.57	3.68	0.435	2.71

Table 5.2 displays the performance increase of RL-MPC 3 and RL-MPC 5 against the nominal MPC and a comparison with RL at a 1-hour prediction horizon. Incorporating a terminal region and value function offers minimal advantages compared to only using a terminal region, as the marginal improvement in performance results in a significantly larger increase in computational time. This reiterates the importance of reducing the computational cost of the neural network in the optimisation problem. As seen in Figure 5.8, RL-MPC 6 offers promising results as an alternative method to using the value function and could yield even more benefits when more than two initial trajectories and terminal regions are provided by various agents and methods. The computational time of RL-MPC 6 can theoretically be halved by solving the problems in parallel, resulting in a compute cost similar to RL-MPC 3 and a similar performance to RL-MPC 5. Further research should be conducted on this implementation in the future.

While the combination of RL and MPC shows promising results, it is important to note that the simulations remain deterministic. These results primarily indicate which implementations are worth testing further in a stochastic environment to create a more accurate depiction of reality. Moreover, it may be tempting to use RL-MPC 2 (with terminal constraint) to guarantee performance as good as RL.

However, this may not be practically achievable in a stochastic environment. Due to the randomness in the state evolution of a stochastic environment, it is impractical to ensure that the system can always meet the terminal point constraint exactly as defined by the reference trajectory. Consequently, allowing a terminal region constrain relaxes this constraint and makes the problem feasible. Finally, a value function derived from a policy that inherently considers uncertainty can provide the MPC with valuable insights regarding the impact of uncertainty on its calculated control actions. Therefore, RL-MPC 3 and RL-MPC 5 are suitable choices for testing in a stochastic environment. RL-MPC 6 was not tested on a stochastic environment for computational reasons; however, further research should be conducted to investigate its efficacy.

6

Stochastic RL-MPC

This chapter outlines the performance gains of various RL-MPC implementations as selected from chapter 5 in a stochastic environment. The initial performances of the standalone RL agents and MPC in various levels of uncertainty will be investigated followed by comparisons with the RL-MPC implementations. This chapter seeks to demonstrate the effectiveness of RL-MPC controllers in a realistic setting, providing insight into the controller's potential performance in real-world applications.

6.1. Initial RL and MPC Performance

The uncertainty present in the environment, as mentioned in section 3.2, is characterised as parametric uncertainty. This uncertainty affects the parameters utilised in the computation of the system's outputs (output noise). Consequently, it is customary to use a state estimator for MPC to mitigate this noise. An approach that is both common and effective for MPC is to use the Moving Horizon Estimation (MHE) technique, which can be naturally integrated into the MPC framework. However, this thesis does not incorporate such an estimator in order to clearly observe the impact of combining RL with MPC in a stochastic environment. Moreover, RL was trained on a stochastic environment and the learned policy inherently takes the uncertainty into account. Finally, it is also important to note that for each level of uncertainty a new agent was trained (see section 3.5). Unless specified otherwise, it can be assumed that the corresponding agent is used for each uncertainty level, both in the RL and RL-MPC controllers. Performance metrics include the mean and variance of the final cumulative reward achieved (averaged across 30 simulations). The computational time required to calculate a control action is not considered, as the presence of uncertainty does not affect it.

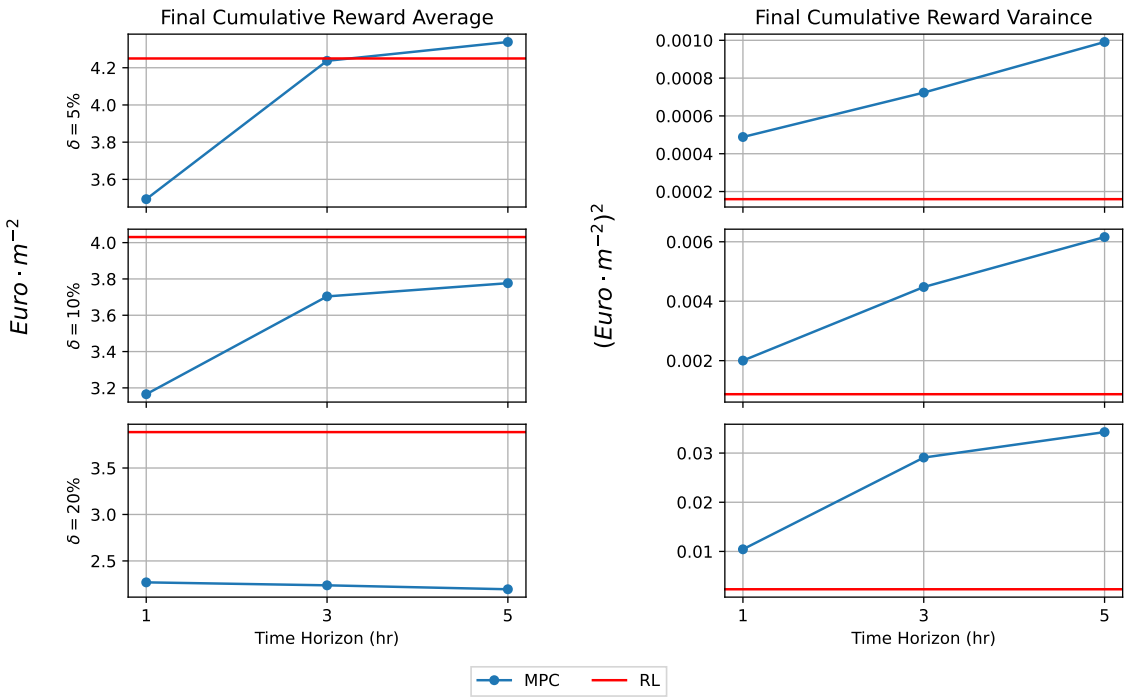

Figure 6.1: RL vs MPC in stochastic conditions

Figure 6.1 demonstrates the performance of RL and MPC for each prediction horizon and uncertainty level. Contrary to the nominal case shown in 5.1, RL consistently outperforms MPC for all prediction horizons and demonstrates superior performance particularly when faced with higher levels of uncertainty. The variance obtained through RL is significantly lower than that of MPC, indicating even greater performance superiority. While it is not surprising given that RL was trained on stochastic data, the findings from Figure 6.1 indicate that as uncertainty levels increase, the effect of increasing the prediction horizon on performance improvement diminishes and may even hinder the performance of MPC in extreme cases of uncertainty. Furthermore, it is worth noting that while both MPC and RL demonstrate comparable performance under conditions of low uncertainty, the final cumulative reward of MPC exhibits significantly higher variance compared to RL. This highlights the robustness of the RL policy. It is investigated whether this robustness to uncertainty is also applied to the RL-MPC controllers.
As stated in section 5.7, RL-MPC 3 (which includes a terminal region constraint) and RL-MPC 5 (which includes both a terminal region constraint and a value function as a terminal cost function) are evaluated at different levels of uncertainty. The performance of these implementations is then compared to that of the standalone RL and MPC controllers. The reference trajectory is generated by the RL agent and is based on deterministic predictions.

