Robust Deep Reinforcement Learning for Greenhouse Control and Crop Yield Optimization

van den Bemd, W.J.G.M.

Award date:
2022

Department of Mathematics and Computer Science
Data Mining Research Group

Robust Deep Reinforcement Learning for Greenhouse Control and Crop Yield Optimization

Master Thesis

Wouter van den Bemd
(0948482)

Supervisors:
Dr. ir. Joaquin Vanschoren
Dr. ir. Albert van Breemen
Prof. dr. Jakob de Vlieg
MSc. Andrei Simion-Constantinescu

Eindhoven, Apr 2022

Abstract

In this study we compare adaptations of two state of the art reinforcement learning algorithms, PPO and SAC, to optimize grower profits in our own simulated hydroponics lettuce greenhouse. We have shown that the application of physics randomization significantly boosts the worst case performance of both SAC and PPO. In our analysis, we trained the algorithms in a single greenhouse and evaluated it in many different greenhouses. While the bare version of PPO and SAC perform better than our baselines when evaluated in the greenhouse it was trained on we observed severe performance degradation when the evaluation greenhouse was altered. To solve this, we compared three different methods. We tested RARL: Robust Adversarial Reinforcement Learning, physics randomization and Gaussian observation noise as three adaptations for PPO and SAC. We have shown empirically that reinforcement learning agents using physics randomization perform better than the baselines in the greenhouse it was trained on. The adapted algorithms show improved performance in the worst case, maintaining performance in the average case, and improved performance in the best case.

Preface

This work is the final result of my graduation research for the master program Data Science in Engineering at Eindhoven University of Technology. I would like to take this opportunity to express my gratitude to those who have supported me throughout the process.

Firstly, I would like to thank Joaquin Vanschoren for his supervision during the process. I appreciate the opportunity I obtained to do this research at the Data Mining group. I would like to thank Andrei Simion-Constantinescu for providing helpful tips and support during weekly meetings.

Secondly, I would like to thank the VBTI team for the great time I had at the company. Under the lead of Albert van Breemen, VBTI is a wonderful company with excellent people. I would like to thank Albert not only for the fresh ideas and a critical view, but also for having me at VBTI. I will remember this as a time of hard work, but also great fun.

Thanks to all my fellow teammates who worked hard with me on the Autonomous Greenhouse Challenge: Mickey Beurskens, Albert van Breemen, Carlo Lepelaars and Koen Botermans. It was fun working with you on the challenge, especially the night shift.

Special thanks to Mickey Beurskens and Illya Kaynov, who provided me with the most useful tips and feedback whenever I was in doubt. It was truly a pleasure working with you.

Finally, special thanks to Lisa Klompers and my parents, who unconditionally supported me in a non-technical way.

Contents

List of Figures

List of Symbols

Symbol	Description	Unit
a	Complete action applied on the environment	
a_t	Complete action applied on the environment at time t	
$a_{t,\cdot}$	Single specific property of the action a_t	
$\hat{A}$	Estimated advantage function	
B	Batch of training data	
c_t	Climate state at time t	
d_t	True if the environment its episode is finished at time t	
$\mathcal{D}$	A replay buffer	
EC	Electrical conductivity of a nutrient solution	$mS\ cm^{-1}$
$\hat{\mathbb{E}}_t$	Expected value at time t	
G_t	Cumulative discounted reward	
$H(P)$	Entropy Function in Soft Actor Critic	
i_t	Information state at time t	
L	Rollout fragment length	
$L^{CLIP}(\theta)$	Surrogate objective function in Proximal Policy Optimization	
m	Model component	
M	Minibatch size	
pH	Quantitative measurement for acidity or basicity	
p_t	Plant state at time t	
P	Probability density function	
$P_a(\cdot)$	State transition matrix for action a in a Markov Decision Process	
$Q(\cdot)$	State-action value function that maps state action pairs to its value	
$Q_\theta(\cdot)$	State-action value function as a result of weights θ	
r	Reward	
r_t	Reward at time t	
$R_t(\theta)$	Ratio between two policies as a result of θ	
s	Observable environment state	
s_t	Observable environment state at time t	
$s_{t,\cdot}$	Single specific property of observable environment state s_t	
t	Time in simulation	h
$V(\cdot)$	State value function that maps a state to its value	
$V^\pi(\cdot)$	State value function that maps a state to an estimate of its value	
w_t	Weather state at time t	
x	Random variable	
y	The target network in Soft Actor Critic	

Greek Symbols Symbol	Description	Unit
α	Trade of coefficient between entropy and reward exploitation in Soft Actor Critic	
γ	The discount factor for future rewards	
δ_t	State value function estimation error	
ϵ	Probability of selecting a randomly sampled action by the epsilon greedy method	
θ	Weights for the policy network	
λ	General Advantage Estimation tunable parameter	
μ	The protagonist policy in Robust Adversarial Reinforcement Learning	
υ	The adversarial policy in Robust Adversarial Reinforcement Learning	
ξ	Training environment	
π	A policy	
π^*	The optimal policy for a task	
π_θ	The policy as a result of weights θ	
$\pi_{\theta_{old}}$	The policy as a result of set of weights of a past moment θ_{old}	
τ	The target smoothing operator in Soft Actor Critic	
ϕ	Weights for the Q network in Soft Actor Critic	

Acronyms	Description
A3C	Asynchronous Actor Critic
AGC	Autonomous Greenhouse Challenge
ALE	Atari Learning Environment
CHP	Combined Heat and Power
CI	Confidence Interval
DDQN	Double Deep Q Network
DFT	Deep Flow Technique
DQN	Deep Q Network
GAE	General Advantage Estimation
KL	Kullback Leibler
MDP	Markov Decision Process
MPC	Model Predictive Control
PID	Proportional Integral Derivative Control
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAC	Soft Actor Critic
TD3	Twin Delayed DDPG
TRPO	Trust Region Policy Optimization

List of Tables

Chapter 1

Introduction

The world population is growing. In a recent publication in Nature, a meta-analysis is conducted on various forecasts for the future food demand [75]. The expected food demand increase is between 35 % and 56 % between 2010 and 2050. To keep up with the food demand agricultural land use is also growing [79]. This negatively affects the biodiversity and therefore sustainability [8]. An increase in agricultural land does not hold for all regions, but mostly for the southern hemisphere [79]. In North America and Europe, the amount of agricultural land is reducing each year. In the Netherlands the yield per unit of growing area has shown a steady increase for vegetables grown in greenhouses from 1998 until today, which is shown in Figure 1.1:

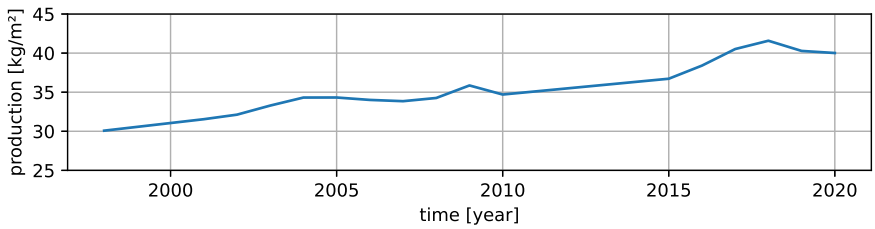


Figure 1.1: Vegetable production in Dutch greenhouses over the years in $kg\ m^{-2}$. Data from [78]

In this Figure, one can see that the spatial efficiency of greenhouses has increased by approximately 33% over 22 years. In 2018 the global vegetable production was 1089 million tonnes [69]. Using the Dutch greenhouse vegetable production density of 40 $kg\ m^{-2}$ one could produce this amount using a growing surface of 27,000 km^2, which is roughly the land surface of the Netherlands. This rough estimate demonstrates the spatial efficiency of greenhouses and their ability to provide enough food for the world.

