# RL-MPC for Autonomous Greenhouse Control

Murray Harraway
*Delft Center for Systems and Control*

**The efficient operation of smart greenhouses is essential for enhancing crop yield while minimizing energy costs. This paper investigates a hybrid control strategy that integrates Reinforcement Learning (RL) and Model Predictive Control (MPC) to optimize economic benefits in autonomous greenhouses. While previous research has explored the use of RL and MPC individually, this study examines the effect of modifying MPC's objective function with terminal constraints and a cost function informed by an independently trained RL agent. This approach leverages RL's ability to handle uncertainties and MPC's online optimisation to improve overall control performance in both a deterministic and stochastic environment. Simulation results demonstrate that in a deterministic environment, RL-MPC outperforms both MPC and RL, particularly showing greater performance gains at lower prediction horizons. Additionally, the results indicate that RL trained in an uncertain environment can transfer its understanding of uncertainty into the RL-MPC framework. This integration allows RL-MPC to maintain superior performance in low uncertainty conditions while exhibiting greater robustness compared to MPC under higher uncertainties.**

## I. Introduction

Smart greenhouses are designed to enhance crop yield per hectare using climate-controlled environments [1]. These smart greenhouses are essential in combating the degrading effects of climate change on crop quality and yield. However, efficiently maintaining such an environment requires advanced control methods. These control methods must be able to adjust factors such as temperature, humidity, lighting, and CO2 levels to accommodate ideal conditions for crop growth [2] while keeping energy costs at a minimum. The advent of smart and advanced greenhouses necessitates skilled labor for operation, yet there is a scarcity of qualified personnel [3]. Along with escalating labor costs, the move to autonomous greenhouses is attractive.

Control strategies such as Reinforcement Learning (RL) and Model Predictive Control (MPC) have been extensively used in autonomous greenhouses [1, 4–6]. Both control schemes have advantages and disadvantages. However, there is a notable similarity between the two, suggesting that combining them could lead to a more efficient solution for autonomous greenhouse control.

Several methods seek to integrate the two control schemes, and successful implementations have been demonstrated in [5, 7, 8]. However, very little research exists on combining RL and MPC for optimizing economic benefit. Lubbers [5] investigates using MPC as the function approximator for the RL agent in the context of greenhouse control. While this is one manner of combining them to optimise for economic benefit, this study aims to investigate the effect of modifying MPC's objective function with terminal constraints and a cost function provided by an independently trained RL agent. This approach allows RL to provide knowledge of the uncertainties present and offers future insights beyond MPC's prediction horizon.

### A. MPC and RL

RL and MPC each have distinct strengths and limitations in controlling complex environments. RL aims to learn optimal policies through interactions with the environment, even in the presence of uncertainty. However, RL lacks online flexibility due to its reliance on feed-forward policy evaluation. Additionally, training RL models often requires a large amount of data, which can be laborious.

MPC, on the other hand, relies on an accurate prediction model, which is often simplified for computational

efficiency, potentially leading to sub-optimal control. Despite this, MPC is known for its easier implementation, better constraint handling, and sample efficiency. However, model mismatches and unforeseen uncertainties can significantly degrade MPC's performance. A key aspect of both strategies is their prediction horizon. MPC uses explicit optimization over a finite horizon, which can lead to short-sighted decisions, especially with sparse rewards and slow system dynamics. RL employs a discounted infinite horizon, allowing it to consider long-term rewards and bears similarities to MPC's prediction horizon.

Combining both approaches can mitigate their respective drawbacks. An MPC controller can optimize a short prediction horizon while incorporating future information from RL, particularly useful in contexts like greenhouse dynamics where actions have long-term impacts. Moreover, RL's knowledge of the system's uncertainty is also valuable and can be transferred to the MPC. This hybrid approach leverages MPC's computational efficiency and RL's ability to handle uncertainty and long-term planning, thereby improving overall control performance.

## II. Greenhouse Model

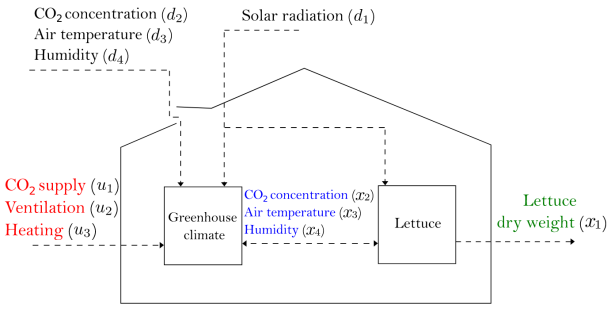


**Figure 1. Graphical representation of greenhouse crop production [9]**

Figure 1 displays a graphical representation of the greenhouse model from the works of van Henten [9] and has been used in previous works[1, 4, 5]. The control inputs are highlighted in red, the states of the indoor climate in blue, the state of the crop in green and the external weather disturbances in black. The model was discretised with the fourth order Runge-Kutta Method with a sample period $\Delta t = 30$ min, resulting in the following system dynamics:

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

with discrete time $k \in Z^{0+}$, state variable $x(k) \in \mathbb{R}^4$, measurement $y(k) \in \mathbb{R}^4$, control input $u(k) \in \mathbb{R}^3$ and weather disturbance $d(k) \in \mathbb{R}^4$. The parameter $p \in \mathbb{R}^{28}$ represents all parameters used in the model, with $f(\cdot)$ and $g(\cdot)$ as the system's non-linear functions. The states of the system, control inputs, disturbances and outputs are described below.

$$\begin{aligned} x(k) &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T \\ u(k) &= \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T \\ d(k) &= \begin{bmatrix} d_1 & d_2 & d_3 & d_4 \end{bmatrix}^T \\ y(k) &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T \end{aligned} \tag{2}$$

The measured output is essentially the same as the state of the system; however, the units of the indoor $C0_2$ density ($y_2$) and relative humidity ($y_4$) differ in that they report in units used in standard measurement sensors.