6.2. Results - VF and Terminal Region

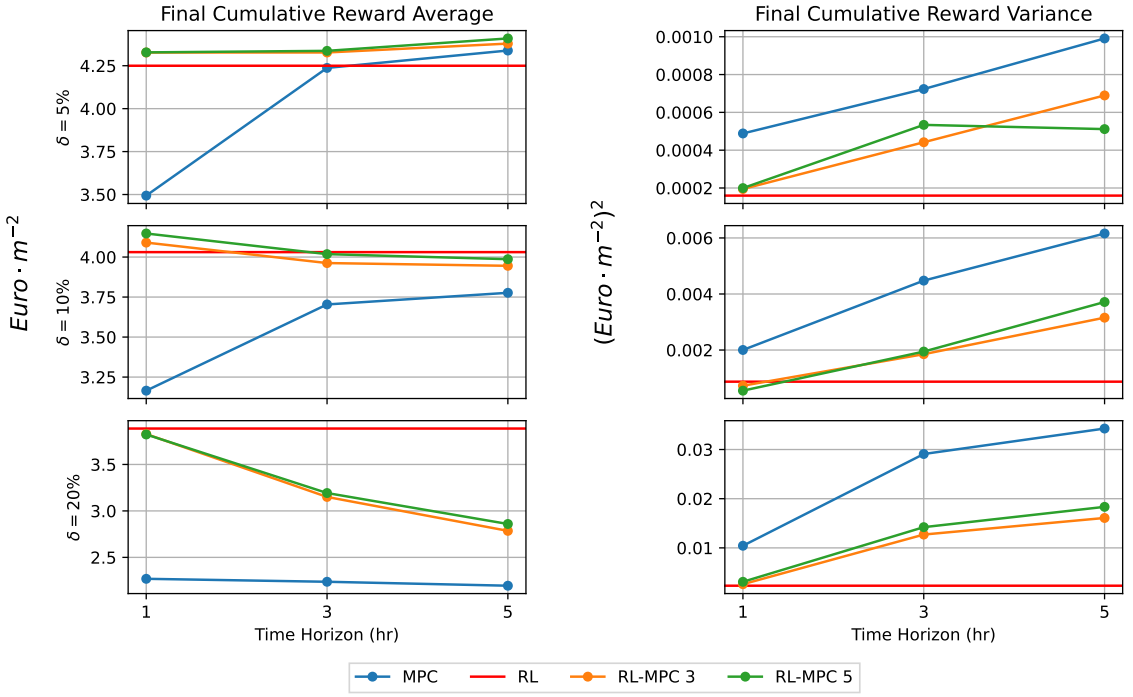

Figure 6.2: MPC vs stochastic RL vs stochastic RL-MPC 3 and RL-MPC 5

The performance comparison between RL-MPC 3 and RL-MPC 5 in a stochastic environment, as well as their comparison with standalone RL and MPC controllers, is shown in Figure 6.2. Similarly to the nominal case, it is evident that when uncertainty is low ($\delta = 5\%$), the RL-MPC controllers outperform both RL and MPC in terms of the mean final cumulative reward. However, it appears that the variance of the resulting RL-MPC controllers is notably greater than that of RL controllers, but notably lower than that of MPC controllers. It appears as if the variance is averaged between the two controllers. However, the behaviour of the RL-MPC controllers becomes intriguing when faced with higher levels of uncertainty. It seems that when there is a high degree of uncertainty, the performance of the RL-MPC controller is significantly impacted due to the poor performance of MPC in such highly stochastic environments. The same can be seen for the variance. While the performance of RL-MPC in a stochastic environment with a $\delta = 10\%$ is comparable to that of RL, it is evident that increasing the prediction horizon significantly affects performance, since it brings the RL-MPC's performance closer to that of MPC. For an uncertainty level of $\delta = 20\%$ the RL-MPC performance is drastically impacted by an increase in prediction horizon, however it still outperforms MPC. A general trend is that as the prediction horizon increases, the RL-MPC controller becomes more similar to the MPC controller. Conversely, with shorter prediction horizons, it becomes more similar to the RL policy. Due to the performance superiority of RL, for higher levels of uncertainty, it is desirable to keep the RL-MPC prediction horizon as short as possible in order to achieve performance, in terms of both the mean and variance of the final cumulative reward. Furthermore, it is clear that the inclusion of a value function as a terminal cost function, in conjunction with a terminal region, has improved performance when compared to using only a terminal region (RL-MPC 3 vs RL-MPC 5). This behaviour is similar to what is observed in the nominal case.

In practice, the exact uncertainty of a model is not known, thus a conservative estimate of uncertainty is assumed during the development of a controller. A parametric uncertainty of $\delta = 20\%$ may be considered too conservative, and if the model and environment is well known, it could have a detrimental effect on performance to design a controller on such a high level of uncertainty. Similarly for when developing a controller assuming a low parametric uncertainty. Therefore a controller is usually developed assuming a middle ground uncertainty level. Thus, it was determined to examine the effectiveness of the RL-MPC

controller when designed with an uncertainty level of $\delta = 10\%$ and evaluated under different uncertainty levels. This evaluation was compared to an RL agent that was also trained with an uncertainty level of $\delta = 10\%$ and the nominal MPC. Moreover, it can be seen in Figure 6.2, that careful consideration must be taken when selecting a predication horizon for both RL-MPC and MPC since a longer prediction horizon may lead to substantially worse performance due to the accumulation of uncertainty across the prediction horizon. Thus, a prediction horizon of 3 hours was chosen for both MPC and RL-MPC 5 (since this is the best performing RL-MPC controller), which was considered to be a suitable compromise for this study.

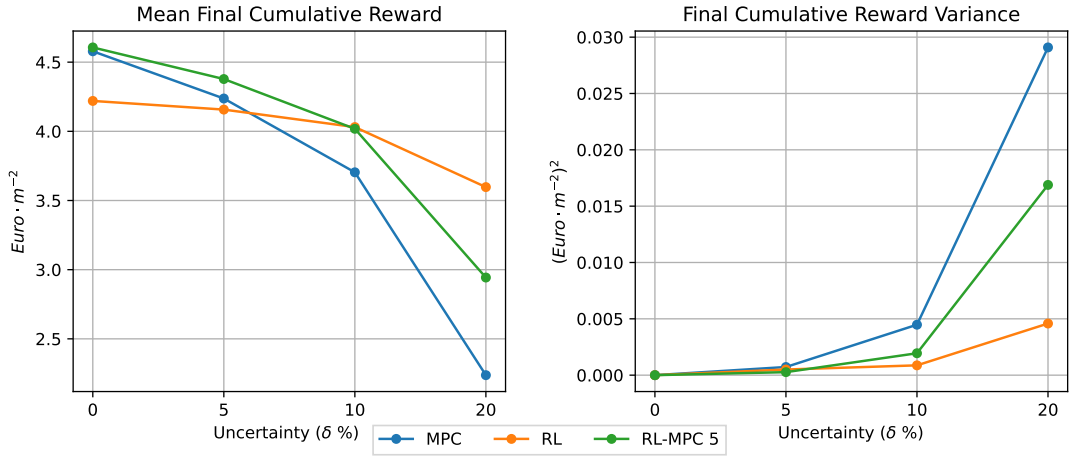

Figure 6.3: MPC vs RL vs RL-MPC 5 using an RL agent trained on $\delta = 10\%$ uncertainty and prediction horizon of 3 hours for both MPC and RL-MPC

The performance of the RL-MPC controller under various levels of parametric uncertainty is shown in Figure 6.3. Both the the RL-MPC controller and the RL agent are designed on an RL agent trained on a $\delta = 10\%$. The superiority of the RL-MPC controller over both the MPC and RL controllers under normal conditions is once again demonstrated in Figure 6.3 in terms of mean final cumulative reward. This trend persists until the uncertainty reaches a threshold where the MPC framework is unable to handle it, resulting in a significant decline in performance, which in turn negatively impacts the RL-MPC's performance. Nevertheless, the RL-MPC consistently outperforms the MPC in terms of mean final cumulative reward, and the decrease in performance is not as severe when compared to MPC. The RL-MPC controller also outperforms the MPC controller in terms of variance in the final cumulative reward. However, the RL agent continues to demonstrate superior performance in terms of variance under conditions of increased uncertainty. Nevertheless, is clear that the ability of RL to deal with uncertainty has been partially transferred to the RL-MPC controller. It is mentioned again that no state estimator is employed in the MPC's framework. By implementing this approach, the performance of the MPC is likely to significantly enhance under conditions of high uncertainty. As a result, the performance of the RL-MPC is also expected to improve, and potentially outperforming RL even at high levels of uncertainty. Moreover, if a stochastic MPC was used, such as in [34], performance of both MPC and RL-MPC controllers could be even greater.

6.3. Conclusion

This chapter has demonstrated the performance gains of the RL-MPC controller in a stochastic environment, which builds upon the deterministic evaluations performed in chapter 5. These results highlights the effectiveness of the RL-MPC controller as well as its robustness in varying levels of uncertainties.