While spatial efficiency is an advantage, there is also a disadvantage in emissions. In 2020, Dutch energy usage equals 2946.2 PJ [77], which accounts for electricity and fossil fuels. The Dutch greenhouses used 110 PJ [65], accounting for 3.7% of Dutch energy usage. After the Paris Agreement [44] was signed in 2015, the Dutch produced their national climate agreement [53]. In this treaty, it is stated that there should be a 49% percent reduction of greenhouse gas emissions by 2030 and a 95% reduction by 2050. This creates the need for optimizing the greenhouse sector, producing more food with less energy and emissions.

Finally, there is also an economic motivation for growers to adopt novel and more efficient techniques in their greenhouses. It was estimated in 2019 that a typical Dutch grower spends approximately 20.8% of its total turnover on variable energy costs, including gas and electricity [55]. Recently, gas prices have surged due to a global gas crisis, increasing the European gas prices by well over 200% in less than half a year in 2021. This leads to a further decline in grower profits and empathizes with the need for efficient and optimal greenhouse control.

1.1 Recent Developments

In recent years, there have been multiple efforts to optimize greenhouse climate controllers. At the time of writing (2022) Wageningen University & Research is organizing its 3rd Autonomous Greenhouse Challenge (AGC). In this challenge, diverse teams with various backgrounds try to grow a lettuce plant in a hydroponics greenhouse using fully autonomous algorithms. In the first part of the challenge, teams grow lettuce in the virtual environment, and in the second challenge, the best teams can apply the algorithm in a greenhouse to grow real lettuce. The past editions of this challenge have shown that algorithms can outperform human experts. In the first edition, the winning team beat the human expert growers [28] and in the second edition, all participants of the real growing experiment have outperformed human experts [29].

At the start of this research project, we participated in the third edition of the challenge, and we made it to the 7th place out of 16 participants. During this period, we also visited a Dutch hydroponics grower in Almere, where we learned fine details about hydroponics growing and the biology of lettuce. The knowledge obtained in these events has proven its value during this project.

In the first two editions, teams have used various techniques including Model Predictive Control (MPC) and Reinforcement Learning (RL) [28][29]. Sample efficiency was an important parameter, as a single greenhouse simulator was released to each team. Even after the challenge researchers continued to improve the reinforcement learning algorithms [10]. In an independent project, it was also shown that model predictive control can outperform human growers [36].

In fields other than agriculture we have seen a similar shift. In 2016 Google's AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0 [63]. Later this algorithm was refined to AlphaZero [64], which outperforms AlphaGo while it can be applied to a wider range of tasks than Go. Another example is Agent57 [9], which can outperform human-level control in all 57 Atari games in the Arcade Learning Environment (ALE) [13].

1.1.1 Practical Problems

While performance in the simulated games is superhuman, we cannot directly apply these solutions to the real world, as many practical problems come into play. Applying it to a greenhouse is an especially challenging environment for reinforcement learning for the following reasons:

1. With the current sample efficiency it is not feasible to train a reinforcement learning algorithm from scratch on a real greenhouse, as this potentially devastates the harvest of a grower in the early stages. Therefore we have to resort to simulated experiences.

2. Performance obtained in a simulated environment does not guarantee performance in the real environment, due to the simulation to reality gap. The issue of sim-to-real transfer is relevant for multiple real-world applications, including reinforcement learning for robotics [56]. Reinforcement learning learns from experience, if the current state is different from the agent its experience, its behavior is unknown.

3. Not all greenhouses are the same. Construction materials, systems such as combined heat and power (CHP) or ventilation, the local climate, and the crop itself vary widely. Therefore, one model fits all solution becomes nearly impossible to implement.

4. While greenhouse simulation models exist, the most accurate models are only commercially available. A search query on Github [24] for "Greenhouse Simulator" shows 14 public repositories at the time of writing, of which none is designed for reinforcement learning.

5. Points (3) and (4) require large initial investments, both for research and for small companies to start research on the subject. This makes the topic more inaccessible to both groups.

1.2 Problem Statement

In this thesis, our main focus is greenhouse optimization using reinforcement learning and the robustness of the trained model to the sim-to-real transfer. While previous research aims to optimize greenhouse performance in either a simulated or real environment, none of the previous papers addressed the possible performance issues caused by the simulation to reality gap, to the best of our knowledge. There are two categories of solutions to solve this problem, shown in the Figure below.

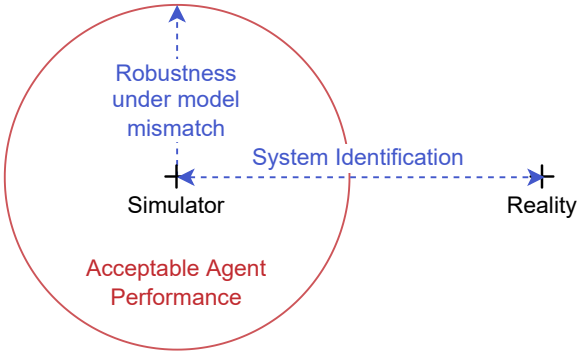


Figure 1.2: Two types of solutions for the simulation to reality gap, system identification and robustness under model mismatch.

Here we illustrated the acceptable agent performance in red. We aim to train an agent with an acceptable performance in reality, which can be achieved using two types of solutions (marked blue in the Figure)

1. **System Identification**
   This solution involves creating a more accurate greenhouse simulation model through experimental data. This brings the simulator and reality closer together to acceptable performance in reality.

2. **Robustness Under Model Mismatch**
   Here the modifications are made to the agent to increase its robustness against a change in environment. The goal is to widen the range of conditions in which the agent operates at an acceptable performance level.

In this thesis, we focus on the second category of solutions for two reasons. Firstly, assuming successful implementation of this technique, the application of reinforcement learning in agriculture becomes much more accessible. This is because less costly experiments and development time are required, as acceptable performance could be achieved with a simulator that is reasonably close to reality, but not a perfect reflection. Secondly, this approach for robustness in reinforcement learning is not necessarily specific to the agriculture domain, and could therefore be beneficial in other domains.

