RL-MPC for Autonomous Greenhouse Control

Combining RL and MPC to maximize 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 maximize 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, June 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

Background and motivation about the need for autonomous greenhouses, and advanced control schemes. How RL and MPC can be used and why they should be combined

1.1. Recent and Related Developments

Recent works and Related Developments, and the similarity between the works and what this thesis does and does not do

1.2. Problem Statement

The aim/objective of the thesis, and the questions that will be answered

1.3. Thesis Contribution

What will this thesis contribute to the research community?

1.4. Thesis Outline

The general outline of the thesis report

2

Background

Description of chapter, Use the present tense when introducing a chapter or section

2.1. Greenhouse Model

Short Introduction and background of how crop and greenhouse dynamics can be modelled. And a motivation on why the van hentron model was selected

2.1.1. Model Description

The description of the van henten model

2.1.2. Model State Equations

The model state equations explained and given

Also explain how uncertainty will be treated in the system

2.1.3. Optimization Goal

The optimization goal of the algorithms. What it is trying to optimize, i.e. the economic benefit

2.2. Reinforcement Learning

2.2.1. Why RL for Greenhouse Control?

why use RL for greenhouse control. and some recent and related works on it

2.2.2. The RL problem

A description of what RL aims to solve
as well as a description of RL algorithms and Q-learning, policy optimization and and actor critic and which RL algorithm was selected

2.2.3. SAC

A brief explanation of how SAC works and how it learns

2.3. MPC

2.3.1. Why MPC for Greenhouse Control?

Why should MPC be used for greenhouse control

2.3.2. The General MPC problem

What MPC aims to achieve and how

2.3.3. Tracking MPC vs EMPC

The differences between EMPC and tracking and the difficulty in determining a suitable cost function and/or terminal constraint for EMPC

2.4. RL and MPC in tandem

To drive home the similarities and differences between the two and why they should be merged

2.4.1. The Approach
2.4.2. The Optimality
2.4.3. Computational Effort
2.4.4. The Prediction Horizon
2.4.5. The combination

3

Reinforcement Learning Setup

This chapter provides an overview of the process of developing the reinforcement learning (RL) agent that will be used in the development phase of the RL-MPC algorithm. This chapter focuses on the description of the environment on which the RL agent is trained, as well as the performance of the agent in both deterministic and stochastic environments. 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 as described in section 2.1 and outlines important features to successfully train an RL agent. 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 that is used for the training period.

Observation Space The observation space of the agent must be carefully selected, in order 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 on 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 **refXX**. 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 in order to make better decisions. As shown in **some section**, since the control input is dependent 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 to be time-aware, and so the current time step is also given to the agent. Although not necessary, it enables the agent to learn a non-stationary policy. Considering that the growing period is 40 days **as in Section XXX**, the problem is episodic. By incorporating time awareness, the agent is able to leverage the current time in order to make more optimal decisions. As discussed in subsection 2.1.3 and later in Equation 3.1, the optimization goal includes maximizing the growth difference between time steps, and so knowledge of the previous dry mass state, the the growth experienced in the previous time step may be beneficial to learning an optimal policy. As a result, the 3 following environment state tuples $s(k)$ at time k are 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 3.1 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 potential infer the system dynamics more accurately. However Equation 3.1 and Equation 3.2 are simpler, with Equation 3.1 facilitating the learning of a value function and integration in the RL-MPC algorithm as discussed in **section XXX**.

Action Space The continuous action space, denoted as A, is defined as $\subseteq [-1, 1]^3$, where $a \in A$. In order to ensure that the current control input, $u(k)$ satisfies the constraints outlined in **show equation**, the agent's action, denoted as $a(k)$, is regarded as a modification to the control input. Consequently, the current control input can be determined as follows:

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

where $\delta u(k)^{max}, u_{min}, u_{max}$ are defined in **section here**

Initial Conditions Initial conditions were kept constant for every episode for both the stochastic and deterministic case and shown in Equation 3.4.The values in question were obtained from the sources as cited in **ref XXX and ref XXX**.

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

Reward Function The reward function is modelled after the optimization goal as defined in **section/equation** and represents the same optimization goal as defined for the Model predictive control 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 are operated within reasonable limits to ensure realistic conditions. As stated in **Section XXX**, state constraints cannot be directly imposed but can be indirectly incorporated through a penalty function within the reward function. It is common practice to impose a linear penalty function for state violations when learning a policy with RL for stability reasons **ref XXX**. As such, the resulting reward function becomes:

$$\begin{aligned} R(k) = \kappa_1 \cdot (y(k) - y(k-1)) - (\kappa_2 \cdot u_1(k) + \kappa_3 \cdot u_2(k) + \kappa_4 \cdot u_3(k)) \\ - (P_{c02} \cdot y_2(k) + P_T \cdot y_3(k) + P_H \cdot y_4(k)) \end{aligned} \tag{3.5}$$

where $\kappa_{let}, \kappa_{c02}, \kappa_{u_v}, \kappa_{u_q}$ are defined in **section XXX** and correspond to the pricing of the the lettuce and control inputs and the penalty terms P_{c02}, P_T, P_H are defined as follows:

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

The penalty constants $c_{p_{CO2}}, c_{p_{T_u b}}, c_{p_{T_l b}}, c_{p_H}$ were found in empirically in **ref XXX** in order 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, it is important for the agent 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 the acceptable levels of CO2 for human brief operation.

parameter	value	units
$c_{p_{CO2}}$	$\frac{10^{-3}}{20}$	€$\cdot(ppm \cdot m^2)^{-1}$
$c_{p_{T_u b}}$	$\frac{1}{200}$	€$\cdot(C^\circ \cdot m^2)^{-1}$
$c_{p_{T_l b}}$	$\frac{1}{300}$	€$\cdot(C^\circ \cdot m^2)^{-1}$
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

Uncertainty It is not the focus of this thesis to accurately model the uncertainty in a greenhouse crop environment, however, it is desirable to see the effect of uncertainty on the generated policy for each of the different algorithms. There are several sources of uncertainty in a greenhouse environment, including parametric uncertainty in the model, unmodeled dynamics, measurement uncertainty, and uncertainty in weather forecasts. The various instances of uncertainty observed in different aspects of the environment can ultimately be attributed to the uncertainty that manifests in the system's outputs. Thus, it was determined that in the stochastic case, there exists uncertainty in the change in the states between consecutive time steps. More specifically,for the discrete time model, it may be modelled as:

$$x(k+1) = x(k) + (f(x(k), u(k), d(k) - x(k)) \cdot (1 + W), \quad W \sim U(-\sigma, \sigma) \tag{3.7}$$

where σ represents the degree of uncertainty in the evolving states, expressed as a percentage. The uniform distribution was chosen for its higher level of aggressiveness compared to the normal distribution. While it may not accurately represent the uncertainty at hand, it is a useful tool for evaluating the potential variability and risks associated with the various policies generated.

3.2. Experimental Setup

The duration of the growing period for lettuce was determined based on the findings of **reference XXX**, which indicate that the growing period typically falls within the range of 30 to 50 days. Therefore, a fixed growing period of fixed 40 day was selected. **Ref XXX** states that discretizing the van Henten model using a time-step between 15 minutes and 1 hour is recommended. Therefore, a time interval of 30 minutes was selected for the purpose of this study. As a result, over a duration of 40 days (1 episode or 1 complete simulation), there is a total of:

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

The initial interval of 30 minutes was originally set at 15 minutes. However, this decision was revised due to the excessive computational burden it imposed on the MPC solver, with minimal benefits. Consequently, extending the time step enables an exponential speedup in the simulation of the 40-day period, while causing only minor deterioration in performance. As a consequence of this, there is a

noticeable increase in the rate at which RL training occurs, thereby allowing for a larger quantity of training episodes.
To facilitate the learning process of RL, the observations returned from the environment were normalized by a running mean and variance. The VecNormalizeWrapper in Stable-Baselines3 is responsible for updating the mean and variance for every observation received from the environment. The observation is normalized as per Equation 3.8 and clipped between $[-10, 10]$.

$$obs_{norm} = \frac{(obs - \mu_{obs})}{\sqrt{\sigma^2_{obs} + 1 \cdot 10^{-8}}} \tag{3.8}$$

where μ_{obs} and σ^2_{obs} represent the running mean and variance, 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, in order to ensure reproducibility, a seed value of 4 was used for the generation of random numbers. This seed value was utilized for both the initialization of neural network weights and the selection of actions for exploration purposes.

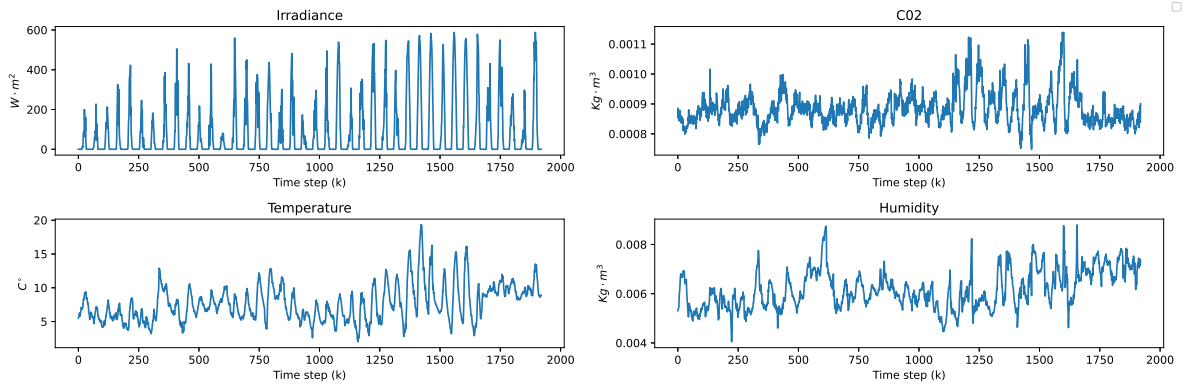


Figure 3.1: Weather Data

Weather Data The weather data used in training is obtained from the VenLow Greenhouse in Bleiswijk from the period January 30–March 11, 2014. The weather data was resampled from its original 5-minute interval to a 30-minute interval, in accordance with the timestep of the environment.The weather data remains consistent throughout the episodes, irrespective of whether the training and/or evaluation is conducted under deterministic or stochastic conditions. As such, the validation data is the same as the training data. In practice, it may be necessary to evaluate the agent on unseen weather data, but the thesis aims to develop an RL policy for incorporation with MPC to investigate the resulting controller. Thus, provided that all algorithms/controllers utilize identical weather data, if the agent learns a suitable policy for this specific weather pattern, it can be considered a suitable basis for comparing algorithms.

Deterministic and Stochastic Case When learning a policy in both 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 3.7. In the stochastic case, three levels of uncertainty were tested, namely σ = 20%,σ = 40% and σ = 80%. Although these uncertainty levels might be considered extreme, it was desirable to learn a policy for each of these uncertainty levels to compare to MPC and RL-MPC under identical conditions of uncertainty. The optimal configuration(s) that produce the most favorable outcomes in the deterministic scenario will be employed in the stochastic scenario, and there will be no reevaluation of hyperparameters. WWhile the stochastic environment provides a representation of 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.

Performance Metrics The primary performance metric used for evaluating RL agents is the 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 (**eq XXX**) 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 is optimizing, and subsequently, what the MPC and RL-MPC controllers are optimizing. 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 metric across policies, since these are all form part of the reward function (with the exception of the computational time) and are not directly optimized. Therefore, making comparisons would not be meaningful, and only observations can be made. However, the computational time taken to compute the optimal control action is an important metric, particularly when combining RL with MPC.

3.3. Hyper-parameter Tuning

The process of hyper-parameter tuning is frequently laborious and requires exhaustive exploration to determine the best configuration to maximize 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 obtained hyper-parameters. The discount factor and activation function are not reported here. The effect of these two hyper-parameters are 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

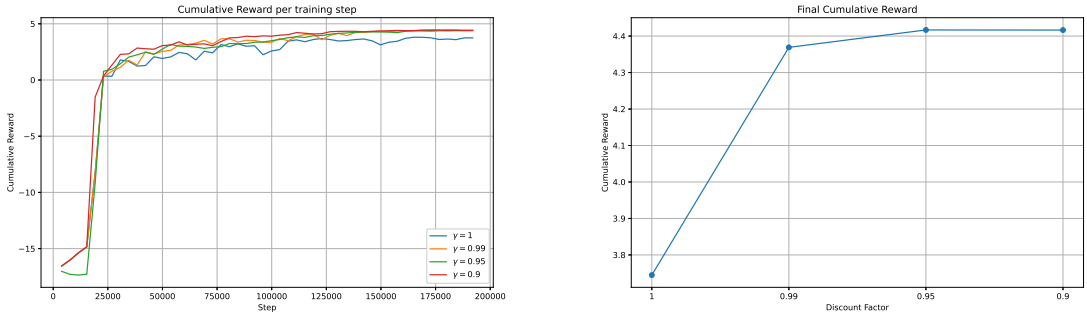
Table 3.2: Hyper-parameters

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 a commonly used activation function due to its simplicity and superior convergence **ref XX**. Although ReLu is differentiable with respect to the weight of the neural network, it is not differentiable with respect to the inputs of the neural network. This is important to note since it is necessary for the trained value function to 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 with respect to the inputs.

Discount Factor Another consideration is the discount factor, denoted as γ. Since the problem is episodic and therefore the cumulative rewards are bounded, it is possible to have a $\gamma = 1$. By setting the discount factor γ to 1, the agent is able to consider the entire prediction horizon when making decisions regarding its actions. Additionally, the value function obtained during training satisfies **equation XXX** indicating that it contains information pertaining to the entire prediction horizon. Having $\gamma < 1$ will shorten the agent's 'prediction horizon' and the resulting value function may not provide significant benefits for the MPC. However, it may stabilize the learning and therefore yield a better policy as compared to when $\gamma = 1$.
Results pertaining to the activation function and discount factor are shown and discussed in subsection 3.4.4

3.4. Deterministic Results

The outcomes of modifying the discount factor, the activation function, and the impact of the three distinct observations provided to the agent (Equation 3.1, Equation 3.2, Equation 3.1) are presented and



(a) Cumulative Reward per Training Step

(b) Final Cumulative Reward

Figure 3.2: Discount Factor vs Cumulative Rewards

examined in this section.Finally, a discussion of the chosen policies and the identification of the desirable characteristics that can be integrated with the MPC framework. While there are several hyperparameters that 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 MPC.

3.4.1. Discount Factor

The tests conducted involved returning the observation to the agent as specified in Equation (2) of the observation tuple, with the hyper parameters as specified in Table 3.2. The impact of the discount factor on the agent's performance and the generated value function is investigated.

Performance The cumulative reward achieved over the training period and the final cumulative reward achieved for each different discount factor are depicted in Figure 3.2a and Figure 3.2b respectively. As can be 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 in finding 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.
In the deterministic case, one can assess the accuracy of the obtained value function by comparing the predicted values of each state and time step visited during the 40-day simulation with the actual values of the visited states. In addition, the predicted value of each state may also be compared to the cumulative rewards obtained at each state. One can determine the precise value of each state visited in the simulation by adding up the rewards from that state onwards, as per **eq XXX**, with the respective discount factor used.