The initial experiments conducted a comparison between standalone RL agents (trained on the stochastic data) and an MPC controller. The results consistently showed that RL outperformed MPC in terms of performance across all prediction horizons and uncertainty levels, except for cases with very low uncertainty and a long prediction horizon. Furthermore, RL's capacity to manage uncertainty was further demonstrated by the significantly reduced variance in the final cumulative reward, in comparison

to MPC. This level of robustness is essential for practical applications in which uncertainty is a common problem.
Further analysis evaluated RL-MPC controllers, specifically RL-MPC 3 and RL-MPC 5, under different uncertainty levels. The findings indicate that RL-MPC controllers consistently outperform the MPC controller and the RL controller when uncertainty is low. However, as uncertainty increases, the performance of RL-MPC approaches that of MPC, especially with longer prediction horizons, resulting in inferior performance as compared to RL. This trend suggests that shorter prediction horizons are preferable for maintaining the superior performance of RL policies within the RL-MPC framework. Moreover, incorporating a value function as a terminal cost function, as in RL-MPC 5, consistently enhances performance over using only a terminal region. Therefore the addition of the value function as a cost function sees benefit in both a deterministic and stochastic setting.

In real-world scenarios, exact uncertainty levels are often unknown, necessitating conservative estimates during controller development. The performed study with a conservative uncertainty level of $\delta = 10\%$ shows that the RL-MPC controller, with a prediction horizon of 3hrs, can effectively manage varying uncertainty levels, outperforming both RL and MPC at low uncertainty levels with a more gradual drop in performance at higher uncertainty levels as compared to MPC, indicating that the RL-MPC controller has a higher level of robustness. The MPC framework may be adapted in order to accommodate for uncertainty, specifically a state estimator for output uncertainty and a sample-based approach for model parametric uncertainty as developed in [34]. By implementing this approach, the performance and robustness of the MPC can be expected to improve in a stochastic environment. Furthermore, if these methodologies is also applied to the RL-MPC framework, even greater performance and robustness improvements can be anticipated.

In conclusion, this chapter has demonstrated that the RL-MPC controller is a highly efficient controller, even in a stochastic environment. While RL still performs better than the RL-MPC controller under extreme levels of uncertainty, significant performance enhancements can be achieved at low uncertainty levels.

7

Computational Speed Up of RL-MPC

The purpose of this chapter is to provide an overview of the application of making the best performing RL-MPC (RL-MPC 5) algorithm more computationally efficient. While the controller has demonstrated a marginal increase in computational time as compared to nominal MPC at lower prediction horizons, incorporating a neural network as a cost function in larger and more intricate systems may introduce excessive delays and hinder performance. 2 Experiments were conducted in order to achieve speedup and the resulting performance and computational time were investigated. Given that the longer computation time is caused by the inclusion of the value function in the formulation of the MPC algorithm, it is logical to focus on optimising this aspect to improve speed. Three approaches were devised to accelerate the algorithm, with the initial one involving the training of a less intricate surrogate value function instead of employing the full complex neural network for the terminal cost function. The aforementioned procedure has previously been performed, and the resulting outcomes were documented in chapter 5. V_{ϕ_4} can be regarded as a surrogate value function for V_{ϕ_1}. The surrogate value function exhibited greater stability and computational efficiency compared to the full order model. Consequently, it was employed in all subsequent experiments due to its stability. The following experiments are designed to accelerate the algorithm by implementing further simplifications to the neural network. It is important to note that the following experiments were performed on the deterministic environment with zero parametric uncertainty.

7.1. Reducing Neurons and Hidden Layers

A simple, and yet an effective approach was to reduce the size of the neural network representing the value function. The policy on which the value function was trained on remained fixed. All trained neural networks were trained with the same hyper parameters as outlined in section 3.6. An investigation was conducted on multiple network architects. The initial architecture, on which previous results have been established on and will be considered the baseline performance, comprised of a deep neural network with 2 hidden layers, each containing 128 neurons. To create the smaller neural networks, it was decided to have 3 deep and 3 shallow neural networks in total. For both the deep and shallow networks, there are 128, 64 and 32 neurons in each hidden layer respectively. By doing this, it effectively generates neural networks that become simpler as the number of neurons and hidden layers decreases. Thus is makes it possible to examine the impact of the complexity of the neural network on the RL-MPC's performance and computational time. It is important to note that for the deep neural networks, both hidden layers use the same number of neurons.

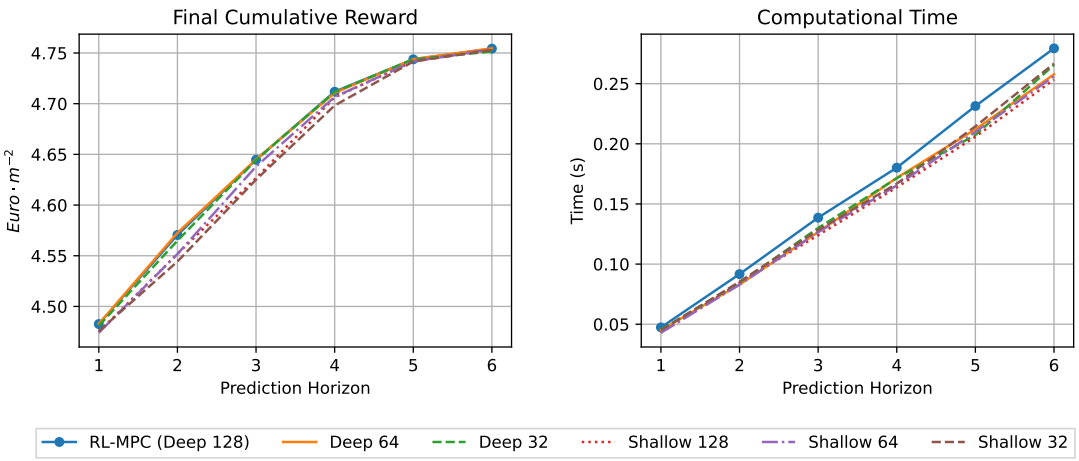

Figure 7.1: Fast RL-MPC with Reduced Neurons

The performance gains, in terms of final cumulative reward, and the computational time of the RL-MPC controller with different neural network architectures vs prediction horizon are shown in Figure Figure 7.1. It is evident that the simpler models reduce computational time, but also at the expense of a decrease in performance. To be more precise, it appears that all simpler neural networks have comparable computational times. However, shallow neural networks have a much more pronounced negative effect on performance compared to deep neural networks, which have a minimal impact on performance.

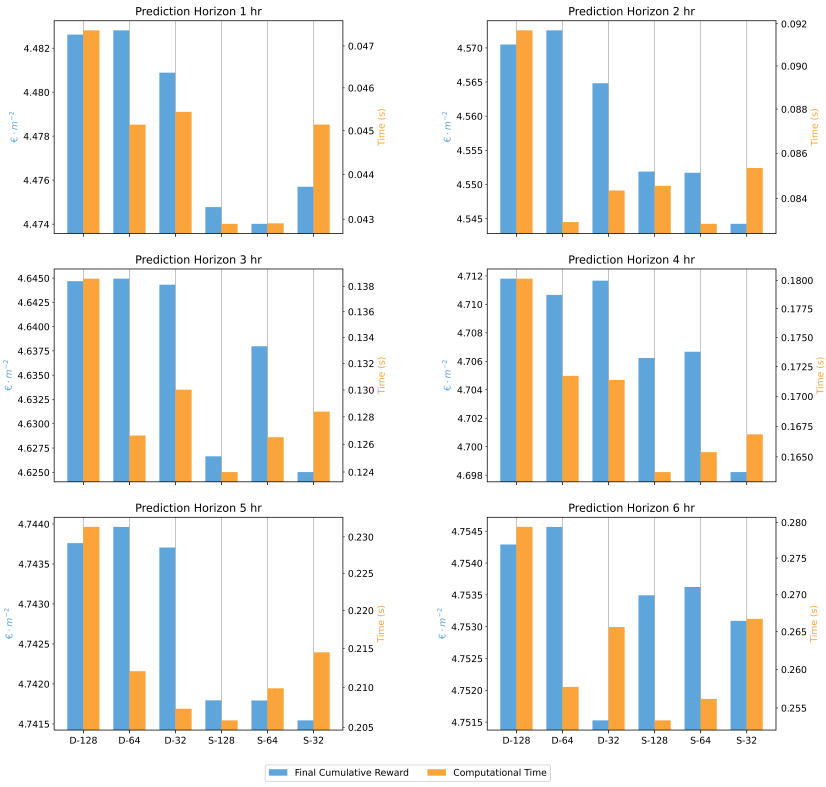

Figure 7.2: RL-MPC with reduced neural network complexity for its terminal cost function, where D-128 would stand for "Deep neural network of 128 Neurons per hidden layer".