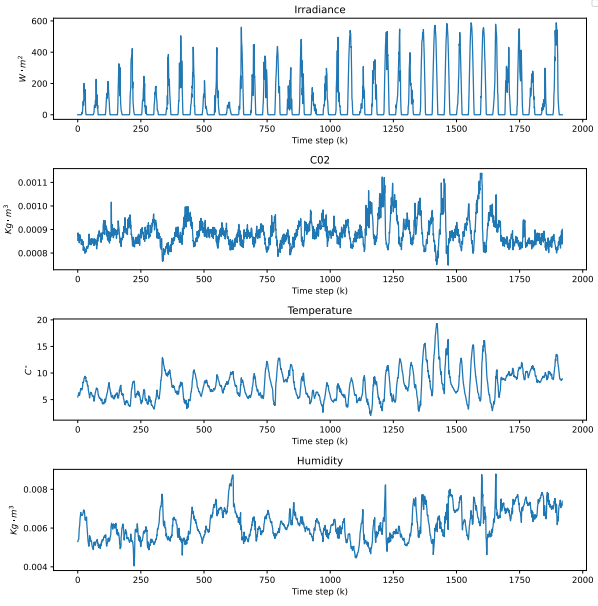


**Figure 2. Weather Data**

The weather data for simulations and training was sourced from the Venlow Greenhouse in Bleiswijk,

covering the period from January 30 to March 11, 2014. The weather is assumed to be deterministic and known at each time step. This dataset, shown in Figure 2, spans a 40-day growing period with a time step of 30 minutes, resulting in a total of 1920 time steps.

### A. Model Uncertainty

As done in [4, 5], the source of uncertainty in the stochastic environment is modeled as parametric uncertainty, aiming to capture all the uncertainties in the greenhouse environment. The parameters of the environment are represented as uncertain with well-defined probabilistic properties. It is assumed that the uncertain parameters, $\hat{p}$, follow a uniform probabilistic distribution:

$$\hat{p} \sim U(\mu_p, \delta) \tag{3}$$

where $\mu_p$ is the mean value of the parameters and $\delta_p$ is expressed as a percentage such that the upper and lower bounds of the distribution is expressed as $p(1-\delta_p)$ and $p(1+\delta_p)$. This was done to guarantee that the sampled model parameters are always non-negative. Since all parameters are perturbed, including parameters of the output measurement equations, output noise is also introduced. Introducing this uncertainty results in the following system dynamics:

$$\begin{aligned} \hat{x}(k+1) &= f(\hat{x}(k), u(k), d(k), \hat{p}(k)) \\ \hat{y}(k) &= g(\hat{x}(k), \hat{p}(k)) \end{aligned} \tag{4}$$

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

### B. Optimization Goal

To directly optimize the economic profit of the greenhouse, it is essential to maximize lettuce size while minimizing resource usage during its growth period, considering the pricing of energy and lettuce. This is referred to as the Economic Profit Indicator (EPI), and optimizing it forms the basis of the following optimization objective:

$$\min_{u_1,u_3} \sum_{k=t_s}^{t_f} (c_{p_1} \cdot u_1(k) + c_{p_2} \cdot u_3(k)) \cdot \Delta t - c_{p_3} \cdot y_1(t_f) \tag{5}$$

where $t_s$ denotes the start time of the growth period, $t_f$ the end time, $\Delta t$ the sample time interval, $c_{p1}, c_{p2}$ the pricing coefficients of injecting $C0_2$, and heating, respectively and $c_{p3}$ as the price of lettuce. These pricing factors are shown in Table 1. It is important to highlight that ventilation, such as opening windows, does not incur any costs and, therefore, does not have a pricing factor. $t_s$ and $t_f$ was selected so that the growing period is fixed at 40 days.

However, this optimization goal makes the reward incredibly sparse, making it difficult to optimize directly with both MPC and RL. Consequently, a stage cost received at every time interval is used, where the total sum of these stage costs throughout the growing period corresponds to the optimization objective in Equation 5. Therefore, the optimization goal that RL, MPC, and RL-MPC aim to optimize is:

$$\min_{u_1,u_3} \sum_{k=t_s}^{t_f} [l(y(k), u(k))] \tag{6a}$$

$$\text{s.t.} \quad \hat{x}(k+1) = f(\hat{x}(k), u(k), d(k), \hat{p}(k)), \tag{6b}$$

$$\hat{y}(k) = g(\hat{x}(k+1), \hat{p}), \tag{6c}$$

$$-\delta u_{max} \leq u(k) - u(k-1) \leq \delta u_{max}, \tag{6d}$$

$$u_{\min} \leq u(k) \leq u_{\max}, \tag{6e}$$

$$x(k_0) = x_{k_0}. \tag{6f}$$

where

$$l(y(k), u(k)) = (c_{p_1} \cdot u_1(k) + c_{p_2} \cdot u_3(k)) \cdot \Delta t - c_{p_3} \cdot (y_1(k) - y_1(k-1)) \tag{7}$$

where the constraints are detailed below:

$$\begin{aligned} u_{min} &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \\ u_{max} &= \begin{pmatrix} 1.2 & 7.5 & 150 \end{pmatrix}^T \\ \delta u_{max} &= \frac{1}{10} u_{max} \\ y_{min} &= \begin{pmatrix} 0 & 500 & 10 & 0 \end{pmatrix}^T \\ y_{max} &= \begin{pmatrix} \infty & 1600 & 20 & 100 \end{pmatrix}^T \end{aligned} \tag{8}$$

The values in Equation 8 were obtained from typical horticultural practices and previous works [1, 4, 6].