We assume the sim-to-real transfer problem is similar to the model mismatch problem, where an agent is trained and evaluated in two different simulated environments. We do this to prevent the need for an actual greenhouse, while we can still analyze the effect of a gap between a training environment and an evaluation environment.

We narrow the scope of this thesis by focusing primarily on growing lettuce in a hydroponics greenhouse. Benefits of this approach include faster crop growth, higher crop density, and growing the crops all year round [62] [51]. This comes at higher construction costs, as it requires high-end construction materials such as glass and control systems for the nutrient mixture and water temperature. We select lettuce for three reasons: Firstly, according to Sharma et al. [62] lettuce

is one of the most promising species to grow in hydroponics because of its high growth rate and nutrient uptake ability. Secondly, it is a common crop that can be found all across the globe [23] that is produced in large quantities (22.4 million metric tons shipped worldwide in 2006 [66]). Finally, the Autonomous Greenhouse Challenge provided insights into the cultivation of hydroponics lettuce.

1.3 Objective and Main Contributions

To address the most important issues mentioned in the scope of the thesis, we define the following research questions:

1. How can reinforcement learning be applied to maximize grower profit?
2. Is there a difference in robustness of reinforcement learning algorithms compared to traditional control methods?
3. How can we make the performance of a reinforcement learning agent robust to differences between the training and evaluation environment?

To answer the questions above we implemented a greenhouse simulation environment. The derivation of this simulator and its validity is an important part of this project, but it is not the main focus. We assume that this simulation model is accurate enough, as it models the most important effects related to the growth of the plant and the greenhouse climate.

Here the second question is our main concern. During this thesis we make the following novel contributions to the topic:

1. We show how our implementation of PPO with physics randomization not only outperforms traditional climate control methods in average performance, but also worst-case performance.
2. We show how robustness in terms of worst-case performance of both PPO and SAC can significantly be improved through the application of domain randomization.
3. We show the effect of model mismatch on the performance of a traditional rule-based climate controller in two state-of-the-art reinforcement learning algorithms, Soft Actor Critic (SAC) [26] and Proximal Policy Optimization (PPO) (Clipping Variant)[60].
4. We show three methods that improve the robustness of the state-of-the-art algorithms against model mismatch. These techniques include Robust Adversarial Reinforcement Learning [50], physics randomization [47] and Gaussian noise on the observations.
5. We created a parametric greenhouse simulator with standardized interfaces, suitable for simulating many forms of greenhouses, using the standard Open AI Gym interface for reinforcement learning [16]. Additionally, a parametric standard plant model has been defined that can be tuned to match other species.

1.4 Outline

This thesis is structured as follows. In Chapter 2 we explain the relevant background knowledge about growing lettuce in hydroponics, greenhouse control, reinforcement learning, and solutions for robustness under model mismatch. In Chapter 3 we explain the structure and validity of the designed and implemented greenhouse simulation environment. In Chapter 4 we focus on optimizing reinforcement learning algorithms to maximize the greenhouse yield. In Chapter 5 we analyze the sensitivity of the trained models to model mismatch, and we investigate the effectiveness of the three proposed methods. Finally, in Chapter 6 we show our most important findings, discuss the results and provide suggestions for future research.

Chapter 2

Background

In this chapter, the relevant literature will be elaborated. We aim to provide a concise background on the biology of lettuce growing and an overview of recent developments in the autonomous greenhouse field. Secondly, an overview of relevant reinforcement learning concepts, and an explanation of the used algorithms is given. Finally, we explain the background of two relevant techniques for increasing the robustness of an agent under model mismatch.

2.1 The Hydroponics Lettuce Greenhouse

In the first part of this section, we examine the contribution of light, carbon dioxide and temperature to the growth of lettuce, and what factors affect the quality of a lettuce plant. Secondly, we elaborate on the different components of a hydroponics greenhouse.

2.1.1 Growing Lettuce

The process of photosynthesis is fundamental to the growth of larger plants [52]. Photosynthesis converts water, carbon dioxide (CO_2) and light into glucose using the following chemical reaction:

$$12H_2O + light + 6CO_2 \longrightarrow C_6H_{12}O_6 + 6O_2 + 6H_2O \tag{2.1}$$

Through photosynthesis, plants capture the energy of sunlight in chemical energy, which is important for the vital processes in the plant. A plant deprived of sunlight will starve to death, even with adequate fertilization [52]. The three major factors for photosynthesis include:

1. Light
2. Carbon dioxide
3. Temperature

All these factors should be within the right range for photosynthesis. If one of the factors is missing, photosynthesis cannot occur at all. The effect of each of these components on photosynthesis will be elaborated below.

Light
The intensity of light is an important parameter for plant growth. If there is no light, no photosynthesis can occur. The relationship between light intensity and photosynthesis rate is illustrated in Figure 2.1.

Here we can see that photosynthesis rate increases linearly with light intensity up to a certain point. After this point, there are nearly no gains from additional light. The optimum depends on

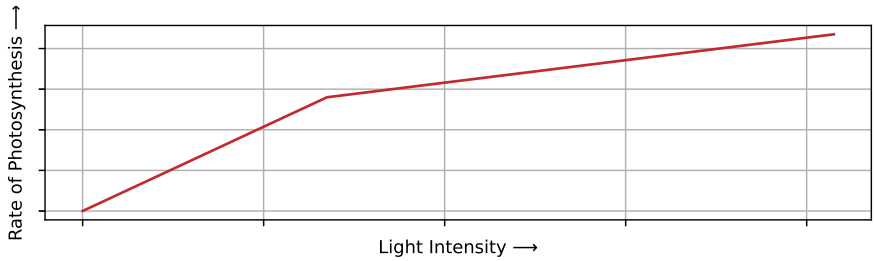


Figure 2.1: Effect of light intensity on the photosynthesis rate, without units. Recreated from [11].

the temperature and carbon dioxide. At low temperatures and low values for carbon dioxide, the optimal light intensity also decreases [11].

While light intensity is important, there is a limit to the amount of hours a plant should receive light each day. The daylight duration can trigger photoperiodic responses in plants, which, depending on the length of the light period, can trigger flowering or other plant processes [52]. Therefore, it is important for the plant to experience a balanced amount of light each day.

Photosynthesis occurs in the chloroplasts, which can capture a specific bandwidth of the visible light for the reaction. This spectral band is referred to as photosynthetically active radiation (PAR) [21][52]. Only photosynthetically active radiation affects the rate of photosynthesis.

The ratio of PAR light to light intensity depends on the light source. A typical greenhouse consists of natural light supplemented by an artificial light source, such as LEDs or HPS lamps [21].

Carbon Dioxide

During the photosynthesis reaction described in Equation 2.1 carbon dioxide inside the greenhouse air is absorbed by the plant. As photosynthesis is vital to plant growth it is important to have a sufficient carbon dioxide concentration. Typically, a higher carbon dioxide concentration increases the photosynthesis rate and therefore the relative production rate of the plant. This is shown in Figure 2.2.