Figure 7.2 displays a more in depth analysis into the performance gains and computational times of

each of the tested neural networks vs prediction horizon. From Figure 7.2, There is no observable correlation between the complexity of the neural network and the improvements in performance and computational time. Contrary to expectations, decreasing complexity does not necessarily result in a decrease in performance time. Although it does suggest the a simpler network can most definitely result in lower computational times albeit at the cost of performance. Furthermore, Figure 7.2 does suggest that the shallow networks offer less benefit, due to their substantial decrease in performance as compared to the simpler deep neural networks. The most effective neural network architecture appears to be a deep neural network with 64 neurons per hidden layer. This architecture achieves performance that is very similar to the original deep neural network with 128 neurons per layer, but with a noticeable reduction in computational time for all prediction horizons. This can both be seen in Figure 7.1 and Figure 7.2. Thus, these findings indicate that while reducing network complexity can speed up the algorithm with minimal to no performance degradation, thorough testing of the neural network architecture is necessary to achieve this.

7.2. Taylor Approximation

Perhaps a more intuitive approach would be to use a taylor expansion around a point to locally approximate the neural network in order to achieve speedup. The taylor expansion provides a first order (linear approximation) or a second order (quadratic approximation) of the neural network's outputs with respect to its inputs. While the calculation of the Jacobian of a neural network, which is required for the first-order Taylor approximation, is generally considered to be straightforward, determining the Hessian, used in the second order taylor approximation, of a neural network is considerable more difficult and computationally intensive. Nevertheless, employing a second order Taylor approximation would result in more precise approximations around the chosen point, potentially leading to better value function approximations and therefore increase in performance. Fortunately, since the structure of the neural network does not change over the course of the control period, both the Jacobian and Hessian only needs to be calculated once.

The taylor expansion of the neural network was accomplished using Casadi and L4Casadi. The first and second order Taylor expansions were both conducted around the terminal point, which is determined by the initial guess ,$(\tilde{\mathbf{x}}_{k|k}, \tilde{\mathbf{u}}_{k|k})$ for every time step. The performance and computational time of these approximations were then compared to that of the original RL-MPC 5 implementation.

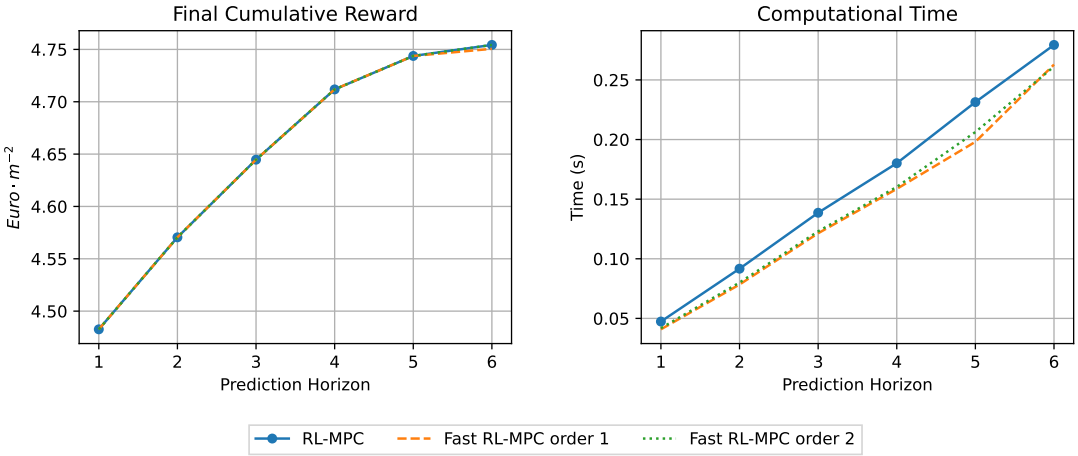

Figure 7.3: Fast RL-MPC with Taylor Expansion

The impact of a linear and quadratic approximation of the neural network around the terminal guess is illustrated in Figure 7.3. Locally approximating the neural network has evident advantages, as both first and second order approximations result in a substantial reduction in computational time without any noticeable decline in performance. It appears that a second order approximation is unnecessary and a first order approximation may be preferable due to its lower computational time without sacrificing

performance.

7.3. Combined

The following study investigated the combined effects of the two speedup techniques on the RL-MPC algorithm. The experiment includes combining the best results from the previous two experiments.

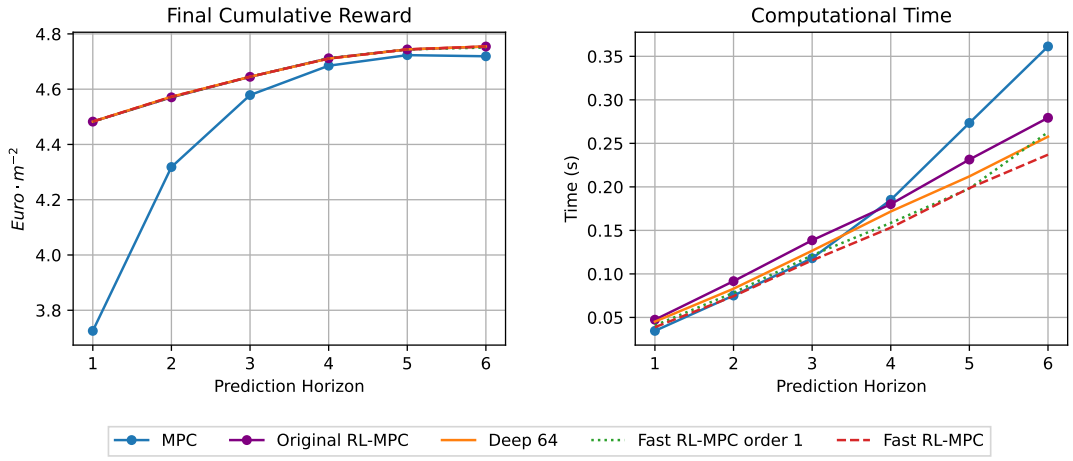

Figure 7.4: Fast RL-MPC

The results of utilising a deep neural network with 64 neurons per hidden layer, along with a first order Taylor approximation around the terminal guess, are presented in Figure 7.4. These findings are compared to the individual speedup techniques used, the original RL-MPC and the nominal MPC. Similar to previous results, Figure 7.1 and Figure 7.3, the combined techniques have zero or little impact on the performance. Although using a first-order Taylor expansion reduces computational time more compared to using a smaller deep neural network, the combination of both methods further decreases computational time. The final reduction is significant enough that the computational cost becomes equivalent to or less than that of MPC, even at shorter prediction horizons of 1, 2, and 3 hours. These results are significant because they demonstrate the successful implementation of methods that achieve speedup without compromising performance. These findings could have even greater implications for more complex (greenhouse) systems, where achieving real-time control or minimizing delay is crucial for system performance.

7.4. Discussion and Conclusion

Based on the findings of this chapter, it is evident that speedup of the RL-MPC can be achieved. This speedup can be achieved by focusing on strategic adjustments to the neural network used in the terminal cost function. The aim of the experiments detailed in this chapter was to reduce computational time with little to no performance impact.

Initially, efforts focused on making the neural network smaller by reducing neurons and hidden layers. This approach showed that changing the number of neurons and hidden layers has a noticeable impact on both performance and computational time. However, there was no discernible relationship between simpler models and faster computational time or performance improvements. Nevertheless, the results indicated that shallow neural networks generally have a higher trade-off between performance improvements and computational time compared to deep neural networks. It was found that a deep neural network with 64 neurons per hidden layer reduced computational time with no performance losses as compared to the original deep neural network with 128 neurons per hidden layer. Thus it is necessary to do a thorough search of the design space to achieve the desirable reduction in computational time when designing the RL-MPC algorithm for other systems.