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

**Table 1. Pricing factors**

## III. Problem Formulation

### A. RL

The Soft-Actor-Critic (SAC) algorithm is used in developing the Rl agent and is effective due to its ability to balance exploration and exploitation, handle continuous state and action spaces, and provide stable and efficient learning. SAC leverages both policy gradient methods and value function approximation, incorporating a value function to guide policy improvement and entropy regularization to encourage exploration. Moreover, such an actor-critic method provides both a policy (actor, $\pi(\cdot)$) and a critic (value function, $V_\pi(\cdot)$), that can be incorporated into the MPC's objective function. Interested readers are referred to [10] to learn how $\pi(\cdot)$ and $V_\pi(\cdot)$ are learned. It is noted that both the actor and critic are represented by neural networks.

*1. Agent Description*

The agent's description is influenced by its observation tuple (agent's state), its discount factor and reward function. The model of the environment is the same for all controllers used, i.e. RL, MPC and RL-MPC.

$$\begin{aligned} s(k) = (&y_1(k), y_2(k), y_3(k), y_4(k), \\ &u_1(k-1), u_2(k-1), u_3(k-1), k, d(k)) \end{aligned} \quad (9)$$

The agent's state, Equation 9, includes all the measurement outputs, the previous control inputs, the current time and weather disturbances.

The reward function is modelled after the optimisation goal, as defined in Equation 6. State constraints cannot be directly imposed but can be indirectly incorporated by a penalty function in the reward function. It is common practice to impose a linear penalty function for state violations when learning a policy with RL. Therefore, the resulting reward function becomes:

$$\begin{aligned} R(k) = -(&c_{p_1} \cdot (u_1(k)) + c_{p_2} \cdot (u_3(k))) + \\ &c_{p_3} \cdot (y_1(k) - y_1(k-1)) \\ -(&P_{c02} \cdot (y_2(k)) + P_T \cdot (y_3(k)) + P_H \cdot (y_4(k))) \end{aligned} \quad (10)$$

where the penalty terms $P_{c02}, P_T, P_H$ are defined in Equation 11:

$$\begin{aligned} P_{\mathrm{CO2}} &= \begin{cases} c_{p_{\mathrm{CO2}}} \cdot (y_2(k) - y_2^{\max}) & \text{if } y_2(k) > y_2^{\max}, \\ c_{p_{\mathrm{CO2}}} \cdot (y_2^{\min} - y_2(k)) & \text{if } y_2(k) < y_2^{\min}, \\ 0 & \text{otherwise} \end{cases} \\ P_T &= \begin{cases} c_{p_{T_{ub}}} \cdot (y_3(k) - y_3^{\max}) & \text{if } y_3(k) > y_3^{\max}, \\ c_{p_{T_{lb}}} \cdot (y_3^{\min} - y_3(k)) & \text{if } y_3(k) < y_3^{\min}, \\ 0 & \text{otherwise} \end{cases} \\ P_H &= \begin{cases} c_{p_H} \cdot (y_4(k) - y_4^{\max}) & \text{if } y_4(k) > y_4^{\max}, \\ c_{p_H} \cdot (y_4^{\min} - y_4(k)) & \text{if } y_4(k) < y_4^{\min}, \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (11)$$

The penalty constants $c_{p_{CO_2}}, c_{p_{T_{ub}}}, c_{p_{T_{lb}}}, c_{p_H}$ were found empirically in Jansen [6] to effectively account for deviations from desired states and their impact on the economic benefit. The penalty constants and their respective units are displayed in Table 2.

To accommodate for the control input constraints (Equation 6e, Equation 6d), the actor ($\pi(\cdot)$) has a continuous action space, denoted as $\mathcal{A}$, and is defined as $\mathcal{A} = [-1, 1]^3$, where $\mathcal{A} \subseteq \mathbb{R}^3$. The agent's action, denoted as $a(k) = \pi(s(k))$, where $a \in \mathcal{A}$, is regarded as a modification to the control input. Consequently, the current control input can be determined as follows:

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

where $\delta u_{max}(k), u_{min}, u_{max}$ are defined in Equation 8.

| Parameter | Value | Units |
|---|---|---|
| $c_{pCO2}$ | $\frac{10^{-3}}{20}$ | $€\cdot(ppm \cdot m^2)^{-1}$ |
| $c_{pT_{ub}}$ | $\frac{1}{200}$ | $€\cdot(C^\circ \cdot m^2)^{-1}$ |
| $c_{pT_{lb}}$ | $\frac{1}{300}$ | $€\cdot(C^\circ \cdot m^2)^{-1}$ |
| $c_{pH}$ | $\frac{1}{50}$ | $€\cdot(RH_\% \cdot m^2)^{-1}$ |
| $y_2^{max}$ | 1600 | ppm |
| $y_2^{min}$ | 500 | ppm |
| $y_3^{max}$ | 20 | $C^\circ$ |
| $y_3^{min}$ | 10 | $C^\circ$ |
| $y_4^{max}$ | 100 | $RH_\%$ |
| $y_4^{min}$ | 0 | $RH_\%$ |