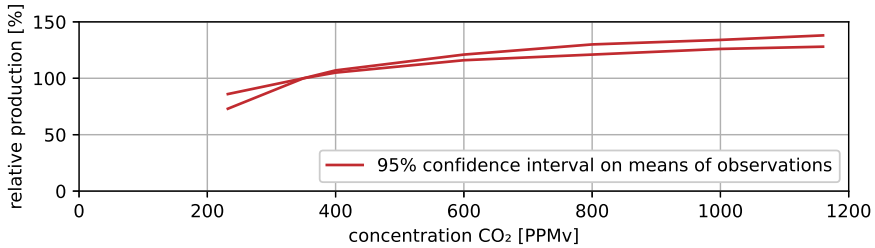


Figure 2.2: Simplified recreation of the work by Nederhof [45]. At the time the outside carbon dioxide concentration was 340 PPM_v. It shows the 95% confidence interval (CI) of the mean relative production change in alternative carbon dioxide concentrations.

By increasing the concentration of carbon, one can improve the relative production rate. We also observe that there are diminishing returns for higher carbon concentration levels. This creates a trade-off for the grower, as the grower needs to balance the costs and environmental effects of carbon dioxide with the relative production increase.

Photosynthesis is not the only way a plant interacts with the outside carbon dioxide level. During its lifetime a plant will continuously emit carbon dioxide due to aerobic respiration, defined by the following chemical reaction [12]:

$$C_6H_{12}O_6 + 6O_2 \longrightarrow 6CO_2 + 6H_2O \tag{2.2}$$

The respiration reaction is the inverse of the photosynthesis reaction. While photosynthesis occurs only under the effect of light, respiration occurs all day. However, respiration occurs at a lower rate than photosynthesis, which causes the plant to absorb more carbon dioxide than emit on average.

Temperature

Temperature has a direct influence on the rate of many chemical reactions, including photosynthesis and respiration [52]. For most plants, the ideal productivity range is between 20 to 30 $°C$. Under normal conditions the photosynthesis rate of the average plant is related to temperature in the following way:

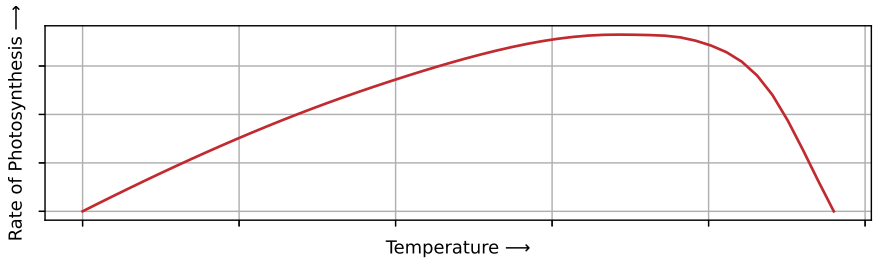


Figure 2.3: Effect of air temperature on the photosynthesis rate. Recreated from [11]

Here we can see a skewed parabolic shape. There exists a single optimum temperature, and deviation from this temperature reduces the rate of photosynthesis of the plant. The reduction of photosynthesis rate is faster for higher temperatures compared to lower temperatures. For lettuce, temperatures up to 30 $°C$ can be tolerated and after that no growth occurs [30].

At temperatures below 0 $°C$ freezing of the water inside the plant may occur, either inside its cells (intracellular freezing) or outside its cells (extracellular freezing). Intracellular freezing causes much more plant injury as the frozen water expands and ruptures the cell, leading to cell death [52].

High temperatures can also induce plant stress. Here the exposure duration and the temperature both influence the amount of stress on the plant. In general, higher temperatures require shorter exposure times before plant death. [52]. In an experiment with 10 different lettuce cultivars, a single cultivar was not able to germinate at 30 $°C$ and at 35 $°C$ none of the cultivars could germinate.

It is also known that acclimatization to hot and cold temperatures can help a plant resist these temperatures better. Also, plants that are actively growing are more susceptible to plant stress from extreme temperatures [52].

Tipburn

Plant injury can also be caused by mineral deficiency due to fast growth under the wrong conditions. An example of this is tipburn, where rapid growth causes the calcium absorbed by the roots cannot be transported to the leaves in time, resulting in browning leaf edges. These brown leaf edges reduce the quality of the plant and may result in a lower selling price. It has been shown that tipburn increases during a long photoperiod with high intensity light, high temperatures and high carbon levels [18].

2.1.2 Common Greenhouse Equipment

In this section, we will elaborate commonly present systems in hydroponic greenhouses. Most commercial greenhouses are equipped with various actuators to regulate the climate, compared to

passive greenhouses where there are no actively controlled systems [51]. The selection of equipment depends on the climate in which the greenhouse is located and which crop is grown. In the Netherlands the temperate climate requires efficient heating and maintaining the heated air [51]. The Venlo type greenhouse in the figure below is a commonly used greenhouse type in Northern Europe.

This greenhouse uses roof openings and is made of glass and steel or aluminum. It has a better thermal performance and high environmental control, with the downside of high construction costs [51]. The high degree of control is due to the rigid construction that divides the outside climate from the inside, and the control equipment inside the greenhouse. An overview of the common equipment in a Venlo type hydroponics greenhouse is shown in the figure below:

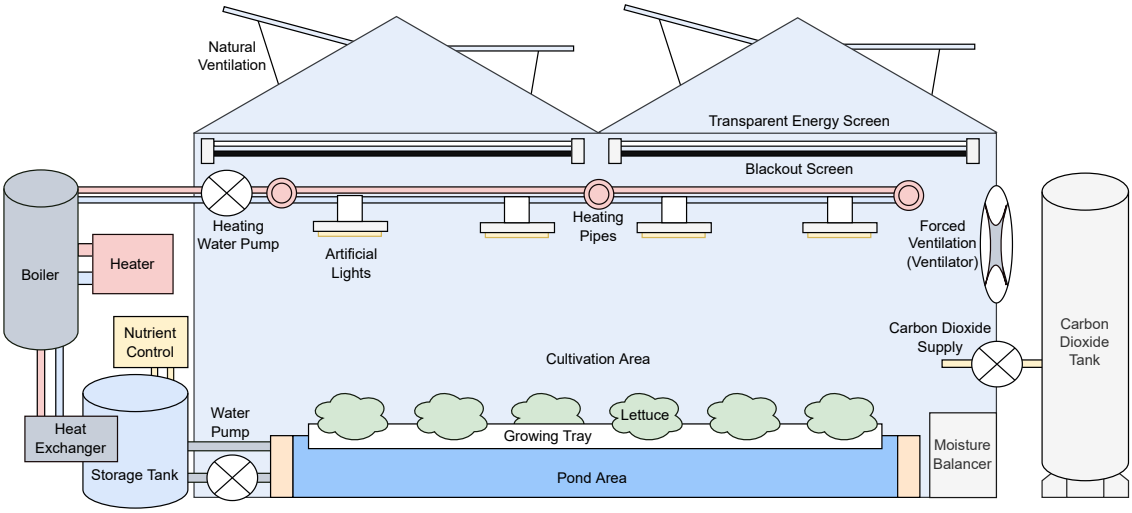


Figure 2.4: An overview of common hydroponic greenhouse components. The selection is based on the descriptions found in the following papers [51][80][61]

.