Value Function Approximation It is difficult to determine whether the critic has converged. Given that the critic serves as a q-value function approximator, the actor must identify the optimal action for a given state in order to compute the value of that state using the critic. Therefore, convergence of the value function is dependent upon the actor policy as well. Although the training curves indicate that the critic has converged, it has only converged in alignment with the actor. In order to assess whether the critic (and actor) has achieved convergence in predicting the value function, it is possible to visualize and compare the actual and predicted value of a given state.
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. It is noted that for the deterministic case, the realizations of state trajectories and rewards received do not differ between simulations, hence the exact value of each state may be calculated.

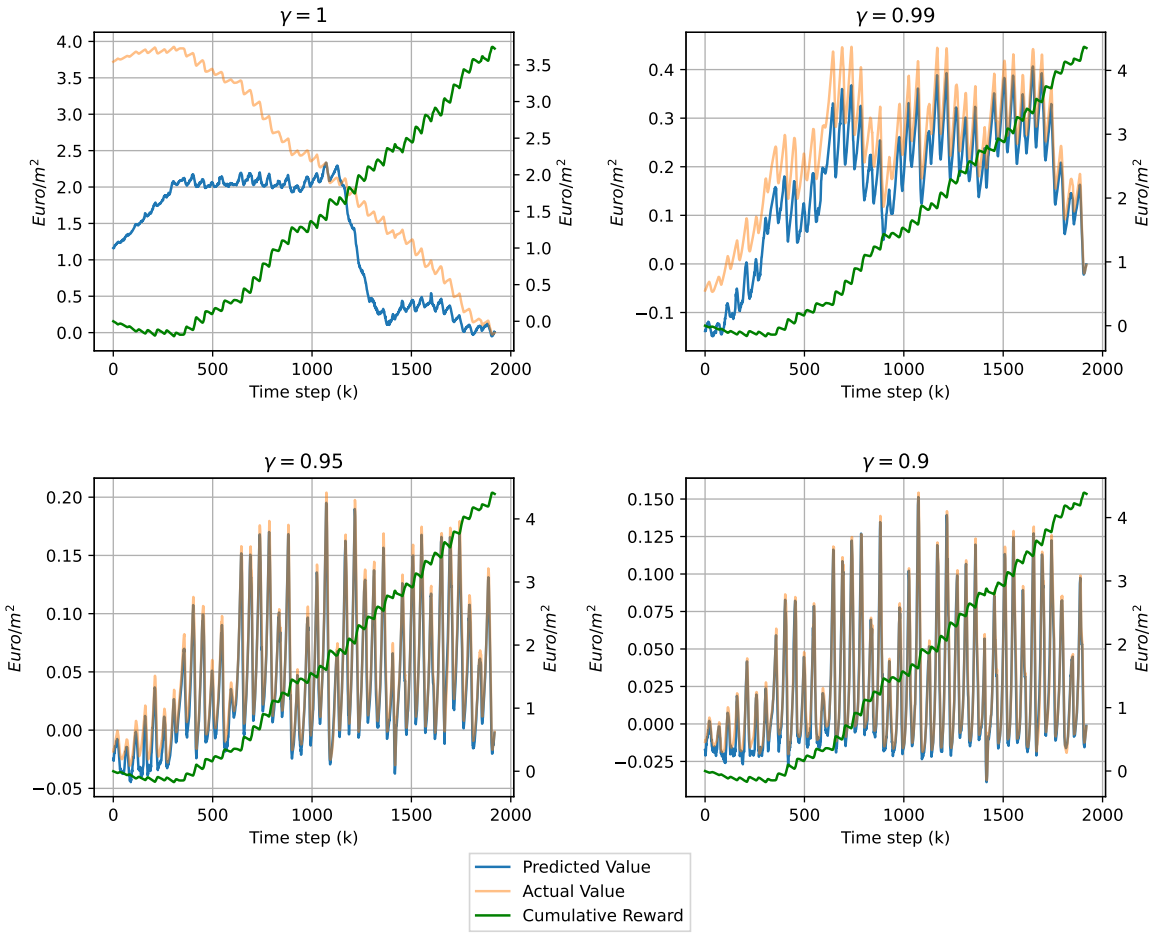


Figure 3.3: Value vs Discount factor

The figure shown in Figure 3.3 illustrates the comparison between the predicted and actual value of each state for each discount factor. In addition the cumulative reward at each time step is also logged. The left-hand side axis represents the values, while the right-hand side axis represents the cumulative reward. It is acknowledged that the trajectory of the actual value (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. It is seen that the lower the discount factor, the more accurate the predicted value of a state is. This may be due to the decrease in problem complexity and therefore more accurate approximations. When $\gamma = 0.9$, the critic is capable of accurately predicting the actual value of a state. However, it falls short in 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, despite having a worse performing policy, could still provide a reasonable estimation of the value of a state throughout the entire simulation. Upon 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 accurately estimate the value of a state in all cases. Figure 3.3 is just one such realization.

3.4.2. Activation Function

It is also important for the trained critic to utilize a differentiable activation functions to ensure differentiability. This is done so that it may be used within the MPC framework. However, such an activation function, such as the commonly used tanh activation function may or may not yield desirable results in terms of maximizing cumulative reward. This section compares the performance of a learned

agent with tanh activation functions against and agent with ReLu activation functions for a $\gamma = 1$ and $\gamma = 0.95$ with the ReLu acting as the baseline performance.

	Performance		
Discount Factor	ReLu	tanh	SpeedUp (%)
1	3.72	3.44	-7.53
0.95	4.40	4.17	-6.08

Table 3.3: Effect of the tanh activation function

Table 3.3 displays the effect of the tanh activation on the agents final performance. it is clear that ReLu significantly 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. By incorporating the concept of differentiability into the critic, it seems that there is a decline in performance.While additional investigation into the hyperparameters may be necessary to reject this notion, the focus of this thesis does not lie in the development of the optimal RL generated policy. Rather, chapter 3 aims to create a policy, and accurate critic, that is competitive with MPC so that when merged, produces a potentially better policy.

~~3.4.3. Observation Tuples~~

Might go into appendix

3.4.4. Final Results and Conclusion

From the findings, the best policy produced by RL is with the hyperparameter shown in Table 3.2, with a $\gamma = 0.95$ and with ReLu activation functions. And 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 it is not differentiable. And an agent, learned with a $\gamma = 1$ and tanh activation function results in a worse performing policy and a critic that struggles to accurately predict the value of states. With that being stated, a critic that has undergone training with a discount factor of $\gamma = 0.95$ is able to accurately predict the value of a state and may still possess sufficient knowledge regarding future rewards to be advantageous for MPC. Additionally, a critic learned with a $\gamma = 1$, albeit bad approximations, may also provide enough information. Therefore, the selected agents (the actor and the critic) to be attempted to be merged with MPC and shown in Table 3.4.

Agent	γ	Activation Function	Final Performance
Agent 1	0.95	ReLu	4.27
Agent 2	0.95	Tanh	4.18
Agent 3	1	Tanh	3.03

Table 3.4: Selected Agents

Although Agent 1, Table 3.4 performs the best, its critic cannot be used in the MPC formulation. However section 3.6 discusses how this issue may be addressed.

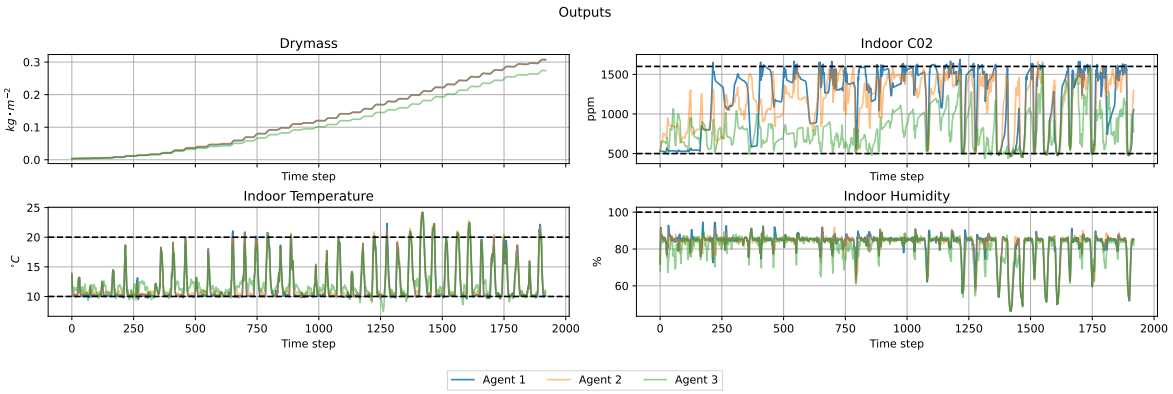


Figure 3.4: Time series of system outputs

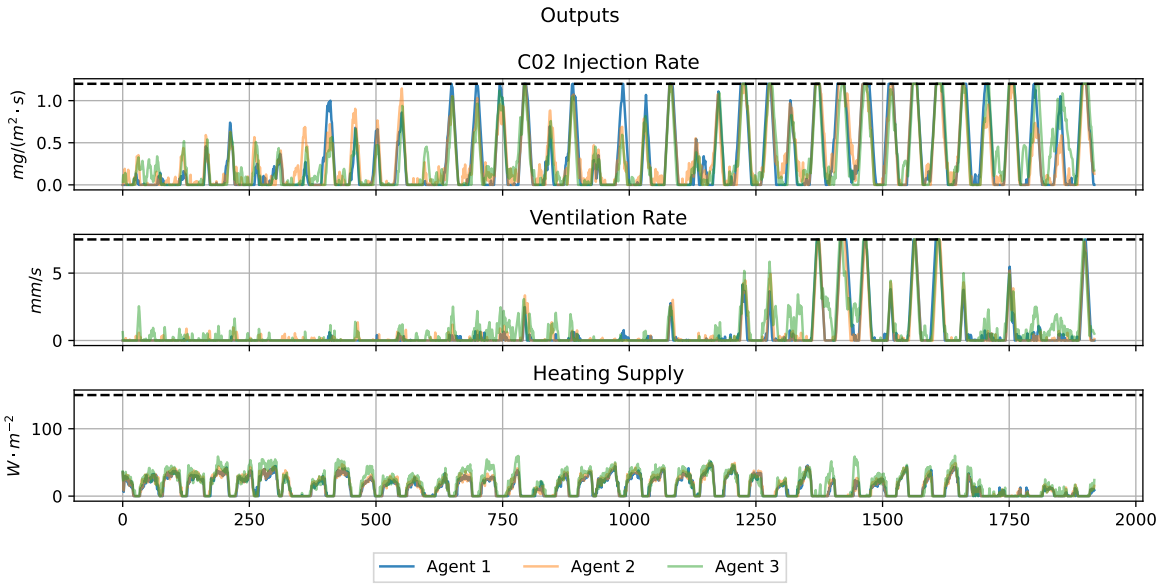


Figure 3.5: Time series of system outputs

The time series plot of the system outputs and inputs across the 40-day period for every agent as selected in Table 3.4 is shown in Figure 3.4. These time series plots are similar to as those reported in **ref XXX and XXX**. Direct comparisons are not possible since in **ref XXX** does not specify the weather data range used and the reward function differs from what is used in this thesis. Furthermore, **ref XXX** include additional constraints on the temperature levels during the day to encourage heating by solar radiation during the day and the heating system by night. These constraints were not imposed in this paper as it aimed to provide RL with greater autonomy in optimizing EPI while minimizing constraint violations deemed dangerous for plant and/or human operation. Furthermore, **ref XX** also reports similar results, however a direct comparison is not possible, since hyper-parameters and reward function differs. However, it is noted that results, time series and cumulative rewards, are similar enough to give confidence in the 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 is noted to take $\approx 0.2ms$. This is expected since a simple inference of the actor network is required to calculate the optimal action.

3.5. Stochastic Results

In training the stochastic RL agent, the best hyper-parameter configuration was used, namely the same as Agent 1 (Table 3.4). Although this produces a critic that is problematic for the integration with MPC, this is solved in section 3.6. Three stochastic agents were trained, each trained on a different level of uncertainty as specified in section 3.1 with the uncertainty model according to Equation 3.7. All stochastic agents used the same hyper-parameters as Agent 1. 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 $\sigma = 20\%$ is referred to as 'Agent 0.2'. The agent named 'Agent 1' will be referred to as the 'Nominal Agent'. Each Agent is compared to the other stochastic Agents in an environment with each level of uncertainty.

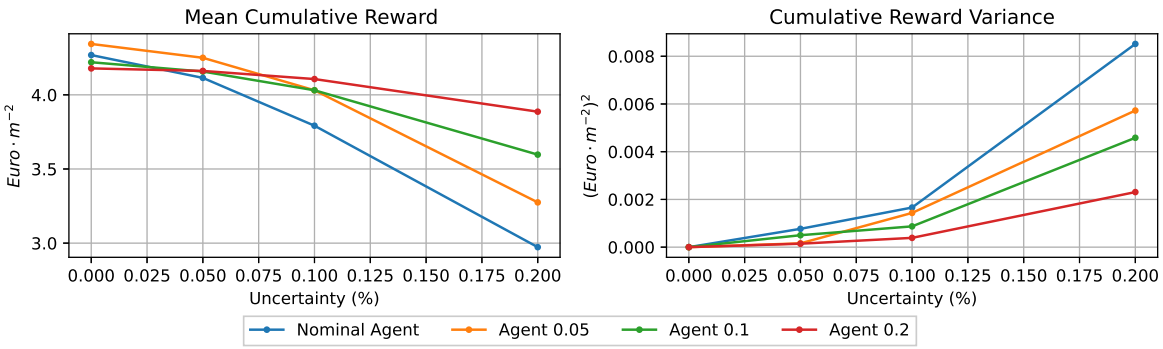


Figure 3.6: Stochastic RL policy performances

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, mean cumulative reward decreases and the variance increases. This is clear across all agents. It is also evident that each agent outperforms others in terms of performance, as indicated by higher mean cumulative reward and lower variance, when tested on the same level of uncertainty on which it was trained. However, it seems that in the nominal case ($\sigma = 0\%$), the agent trained on a 5% uncertainty level (Agent 0.05) outperforms the nominal agent, this could be due to the agent having explored more due to the noise. Moreover, Agent 0.2 seems to achieve a higher average cumulative reward than Agent 0.1 when tested on a 10% uncertain model.

3.5.1. Conclusion

It is widely recognized that RL has the capability to address uncertainty through 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 a RL policy learned from stochastic data can transfer its characteristics to the RL-MPC framework. Finally, each agent, specifically the Nominal agent, Agent0.05, Agent0.1, and Agent0.2, is employed in its corresponding stochastic environment. Although Agent0.05 may outperform the Nominal Agent under nominal conditions, for the purpose of this thesis, its corresponding agent would be used.