**Table 2. Penalty Constants**

*2. Agent Training*

Stable baselines 3 (SB3) [11] was used to facilitate the development and training of the SAC algorithm. The final hyper-parameters and actor-critic network structures are posted in Table 3 and were found empirically. The defaults provided by SB3 are used for hyper-parameters that are not reported. All random generators in the experiments were seeded with a value of 4 to ensure reproducibility of the results.

| Parameter | Value |
|---|---|
| **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 |
| **Discount Factor** | 0.95 |
| **Activation Function** | ReLu |

**Table 3. Hyper-parameters**

In RL problems with a fixed, long-term horizon, a discount factor ($\gamma$) of 1 is often desired. This setting ensures that the RL agent's prediction horizon covers the entire growing period, allowing the critic to evaluate and retain information about the full state trajectory and the actor to make long-term decisions. Consequently, the critic can provide accurate value estimations based on long-term outcomes.
However, using $\gamma = 1$ can make it challenging to find a competitive policy. To address this, a lower discount factor was employed, which facilitated the discovery of a more effective policy. Nevertheless, this adjustment means that the critic's value estimates are based on discounted returns rather than the true long-term expected return.

To reconcile these issues, we trained a separate critic with $\gamma = 1$ on the fixed policy obtained through SAC. This approach leverages the advantages of a practical policy while ensuring that the critic provides accurate value estimations over the entire trajectory.

*3. Value Function Learning*

In order to train a value function, multiple trajectories are sampled and for each state visited, the expected return is calculated as per Equation 12.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T = \sum_{k=0}^{T} R_{t+k+1} \tag{12}$$

All initial states and inputs were uniformly sampled around a region of the nominal trajectory at time $k$. Initial states and inputs were sampled from $\hat{\mathbb{X}}^4$ and $\hat{\mathbb{U}}^3$ and the initial time step $k$ is uniformly sampled across the entire time horizon as shown in Equation 13.

$$\begin{aligned}
&k \sim U(t_s, t_f)\\
\hat{\mathbb{X}}^4 = \{&(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4) \mid \hat{x}_1 \in [\hat{x}_{1\,\min}(x_{1_k}), \hat{x}_{1\,\max}(x_{1_k})],\\
&\hat{x}_2 \in [\hat{x}_{2\,\min}(x_{2_k}), \hat{x}_{2\,\max}(x_{2_k})],\\
&\hat{x}_3 \in [\hat{x}_{3\,\min}(x_{3_k}), \hat{x}_{3\,\max}(x_{3_k})],\\
&\hat{x}_4 \in [\hat{x}_{4\,\min}(x_{4_k}), \hat{x}_{4\,\max}(x_{4_k})]\}\\
\hat{\mathbb{U}}^3 = \{&(\hat{u}_1, \hat{u}_2, \hat{u}_3) \mid \hat{u}_1 \in [\hat{u}_{1\,\min}(u_{1_k}), \hat{u}_{1\,\max}(u_{1_k})],\\
&\hat{u}_2 \in [\hat{u}_{2\,\min}(u_{2_k}), \hat{u}_{2\,\max}(u_{2_k})],\\
&\hat{u}_3 \in [\hat{u}_{3\,\min}(u_{3_k}), \hat{u}_{3\,\max}(u_{3_k})]\}
\end{aligned} \tag{13}$$

where the minimum and maximum of the sampled state space for a specific state and input are calculated as per Equation 14

$$\begin{aligned}\hat{x}_{i\,\min} &= x_{i_k} \cdot (1-\sigma), \hat{u}_{i\,\min} = u_{i_k} \cdot (1-\sigma)\\ \hat{x}_{i\,\max} &= x_{i_k} \cdot (1+\sigma), \hat{u}_{i\,\max} = u_{i_k} \cdot (1+\sigma)\end{aligned} \tag{14}$$

where $x_{i_k}$ and $u_{i_k}$ represents the nominal trajectories of the states and inputs respectively. $\sigma$ denotes the desired spread of sampled initial states around the nominal trajectory, which is expressed as a percentage. Given that the actor was found to compute a control action in $0.2ms$, 1000 trajectories were sampled to achieve appropriate coverage of state and input spaces. A neural network was trained with inputs as the state, $s_k$, and labels as the total return,$TR$ and the loss function in Equation 15 is minimised with the Adam optimiser:

$$L(\phi, \mathcal{D}) = V_\phi(s_k) - \mathbb{E}(TR) \tag{15}$$

where $V_\phi$ is the function approximator with weights $\phi$ and $G_t(s_k)$ is the total return of state $s_k$. Parameters include a 2 hidden layer network with 128 neurons per layer, a $1 \cdot 10^{-3}$ learning rate and a batch size of 1024 trained on 200 epochs. Initially, input, $s(k)$, for the value function was the same as that used by the RL agent (Equation 9). However, it was discovered that the expected return of a state could be reasonably estimated using only the current dry mass and time. This simplification also reduced the non-linearity of the neural network, making it suitable for use in the MPC formulation. While training on the full state of the agent allowed for more accurate predictions of the expected return, the increased non-linearity of the neural network adversely affected the MPC optimizer when integrated as a cost function.