We will now elaborate each of the components shown above in more detail.

1. **Cultivation Area** - Area where the greenhouse plants are grown, consisting of the pond and the growing trays that contain the lettuce.
2. **Pond Area** - The water basin that defines the hydroponics setup. In the figure, the Deep Flow Technique is shown, where the growing trays float in the water, and the lettuce roots are suspended in the water.
3. **Lettuce** - Any lettuce cultivar suitable for hydroponics growing.
4. **Growing Tray** - The growing tray in which the lettuce plants are spaced with typical densities of 20 to 30 m^{-2} [32]. It contains holes for the roots of the plant to fit through, such that the lettuce leaves are above the water and its roots are suspended in the water.
5. **Storage Tank** - Water storage buffer for the pond area.
6. **Water Pump** - The pump that provides fresh water from the storage tank to the pond area. The excess water from the pond is drained and returned to the storage tank.
7. **Nutrient Controller** - The water temperature needs to be regulated, and the water is continuously monitored to keep the nutrient mixture optimal for growing lettuce. The water also requires cleaning from harmful viruses and bacteria that build up over time. This can be achieved by circulating the water through an UV light [33].
8. **Boiler** - Typically a gas boiler or combined heat and power (CHP) system. It uses gas as a combustible to warm the water in the heat buffer tank. In case of a CHP system the

combustible is used to generate electricity, from which two byproducts are formed: heat to warm the greenhouse and carbon dioxide to increase the photosynthesis rate of the crops. In case of an electricity surplus growers can sell the electricity back to the net [55].

9. **Heat Buffer Tank** - Storage tank that serves as a heat buffer for the heating system. It is insulated to prevent heat loss and increase heating efficiency.
10. **Heating Water Pumps** - Water pumps that determine the flow rate of hot water to the heat exchanger or to the overhead heating pipes. The climate controller controls the flow rate of the pumps.
11. **Overhead Heating Pipes** - These heating pipes distribute the heat contained in the water warmed by the boiler through the greenhouse. The hot water flows through the pipes, and heat is emitted through radiation [51]. The cold water then flows back to the boiler.
12. **Heat Exchanger** - The heat exchanger in the water storage tank creates a large surface area where heat can be transferred from the hot water of the boiler to the water of the storage tank. It typically consists of a large quantity of metal pipes as metal has a high heat transfer rate. The water of the different systems does not mix in the process. The heat exchanger is used to heat the water up to between 24 and 26 $^{\circ}C$ [15].
13. **Forced Ventilation (Ventilator)** - Electric ventilation system controlled by the climate computer to increase homogenization of temperature in the greenhouse [51]. The increased airflow also increases plant transpiration and prevents tipburn due to the greater rate of nutrient transport [15].
14. **Natural Ventilation (Windows)** - Electronically controlled windows located on the roof of the greenhouse. Typically, the ventilation window is automatically controlled using the greenhouse air temperature and the heater setpoint. By setting the opening temperature 2 to 4 $^{\circ}C$ above the heating setpoint, one can control the temperature in the greenhouse to stay within bounds.
15. **Carbon Dioxide Supply Valve** - A set of valves controlled by the climate computer that release carbon dioxide from the carbon dioxide tank into the greenhouse when opened. Carbon dioxide is artificially supplied to improve photosynthesis rate for the plant.
16. **Carbon Dioxide Tank** - A supply tank that contains carbon liquid carbon dioxide gas.
17. **Artificial Lights** - Supplemental artificial lights to increase growth on dark days or during a winter growing cycle. Typical lamp types are high pressure sodium (HPS) or LED [31]. HPS is typical for older greenhouses and produces less light per unit of electricity consumed compared to LEDs, while emitting extra heat.
18. **Blackout Screen** - A black screen near the ceiling of the greenhouse. Can be closed to block sunlight under extreme conditions and increase insulation during the night.
19. **Transparent Energy Screen** - A transparent screen located near the ceiling of the greenhouse. It diffuses sunlight to lower plant stress under extreme sunlight conditions. It can be closed during the night for increased insulation.
20. **Moisture Balancer** - Increases or decreases the humidity in the greenhouse artificially.
21. **Climate Controller** - A computer that handles the sensor input and contains the setpoints for all controlled environmental variables, including air temperature, light level, humidity, nutrient mixture pH, EC-values, water temperature and carbon dioxide levels. To maintain the desired climate an algorithm is used that returns control inputs that are sent to the matching equipment in the greenhouse.

2.1.3 Yield Optimization

In this section we elaborate yield optimization techniques in a state of the art and state-of-the-practice context.

State of the Practice

In [51] the author describes three common methods for greenhouse control. The least advanced is ON-OFF, where employees manually set the valve position for the heating valves or change the ventilation position. This form of control requires the least automation, but skill of the grower. The second and third methods are PID and Fuzzy control. Both methods operate autonomously to control the greenhouse. Here PID minimizes the error to a predefined setpoint and fuzzy logic learns how to achieve optimal conditions.

Regardless of the automated systems the grower has to make many decisions. Growers have contracts with wholesalers that specify what amount of crops will be ready on a specific date. To reach this goal, a grower needs to make decisions on the set points of the systems mentioned above. The decisions the grower has to make are strongly influenced by weather, as warm and sunny weather may produce many plants, while cold and cloudy weather requires artificial lighting and heating for optimal growth.

To aid in these decisions there exist manuals that describe ideal growth temperatures for the plant, ideal nutrient mixtures, and how to prevent or handle disease in plants. A noteworthy example of this is the KWIN [76], or the Complete Guide for Growing Plants Hydroponically [30].

State of the Art

State of the art greenhouse control systems as the ones found in [28][29][36][10][6] have two major components in common. There is no human in the loop and they all apply digital twinning. By applying Digital Twinning one creates a replica of the greenhouse in a simulated virtual environment. This can save costs on experimentation as in a digital environment, experiments can be performed at minimal costs, while real experiments are costly and require time. This makes it a useful method for experimentation and optimization. Digital Twinning is a core part of every participant in the Autonomous Greenhouse Challenge [28] [29]. But also for others that use model predictive control to optimize a greenhouse [36] or reinforcement learning [10].

Greenhouse Simulators

Simulators exist in many forms with many purposes. A noteworthy greenhouse simulator is KASPRO [19], a parametric model that simulates the greenhouse climate. KASPRO can be coupled to INTKAM, a model that simulates the growth of plants inside the greenhouse. This combination results in a complete simulation of the greenhouse climate, the growth of the plant, and the yield for the grower. While this simulator is reputable and applied to optimize real greenhouses, it is only commercially available, and it was not possible to access it for this project.