3.6. Trained Value Function

Although training an agent using SAC produces an actor and a critic network, it was shown in subsection 3.4.4 that the produced critic had undesirable characteristics. This critic was trained based on a changing policy that was dependent on the critic itself, therefore it is not surprising that the approximation to the value function was sub-optimal. However this section aims at training a value function approximator with a fixed policy[1]. Therefore, a value function may be trained on the best policy obtained. 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 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 is mostly dependent 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. Two methods were employed to various value function approximators, namely the temporal difference learning method and expected return method. It is noted that this value function is only accurate under the policy it was trained on, and that the neural networks used for as the value function approximator must be made up with tanh activation functions to ensure differentiability.

3.6.1. Temporal Difference Learning

This method uses a similar method by which SAC, DDPG and TD3 update their critic. Most similar to DDPG. Two neural networks are used to represent the value function, a current and target network. The mean squared bellman error is minimized between the target values (from the target network) and the current values (from the current network) as shown in **eq XXX**. Moreover, a polyak averaging is used to update the target networks.

Obtaining Data To obtain data, the nominal Agent (or Agent 1 Table 3.4) was used. A similar approach in obtaining data as in **ref xxx** was used. Along the nominal trajectory, q ($q \in \mathbb{N}_{>0}$) internal states, x and inputs, u were uniformly sampled from $\hat{\mathbb{X}}^4$ and $\hat{\mathbb{U}}^3$ 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 normalized as per **??** 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 day period is over. The transition tuple of the actual system is also stored. To ensure a value function approximator generalizes well across the state space, the quality of sampled internal states and inputs is important. From the time series plot Figure 3.4 and Figure 3.5 is 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 the entire set $\mathbb{U}^3$ as shown in **eq XXX**. States x_2, x_3, x_4 were sampled with a range slightly larger than their respective minimum and maximum constraints 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:

[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

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

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

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

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

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 optimizer is used to minimize 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, which is a hyperparameter that needs to be tuned.

Show the sampled state space

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 has the advantage of training being significantly more stable since, in contrast to the TD method, targets remain unchanged as the function approximator's weights are updated. However, this method of learning requires a lot of data. Many trajectories must be simulated until the end, and the return must be calculated for each state visited. More importantly, only starting states are sampled, which makes it harder to obtain the same data spread as the TD-method. In contrast, the TD-method allows for higher data spread because trajectories only include one time step, which makes it possible to sample more initial states across the state space.

Obtaining Data The same number of starting points must be sampled in order to obtain a spread that is comparable to the TD-Method; however, because the trajectory must be run through to the end of the simulation, a significantly larger amount of computational data is needed. However, targets are calculated not only for the initial state, but also for each state encountered along the trajectory. This means that fewer initial points are required, but it is still necessary to select them appropriately. 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. Therefore, initial states

and inputs were sampled from $\hat{\mathbb{X}}^4$ and $\hat{\mathbb{U}}^3$ and time k is uniformly sampled across the entire time horizon as shown in Equation 3.12.

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

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

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

where represent the minimum and maximum range of the sample state space for a specific state, respectively, and 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 sampled around/near the nominal trajectory. As can be seen from Figure 3.5 and Figure 3.4 and later in **fig xxx**, it can be observed that the performance of policies can vary significantly with minimal changes in the nominal state and input trajectories. Therefore, this approach of sampling initial states and inputs is deemed appropriate. Finally , Agent 1 (or the nominal Agent) was used for the fixed policy. Given that the computation of a control action requires a time of $0.2ms$, it is possible to sample a large number of trajectories in order to achieve appropriate coverage of both 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 is sampled, for each state observed/visited, the total return is calculated, and the tuple (s, TR) 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 and labels as the total return and the loss function in Equation 3.14 is minimized with the Adam optimizer.

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}[(V_\phi(s_k) - TR)^2] \tag{3.14}$$

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 To investigate the effect of the value function in the MPC framework, it was decided to train four value functions that were based on different architects and/or states used as inputs. 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_1	Equation 3.1	2	128
V_2	Equation 3.1	2	32
V_3	Equation 3.1	1	128
V_4	$(y_1(k), k)$	2	128

Table 3.7: Value Functions

Each value function was trained on the nominal agent, Agent 1. Additionally, V_4 was trained on each stochastic policy, namely 'Agent 0.05', 'Agent 0.1', and 'Agent 0.2'. 1000 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 inputs spaces. The architects are chosen based on the principle that each subsequent architecture model becomes less complex, with V_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 total return and the actual total return as shown in Equation 3.14. Moreover, the accuracy of the resulting value function across the simulation period will be visualized by using Equation 3.15.

$$\begin{aligned} V(s_k) &= r_k + V(s_{k+1}) \\ \therefore V(s_{k-1}) &= r_{k-1} + V(s_k) \\ \therefore V(s_{k-2}) &= r_{k-2} + r_{k-1} + V(s_k) \\ \therefore V(s_0) &= \sum_{i=0}^{k-1} r_i + V(s_k) \end{aligned} \tag{3.15}$$

During each time step, the computation of the initial state's value will be determined by the cumulative rewards received up to that point, as well as the approximation of the current state's value using the value function. If the value function is able to approximate the value of a state accurately, then Equation 3.15 should yield the same result for every time step, resulting in a horizontal line, $y = V(s_0)$.

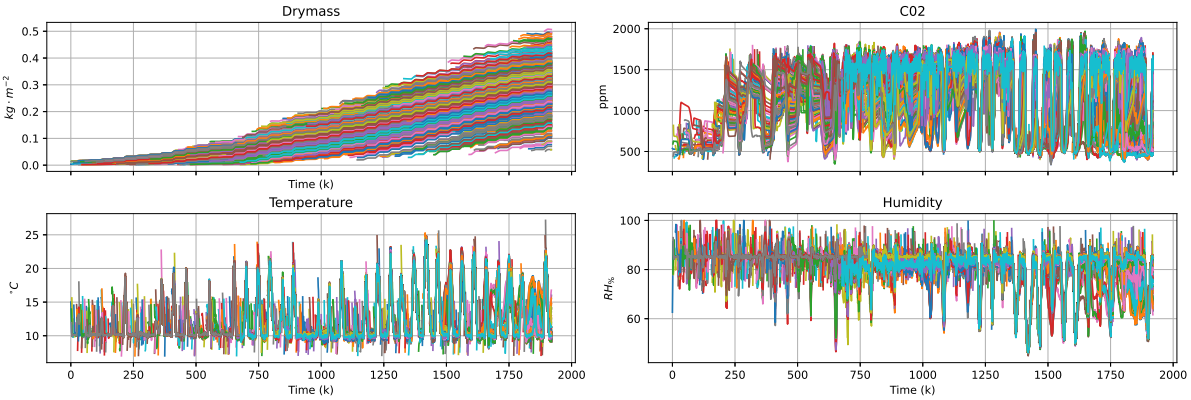


Figure 3.7: Sampled States

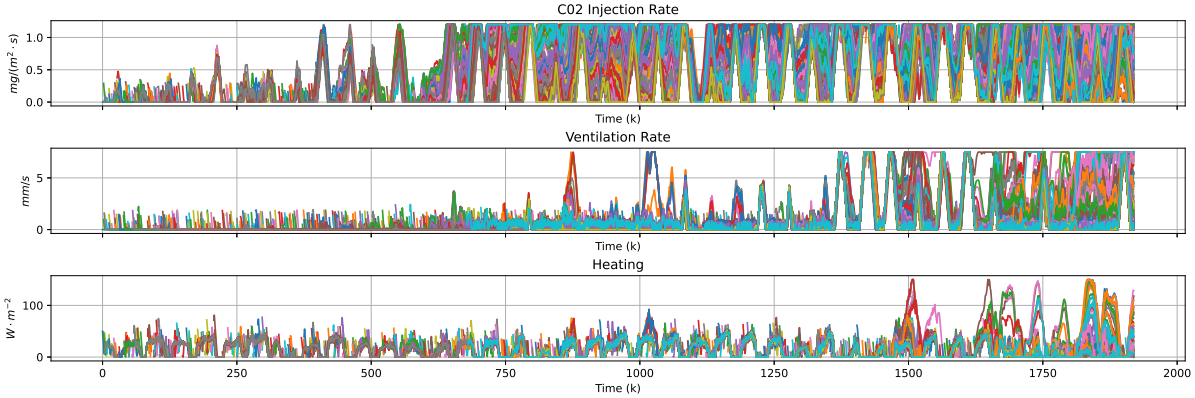


Figure 3.8: Sampled Inputs from Nominal conditions

Figure 3.7 and Figure 3.8 are the results of all the trajectories sampled from the nominal agent. The figures reveals that the sampled trajectories exhibit a lesser extent of coverage of the state and input spaces as compared to the temporal difference learning. Nevertheless, a sufficient level of coverage is achieved. Following the completion of training and validation, a small number of additional trajectories will be sampled in order to verify the sufficient accuracy of the prediction model.

3.6.3. Results

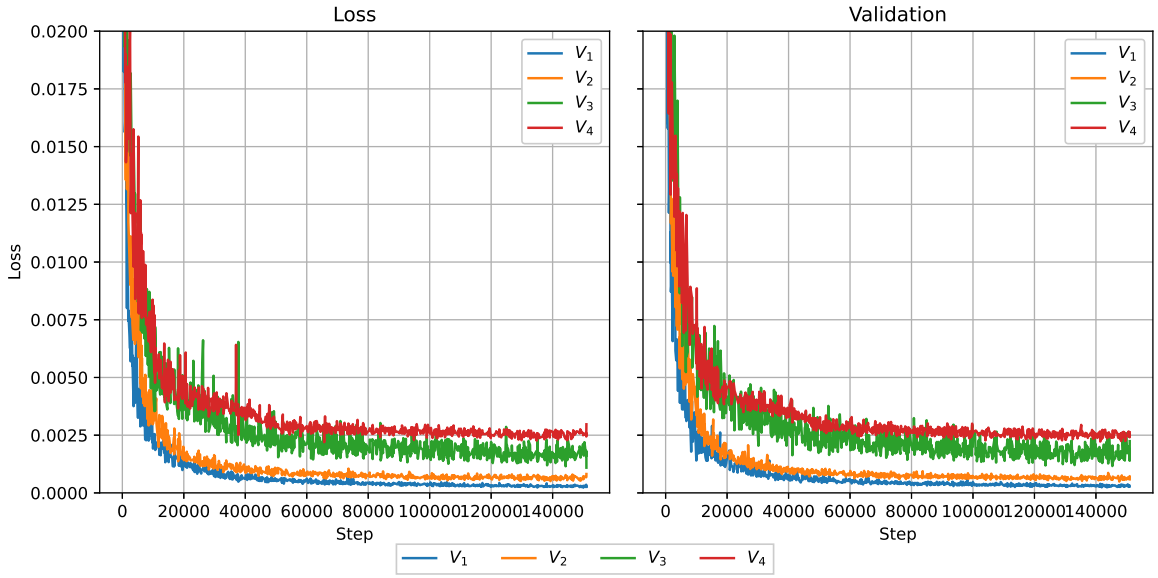
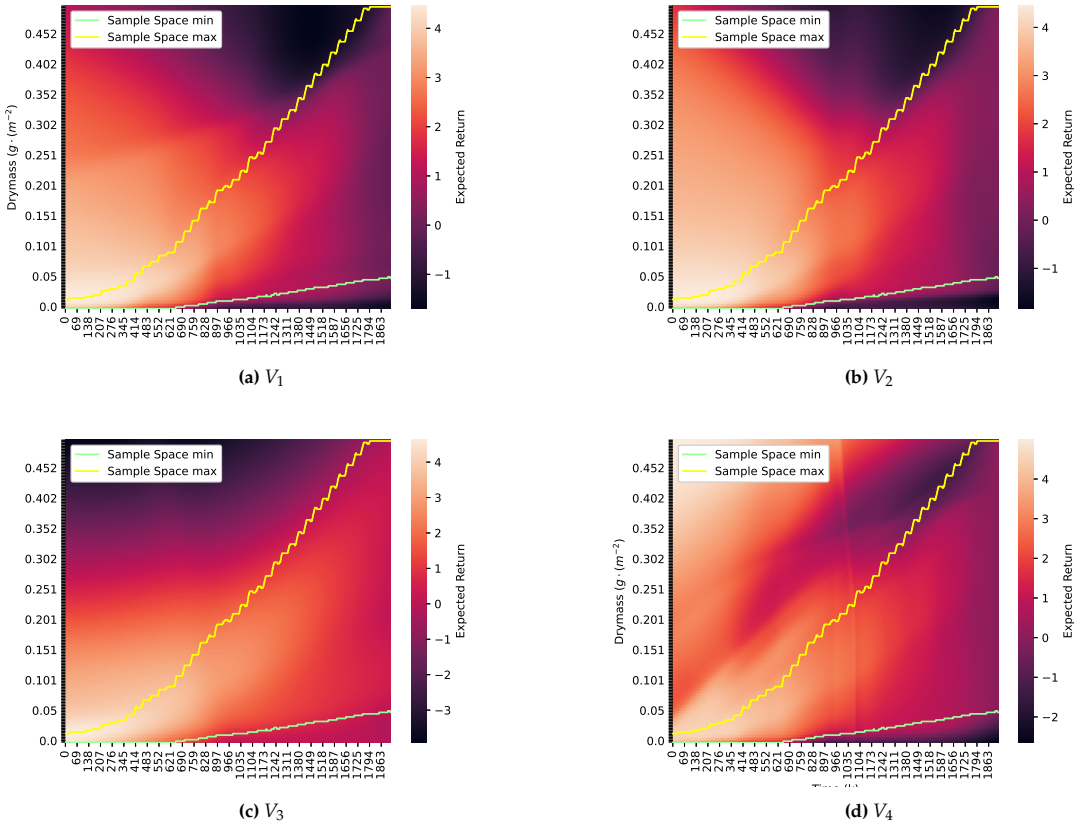


Figure 3.9: Performance Curves, trained on the nominal Agent

Figure 3.9 displays the loss curves of all four models trained on data generated by the nominal agent. As expected, the baseline model (V_1) is able to achieve the highest accuracy compared to the simpler models.This can be attributed to its more complex structure, with each subsequent simpler model exhibiting lower accuracy. Nevertheless, when utilizing the reduced observation space model, denoted as V_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.



(a) V_1 **(b)** V_2

(c) V_3 **(d)** V_4

Figure 3.10: Value vs Drymass vs time

Figure 3.10 gives a visual representation of the value of a state given the dry mass and time. It is noted that although V_1, V_2 and V_3 require an observation space as given in Equation 3.1 to determine its value, since the value is highly dependent on the dry mass state and time, the other states in the observation space were averaged. V_4 naturally has an observation space of only time and the dry mass state. The lower and upper limits in Figure 3.10 are displayed to 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. The intuitiveness of Figure 3.10 stems from the fact that the highest return is observed at the beginning of the growing period, which can be attributed to the longer duration of the growing period. Moreover, having a higher dry mass at any specific time leads to greater rewards, this behavior 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 $400g \cdot m^-2$, therefore for dry masses close to this value, very little reward can be expected, hence the lower returns in Figure 3.10. The primary difference is that the level of smoothness in V_3 as depicted in Figure 3.10c seems notably higher than that of even Figure 3.10d, which is exclusively trained on the present time and dry mass. It is anticipated that the opposite would be true, as more intricate models would be capable of fitting the data with greater precision.