### B. MPC

Similarly to the RL agent, the MPC aims to optimize the objective function defined in Equation 6. To facilitate direct comparisons between RL, MPC, and RL-MPC, the same linear penalty constraints, as described in Equation 11, were used in the form of slack variables. Therefore, the following optimization goal is solved at every time step:

$$\begin{aligned}
&\min_{u(k),x(k)} \sum_{k=k_0}^{k_0+N_p-1} \left[ l(y(k),u(k)) + \sum_{i=1}^{6} s_i(k) \right] \\
&\text{s.t.} \quad x(k+1) = f(x(k),u(k),d(k),\mu_p),\\
&y(k) = g(x(k+1),\mu_p),\\
&-\delta u_{max} \le u(k) - u(k-1) \le \delta u_{max},\\
&u_{\min} \le u(k) \le u_{\max},\\
&x(k_0) = x_{k_0},\\
&s_i(k) \ge 0,\\
&s_1(k) \ge c_{p_{C02}} \cdot (y_2^{min} - y_2(k)),\\
&s_2(k) \ge c_{p_{C02}} \cdot (y_2(k) + y_2^{max}),\\
&s_3(k) \ge c_{p_{T_{lb}}} \cdot (y_3^{min} - y_3(k)),\\
&s_4(k) \ge c_{p_{T_{ub}}} \cdot (y_3(k) + y_3^{max}),\\
&s_5(k) \ge c_{p_H} \cdot (y_4^{min} - y_4(k)),\\
&s_6(k) \ge c_{p_H} \cdot (y_4(k) + y_4^{max}).
\end{aligned} \tag{16}$$

The policy generated by the MPC is denoted $\kappa(x,u,d,p)$ where the optimal control action to take at time $k$ is

$$u_k^* = \kappa(x_k, u_{k-1}^*, d_k, \mu_p) \tag{17}$$

The open-source software CasADi [12] and solver IPOPT [13] are used in Python to solve Equation 16.

### C. RL-MPC

According to Ellis et al. [14] and Amrit et al. [15], an EMPC without a terminal constraint and terminal cost function faces challenges in proving performance and stability guarantees. Specifically, Ellis et al. [14] asserts that a terminal point constraint is necessary to ensure closed-loop performance. Additionally, Amrit et al. [15] argues that a terminal cost function with a terminal region constraint is superior to a terminal point constraint, as it enlarges the feasible set of initial conditions and may enhance closed-loop performance. However, identifying suitable terminal constraints and cost functions is challenging. This study aims to determine whether an RL agent can provide these elements. Furthermore, from an RL perspective, the learned value function is only an approximation. When used in MPC, this value function is effectively

unrolled, executing value iterations, which can lead to an improved policy compared to the original policy that generated the value function.

The solution to the OCP in Equation 16 at time $k$ can be denoted as:

$$\begin{aligned}\mathbf{x}_{k|k} &= [x_{k|k}, x_{k_+1|k}, x_{k+2|k}, ..., x_{k+N_p|k}]^T \\ \mathbf{u}_{k|k} &= [u_{k|k}, u_{k+1|k}, ..., u_{k+N_p-1|k}]^T\end{aligned} \tag{18}$$

Where the initial guess generated by the RL agent at time $k$ is denoted as:

$$\begin{aligned}\tilde{\mathbf{x}}_{k|k} &= [\tilde{x}_{k|k}, \tilde{x}_{k+1|k}, ..., \tilde{x}_{k+N_p|k}]^T \\ \tilde{\mathbf{u}}_{k|k} &= [\tilde{u}_{k|k}, \tilde{u}_{k+1|k}, ..., \tilde{u}_{k+N_p-1|k}]^T\end{aligned} \tag{19}$$

such that

$$\begin{aligned}&\tilde{\mathbf{x}}_{k|k} = [\mathbf{x}_{k|k-1}, \\ &f(x_{k-1+N_p|k-1}, \pi(x_{k-1+N_p|k-1}), d_{k+Np|k}, p)]^T \\ &\tilde{\mathbf{u}}_{k|k} = [\mathbf{u}_{k|k-1}, \pi(x_{k-1+N_p|k-1})]^T\end{aligned} \tag{20}$$

Equation 20 takes the previous time steps solution, shifts it in time and uses the policy $\pi(\cdot)$, as provided by the actor, to calculate the optimal action and resulting state to take at the last time step. This method can be interpreted as extending the MPC's horizon. It must be noted that, for the first time step, $k = 0$, the intial guess is generated by unrolling the RL policy for the entire horizon. Based on prior analyses, generating these initial guesses is extremely fast. Furthermore, since it comes from a policy comparable to the MPC's which optimises the same goal, these initial guesses can be used for more than just initial guesses, but also to generate terminal constraints. The terminal region is defined as:

$$\begin{aligned}&(1-\epsilon)\tilde{x}_{k+Np|k} \leq x_{k+Np|k} \leq (1+\epsilon)\tilde{x}_{k+Np|k} \\ &(1-\epsilon)\tilde{u}_{k+Np-1|k} \leq u_{k+Np-1|k} \leq (1+\epsilon)\tilde{u}_{k+Np-1|k}\end{aligned} \tag{21}$$