Moreover, applying Taylor approximations, both linear and quadratic, around the terminal guess provided further insights to reducing computational time. For both linear and quadratic approximations, the speedup in computational time was noticeable while having no impact on the performance,

particularly the first order approximation. In addition, the Jacobian of a neural network is substantially easier to compute than the Hessian. Consequently, opting for a linear approximation may be a more attractive option.
Combining these two approaches yielded the most promising results. Using a smaller deep neural network of 64 neurons per hidden layer with a first-order Taylor approximation achieved computational speeds comparable to or faster than the nominal MPC for every prediction horizon tested. Not only was the amount of time required for computation decreased, but there was also no noticeable decline in performance.
In conclusion, these findings underscore the feasibility of enhancing RL-MPC's computational efficiency through thoughtful adjustments in neural network complexity and approximation techniques. Such optimizations are crucial for scaling RL-MPC applications to more intricate systems, potentially facilitating real-time control and minimizing delays, consequently enhancing overall system performance. Future research may explore further refinements to the structure of the cost function, potentially using alternative, more intuitive non-linear function approximators instead of neural networks.

8

Discussion and Conclusion

In this thesis, an RL-MPC algorithm was proposed in order to assess the performance gains as compared to the standalone RL and MPC controllers on a greenhouse environment. Two similar performing policies were created with RL and MPC respectively both for a deterministic and stochastic environment. The RL-MPC algorithm was developed in the deterministic environment by testing various combinations of the two controllers. The resulting RL-MPC was tested on a deterministic and stochastic setting. Lastly, modifications were investigated that could reduce the computational demand of the RL-MPC controller. This chapter assess the research questions asked in the beginning of this thesis and draws final conclusions of this RL-MPC controllers, with recommendations for future research .

8.1. Conclusion

This thesis aimed to answer two research questions, namely:

- *How does the economic performance of the RL-MPC algorithm compare to the standalone RL and MPC algorithms in a deterministic environment?*
  The results presented in this thesis clearly demonstrate that the RL-MPC algorithm greatly enhances economic performance in comparison to both the standalone RL and MPC algorithms. The MPC policy outperformed the policy generated from RL in this particular setting, and its superiority was further enhanced with an increase in the prediction horizon.The RL-MPC algorithm demonstrated superior economic performance across all prediction horizons of the MPC, with particularly notable results at shorter prediction horizons.
- *How does the economic performance of the RL-MPC algorithm compare to the standalone RL and MPC algorithms in a stochastic environment?*
  Evidence demonstrated that RL exhibits superior ability to manage uncertainty when trained using the corresponding stochastic data. The MPC was not adjusted to address this uncertainty, resulting in the RL outperforming the MPC, particularly under conditions of high uncertainty.The study demonstrated that the developed RL-MPC algorithm can achieve superior performance compared to both RL and MPC controllers under conditions of low uncertainty, similar to the deterministic scenario. Nevertheless, when faced with high levels of uncertainty, the performance of RL-MPC is hindered by the subpar performance of MPC in the stochastic setting. While it continues to surpass MPC in terms of performance, it faces challenges in achieving the same level of performance as RL in situations with high levels of uncertainty. Additionally, increasing the prediction horizon for the RL-MPC scheme may have negative consequences. This is because a longer prediction horizon causes it to behave more similarly to the MPC, which is undesirable in situations with high uncertainty.
- *What modifications and/or approximations can be employed to reduce the computational time of the RL-MPC algorithm?*
  Additional modifications can be made to reduce the computational burden of the RL-MPC algorithm. These modifications include training a simpler neural network with less neurons and

hidden layers, as well as using a 1st and 2nd order Taylor approximation of the neural network around the terminal guess. It was shown that the computational demand and performance is dependent on the size and complexity of the neural network, however no direct relationship was noticed between reduced computational burden and complexity of the neural network. However it was found that a simpler neural network did result in decreased computational time with no impact on performance. Moreover, it was shown that using both a 1st and 2nd order Taylor approximation noticeably decreased computational burden while resulting in very little to no performance losses. The combination of both techniques yielded a RL-MPC controller that exhibited computational times similar to those of MPC for short prediction horizons, and faster computational times for longer prediction horizons. Furthermore, the overall computational times of the RL-MPC new controller were faster than those of the original RL-MPC, while preserving its original performance.

The superiority of the RL-MPC controller over its standalone counterparts is evident, but the most significant enhancement in performance is achieved by incorporating a terminal constraint region. While incorporating a suitable value function does enhance performance, it also substantially increases the computational demand. The terminal region constraint enhances performance and decreases the computational load. Even so, it should be acknowledged that the greenhouse model used in this thesis is relatively uncomplicated. It is probable that including this value function may produce greater benefits in larger and more complex systems.

The computational burden of the RL-MPC algorithm can be similarly argued. Despite the RL-MPC algorithm demonstrating relatively fast computational speeds, even outperforming MPC at long prediction horizons, one might question the need to introduce modifications to further enhance its speed. The RL-MPC algorithm provides real-time control for a relatively simple non-linear system. However, when dealing with much larger systems, achieving real-time control may become difficult and essential for performance. Moreover, the modifications made are simple to incorporate and have no impact on the performance, thus there is no reason to not implement them.

It was noted that in this thesis, training an RL policy was significantly more difficult and time consuming than generating a policy through MPC, and thus in a real world application an MPC would likely be used. However results show even an okay RL policy can be integrated integrated into MPC to produce an RL-MPC algorithm that surpasses MPC's performance for all prediction horizons, showcasing the importance of integrating the two methods.

8.2. Recommendations & Future Work

A further study into the hyperparameters of the RL agent may be necessary to generate an RL policy capable of outperforming the MPC, even in a deterministic environment. Since the value function is learned separately to the training of the RL agent, it is possible to use other RL algorithms such as PPO, TRPO or other policy optimization algorithms to try increase the policies performance. Moreover, since the problem is episodic it would be desirable to find a set of hyperparameters that learns a very good policy for a discount factor of one, however since the value function can be learned separately, this may not be necessary and thus lower discount factors can be used in order to stabilize the RL training. In addition, it is important to note that a neural network is not the only non-linear function approximator available, and there may be other forms that are more suitable for effectively learning a value function in relation to the dynamics of the greenhouse. Additional types of non-linear function approximators include radial basis functions or gaussian processes. However, an in-depth investigation is necessary to determine which is most suitable.

It is also noted that soft constraints on the states are used for the MPC formulation. It is possible to implement hard constraints to ensure that no constraint violations (or very little) occur. This takes away one of MPC's strengths and a possible recommendation would be to compare the RL-MPC algorithm with MPC under hard constraints to determine whether the RL may still provide useful knowledge to the MPC. However it makes it difficult to compare to the RL's policy since the objective function changes. Moreover, a more accurate uncertainty model can be used in order to better predict the controllers behaviour in a realistic environment. Additionally, the studies performed in this thesis can be conducted on a more complex environment too such as in [60].

RL-MPC 6 is also a promising combination of RL and MPC. Guesses and terminal region constraints can be provided by multiple agents and/or other policies generated by other controllers and the best one selected by a common value function. Determining the policy on which this value function is trained on can also be investigated to ensure the best performance of the resulting RL-MPC controller.

In the case of the stochastic environment, RL outperforms MPC since it is given information about the uncertainty present in the environment, whereas MPC is not. This was done to determine whether RL is able to transfer its knowledge about this uncertainty into the MPC's framework through a terminal region constraint and a value function. However, estimation techniques such as the Moving Horizon Estimator (MHE) is commonly used in MPC to mitigate the noise on the output. Moreover stochastic MPC controllers such as in [34] targets parametric uncertainty as used in this thesis. One can expect the performance of MPC to significantly increase in a stochastic environment if these techniques are used. Therefore a study may be performed whereby the RL-MPC framework also incorporates these and the resulting performance gains examined.

Finally, in order to implement this in a real greenhouse, a theoretical foundation must be established to ensure that the combination of RL-MPC will yield performance guarantees. This thesis serves as an initial investigation into the performance gains expected and the resulting benefits of the RL-MPC.

References

[1] V. Blazhevska, *Growing at a slower pace, world population is expected to reach 9.7 billion in 2050 and could peak at nearly 11 billion around 2100: UN Report*, Jun. 2019. (visited on 12/01/2023).

[2] FAO, *The Future of Food and Agriculture: Trends and Challenges*. Rome: Food and Agriculture Organization of the United Nations, 2017, ISBN: 978-92-5-109551-5.