Open source simulators exist, but most of them do not model all the required aspects of greenhouse control such as heating, ventilation, lamps, growth of the plant and its yield. An example of this is CropGym [48], which simulates fertilization.

A different approach to creating a simulator beforehand is to create a simulator while learning in a real greenhouse. In [6] samples from the real world were stored in a database that was then used to improve the performance of the greenhouse simulator on the go.

2.2 Deep Reinforcement Learning

In this section we briefly touch on the fundamentals of deep reinforcement learning, followed by an explanation of two state of the art algorithms, Soft Actor Critic and Proximal Policy Optimization.

2.2.1 The Reinforcement Learning Problem

In [68] a formal definition for the reinforcement learning is given. In this report we will use the same conventions. In the standard reinforcement learning setting there is an agent and an

environment. The agent is the learner and decision maker, while the environment is what the agent interacts with, giving a response in return.

In this definition, the agent and the environment act at discrete time steps $t \in \{0, 1, .., n\}$. At each time step t the agent receives the current state of the environment s_t and the reward for the last time step r_t. On the basis of s_t and r_t, it decides on an action a_t, which is received by the environment. The environment is subject to action a_t and produces a s_{t+1} and r_{t+1} pair. This iterative process continues until the current episode is finished. The cycle is visualized in Figure 2.5.

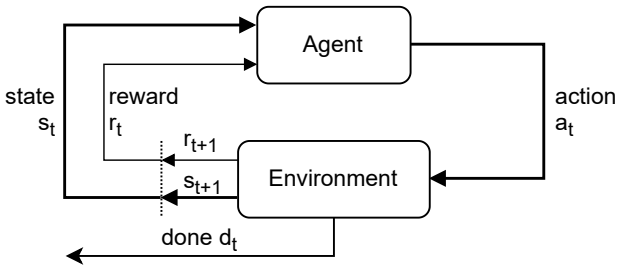


Figure 2.5: The agent-environment interaction, recreated from [68]

The objective of the agent is to maximize the expected future return, which is a measure of total future rewards for all time steps $t \in \{0, 1, ..., n\}$ until termination of the episode. This measure is commonly expressed using the following equation [68]:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.3}$$

Here G_t equals the cumulative discounted reward following t, and γ is the discount factor that determines the importance of rewards obtained in future time steps. Typical ranges for gamma are between 0.9 and 0.995, and for these values the contribution of expected future rewards on G_t decay as time progresses.

Markov Decision Process

The discrete time environment in Figure 2.5 is a Markov Decision Process (MDP). A transition from s_t to s_{t+1} is due to the taken action a_t and the state transition matrix $P_a(s_t, s_{t+1})$ for action a_t. The resulting transition is therefore partially stochastic and partially under control by the decision maker that selects a_t. An example of an MDP is given in Figure 2.6.

The MDP process allows uncertainty in the environment, which is the case for many problems. An example of this is optimizing a policy for the game of blackjack. The cards drawn in blackjack are random, but by taking the optimal policy π^* we achieve the maximum expected reward, which results in the best possible outcome in the long term.

The Markov Property

If the state contains all the information about the state of the environment the Markov property holds. In this case, the optimal policy can always select the best possible chain of actions to maximize the reward as all the required information is available. Regardless of this advantage we should not expect this to always be the case [68]. In the case of blackjack, forming an optimal policy is simple when we know what the next card in the deck is. However, the resulting policy is unfair, as it violates the rules of blackjack.

In the context of growing lettuce, a simulation model could store attributes that are internal to the lettuce, such as nutrients inside the lettuce. We should not encode this information in the state, as it is unrealistic to sense the values of these attributes in a realistic greenhouse. Therefore, we form an environment that violates the Markov property.

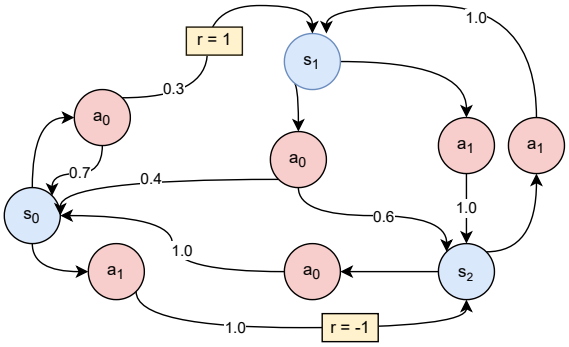


Figure 2.6: An example Markov decision process with 3 states denoted by s, and from each state there are two actions a the agent can pick. Upon selecting an action the outgoing arrows and the associated transition probabilities represent the state transition matrix. On some transitions a reward r is granted.

The Credit Assignment Problem

In an MDP state, transitions may have a reward r associated with them. We can apply positive rewards if the agent performs well, and negative rewards if the agent takes a bad action. Note that well and bad are subjective words open to interpretation. There are many ways an agent can be rewarded for good behavior.

In the blackjack example, we can choose to give a positive reward for winning. While this might work, we can possibly speed up the learning by applying a negative reward for obvious bad actions. The fundamental problem here is assigning a reward that reflects the value of the taken action. Imagine a game of football where the objective is to score more points than the opposing team. We could reward the agent 1 point upon scoring a goal, and -1 when the opponent scores a goal. However, when the first goal is scored at time t, we have taken t actions. For small t, most actions have contributed towards scoring the goal, but for large t we have almost certainly performed bad actions. How can the agent discriminate between when actions are correlated with the reward and when there is a causal relationship between actions and reward [5]?

We could solve this by applying a more immediate reward to the agent. In the ideal case we can quantify the contribution of an action directly to the objective and reward the agent accordingly. Unfortunately, this is not trivial as the contribution of the taken action can be complex. An example can be seen in the football case where an action such as "shoot" sometimes contributes to the reward and sometimes does not. The value of this action depends on the policy of the opponent, the agent's own policy, and the environment rules. As the environment is unaware of both these policies, we cannot possibly define a function that reflects the contribution of the action exactly.

Deep Reinforcement Algorithms

In the past decade, many new discoveries were made in deep reinforcement learning. In this section, we will briefly elaborate on key concepts behind these algorithms, and give an overview of the most important algorithms.

Model Based versus Model Free Methods

There are two families of algorithms in deep reinforcement learning. One is model free and the other is model based. The model free methods obtain experience by interacting with the environment, but they are unaware of the exact definition of the environment itself. In the model based family, the algorithms either learn to replicate the environment or have the exact definition of the algorithm. Where model free methods only interact with an external environment, model based methods have a copy of the environment that can be used to examine possible future scenarios. A well known example of a model based algorithm is Monte Carlo Tree search, which

is applied in AlphaGo to beat human Go champions in 2016 [63].

Policy Optimization versus Q-Learning

In the model free family there are two groups of algorithms, algorithms that directly optimize a policy $\pi_\theta(a|s)$ and algorithms that optimize a state-action value function $Q_\theta(s,a)$ as in Q-learning [2].