The terminal cost function takes the form of the value function learned $V_\phi$ and therefore, the resulting RL-MPC OCP is defined as:

$$\begin{aligned}\min_{u(k),x(k)} \quad & \sum_{k=k_0}^{k_0+N_p-1} l(u(k), y(k)) - V_\phi(y_1(k_0+N_p), k_0+N_p) \\ \text{s.t.} \quad & x(k+1) = f(x(k), u(k), d(k), p), \\ & y(k) = g(x(k+1), p), \\ & -\delta u \leq u(k) - u(k-1) \leq \delta u, \\ & u_{\min} \leq u(k) \leq u_{\max}, \\ & x(k_0) = x_{k_0}, \\ & \tilde{\mathbf{x}}_{k|k} = [\mathbf{x}_{k|k-1}, f(x_{k-1+N_p|k-1}, \pi(x_{k-1+N_p|k-1}), d_{k+Np|k}, p)] \\ & \tilde{\mathbf{u}}_{k|k} = [\mathbf{u}_{k|k-1}, \pi(x_{k-1+N_p|k-1})]^T, \\ & (1-\epsilon)\tilde{x}_{k+Np|k} \leq x_{k+Np|k} \leq (1+\epsilon)\tilde{x}_{k+Np|k}, \\ & (1-\epsilon)\tilde{u}_{k+Np-1|k} \leq u_{k+Np-1|k} \leq (1+\epsilon)\tilde{u}_{k+Np-1|k}.\end{aligned} \tag{22}$$

This implementation was investigated to determine whether RL could provide an adequate terminal region and cost function to improve the original EMPC's performance, thereby producing a high-performing EMPC, also known as RL-MPC. The goal was to achieve these improvements for shorter prediction horizons and to transfer knowledge of the system's uncertainty to enhance the EMPC's ability to handle such uncertainties. In conjuction with Casadi and IPOPT, the open-source software L4Casadi [16, 17] was used to develop the RL-MPC framework and solve Equation 22.

## IV. Simulation Results

The simulation results are categorized into two components: the nominal (deterministic) environment and the stochastic environment.

In a nominal environment with no parameter deviation ($\delta = 0\%$), RL, MPC, and RL-MPC algorithms were simulated and trained using a deterministic prediction model($\hat{p}(k) = \mu_p$). This setup ensures that repeated simulations yield identical results. The nominal environment was crucial for developing the RL-MPC algorithm and evaluating the effectiveness of RL in providing useful terminal constraints and cost functions. Additionally, it served as the basis for generating the nominal trajectories for states and inputs.

In the stochastic environment, we test three levels of uncertainty: $\delta = 5\%$, 10%, and 20%. For these cases,

MPC and RL-MPC continue to use a deterministic prediction model. However, three RL agents are trained, each specifically for one level of uncertainty. The stochastic environment is used to evaluate whether RL can impart its knowledge of the system's uncertainty to the EMPC, enabling the resulting RL-MPC algorithm to better handle the present uncertainty. Although the RL-MPC formulation relies on a deterministic prediction model, it leverages the RL agents' knowledge of uncertainties. For each level of uncertainty, the RL-MPC's initial guesses, terminal constraints, and terminal cost function are derived from the RL agent trained for that specific uncertainty level. Finally, prediction horizons of 1-6 hours were tested for the MPC and RL-MPC. Lastly, modification to the RL-MPC is made to ease the computational burden and tested against the original RL-MPC. For all simulations it was decided to have $\epsilon = 5\%$.

*1. Performance Metrics*

The performance metrics include the final cumulative reward achieved over the simulation period. This reward is the sum of stage costs, which encompasses Equation 6, along with penalties for state and output violations. In other words, the sum of stage costs of Equation 10 or Equation 16. Performance metrics for the nominal case will also include the computational time of each algorithm. For the stochastic setting, the performance metrics are detailed by including the mean final cumulative reward, averaged across 30 simulation runs, as well as the variance of the final cumulative reward.

To note, time series plots of the evolution of the system's states and inputs are not shown, since they are similar to those posted in [1, 4, 6].

### A. Deterministic

Figure 3 displays the performance and the computational time of the RL, MPC and the RL-MPC algorithms with various prediction horizons. It is clear that MPC is not myopic as previously thought. It noticeable outperforms the RL agent, with increased performance as prediction horizons increases, however; naturally, the computational time also increases. One could argue that it might not be necessary to introduce terminal constraints and a cost function from RL since performance of the EMPC is satisfactory.


**Figure 3. Comparison of RL, MPC and RL-MPC on the nominal environment**

However, introducing these constraints and cost function boosts performance further across all prediction horizons, and significantly so at lower prediction horizons. Although the addition of a neural network as a cost function increases the computational time, it was found that the addition of the terminal constraints significantly lowers it to the point where the complete RL-MPC algorithm is faster the original MPC at higher prediction horizons. At lower prediction horizons, specifically 1 and 2 hours, the trade off between increased performance and increased computational costs is clearly in favour of increased performance. It is important to note that these terminal constraints and cost function are provided by an RL policy that performs considerably worse than MPC; however, they still visibly improve the RL-MPC performance. This demonstrates the importance of providing an EMPC with appropriate terminal constraints and cost functions to achieve superior performance.

Lastly, one could argue that since the computational speeds are relatively fast compared to the control time interval (sub-second computational speeds versus a 30-minute interval), the prediction horizon can be increased until performance requirements are met. However, this model of the greenhouse system is relatively simple. More complex models may not offer the same flexibility in extending the prediction horizon.