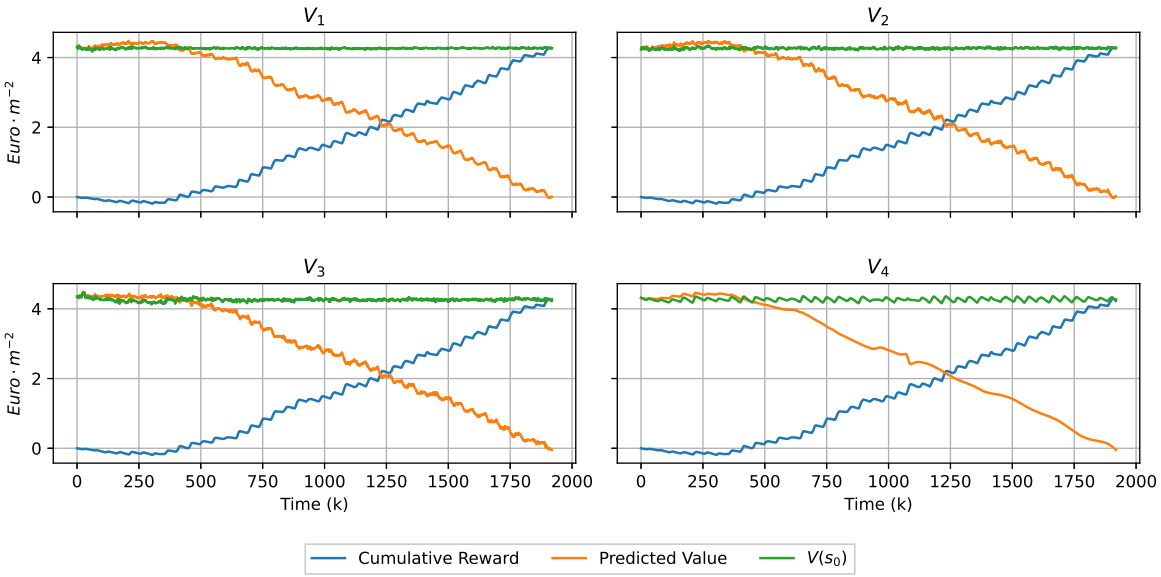


Figure 3.11: Value predictions - Entire Time Horizon

Figure 3.11 displays a time series plot of the cumulative rewards plotted against the predicted value at each time step. This plot is similar to what is shown in Figure 3.3. Additionally, it includes the calculated $V_(s_0)$ at each time step using Equation 3.15 as a visual indicator of the accuracy of the value function. As demonstrated in Figure 3.11 in conjunction with Figure 3.9, the predicted values show a high level of accuracy. This is evident from the nearly perfect horizontal line at $V_(s_0)$ that spans across the prediction horizon. However, what is interesting and that cannot be seen in Figure 3.10 is the level of noise present in each prediction. Naturally, V_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 a lot smoother than all the others. This observation strongly suggests that the primary factors determining the value of a state are its drymass and time, while the minor fluctuations in rewards and corresponding expected return resulting from other factors have negligible influence across the entire time horizon.

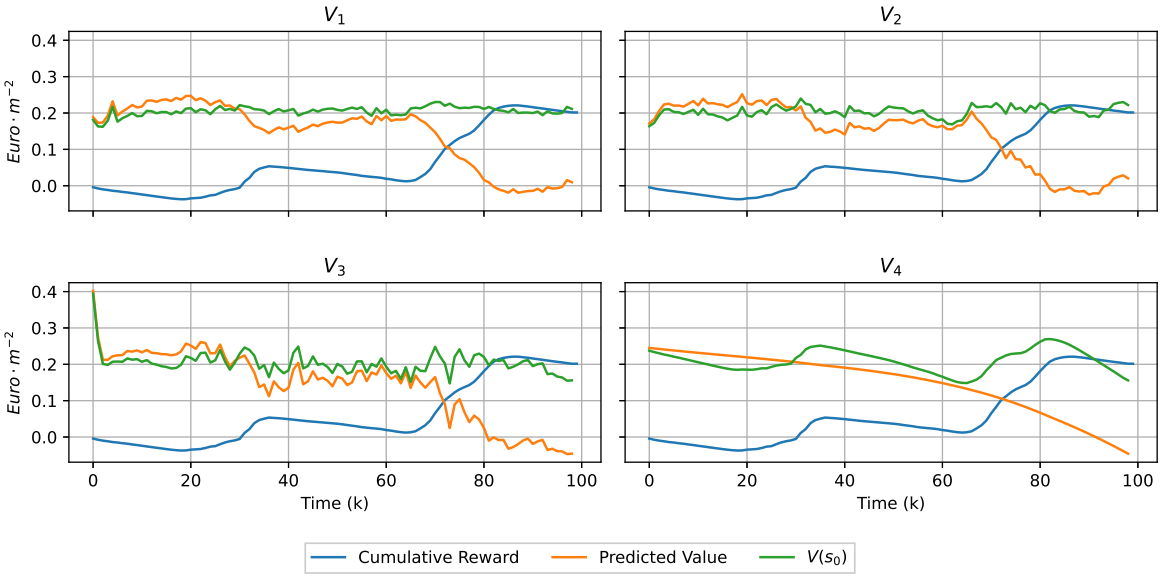


Figure 3.12: Value predictions - 2 Days

Figure 3.12 shows a time series of the cumulative reward and predicted values over a two day span.

Remark 1 *It is evident that while V_1, V_2, V_3 can produce more precise estimations (as demonstrated by the proximity of their prediction line V_{s_0} to the actual value), V_4 remains significantly smoother. Moreover, V_3 is not as smooth as it may have seemed in Figure 3.10 and displays the highest level of noise across the four models trained. Although this is one realization of a trajectory,this behavior was observed in all simulated trajectories.*

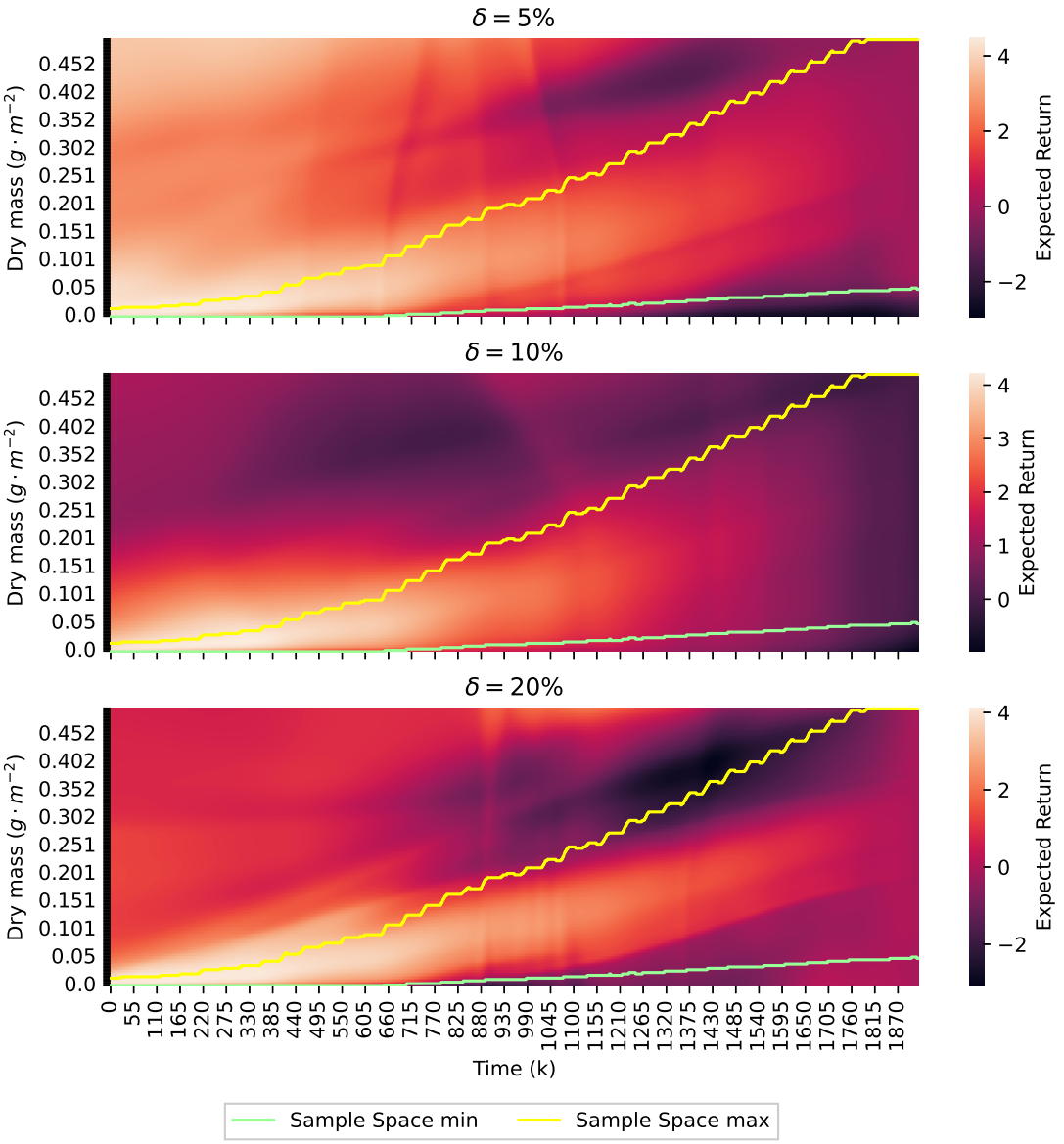


Figure 3.13: Drymass vs Time vs Value - Stochastic

Finally, for each level of uncertainty in the environment, a value function was trained with its corresponding agent. Figure 3.13 displays the same heatmap as Figure 3.10, yielding similar behaviour as compared to the nominal conditions. However, it appears to be more coarse, particularly beyond the range of the training data.

3.7. Conclusion

The objective of this chapter was to develop a policy that is competitive with MPC and 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 provides inaccurate value estimations and a policy that performs 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, in order to integrate the critic with MPC, it is necessary for the critic to be differentiable. This requires the use of the tanh activation function. However, using this activation function leads to a policy that is not as effective as when using a non-differentiable activation function like ReLu.
Given these obstacles, it became evident that additional measures were necessary. It was decided to train separate critics on the best rl policies obtained. Therefore lower discount factors and non-differentiable activation functions may be used. The best policy found and trained on the nominal data was denoted as Agent 1 with parameters shown in Table 3.4. Using the same hyper parameters as Agent 1, 3 different agents were learned based on each level of uncertainty in the greenhouse model, namely Agent0.05, Agent0.1 and Agent0.2.
Empirical evidence demonstrated that the stochastic agents outperformed the nominal model in terms of both the average cumulative reward and the variability in reward across different levels of uncertainty.
After generating sufficient policies for both the nominal and stochastic environment, it was necessary to train an appropriate critic/value function approximator. Four different model architectures were used to train the critics, where each model architecture becomes progressively less complex. Every agent, whether in the nominal or stochastic case, had a critic that was 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, the predictions were very noisy, which could be problematic later on 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. It was this model architecture that was used to train value function approximators for the stochastic agents.
These agents and corresponding value function establishes 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 on the greenhouse environment. Moreover, the effect of the prediction horizon is investigated on both nominal and stochastic conditions. The works in this chapter is similar to what is presented in [1] and [2]. However, direct comparisons are not possible. [1] develops to create a robust sample based controller, while this chapter will focus on constructing a conventional Model Predictive Controller (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 [2], such an MPC controller is developed. However, it does not provide specific weather data. Finally the optimization goal used in this thesis differs from that in both papers. However, a qualitative comparison may be done.

4.1. Greenhouse MPC problem formulation

In order to directly compare to RL and to the RL-MPC controller, it is important that the optimization goal is equivalent. As discussed in subsection 2.1.3, it is desired to optimize the economic benefit of the greenhouse environment. Similarly to section 3.1, the optimization goal of the MPC is done to ensure that the sum of stage costs are equal to the actual economic benefit of the system. As such the following optimization goal is solved at every time step:

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

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

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

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

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

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

To ensure that the optimization goal is exactly the same as the Rl reward function (section 3.1), the cost function $V(u(k), y(k))$ becomes:

$$
\begin{aligned}
l(u(k), y(k)) = -\kappa_1(y(k) - y(k-1)) + \sum_{j=2}^{4} \kappa_j u_{j-1} + \sum_{i=1}^{4} s_i(k) \\
\text{where} \quad s_i(k) \geq 0, \\
s_1(k) \geq c_{p_{C02}} \cdot (y_2^{min} - y_2(k)), \\
s_2(k) \geq c_{p_{C02}} \cdot (y_2(k) + y_2^{max}), \\
s_3(k) \geq c_{p_{T_{lb}}} \cdot (y_3^{min} - y_3(k)), \\
s_4(k) \geq c_{p_{T_{ub}}} \cdot (y_3(k) + y_3^{max}), \\
s_5(k) \geq c_{p_H} \cdot (y_4^{min} - y_4(k)), \\
s_6(k) \geq c_{p_H} \cdot (y_4(k) + y_4^{max}),
\end{aligned}
\tag{4.2}
$$

where the slack variables are introduced in order to accommodate the output constraint in equation Equation 4.1, resulting in an optimization 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 what is used in the Rl problem formulation and given in Table 3.1.

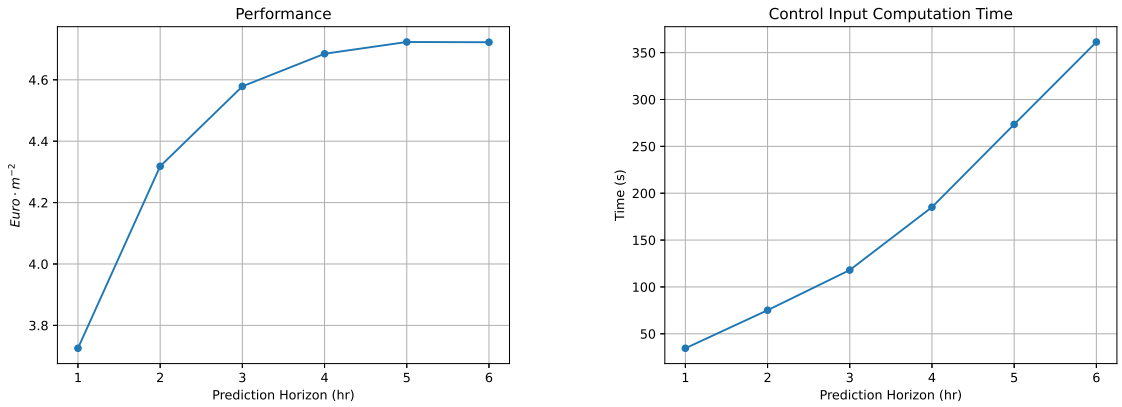
Experimental Setup Furthermore, the experimental setup is identical to what is used in section 3.2 with the exception of normalizing observations. The performance metrics are the sum of the stage costs during the simulation period, which is equivalent to the EPI minus the sum of the state violations. For the stochastic environment, the performance metric is calculated by taking the average of the sum of stage costs over 30 simulation periods, as well as considering the variance of the resulting stage costs. In the case of stochasticity, the MPC algorithm still solves the optimization problem defined by equation Equation 4.1 using the nominal system parameters. However, it is only during the system evolution that the uncertainty in parameters is taken into account.