In policy optimization the network can be optimized based on the performance objective $J(\pi_\theta)$. Typically, a state value function is included to estimate the value of the current state $V^\pi(s)$. The agent uses this state value function in combination with the reward to estimate the value of the taken action. It can use this estimate to modify the chance to select that action again given state s. The value for a particular state is directly dependent on the policy that determines the course of action. This implies that the state value function can only be optimized using experience obtained directly by the policy.

In Q-learning, a state-action value function $Q_\theta(s,a)$ is learned to evaluate the value of action a given state s. The corresponding learned policy is achieved by selecting the action a in state s for which $Q_\theta(s,a)$ is maximized. In the case of Q-learning, the state-action value function does not depend on the policy, and past experiences can be stored in a replay buffer. As past experiences can be reused for training, this approach typically results in higher sample efficiency. A comparison of recent algorithms shows the difference, in [26] SAC achieves high rewards using less samples than PPO.

Reinforcement Learning Taxonomy

Using the previously mentioned families of algorithms, we can create a taxonomy of algorithms similar to Open AI [2]. The taxonomy in Figure 2.7 contains all the major milestones in reinforcement learning mentioned in this work.

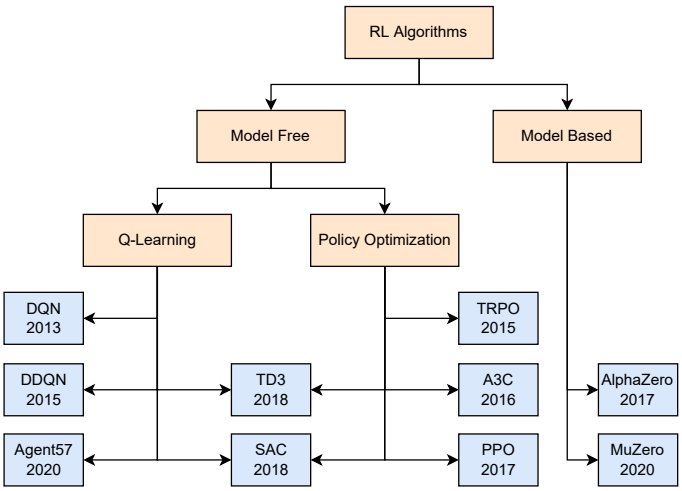


Figure 2.7: The taxonomy of important deep reinforcement learning algorithms of the past decade. In orange we show different categories, and in blue we display abbreviations of algorithms and the year they were published. Inspired by the taxonomy chart of Open AI [2]

In this work, we focus on model free methods, as our goal is to apply reinforcement learning in case the environment is not known, hence our need for robustness under model mismatch.

In 2013 Deep Q-learning was published [41]. In this algorithm, a state-action value function is learned by minimizing the temporal difference error of state transitions stored in the replay buffer. Exploration of the state space is achieved by applying the epsilon greedy algorithm. At every step t there is a probability ϵ that an uniformly sampled action replaces the policy its action. In 2015, the algorithm was improved using the double-Q trick. This trick corrects for the state-action value function overestimation by minimizing the outcome of two learned Q functions [27].

In 2015, Trust Region Policy Optimization (TRPO) was published [58]. It addresses the need to use slight off policy data for training, as this increases sample efficiency. The algorithms update function clips the gradient by bounding the maximum difference between the updated policy and the policy used to gather the training data. The Kullback Leibler (KL) Divergence is applied to calculate the difference between the updated policy and the policy used for training. As the difference in policies is bounded, the difference in state value function becomes negligible, which allows the algorithm to perform multiple update batches.

In 2016 Asynchronous Actor Critic (A3C) was published [40]. In the paper, a novel algorithm was proposed for asynchronous training of existing deep learning methods. It uses a single network shared between multiple workers. Each of the workers gathers experience and updates the globally shared network. Then a synchronization step is performed. As workers update and synchronize to the global network at different moments in time they may have slightly different policies. This method has two advantages. Training time is significantly reduced as the speed at which episodes are completed scales roughly linear with the number of workers used. Secondly, by using multiple workers with slightly different policies, the experience is more diverse which stabilizes training.

In 2017, an iterative improvement was made on TRPO, called Proximal Policy Optimization (PPO) [60]. Proximal Policy Optimization is similar to TRPO, but it applies a first order method to bound the maximum difference between policies. Instead of KL Divergence, PPO bounds the probability of selecting the action a given state s between the new policy and the policy used during training. This makes PPO easier to implement, and it empirically seems to perform similar to TRPO [2].

In 2018 Twin Delayed Deep Deterministic Policy Gradient (TD3) was released [22]. The work was inspired by DDQN and applied a variation of the double-Q trick to address value overestimation in actor critic methods. It combines two state-action value functions with a policy function. It also applies target policy smoothing to prevent deterministic policies from updating the policy using similar state-action value estimates, which causes overfitting and narrow peaks in the value estimate [22].

The resulting algorithm outperformed other actor critic methods in 6 Atari benchmarks.

In the same year, the second version of Soft Actor Critic (SAC) was published [26]. Soft Actor Critic is similar to TD3, with three key differences. Where TD3 uses target policy smoothing by applying random noise to the calculated action to prevent bootstrapping similar experiences in training, SAC uses an entropy regularized objective. The entropy regularized objective stimulates the agent to select low probability actions in the actor network, effectively stimulating exploration. This removes the need for target policy smoothing as in TD3, hence this is not used in SAC. Finally, in SAC, the current policy directly determines the actions, whereas in TD3 the target policy determines the actions [2].

In 2020, Agent57 outperformed humans in all Atari games in the human Atari benchmark on a single set of hyper parameters. Agent57 adjusts the reward by adding an internally generated intrinsic reward for exploration. There are two types of intrinsic rewards: short term novelties within an episode and long term novelties between episodes [9]. Agent57 uses this to train a family of policies from exploratory to exploitative. An adaptive mechanism selects which policy to apply throughout the training process.

On the model based side, we have Alpha Zero from 2017 [64] and Mu Zero [57] from 2019. Both methods are model-based methods that apply MCTS to imagine future scenarios and to choose actions from the environment. Where Alpha Zero uses a given dynamics model depending on the task, Mu Zero eliminates this need for the dynamics model, thus it can be applied to a wide range of problems.

Open Source Frameworks

In the past years, multiple efforts were made to standardize reinforcement learning algorithms, including Open AI Baselines [3], a fork called Stable Baselines [4] and RLLIB [38].

Open AI baselines is a useful repository to understand the implementation of the algorithms and train baselines for comparisons in scientific work. Stable Baselines is a fork that adds documentation and additional algorithms. RLLIB is integrated with Ray [42], which allows the user to

speed up training using multiprocessing. Both Stable Baselines and RLLIB include state of the art reinforcement learning algorithms, including TD3, PPO and SAC. Both allow logging results to Tensorboard [1], a dashboard that visualizes the training performance of a model.

In this work, we will use the algorithms from RLLIB 1.7.10. We chose not to implement the reinforcement learning algorithms ourselves, as our main purpose is not the design and implementation of the algorithm, but to investigate its effectiveness and address the sim-to-real deployment issues.