[3] M. van Dijk, T. Morley, M. L. Rau, and Y. Saghai, "A meta-analysis of projected global food demand and population at risk of hunger for the period 2010–2050," *Nature Food*, vol. 2, no. 7, pp. 494–501, Jul. 2021, ISSN: 2662-1355. DOI: 10.1038/s43016-021-00322-9. (visited on 12/01/2023).

[4] Nishat, *The "Green Deal" Greenhouse: A promise for sustainable food supply*, Sep. 2020. (visited on 12/01/2023).

[5] K. Winkler, R. Fuchs, M. Rounsevell, and M. Herold, "Global land use changes are four times greater than previously estimated," *Nature Communications*, vol. 12, no. 1, p. 2501, May 2021, ISSN: 2041-1723. DOI: 10.1038/s41467-021-22702-2. (visited on 12/01/2023).

[6] P. Muñoz, A. Antón, M. Nuñez, *et al.*, "Comparing the environmental impacts of greenhouse versus open-field tomato production in the Mediterranean region," *Acta Horticulturae*, vol. 801, pp. 1591–1596, Nov. 2008. DOI: 10.17660/ActaHortic.2008.801.197.

[7] C. F. Alvarez and G. Molnar, *What is behind soaring energy prices and what happens next? – Analysis*, https://www.iea.org/commentaries/what-is-behind-soaring-energy-prices-and-what-happens-next, Oct. 2021. (visited on 12/01/2023).

[8] A Breukers, O Hietbrink, and M. N. A. Ruijs, "The power of Dutch greenhouse vegetable horticulture : An analysis of the private sector and its institutional framework,"

[9] B. Morcego, W. Yin, S. Boersma, E. van Henten, V. Puig, and C. Sun, *Reinforcement Learning Versus Model Predictive Control on Greenhouse Climate Control*, Mar. 2023. arXiv: 2303.06110 [math]. (visited on 12/01/2023).

[10] DevOps, *Greenhouse Climate Control – How to Improve Plant Growth - DryGair*, Aug. 2021. (visited on 12/01/2023).

[11] P. Rusnak, *What Is the Current State of Labor in the Greenhouse Industry?* Nov. 2018. (visited on 12/01/2023).

[12] S. Zhang, Y. Guo, H. Zhao, Y. Wang, D. Chow, and Y. Fang, "Methodologies of control strategies for improving energy efficiency in agricultural greenhouses," *Journal of Cleaner Production*, vol. 274, p. 122 695, Nov. 2020, ISSN: 09596526. DOI: 10.1016/j.jclepro.2020.122695. (visited on 12/01/2023).

[13] D. Bertsekas, "Newton's method for reinforcement learning and model predictive control," *Results in Control and Optimization*, vol. 7, p. 100 121, Jun. 2022, ISSN: 2666-7207. DOI: 10.1016/j.rico.2022.100121. (visited on 12/01/2023).

[14] J. Arroyo, C. Manna, F. Spiessens, and L. Helsen, "Reinforced model predictive control (RL-MPC) for building energy management," *Applied Energy*, vol. 309, p. 118 346, Mar. 2022, ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2021.118346. (visited on 12/01/2023).

[15] L. Beckenbach, P. Osinenko, and S. Streif, "Addressing infinite-horizon optimization in MPC via Q-learning," *IFAC-PapersOnLine*, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018, vol. 51, no. 20, pp. 60–65, Jan. 2018, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.10.175. (visited on 12/13/2023).

[16] J. Lubars, H. Gupta, S. Chinchali, *et al.*, *Combining Reinforcement Learning with Model Predictive Control for On-Ramp Merging*, Sep. 2021. arXiv: 2011.08484 [cs]. (visited on 12/24/2023).

[17] S. Lubbers, "Autonomous greenhouse climate control with Q-learning using ENMPC as a function approximator," 2023. (visited on 12/01/2023).

[18] H. Sikchi, W. Zhou, and D. Held, *Learning Off-Policy with Online Planning*, Oct. 2021. arXiv: 2008.10066 [cs]. (visited on 01/10/2024).

[19] D. P. Bertsekas, "Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control,"

[20] M. Lin, Z. Sun, Y. Xia, and J. Zhang, "Reinforcement Learning-Based Model Predictive Control for Discrete-Time Systems," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2023, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2023.3273590. (visited on 12/01/2023).

[21] G. De la Torre-Gea, D. Soto-Zarazúa, I. Lopez-Cruz, I. Pacheco, and E. Rico-García, "Computational fluid dynamics in greenhouses: A review," *AFRICAN JOURNAL OF BIOTECHNOLOGY*, vol. 10, pp. 17 651–17 662, Dec. 2011. DOI: 10.5897/AJB10.2488.

[22] Y. Jansen, "Optimal Control of Lettuce Greenhouse Horticulture using Model-Free Reinforcement Learning," M.S. thesis, 2023. (visited on 12/01/2023).

[23] I. Lopez-Cruz, E. Fitz-Rodríguez, S. Raquel, A. Rojano-Aguilar, and M. Kacira, "Development and analysis of dynamical mathematical models of greenhouse climate: A review," *European Journal of Horticultural Science*, vol. 83, pp. 269–279, Nov. 2018. DOI: 10.17660/eJHS.2018/83.5.1.

[24] L. Gong, M. Yu, S. Jiang, V. Cutsuridis, and S. Pearson, "Deep Learning Based Prediction on Greenhouse Crop Yield Combined TCN and RNN," *Sensors*, vol. 21, no. 13, p. 4537, Jan. 2021, ISSN: 1424-8220. DOI: 10.3390/s21134537. (visited on 12/05/2023).

[25] B. Maestrini, G. Mimić, P. A. Van Oort, *et al.*, "Mixing process-based and data-driven approaches in yield prediction," *European Journal of Agronomy*, vol. 139, p. 126 569, Sep. 2022, ISSN: 11610301. DOI: 10.1016/j.eja.2022.126569. (visited on 01/15/2024).

[26] M. Ławryńczuk, "MPC Algorithms," in *Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach*, ser. Studies in Systems, Decision and Control, M. Ławryńczuk, Ed., Cham: Springer International Publishing, 2014, pp. 1–30, ISBN: 978-3-319-04229-9. DOI: 10.1007/978-3-319-04229-9_1. (visited on 12/19/2023).

[27] G. Dulac-Arnold, D. Mankowitz, and T. Hester, *Challenges of Real-World Reinforcement Learning*, Apr. 2019. arXiv: 1904.12901 [cs, stat]. (visited on 01/16/2024).

[28] W. J. Knibbe, L. Afman, S. Boersma, *et al.*, "Digital twins in the green life sciences," *NJAS: Impact in Agricultural and Life Sciences*, vol. 94, no. 1, pp. 249–279, Dec. 2022, ISSN: 2768-5241. DOI: 10.1080/27685241.2022.2150571. (visited on 01/16/2024).

[29] E. J. van Henten, *Greenhouse climate management: an optical control approach*. 1994, ISBN: 978-90-5485-321-3.

[30] E. J. Van Henten, "Validation of a dynamic lettuce growth model for greenhouse climate control," *Agricultural Systems*, vol. 45, no. 1, pp. 55–72, Jan. 1994, ISSN: 0308-521X. DOI: 10.1016/S0308-521X(94)90280-1. (visited on 01/16/2024).

[31] G. Van Straten and E. J. V. Henten, "Optimal Greenhouse Cultivation Control: Survey and Perspectives," *IFAC Proceedings Volumes*, vol. 43, no. 26, pp. 18–33, 2010, ISSN: 14746670. DOI: 10.3182/20101206-3-JP-3009.00004. (visited on 12/14/2023).

[32] M. Ghoumari, H.-J. Tantau, and J. Serrano, "Non-linear constrained MPC: Real-time implementation of greenhouse air temperature control," *Computers and Electronics in Agriculture - COMPUT ELECTRON AGRIC*, vol. 49, pp. 345–356, Dec. 2005. DOI: 10.1016/j.compag.2005.08.005.

[33] G van Straten, H Challa, and F Buwalda, "Towards user accepted optimal control of greenhouse climate," *Computers and Electronics in Agriculture*, vol. 26, no. 3, pp. 221–238, May 2000, ISSN: 0168-1699. DOI: 10.1016/S0168-1699(00)00077-6. (visited on 06/25/2024).