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 **ref XXX** and the non-linear solver IPOPT **ref xxx** 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 hour, 2 hours, 3 hours, 4 hours, 5 hours, and 6 hours. Results include the final cumulative reward and the average time required to solve Equation 4.1 for each prediction horizon.

[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



(a) Final cumulative reward of MPC on nominal conditions

(b) Computational Time of MPC control actions

Figure 4.1: MPC Performance Metrics

Figure 4.1a and Figure 4.1b exhibit the performance and the computational time of the MPC for each prediction horizon respectively. It it can be seen that performance increases with an increase in prediction horizon up until 6 hrs. This increase in performance however is not guaranteed for an economic model predictive controller as stated in [3] and [4], without a sufficiently long time horizon or an appropriate terminal cost function or constraints. Although not entirely clear in Figure 4.1a, a prediction horizon of 6hr, actually produces a slightly lower performing policy than with a prediction horizon of 5 hours.
This is also clear inFigure 4.1a that increasing the MPC's time horizon past 6hrs to 12 and 24 hrs produces a less optimal policy. As expected though, the computational time in computing the control action seems to increase with an increase in complexity. It is noted that while RL can compute the optimal control action in $\sim 0.2ms$, whereas even the fastest MPC controller (1hr prediction horizon) takes $\sim 35ms$. It is nearly 175x slower. While this outcome is not surprising, it effectively demonstrates the computational demand of MPC, specifically for a highly non-linear model.

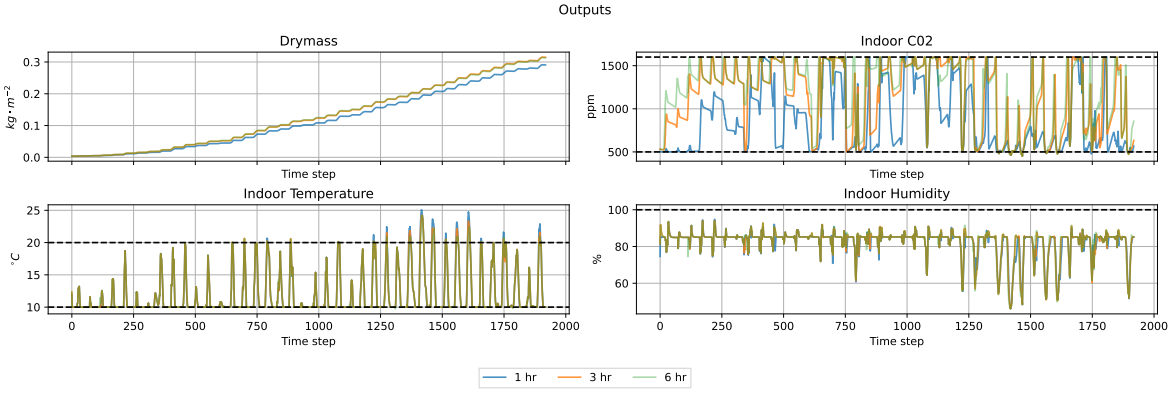


Figure 4.2: MPC 1hr and 5hr Time series of greenhouse outputs

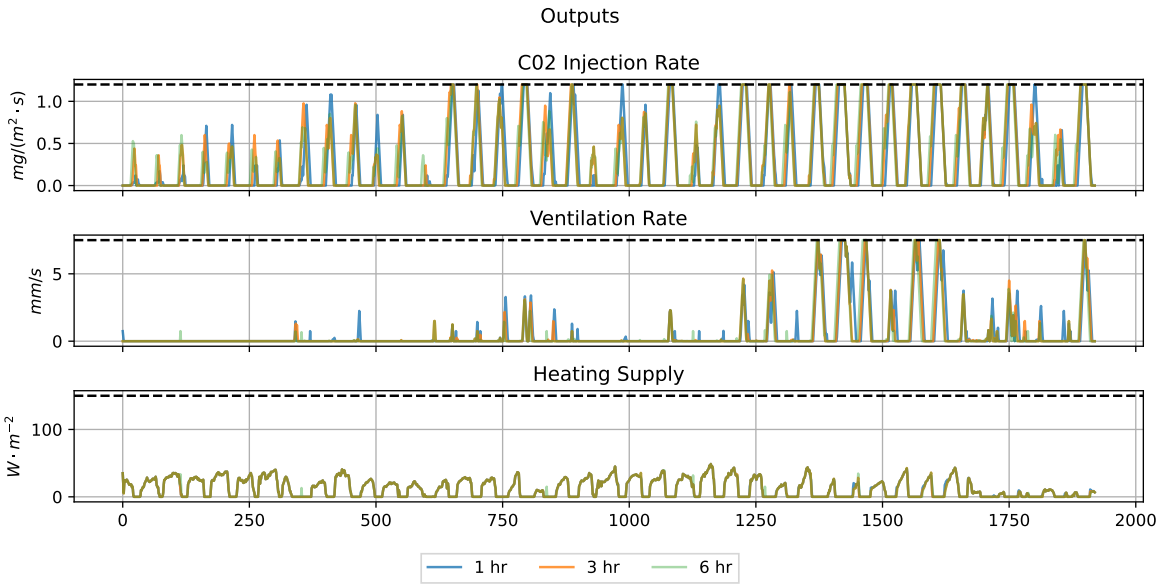


Figure 4.3: MPC 1hr and 5hr time series of controller inputs

Figure 4.2 and Figure 4.3 displays the trajectories of the states and inputs of the best performing MPC controller (5hr prediction horizon) and the worst performing (1 hr). These bear similarities to those of 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 raises 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 to be unchanged. Either the temperature does not play a vital role in plant growth or the prediction horizon is not long enough to see the effect of temperature at initial growth stages. However, it is clear that the MPC controller uses heating at night and the irradiance during the day to ensure that temperature constraints are met. These results are very similar to [2] and [1] although difficult to compare quantitatively due to the difference in the optimization goal and MPC problem formulation.

4.3. Stochastic Results

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

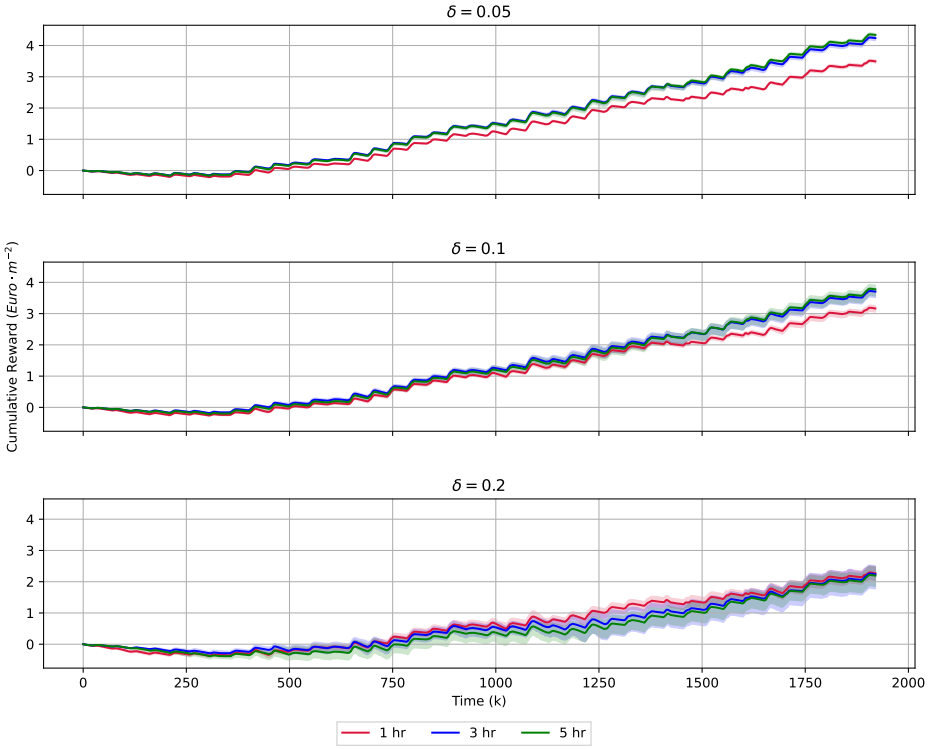


Figure 4.4: Time evolution of cumulative Rewards. Displays the evolution of the cumulative reward over the growing period, for each level of stochastic and prediction horizon. Solid lines represent the mean cumulative reward trajectory with a range indicating the minimum and maximum trajectory recorded 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 impact performance noticeable, there is significant performance degradation for $\delta = 0.2$. Also to note, 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 results in worse performance.

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 performance. Longer prediction horizons appear to have a greater negative

impact on the system beyond a certain level of uncertainty. Moreover, there is a clear adverse affect on variance as prediction horizon is increased for all uncertainty levels. This can be attributed to the fact that longer prediction horizons become progressively less accurate due to the increasing uncertainty of the model parameters. It is widely acknowledged that MPC is not adept at handling uncertainty, and these results are confirmation of this. Although it is possible to formulate a robust MPC, such as in [1], 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. Thus, it is worth exploring whether the RL agent can assist the MPC in mitigating the adverse impacts of parametric uncertainty.

4.4. Conclusion

MPC is a highly beneficial and efficient control strategy that has shown remarkable achievements in diverse fields. The formulation of the EMPC OCP was formulated and aimed to maximize the economic advantage of the greenhouse by aligning its optimization goal with that of the RL agent. This was done to facilitate direct comparisons between the two control schemes. Minimal adjustments were necessary to attain the level of performance showcased in this chapter. While the performance of the MPC is satisfactory,as anticipated for an EMPC, increasing the prediction horizon may not yield an increase in performance without an adequate terminal constraint or region. Moreover, the computational time required to calculate a control input is costly.The scalability of the computational time is considerably inferior to that of the RL agent. Therefore in more complex systems, the model is typically linearized or simplified, leading to a suboptimal policy. Furthermore, the performance of the MPC is significantly affected by the presence of parametric uncertainty, and increasing the prediction to address only exacerbates this.
EMPC has clearly shown to have advantages over its RL counterpart however clear downfalls exist. Lastly, to guarantee performance for an EMPC, appropriate additions to its formulation must be made.

5

Deterministic RL-MPC

This chapter aims to construct the RL-MPC framework and examines various implementations to determine their effectiveness. The resulting contollers are evaluated by comparing it with the MPC and RL controllers developed in previous sections. In addition, this chapter will analyze the nominal case in which the model is precisely known.

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

5.1. Implementation

While there are numerous implementations of RL-MPC, there is limited research focused on maximizing economic benefit specifically for continuous state and action spaces while training RL separately from MPC. As stated in [3] and [4], an EMPC without a terminal constraint and/or terminal cost function does not provide performance and stability guarantees. Specifically [3] states that a terminal constraint is required to ensure closed-loop performance, while [4] extends this concept by proving that applying a terminal region constraint with an appropriate terminal cost function is required to guarantee closed-loop performance. [4] 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 prove to be very difficult. It is the objective of this thesis is to ascertain whether the RL agent is capable of providing this.

The integration of RL into MPC will increasingly involve more complex implementations to analyze the impact at each stage. Firstly, initial guesses from the actor will be examined. Subsequently, a terminal constraint will be established by the RL agent. Following this, a terminal constraint region will be defined, also determined by the RL agent. The various value functions trained by the nominal agent will then be used at the terminal cost function, with and without the terminal region constraint. Lastly, a parallel problem will be presented in order to explore an slightly alternative application of the value function.

5.1.1. RL-MPC problem formulations

Implementation 1 Implementation 1 is the same as Equation 4.1, with the addition of initial guesses. Two sets of initial guesses will be tested and compared with one another. Given that the solution to the OCP in Equation 4.1 at time k_0 denoted as:

$$\begin{aligned} \mathbf{x}^{k_0} &= [x_{k_0}, x_{k_0+1}, x_{k_0+2}, \ldots, x_{k_0+N_p}] \\ \mathbf{u}^{k_0} &= [u_{k_0}, u_{k_0+1}, \ldots, u_{k_0+N_p-1}] \end{aligned} \tag{5.1}$$

then the two sets of initial guesses at the next time, k_1 step can be denoted as:

$$\begin{aligned}\tilde{\mathbf{x}}^{k_1} &= [x_{k_0+1}, ..., x_{k_0+N_p}, f(x_{k_0+N_p}, \pi(x_{k_0+N_p}), d_{k_0+Np}, p)]^T \\ \tilde{\mathbf{u}}^{k_1} &= [u_{k_0+1}, ..., u_{k_0+N_p-1}, \pi(x_{k_0+N_p})]^T\end{aligned} \tag{5.2}$$

$$\begin{aligned}\tilde{\mathbf{x}}^{k_1} &= [x_{k_1}, f(x_{k_1}, \pi(x_{k_1}), d_{k_1}, p), ..., f(x_{k_1+N_p-1}, \pi(x_{k_1+N_p-1}), d_{k_1+Np-1}, p)]^T \\ \tilde{\mathbf{u}}^{k_1} &= [\pi(x_{k_1}, \pi(x_{k_1+1}), ..., \pi(x_{k_1+Np-1})]^T\end{aligned} \tag{5.3}$$

The first initial guess (Equation 5.2) takes the previous time steps solution, shifts it 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. Essentially the actor is unrolled once from the last time step of the previous solution to Equation 4.1. The second initial guess,Equation 5.3, involves unrolling the actor Np steps from the current state, and using the resulting actions and states as initial guesses.

Implementation 2 The second implementation incorporates a terminal constraint. The asymptotic average performance of the EMPC can be guaranteed to be no worse than the performance of a optimal reference trajectory under certain assumptions. From [4], these assumptions are:

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

$$V_f(f(x, \kappa_f(x))) \leq V_f(x) - l(x, \kappa_f(x)) + l(x_s, u_s) \quad \forall x \in \mathbb{X}_f \tag{5.4}$$

Assumptions 1 and 2 holds since all states and inputs are bounded and the stage cost, as defined in Equation 4.2 is continuous. For this implementation, $V_f(\cdot) \equiv 0$, therefore, assumption 3 clearly holds since $x, \kappa_f(x)$ is constrained to x_s, u_s. [5], [4] proves that to ensure this for a time-varying system a reference trajectory $(\tilde{\mathbf{x}}, \tilde{\mathbf{u}})$ that serves as a bases for a meaningful economic performance for the system must be provided. 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. In order to keep the EMPC close to RL's reference trajectory, initial guesses as given in Equation 5.2 with a terminal constraint equal to the last initial guess such that:

$$\begin{aligned}x_{Np}^{k_0} &= \mathbf{e}_{Np}^T \tilde{\mathbf{x}}^{k_0} \\ u_{Np-1}^{k_0} &= \mathbf{e}_{Np-1}^T \tilde{\mathbf{u}}^{k_0}\end{aligned} \tag{5.5}$$

where $\mathbf{e}_{Np+1}$ and $\mathbf{e}_{Np}$ represents the ith standard basis vector in $\mathbb{R}^{Np+1}$ and $\mathbb{R}^{Np}$ respectively. i.e., e_i provides a selection vector to extract the last state and input from the initial guess, to be used as a terminal constraint. Both the last control input and the state must be constrained since the current control action depends on the previous control action (Equation 4.1d).