### B. Stochastic

Stochastic results are presented in Figure 4 and Figure 5 Figure 4 presents the findings of RL, MPC, and RL-MPC in a stochastic environment with varying levels of uncertainty across 1, 3, and 5-hour prediction horizons. It is evident that as uncertainty increases, RL outperforms MPC due to RL's ability to learn and manage uncertainty. Moreover, increasing the

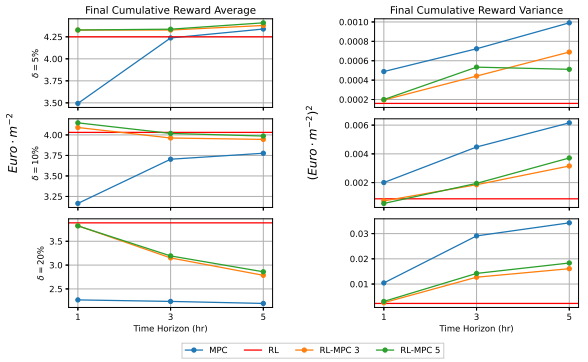


**Figure 4. Comparison of RL, MPC and RL-MPC across all prediction horizons**

prediction horizon for MPC under higher uncertainty yields diminishing performance returns. The variance of RL is significantly lower than that of MPC, with the difference increasing as uncertainty rises.
In the case of RL-MPC, similar to the nominal scenario, at low uncertainties, RL-MPC outperforms both MPC and RL in terms of final mean cumulative reward. For all levels of uncertainty, RL-MPC noticeably outperforms MPC in both cumulative reward and variance across all prediction horizons. This demonstrates that the terminal constraints and cost function provided by RL impart knowledge of the uncertainty in the environment. Although RL still outperforms RL-MPC at higher levels of uncertainty, RL-MPC performs substantially better than the original EMPC.

Often, a conservative estimate of the uncertainty in the environment is used to develop a controller. Hence, Figure 5 includes the performance of an RL agent trained on data with $\delta_p = 10\%$, the RL-MPC algorithm that uses this agent, and the original MPC tested across three different uncertainty levels. The prediction horizon for the MPC and RL-MPC was fixed at 3 hours.

The superiority of the RL-MPC controller over both MPC and RL is illustrated in Figure 5, particularly regarding mean final cumulative reward at low uncertainty levels. This trend persists until uncertainty reaches a threshold where MPC's performance noticeably declines, which negatively affects the RL-MPC's performance. Despite this, RL-MPC consistently outperforms MPC, showing a less severe performance drop as uncertainty increases. The RL agent remains

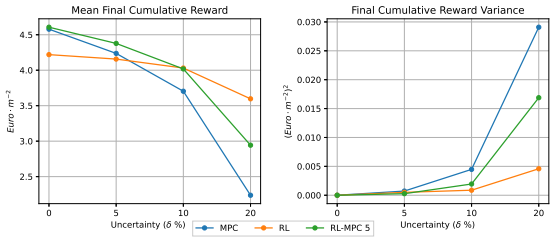


**Figure 5. Comparison of RL, MPC and RL-MPC developed at $\delta_p = 10\%$**

the best performer under high uncertainty. Figure 5 suggests that equipping an EMPC with terminal constraints and a cost function generated by an RL agent trained on stochastic data enhances the EMPC's understanding and management of system uncertainty. Essentially, this approach makes the EMPC more robust against noise without sacrificing performance at lower uncertainty levels, and even enhances it.

### C. Speedup

Although the original MPC and RL-MPC offer real-time computational speeds across all prediction horizons, RL-MPC may not be real-time for more complex systems due to the added computational burden from the neural network used as a cost function. Simplifying the neural network architecture and using approximations can help address this issue. While it was found that reducing the network size can improve computational speeds without performance loss, the focus here is on using Taylor approximations of the neural network around the terminal guess provided by RL. The results of this approach are shown in Figure 6.

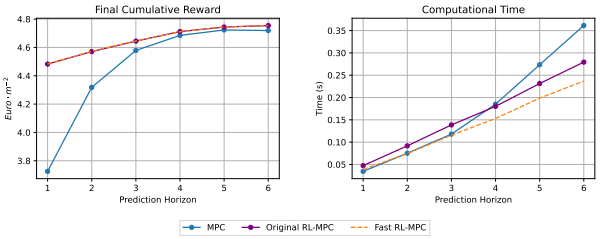


**Figure 6. Fast RL-MPC**

It was found that a second-order Taylor approximation did not provide any advantage over a first-order approximation and hence not shown here. Figure 6 illustrates that the first-order Taylor approximation results in a

substantial increase in computational speeds without sacrificing performance, making the RL-MPC algorithm as fast as MPC at shorter prediction horizons. This approach is both simple and highly effective for making the RL-MPC algorithm faster.

## V. Conclusion

This study has demonstrated the efficacy of integrating RL and MPC for autonomous greenhouse control in optimsing economic benefit. The hybrid RL-MPC approach leverages the strengths of both methods, combining RL's ability to handle uncertainties and long-term planning with MPC's performance and online optimisation.

The study shows that while MPC alone can provide satisfactory performance in a deterministic environment, the addition of RL-derived terminal constraints and cost functions significantly enhances performance, especially in shorter prediction horizons. This hybrid approach not only boosts economic benefits but also improves computational efficiency at higher prediction horizons making it a viable solution for real-time applications in more complex systems.