[34] S. Boersma, C. Sun, and S. van Mourik, "Robust sample-based model predictive control of a greenhouse system with parametric uncertainty," *IFAC-PapersOnLine*, 7th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2022, vol. 55, no. 32, pp. 177–182, Jan. 2022, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2022.11.135. (visited on 12/05/2023).

[35] W. van den Bemd, "Robust Deep Reinforcement Learning for Greenhouse Control and Crop Yield Optimization," Ph.D. dissertation. (visited on 12/01/2023).

[36] S. Hemming, F. de Zwart, A. Elings, A. Petropoulou, and I. Righini, "Cherry Tomato Production in Intelligent Greenhouses—Sensors and AI for Control of Climate, Irrigation, Crop Yield, and Quality," *Sensors*, vol. 20, no. 22, p. 6430, Jan. 2020, ISSN: 1424-8220. DOI: 10.3390/s20226430. (visited on 12/01/2023).

[37] Bonsai, *Why Reinforcement Learning Might Be the Best AI Technique for Complex Industrial Systems*, Nov. 2017. (visited on 12/06/2023).

[38] K. Daaboul, *Uncertainty and Prediction in Model-based Reinforcement Learning*, Oct. 2020. (visited on 12/06/2023).

[39] A. Ajagekar and F. You, "Deep Reinforcement Learning Based Automatic Control in Semi-Closed Greenhouse Systems," *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 406–411, 2022, ISSN: 24058963. DOI: 10.1016/j.ifacol.2022.07.477. (visited on 12/01/2023).

[40] L. Wang, X. He, and D. Luo, "Deep Reinforcement Learning for Greenhouse Climate Control," in *2020 IEEE International Conference on Knowledge Graph (ICKG)*, Nanjing, China: IEEE, Aug. 2020, pp. 474–480, ISBN: 978-1-72818-156-1. DOI: 10.1109/ICBK50248.2020.00073. (visited on 12/01/2023).

[41] A. Ajagekar, N. S. Mattson, and F. You, "Energy-efficient AI-based Control of Semi-closed Greenhouses Leveraging Robust Optimization in Deep Reinforcement Learning," *Advances in Applied Energy*, vol. 9, p. 100 119, Feb. 2023, ISSN: 2666-7924. DOI: 10.1016/j.adapen.2022.100119. (visited on 12/05/2023).

[42] Decardi-Nelson Benjamin and You Fengqi, "Improving Resource Use Efficiency in Plant Factories Using Deep Reinforcement Learning for Sustainable Food Production," *Chemical Engineering Transactions*, vol. 103, pp. 79–84, Oct. 2023. DOI: 10.3303/CET23103014. (visited on 12/05/2023).

[43] W. Zhang, X. Cao, Y. Yao, Z. An, X. Xiao, and D. Luo, *Robust Model-based Reinforcement Learning for Autonomous Greenhouse Control*, Oct. 2021. DOI: 10.48550/arXiv.2108.11645. arXiv: 2108.11645 [cs]. (visited on 12/01/2023).

[44] S. van Mourik, B. van't Ooster, and M. Vellekoop, *Plant Performance in Precision Horticulture: Optimal climate control under stochastic uncertainty*, Jul. 2023. DOI: 10.48550/arXiv.2303.14678. arXiv: 2303.14678 [cs, eess, math]. (visited on 12/01/2023).

[45] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning), Nachdruck. Cambridge, Massachusetts: The MIT Press, 2014, ISBN: 978-0-262-19398-6.

[46] R. Bellman, "Dynamic programming," *Science (New York, N.Y.)*, vol. 153, no. 3731, pp. 34–37, Jul. 1966, ISSN: 0036-8075. DOI: 10.1126/science.153.3731.34.

[47] A. Rao, "OPTIMAL POLICY FROM OPTIMAL VALUE FUNCTION,"

[48] H. Dave, *Understanding the Bellman Optimality Equation in Reinforcement Learning*, Feb. 2021. (visited on 12/11/2023).

[49] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing Atari with Deep Reinforcement Learning*, Dec. 2013. arXiv: 1312.5602 [cs]. (visited on 12/11/2023).

[50] L. Dai, Y. Xia, M. Fu, *et al.*, "Discrete-Time Model Predictive Control," in *Advances in Discrete Time Systems*, IntechOpen, Dec. 2012, ISBN: 978-953-51-0875-7. DOI: 10.5772/51122. (visited on 12/05/2023).

[51] J. K. Gruber, J. L. Guzmán, F. Rodríguez, C. Bordons, M. Berenguel, and J. A. Sánchez, "Nonlinear MPC based on a Volterra series model for greenhouse temperature control using natural ventilation," *Control Engineering Practice*, vol. 4, no. 19, pp. 354–366, 2011, ISSN: 0967-0661. DOI: 10.1016/j.conengprac.2010.12.004. (visited on 12/13/2023).

[52] A. Montoya, J. Guzmán, F. Rodriguez, and J. Sánchez-Molina, "A hybrid-controlled approach for maintaining nocturnal greenhouse temperature: Simulation study," *Computers and Electronics in Agriculture*, vol. 123, pp. 116–124, Apr. 2016. DOI: 10.1016/j.compag.2016.02.014.

[53] C. Bersani, A. Ouammi, R. Sacile, and E. Zero, "Model Predictive Control of Smart Greenhouses as the Path towards Near Zero Energy Consumption," *Energies*, vol. 13, no. 14, p. 3647, Jan. 2020, ISSN: 1996-1073. DOI: 10.3390/en13143647. (visited on 12/01/2023).

[54] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd edition. Madison, Wisconsin: Nob Hill Publishing, 2017, ISBN: 978-0-9759377-3-0.

[55] M. Ellis, H. Durand, and P. D. Christofides, "A tutorial review of economic model predictive control methods," *Journal of Process Control*, Economic Nonlinear Model Predictive Control, vol. 24, no. 8, pp. 1156–1178, Aug. 2014, ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2014.03.010. (visited on 12/05/2023).

[56] J. B. Rawlings, D. Angeli, and C. N. Bates, "Fundamentals of economic model predictive control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dec. 2012, pp. 3851–3861. DOI: 10.1109/CDC.2012.6425822. (visited on 12/01/2023).

[57] R. Amrit, J. B. Rawlings, and D. Angeli, "Economic optimization using model predictive control with a terminal cost," *Annual Reviews in Control*, vol. 2, no. 35, pp. 178–186, 2011, ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2011.10.011. (visited on 12/05/2023).

[58] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006, ISSN: 1436-4646. DOI: 10.1007/s10107-004-0559-y. (visited on 06/26/2024).

[59] M. J. Risbeck and J. B. Rawlings, "Economic Model Predictive Control for Time-Varying Cost and Peak Demand Charge Optimization," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 65, no. 7, 2020.

[60] "GreenLight – An open source model for greenhouses with supplemental lighting: Evaluation of heat requirements under LED and HPS lamps," *Biosystems Engineering*, vol. 194, pp. 61–81, Jun. 2020, ISSN: 1537-5110. DOI: 10.1016/j.biosystemseng.2020.03.010. (visited on 07/04/2024).

[61] S. Hemming, F. de Zwart, A. Elings, I. Righini, and A. Petropoulou, "Remote Control of Greenhouse Vegetable Production with Artificial Intelligence—Greenhouse Climate, Irrigation, and Crop Production," *Sensors*, vol. 19, no. 8, p. 1807, Jan. 2019, ISSN: 1424-8220. DOI: 10.3390/s19081807. (visited on 12/01/2023).

[62] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science (New York, N.Y.)*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, ISSN: 1095-9203. DOI: 10.1126/science.aar6404.

[63] M Jonker, "Model-based Reinforcement Learning: A Survey,"

[64] V. Mnih, A. P. Badia, M. Mirza, *et al.*, *Asynchronous Methods for Deep Reinforcement Learning*, Jun. 2016. DOI: 10.48550/arXiv.1602.01783. arXiv: 1602.01783 [cs]. (visited on 12/11/2023).

[65] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms,"

[66] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, Jul. 2019. DOI: 10.48550/arXiv.1509.02971. arXiv: 1509.02971 [cs, stat]. (visited on 12/11/2023).