Implementation 3 Implementation 3 builds upon implementation 2, in that instead of providing a terminal constraint, a terminal region as provided by the actor will be used. The terminal region is defined as:

$$\begin{aligned}(1 - \delta_T)\mathbf{e}_{Np}^T \tilde{\mathbf{x}}^{k_0} \leq x_{Np}^{k_0} \leq (1 + \delta_T)\mathbf{e}_{Np}^T \tilde{\mathbf{x}}^{k_0} \\ (1 - \delta_T)\mathbf{e}_{Np-1}^T \tilde{\mathbf{u}}^{k_0} \leq u_{Np-1}^{k_0} \leq (1 + \delta_T)\mathbf{e}_{Np-1}^T \tilde{\mathbf{u}}^{k_0}\end{aligned} \tag{5.6}$$

[4] 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 5.4. However, [4] suggest that providing a terminal region may be more beneficial than a terminal constraint. Finally, a terminal constraint and initial guesses will also be provided by Equation 5.3 to investigate performance. However, since the action of unrolling from the current state does not result in following a fixed trajectory, no performance guarantee can be made.

Implementation 4 Implementation 4 consists of only including the value function as learned in section 3.6, and initial guesses as 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:

$$\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) - \tilde{V}(s'(k_0 + N_p)) \tag{5.7}$$

where $\tilde{V}$ represents the learned value function and $s'(k_0 + N_p)$ is the normalization of $s(k_0 + N_p)$. For $\tilde{V}^1$,$\tilde{V}^2$,$\tilde{V}^3$, the unnormalized input is Equation 3.1, where as in $\tilde{V}^4$, this is $(y_{k_0+N_p}, k_0 + N_p)$ as discussed in section 3.6. Normalization of the state observation is performed with Equation 3.8. This implementation aims to evaluate the impact of different neural network architectures, including a deep neural network ($\tilde{V}^1$), a smaller deep neural network ($\tilde{V}^2$), a shallow neural network ($\tilde{V}^3$), and a deep neural network trained to learn the value function on only two system states. This can be considered a naive implementation of RL-MPC, and would essentially equate to the rolling out the value function. According to approximate dynamic programming as stated in [6], this policy could be better than the policy that generated the value function. 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 implementation may be conceptually viewed as either unrolling the value function, or providing the MPC knowledge of the future, essentially extending its prediction horizon. Note that the value function is maximized by minimizing the negative of it, since the learned value function represents total return and not cost.

Implementation 5 Implementation 5 essentially combines Implementation 3 and Implementation 4. [4] states that finding an appropriate terminal cost function and corresponding terminal region proves to be non-trivial in order to satisfy assumption 3. This implementation is also claimed to be superior to the terminal constraint of implementation 2 ([4]) under necessary conditions. Therefore the resulting RL-MPC OCP is defined as:

$$\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) - \tilde{V}(s(k_0 + N_p)) \tag{5.8a}$$

$$\text{s.t.} \quad x(k+1) = f(x(k), u(k), d(k), p), \tag{5.8b}$$

$$y(k) = g(x(k+1), p), \tag{5.8c}$$

$$-\delta u \leq u(k) - u(k-1) \leq \delta u, \tag{5.8d}$$

$$u_{\min} \leq u(k) \leq u_{\max}, \tag{5.8e}$$

$$x(k_0) = x_{k_0}. \tag{5.8f}$$

$$\tilde{\mathbf{x}}^{k_0} = [x_{k_{-1}+1}, ..., x_{k_{-1}+N_p}, f(x_{k_{-1}+N_p}, \pi(x_{k_{-1}+N_p}), d_{k_{-1}+Np}, p)]^T \tag{5.8g}$$

$$\tilde{\mathbf{u}}^{k_0} = [u_{k_{-1}+1}, ..., u_{k_{-1}+N_p-1}, \pi(x_{k_{-1}+N_p})]^T \tag{5.8h}$$

$$(1-\delta_T)\mathbf{e}_{Np}^T \tilde{\mathbf{x}}^{k_0} \leq x_{Np}^{k_0} \leq (1+\delta_T)\mathbf{e}_{Np}^T \tilde{\mathbf{x}}^{k_0} \tag{5.8i}$$

$$(1-\delta_T)\mathbf{e}_{Np-1}^T \tilde{\mathbf{u}}^{k_0} \leq u_{Np-1}^{k_0} \leq (1+\delta_T)\mathbf{e}_{Np-1}^T \tilde{\mathbf{u}}^{k_0} \tag{5.8j}$$

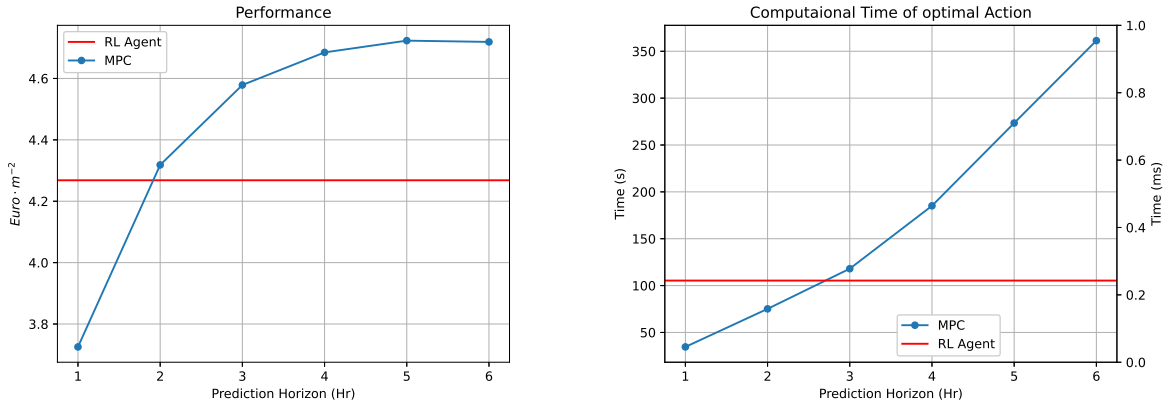
$$\tag{5.8k}$$

This implementation was investigated to determine whether RL might provide both an adequate terminal region and cost function to improve the MPC's performance.

Implementation 6 Implementation 6 serves 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 are compared by evaluating them with the value function. The solution trajectory with the terminal state that yields the most favourable outcome (as 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 2 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 along 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 and MPC policies will be evaluated for the nominal conditions.



(a) MPC vs RL Performance

(b) MPC vs RL compute time

Figure 5.1: MPC vs RL

Figure 5.1 demonstrated the performance of the MPC and RL agent in the nominal setting. Although MPC does perform better for all prediction horizons (except 1 hour) as shown in Figure 5.1a, it does not imply that this is the best RL policy obtainable. A more extensive hyper parameter tuning may have to be forgone in order to achieve a policy that outperforms MPC. However the RL agent is clearly competitive and as seen in Figure 5.1b, can compute actions significantly faster. It is evident that the RL agent can be utilized 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 analyzed in following sections and compared to the baseline performances, as depicted in Figure 5.1.

5.2. Results - Implementations 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.

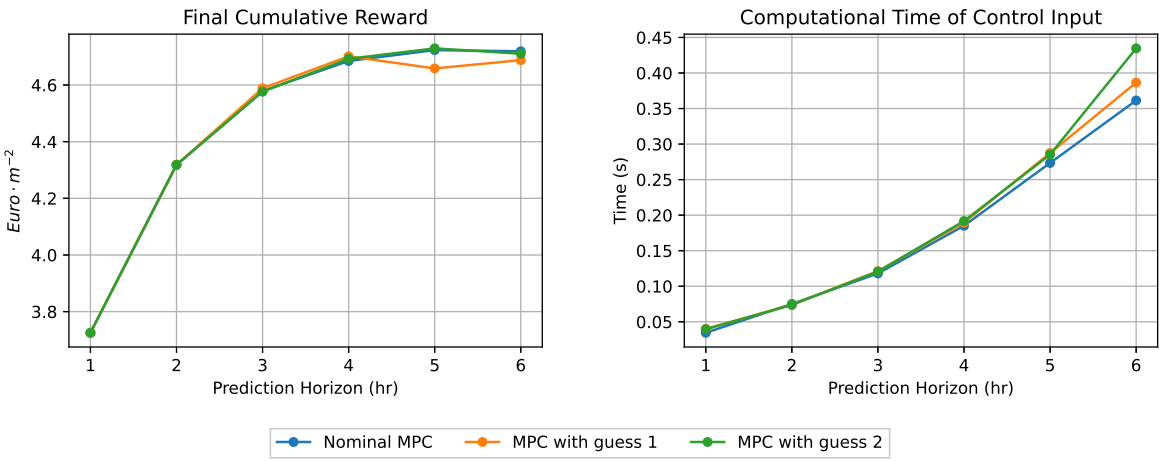


Figure 5.2: MPC vs MPC with initial guesses

Figure 5.2 presents the results of passing in initial guesses from the actor. Unrolling the actor from the current state and using the resulting state and control inputs as initial guesses (Equation 5.3) seems to have very little impact on the final cumulative reward, however it noticeable 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 has minimal impact on the final cumulative reward for shorter prediction horizons, and seems 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 as compared to the nominal MPC. Additionally, computational also time increases. It would be expected that computational time decreases for both initial guesses. A reason for the sub-optimal performance may be due the initial guesses of the Lagrangian multipliers (the found Lagrangian multipliers 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 optimal initial guesses becomes more significant as the prediction horizon is extended, as 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 - Implementations 2

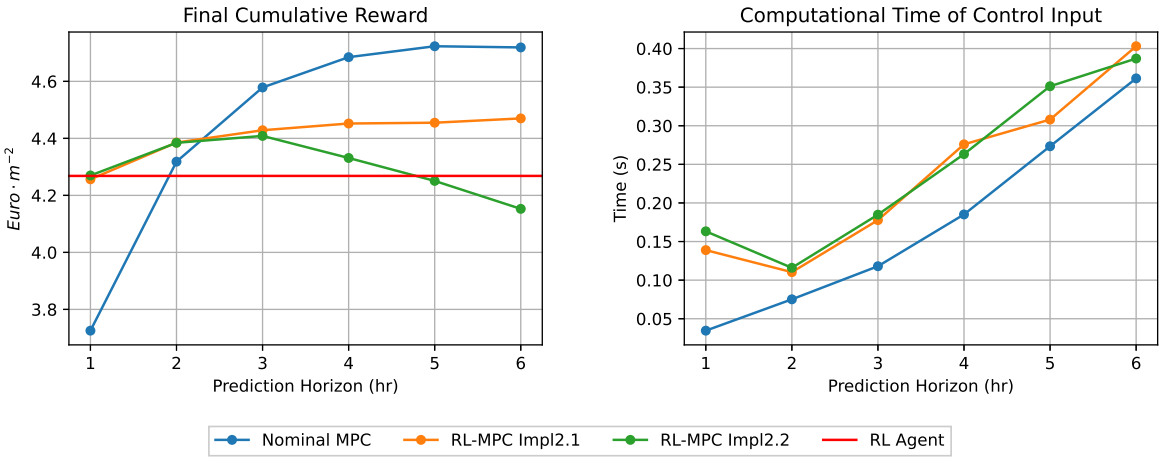


Figure 5.3: MPC with terminal constraints from agent

In order to guarantee that the RL-MPC policy achieves a performance level that is equal to or better than the RL policy, the approach described in Implementation 2 was used. 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, that 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 compared to the MPC policy when the prediction horizon exceeds 3 hours. Beyond the 3-hour mark, the MPC policy consistently outperforms the RL policy and RL-MPC policy. While the implementation of 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. Again, by extending the horizon for the MPC controller is not guaranteed to perform better under economic optimization, therefore a safer choice may be to implement the terminal constraint as 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 that of reinforcement learning, when longer prediction horizons are considered.

The computational time for the control input significantly increases when the prediction horizon is set to 5 or 6 hours. The terminal constraint may excessively limit the MPC due to the presence of a sub-optimal terminal constraint, particularly when dealing with longer prediction horizons.

5.4. Results - Implementation 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 5.6, with a chosen $\delta = 5\%$, allowing for a 10$ deviation in the terminal state. As claimed in [4], 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 $V(s) \equiv 0$.

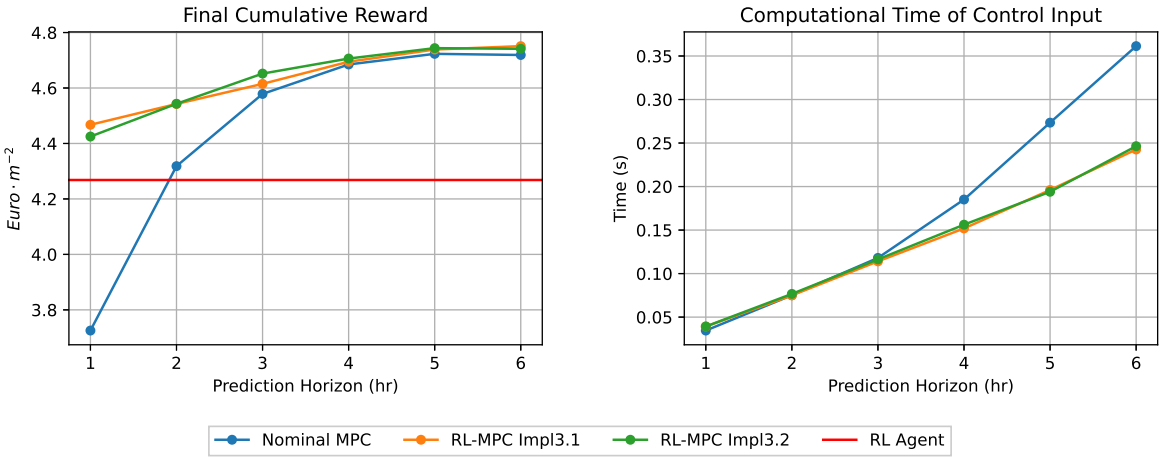


Figure 5.4: MPC with terminal region

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 both the standalone MPC and RL policies. Furthermore, the RL-MPC obtains a marginally superior final cumulative reward in comparison to the MPC even at longer prediction horizons. Lastly, it appears that this performance is increasing monotonically with an increase in prediction horizon, unlike for MPC. This could indicate that an appropriate cost function and terminal region has been found to guarantee performance and stability. However, longer prediction horizons would be required to provide conclusive evidence of this.

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 also providing guidance to the MPC, resulting in reduced computational times.

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 in order to attain optimal economic advantage, performance, and stability.

5.5. Results - Implementation 4

This implementation investigates the effect of a supplying the MPC with a cost function, specifically an approximate value function represented by a neural network.