Furthermore, the RL-MPC framework proves robust against various levels of environmental uncertainty, maintaining high performance where the traditional EMPC's efficacy diminishes. The integration of RL's knowledge of system uncertainties into the MPC framework enhances the overall control strategy, ensuring more reliable and optimal performance in dynamic and unpredictable environments.

Overall, the RL-MPC approach presents a promising advancement in the field of autonomous greenhouse control, providing a balanced solution that addresses both performance and computational challenges. Future research involves the following:

- Further investigation into the hyperparameters of the RL agent may be necessary to develop an RL policy that outperforms MPC in deterministic environments. Other deep RL techniques, such as TRPO, PPO, and TD3, can be employed to develop a more effective RL policy. It is hypothesized that an increase in the RL policy's performance will result in an increase in the RL-MPC's performance. Additionally, exploring different non-linear function approximators, such as radial basis functions or Gaussian processes, could also improve value function learning while optimizing the computational burden in the RL-MPC framework.
- In the presence of uncertainty, RL outperforms MPC because it has access to information about the uncertainty in the environment, whereas MPC does not. This approach aims to determine whether RL can transfer its knowledge of this uncertainty into the MPC framework through a terminal region constraint and a value function. However, estimation techniques such as Moving Horizon Estimation (MHE) are commonly used in MPC to mitigate noise in the output. Moreover, stochastic MPC controllers, such as those described in [4], target parametric uncertainty. When these techniques are employed, one can expect a significant increase in the performance of MPC in stochastic environments. Therefore, a study should be conducted where the RL-MPC framework incorporates these techniques to examine the resulting performance gains.
- Finally, a theoretical foundation should be established to implement this approach in a real greenhouse, ensuring that the combination of RL and MPC will yield performance guarantees. This study served as an initial investigation into the expected performance gains and the resulting benefits of the RL-MPC framework.

## References


[1] Morcego, B., Yin, W., Boersma, S., van Henten, E., Puig, V., and Sun, C., "Reinforcement Learning Versus Model Predictive Control on Greenhouse Climate Control," , Mar. 2023.

[2] DevOps, "Greenhouse Climate Control – How to Improve Plant Growth - DryGair," , Aug. 2021.

[3] Rusnak, P., "What Is the Current State of Labor in the Greenhouse Industry?" , Nov. 2018.

[4] Boersma, S., Sun, C., and van Mourik, S., "Robust Sample-Based Model Predictive Control of a Greenhouse System with Parametric Uncertainty," *IFAC-PapersOnLine*, Vol. 55, No. 32, 2022, pp. 177–182. doi: 10.1016/j.ifacol.2022.11.135.

[5] Lubbers, S., "Autonomous Greenhouse Climate Control with Q-learning Using ENMPC as a Function Approximator," 2023.

[6] Jansen, Y., "Optimal Control of Lettuce Greenhouse Horticulture Using Model-Free Reinforcement Learning," Master's Thesis, 2023.

[7] Arroyo, J., Manna, C., Spiessens, F., and Helsen,

L., "Reinforced Model Predictive Control (RL-MPC) for Building Energy Management," *Applied Energy*, Vol. 309, 2022, p. 118346. doi: 10.1016/j.apenergy.2021.118346.

[8] Lin, M., Sun, Z., Xia, Y., and Zhang, J., "Reinforcement Learning-Based Model Predictive Control for Discrete-Time Systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023, pp. 1–13. doi: 10.1109/TNNLS.2023.3273590.

[9] van Henten, E. J., *Greenhouse climate management: an optical control approach*, 1994.

[10] "Soft Actor-Critic — Spinning Up Documentation," https://spinningup.openai.com/en/latest/algorithms/sac.html, ????

[11] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, No. 268, 2021, pp. 1–8.

[12] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M., "CasADi: A Software Framework for Nonlinear Optimization and Optimal Control," *Mathematical Programming Computation*, Vol. 11, No. 1, 2019, pp. 1–36. doi: 10.1007/s12532-018-0139-4.

[13] Wächter, A., and Biegler, L. T., "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, Vol. 106, No. 1, 2006, pp. 25–57. doi: 10.1007/s10107-004-0559-y.

[14] Ellis, M., Durand, H., and Christofides, P. D., "A Tutorial Review of Economic Model Predictive Control Methods," *Journal of Process Control*, Vol. 24, No. 8, 2014, pp. 1156–1178. doi: 10.1016/j.jprocont.2014.03.010.

[15] Amrit, R., Rawlings, J. B., and Angeli, D., "Economic Optimization Using Model Predictive Control with a Terminal Cost," *Annual Reviews in Control*, Vol. 2, No. 35, 2011, pp. 178–186. doi: 10.1016/j.arcontrol.2011.10.011.

[16] Salzmann, T., Arrizabalaga, J., Andersson, J., Pavone, M., and Ryll, M., "Learning for CasADi: Data-driven Models in Numerical Optimization," , Dec. 2023. doi: 10.48550/arXiv.2312.05873.

[17] Salzmann, T., Kaufmann, E., Arrizabalaga, J., Pavone, M., Scaramuzza, D., and Ryll, M., "Real-Time Neural-MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms," *IEEE Robotics and Automation Letters*, Vol. 8, No. 4, 2023, pp. 2397–2404. doi: 10.1109/LRA.2023.3246839.