[67] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust Region Policy Optimization*, Apr. 2017. DOI: 10.48550/arXiv.1502.05477. arXiv: 1502.05477 [cs]. (visited on 12/11/2023).

[68] C. Yoon, *Understanding Actor Critic Methods*, https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f, Jul. 2019. (visited on 12/13/2023).

[69] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, Aug. 2017. DOI: 10.48550/arXiv.1707.06347. arXiv: 1707.06347 [cs]. (visited on 12/11/2023).

[70] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Jul. 2018, pp. 1587–1596. (visited on 12/13/2023).

[71] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, Aug. 2018. DOI: 10.48550/arXiv.1801.01290. arXiv: 1801.01290 [cs, stat]. (visited on 12/11/2023).

A

RL & RL Training

A.1. Overview of RL Algorithms

Various RL algorithms exist to solve the MDP problem as discussed above (**??**). There are two main categories which exist in RL, model-based and model-free. Where model free reinforcement learning solely learns from the interactions between the agent and the environment, without any knowledge of the underlying dynamics. Both have been successfully applied in the greenhouse control problem with model-based RL being used in [43] and model-free in [22], [35], [39], [61]. Both offer their respective advantages and disadvantages. Most commonly, since model-based reinforcement learning encodes the dynamics of the environment, it offers efficient sampling and faster learning since it allows the agent to plan its action ahead of time, whereas model-free has shown to have better asymptotic performance in complex environments. An interesting model-based algorithm as used in Alpha-Zero [62], utilizes a known model, where the policy is planned over the known model and horizon but uses learning for the global solution.[63]. Alpha-Zero was considered the best chess engine upon its release and continues to be recognized as one of the top contenders [62]. However the ground-truth model of the environment is usually not available, and therefore only the simplified dynamics is used in model-based RL. However, model-free is able to learn the model only from experience.

Additionally, model-free algorithms may be classified into two further categories: Policy Gradient methods (Policy Optimization) and Q-learning. Actor-Critic algorithms are an amalgamation of the two. Figure A.1 displays an overview of the non-exhaustive taxonomy of RL algorithms. These RL algorithms (Figure A.1) are used to learn a various aspects (or a combination) of the MDP. These aspects encompass policies (both stochastic and deterministic), Q-functions, value functions, and/or environment models.

This thesis will explore a model-based method of RL.It is imperative to understand both Q-learning and Policy Optimization (and actor-critic methods), although the developed algorithm may be classified as model-based, model-free RL techniques are used to gain an estimation of the value function.

A.2. Selection of RL Algorithm

In order to select an RL algorithm, it must first be determined what is needed from the respective algorithm. Since the objective of the thesis is to combine RL with MPC with RL as the initial approximate solution to the MPC problem, a value function is required. This value function can either be a Q-function or a State value function. Moreover, the approximated value function needs to accommodate continuous states and actions. **??** outlines the environment, and due to the numerous state variables and actions available, discretizing either the action or state space (or both) would lead to a rapid increase in the dimensionality of the problem (curse of dimensionality). Hence a suitable approach would be the use of actor-critic or policy optimization methods.

All model-free algorithms presented in Figure A.1, except for (Vanilla) Policy Gradient, utilizes an approximated value function. Although A2C/A3C, PPO and TRPO are classified as policy optimization approaches, they are commonly used with critic-networks to reduce variance and increase learning

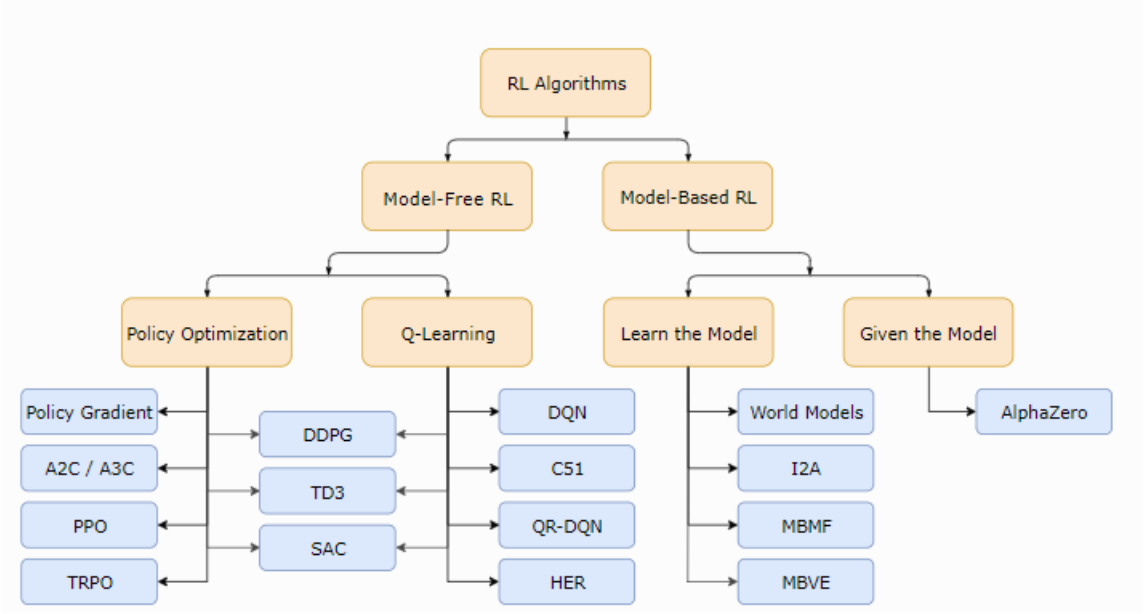

Figure A.1: Taxonomy of algorithms in modern RL

speed, as opposed to using monte-carlo sampling [45].

In recent developments, q-learning algorithms such as DQN (developed in 2013 [64]), originally designed for discrete actions [45], have been extended with Deterministic Policy Gradient approaches [65] to be capable of continuous action spaces. The resulting actor-critic algorithm, known as Deep Deterministic Policy Gradient (DDPG) [66], is an off-policy algorithm capable of both continuous state and action spaces and was developed in 2016.

Furthermore, a policy optimization method, TRPO, was released in 2015 [67] and improved upon the stability and learning speed of vanilla policy gradients by limiting the size of the policy update, although it is a challenging algorithm to implement. The Kullback Leibler (KL) Divergence is applied to ensure that the updated policy does not differ significantly from the previous policy.

In 2016, (Asynchronous) Advantage Actor-Critic A2C/A3C was published [64], whereby A3C is an asynchronous version of A2C. A2C/A3C is on-policy policy optimization algorithm, that uses a value function to optimize the policy gradients. A3C significantly reduces the time of learning by using multiple agents to gather experience to update the shared policy network. The gathered experience is more diverse which stabilizes training as compared to A2C. However, it was found empirically that A2C produces comparable performance to A3C while being more efficient [68].

Proximal Policy Optimization (PPO) was released to further improve on TRPO in 2017 [69]. Instead of using KL divergence to restrict the size of the policy update, first order methods are used to limit the update policy, allowing for easier implementation,faster learning and similar performance.

Finally, in 2018, two algorithms were released. Twin Delayed Deep Deterministic Policy Gradient (TD3) was introduced [70] and expands on the DDPG algorithm. TD3 uses additional Q-functions, delayed updates and policy smoothing to improve the performance and the stability of the DDPG algorithm. Moreover, Soft Actor-Critic (SAC) also extends the framework established by DDPG, but uses entropy regularization for agent exploration and removes the need for policy smoothing. Additional differences are in the way the agents actions are selected during learning, where TD3 uses the target policy, SAC uses the current policy. However, both algorithms are considered state-of-the-art algorithms and have similar performance, with SAC having a slightly faster convergence rate due to its entropy regularization.

Algorithms such as A2C/A3C [64], PPO [69] and TRPO [67], are on-policy methods, albeit state-of-the-art algorithms, have low sample efficiency and high computational cost. The greenhouse is a complex environment and the anticipated learning process for policies in such a complex setting is expected to be long, therefore opting for an off-policy method is deemed desirable. Both Twin-Delayed Deep

Deterministic Policy Gradient (TD3) [70] and Soft Actor-Critic (SAC) [71] are extensions of DDPG that result in a more stable Q-function and thus it was decided to use SAC.

A.3. Agent Training

A.4. Value Function Training

B

RL-MPC