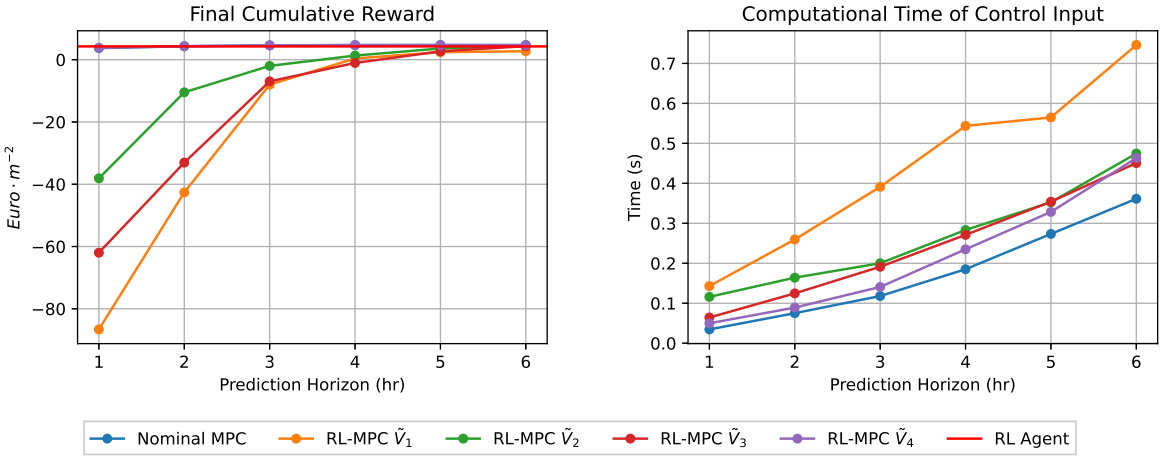


Figure 5.5: MPC with terminal cost function

Figure 5.5 is a naive implementation of merging RL and MPC. However, the outcomes are unsatisfactory in terms of both the overall cumulative reward and the computational time required for generating control inputs. Caution must be exercised when employing a neural network in an optimizer, as they possess a profoundly non-linear nature. As illustrated in Figure 5.5, the performance deteriorates as the neural network becomes more intricate. The neural networks exhibit highly non-convex behaviour that may cause the MPC optimizer to become easily trapped in local optima. The only value function that seems to not destabilize the optimizer is $\tilde{V}_4$ 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, for the other value functions, the MPC needs to optimise across all possible model states and inputs.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 meaningful.
It is natural to observe an increase in computational time in Figure 5.5. Once again, the extremely non-convex nature of the behaviour can disrupt the MPC optimizer and lead to unpredictable computational times. However, it is worth noting that $\tilde{V}_4$ consistently exhibits a predictable computational time, highlighting the significance of obtaining a high-quality value function.

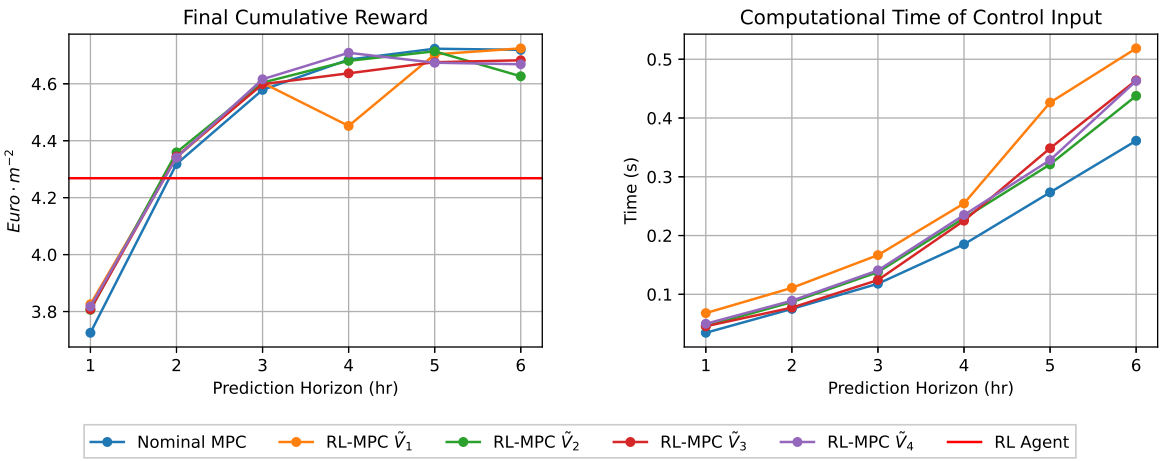


Figure 5.6: MPC with terminal cost function

To minimise the non-linearity of the value functions, the decision variables, except for the drymass, were kept fixed at their initial guesses. This was done because it is evident that the value of a state can be estimated based solely on the current time and the state of the drymass.Thus, the optimisation process was exclusively focused on optimising the dry mass, as was done with $\tilde{V}_4$. The performance of the resulting RL-MPC policy is demonstrated in Figure 5.6. The addition of the value functions improves the final cumulative reward over MPC at shorter prediction horizons but degrades performance for longer prediction horizons (5 and 6hr). Although the value functions have been simplified, increasing the prediction horizon allows the MPC's optimization process to get trapped in local optima, this could be the reason for the observed performance degradation in Figure 5.6. Moreover, the simplification of the value function also resulted in more consistent times, with very little increase as compared to the nominal MPC.

$\tilde{V}_4$ was used for further testing, since it exhibits the greatest performance increase (final cumulative reward) over the nominal MPC (1-4hrs) as compared to the other simplified value functions. The performance of $\tilde{V}_4$ is expected to be superior to the others 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. These results suggests that a value function can increase performance over nominal MPC, but necessarily over the Rl policy that it was derived from. In addition, in order for the value function to be effective, it should have a minimal number of local optima 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, and as 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 - Implementation 5 and 6

Implementation 5 aims to incorporate both a terminal region constraint as well as a terminal cost function. $\tilde{V}_4$ is used as a cost function with a terminal region generated by initial guesses as in Equation 5.2, with a $\delta_T = 5\%$. Implementation 6 solves two problems, each with its own initial guesses and terminal region (Equation 5.2 and Equation 5.3 respectively).It determines the optimal policy by evaluating the terminal state in each solution using the value function.

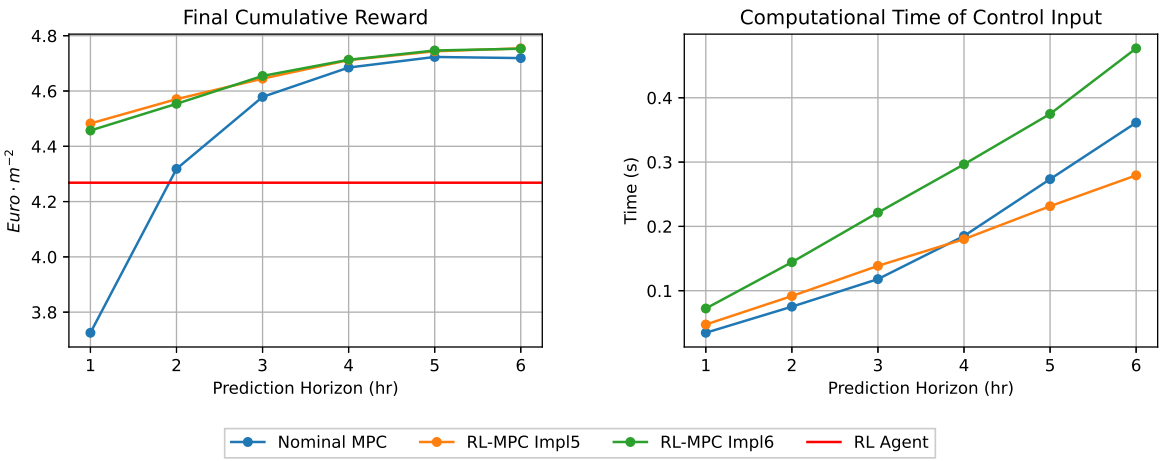


Figure 5.7: MPC vs RL-MPC with terminal region and cost function vs RL-MPC solving two separate problems and evaluating best policy with value function

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 6 is notably higher because it solves two optimisation problems instead of just one. Nevertheless, it is possible to solve the 2 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 majority of this performance increase is due to the terminal region constraint. In addition, this constraint on the terminal region also seems to make the performance increase monotonically, even with the addition of the value function. Furthermore, despite the inclusion of a neural network in the optimisation problem (Implementation 5), the computational time is reduced to a similar level as the standalone MPC when a terminal region constraint is present.

However , whether a value function is necessary when the terminal region

5.7. Final Result and Conclusion

The three best implementations are compared, namely Implementation 3 (with guesses and terminal region provided by Equation 5.2), Implementation 5 and 6.

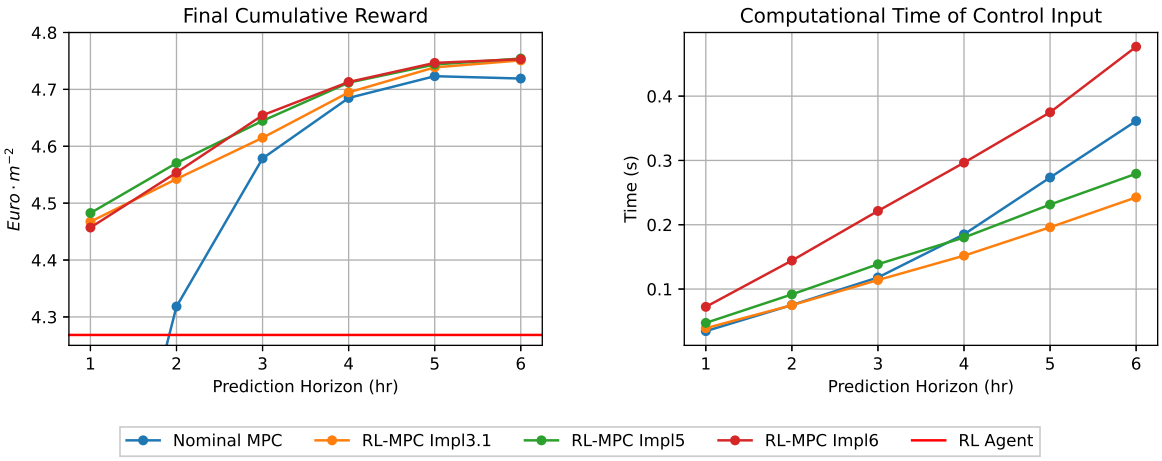


Figure 5.8: MPC final

The final results of the mentioned implementations are shown in Figure 5.8. While the inclusion of a terminal region is the primary factor in improving performance, the incorporation of 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 the necessary assumptions to ensure performance of an EMPC. However, a formal proof would be needed to confirm this conclusively. Moreover, the increase in computational time is not that drastic when a neural network is implemented, especially with the addition of 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 result in destabilizing the system. Therefore any methods in reducing 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 is already so beneficial.

Table 5.2: RL-MPC 3 and 5 vs MPC and RL

	1hr		2hr		3 hr		4 hr		5 hr		6 hr		RL @ 1 hr
	Perf	Time	Perf	Time	Perf	Time	Perf	Time	Perf	Time	Perf	Time	Perf
RL-MPC 3 (%)	19.91	10.91	5.191	2.38	0.79	-5.38	0.21	-8.47	0.328	-16.2	0.67	-18.95	4.67
RL-MPC 5 (%)	20.32	24.26	5.84	18.26	1.44	44.37	0.57	3.68	0.435	2.71	0.746	-7.86	5.02

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 1hr prediction horizon. Incorporating both a terminal region and value function appears to offer 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 optimization 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 be theoretically 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, is may be tempting to use RL-MPC 2 (with terminal constraint) to guarantee performance as good as RL. However, in a stochastic environment, this may not be practically achievable and there are no equivalent performance guarantees available. It may not be feasible because the terminal constraint, as determined by the generated reference trajectory, may not be attainable due to the random nature of the state evolution. Consequently, allowing a terminal region relaxes this constraint and makes the problem computational tractable. 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.

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. Nevertheless, this uncertainty affects the parameters utilised in the computation of the system's outputs (output noise). Consequently, it is customary to utilise a state estimator to mitigate this noise. An approach that is both common and effective for MPC is to utilise 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 pure RL and RL-MPC. 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.

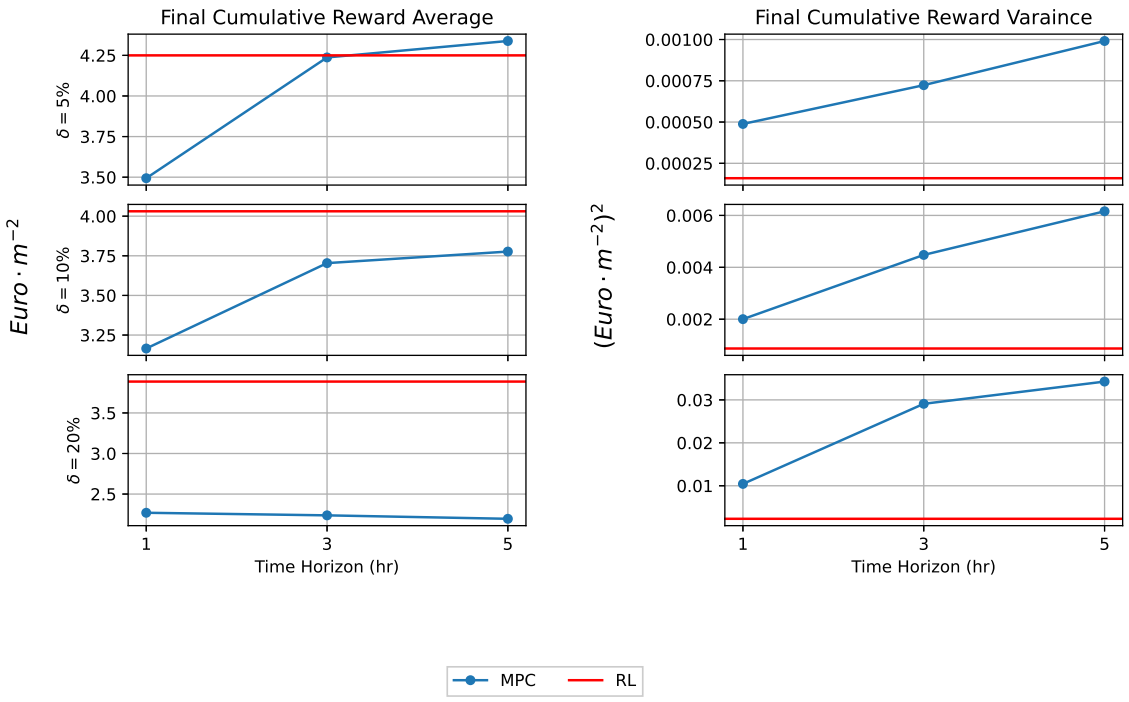


Figure 6.1: RL vs MPC in stochastic conditions

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. 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

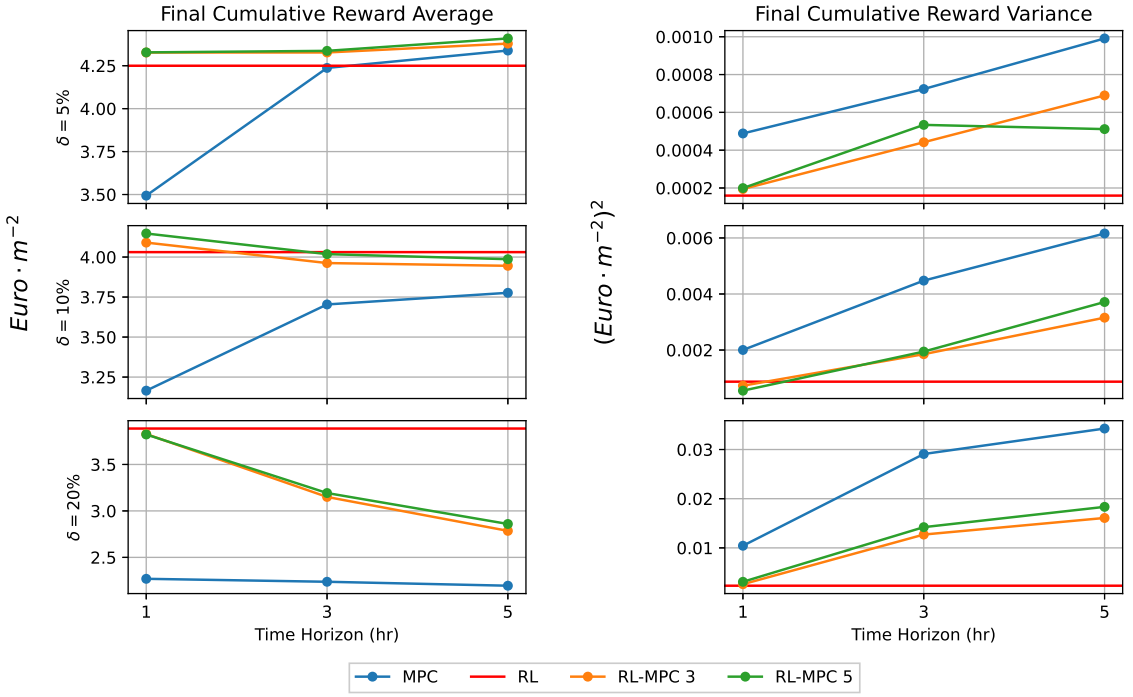


Figure 6.2: MPC vs stochastic RL vs stochastic RL-MPC 3 and RL-MPC 5

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, with the exception of a 1 hr prediction horizon. 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 still outperforming MPC. 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, similar to that of RL. 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, so 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 is indeed highly accurate, 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 a 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 the MPC and RL-MPC, which was considered to be a suitable compromise for this study.

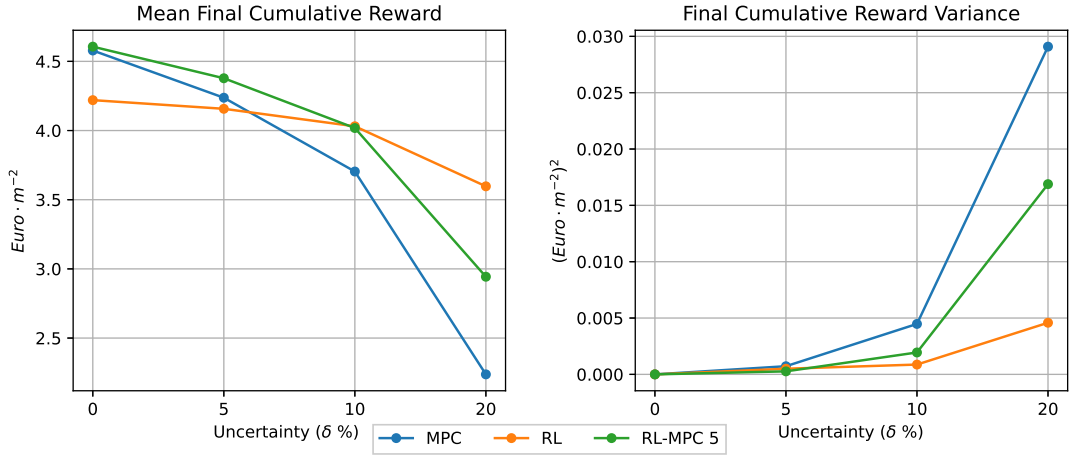


Figure 6.3: MPC vs RL vs RL-MPC 5 using an RL agent trained on $\delta = 10\%$ uncertainty and prediction horizon of 3 hours for both MPC and RL-MPC

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. 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. The RL agent continues to demonstrate superior performance in terms of variance under conditions of increased uncertainty. However, it is evident that RL's capacity to handle uncertainty has been successfully transferred to RL-MPC. 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.

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. 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.

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 balanced prediction horizon, 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. 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 [1]. By implementing this approach, the performance 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 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 high levels of uncertainty, significant performance enhancements can be achieved at low uncertainty levels when the model is well known.

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 on marginal increase in computational time as compared to nominal MPC, 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 was 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 complete complex neural network for the terminal cost function. The aforementioned procedure has previously been performed, and the resulting outcomes were documented in chapter 5. It was determined that $\tilde{V}_4$ can be regarded as a surrogate value function for $\tilde{V}_1$. 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 nominal 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 create 3 deep and 3 shallow neural networks. For both the deep and shallow networks, there 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.

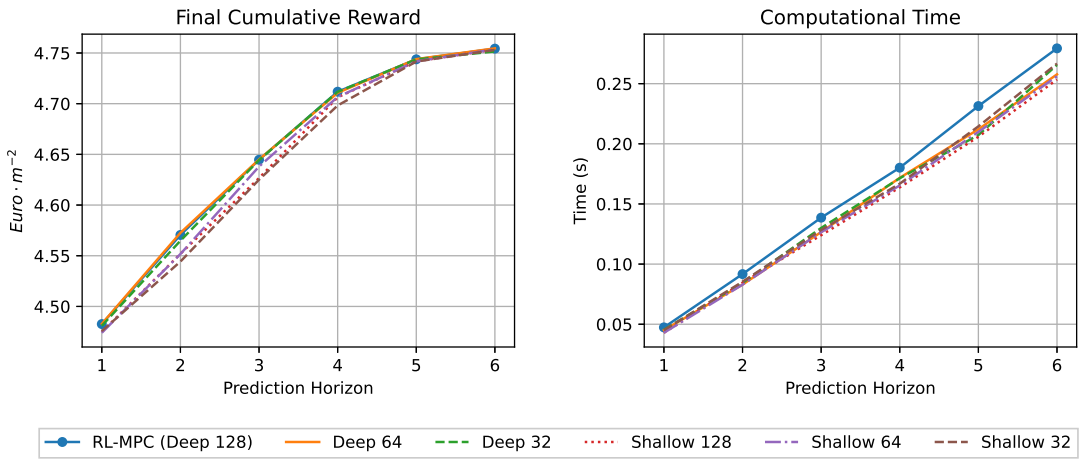


Figure 7.1: Fast RL-MPC with Reduced Neurons

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.

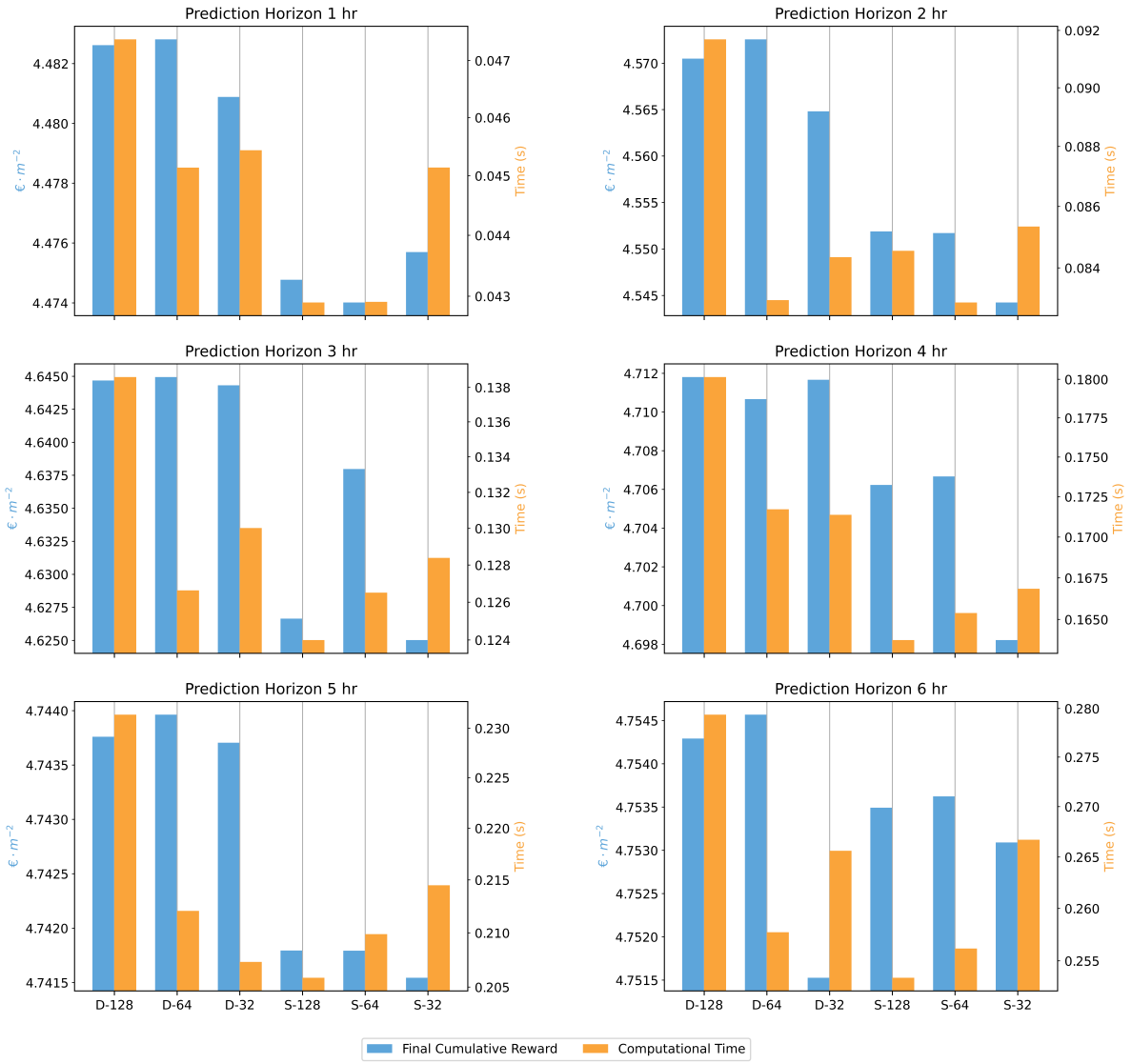


Figure 7.2: RL-MPC with reduced neural network complexity for its terminal cost function. where D-128 would stand for "Deep neural network of 128 Neurons per hidden layer".

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. However, 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 (Equation 5.2) for every time step. The performance and computational time of these approximations were then compared to that of the original neural network when used in the RL-MPC 5 implementation.

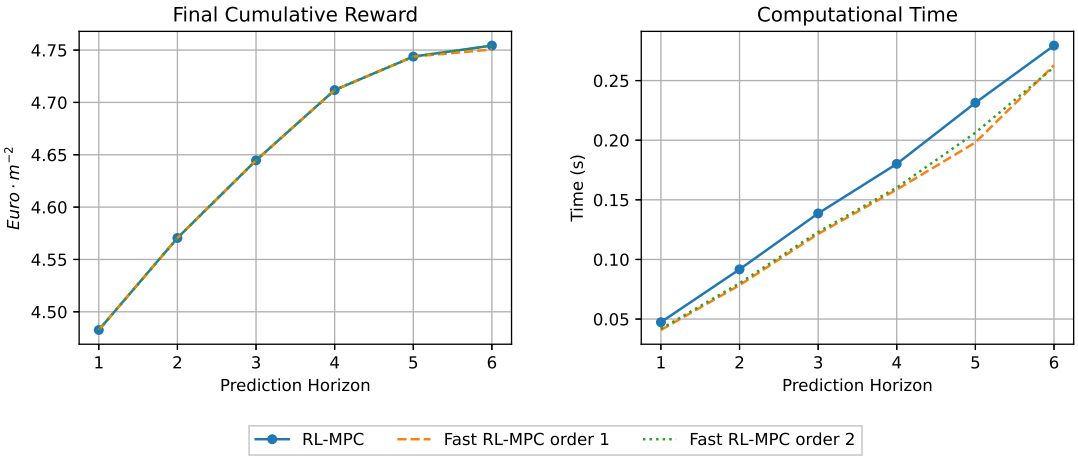


Figure 7.3: Fast RL-MPC with Taylor Expansion

The impact of a linear and quadratic approximation of the neural network around the terminal guess was 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.

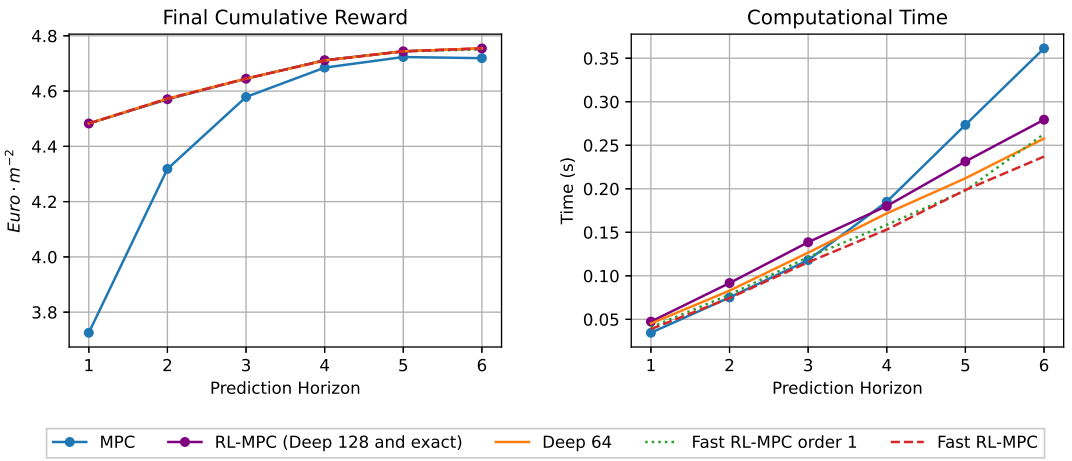


Figure 7.4: Fast RL-MPC

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 combination of the two have zero or little impact on the performance. While using a first order taylor expansion has a greater impact on reducing the computational time than using a smaller deep neural network, the combination seems to slightly decrease computational time even further resulting so much so that the computational cost is the same as the MPC even at lower prediction horizons, specifically for a 1,2 and 3 hr prediction horizon.

7.4. Discussion and Conclusion

something here

8

Discussion and Conclusion

A conclusion...

8.1. Discussion
8.2. Conclusion
8.3. Recommendations & Future Work

References

[1] 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).

[2] 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).

[3] 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).

[4] 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).

[5] 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.

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

A

RL & RL Training

A.1. Selection of RL Algorithm
A.2. Agent Training
A.3. Value Function Training

B

RL-MPC