Selection of algorithms
We base the selection of algorithms on three criteria: The algorithms should be available in RLLIB, the algorithms should perform well, and the algorithms should be commonly applied in other papers. From these criteria, we find that PPO and SAC best match our needs. Both are implemented in RLLIB, they were published in 2017 and 2018 and are commonly presented as baselines in other works. In the following sections, we will elaborate on the details of these algorithms.

2.2.2 Proximal Policy Optimization

[60] As mentioned before, PPO is an on-policy algorithm that can learn from a batch of data gathered by the policy, where the deviation of the updated policy is bounded with respect to the original policy. In this section, we will elaborate the clipping variant of PPO [60]. The first order approximation to determine the distance between two policies is denoted by the following ratio R.

$$R_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}} \tag{2.4}$$

Here $\pi_{\theta_{old}}$ is the policy before the minibatch update, and π_θ is the policy after the latest minibatch update. Note that $R_t(\theta_{old}) = 1$. In PPO, we maximize the following clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[min(R_t(\theta)\hat{A}_t, clip(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{2.5}$$

Here $L^{CLIP}(\theta)$ is the surrogate objective function, $R_t(\theta)$ is the probability ratio defined in the equation above, ϵ is a hyper parameter that defines the maximum distance from the old policy with $\epsilon = 0.2$ and $\hat{A}_t$ is the estimated advantage function. The clip function clips the first argument $R_t(\theta)$ between $1-\epsilon$ and $1+\epsilon$. The advantage function of the previous equation is defined as follows:

$$\begin{aligned} \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{L-t+1}\delta_{L-1} \\ where \quad \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{2.6}$$

Here γ is the discount factor, λ is the general advantage estimation (GAE) parameter, and $V(s_t)$ is the state value function, and L is the roll out fragment length, which is the number of time steps gathered by each actor.

The complete PPO algorithm as described in [60] is shown in Algorithm 1.

2.2.3 Soft Actor Critic

As mentioned previously, Soft Actor Critic (SAC) is an entropy regularized algorithm, in which exploration is part of the objective function. In combination with two value functions and an off-policy replay buffer, SAC can obtain high performance using less experience compared to PPO [26]. In SAC, the exploration rewarding function is the entropy $H(P)$. The entropy is defined in Equation 2.7.

Algorithm 1 PPO, Actor-Critic Style [60].

for *iteration* $= 1, 2, ...$ **do**
  for *actor* $= 1, 2, ..., n$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for L time steps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq n \cdot L$
  $\theta_{old} \leftarrow \theta$

$$H(P) = \mathop{\mathbb{E}}_{x \sim P}[-logP(x)] \tag{2.7}$$

Here x is a random variable with the probability density function P. The lowest possible entropy occurs when the outcome of random variable x is always the same. The highest possible entropy occurs when all outcomes are equally likely. In SAC, the entropy term is directly included in the objective function for the policy, as shown in Equation 2.8.

$$\pi^* = \underset{\pi}{argmax} \mathop{\mathbb{E}}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \Big(r_t + \alpha H(\pi(\cdot|s_t)) \Big)\right] \tag{2.8}$$

Here $\alpha > 0$ is the trade-off coefficient that balances the entropy regularization term and the reward. This term effectively determines the ratio between exploration and exploitation for the algorithm. There are two variations of SAC, one with a fixed value for alpha and one with a separate mechanism that tunes alpha during training. Both methods achieve similar performance.

The policy network is updated using the state value function, in which the entropy term is also included. The modified state value function for SAC is shown in Equation 2.9.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \Big(r(s_t, a, s_{t+1}) + \alpha H(\pi(\cdot|s_t)) \Big) \middle| s_0 = s\right] \tag{2.9}$$

The state-action value function requires a similar modification and is presented in Equation 2.10.

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a, s_{t+1}) + \alpha \sum_{t=1}^{\infty} H(\pi(\cdot|s_t)) \middle| s_0 = s, a_0 = a\right] \tag{2.10}$$

In SAC, the state value function and the state-action value function are connected using Equation 2.11.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[Q^{\pi}(s, a)] + \alpha H(\pi(\cdot|s)) \tag{2.11}$$

The Bellman equation for the state-action value function now becomes the following:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\substack{a' \sim \pi \\ s' \sim P}}[r(s, a, s') + \gamma V^{\pi}(s')] \tag{2.12}$$

The complete algorithm for training SAC is shown in algorithm 2. In this algorithm, $\mathcal{D}$ is the replay buffer, B is a batch of training data, y is the target network, ϕ are the weights for the Q network, τ is the target smoothing coefficient.

Algorithm 2 Soft Actor-Critic [26][2].

for $i = 1, 2, .., n$ **do**
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r and done d
  Store (s,a,r,s',d) in replay buffer $\mathcal{D}$
  if d is True **then**
    Reset environment state
  if Time to update **then**
    for $j = 1, 2, ..., k_{iter}$ **do**
      Sample transition batch $B = (s, a, r, s', d)$ from $\mathcal{D}$
      Compute targets for the Q functions using $\tilde{a}' \sim \pi_\theta(\cdot|s')$:

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

      Update Q-functions by one step of gradient descent using the formula below for $i = 1, 2$.

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d) \right)^2$$

      Update the policy by one step of gradient ascent using the formula below. Here $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which can be differentiated wrt θ using the reparameterization trick.

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right)$$

      Update target networks with the formula below for $i = 1, 2$

$$\phi_{targ,i} \longleftarrow \tau \phi_{targ,i} + (1 - \tau)\phi_i,$$

2.3 Robust Deep Reinforcement Learning

In this section, a survey of different methods is presented that address the problem of robustness against model mismatch. First, we present Robust Adversarial Reinforcement Learning (RARL) [50], and secondly, we present domain randomization as applied by Open AI [47].

2.3.1 RARL: Robust Adversarial Reinforcement Learning

In RARL, the environment is modified to allow two agents to participate. One is the protagonist that seeks to optimize the regular reward r_t^1 while the other is the adversary that maximizes its own reward r_t^2 where $r_t^2 = -r_t^1$. In [50] the protagonist controls various MuJoCo simulated creatures, of which the goal is to move to a different location without falling. The adversary seeks to oppose this goal by applying disturbances in the form of forces. This problem is linked to the Nash equilibrium, where the outcome of a problem for one agent depends on the actions of the other agent. Depending on the strategy of the protagonist, the adversarial tries to apply the worst possible disturbance.

In the paper, a performance analysis is made between TRPO and TRPO with the RARL modifications. Not only is the policy learned by RARL better than the policy learned by TRPO, it is also more robust against a change in the environment. The benchmark is based on 4 MuJoCo games [72], HalfCheetah, Swimmer, Hopper and Walker2d. All the mentioned environments are robotic problems, where the adversary agent can apply forces to parts of the robots to prevent the protagonist from achieving the goal. This implies that an existing environment needs to be extended to support such a second action space.

As mentioned before, evaluation of the policy is performed in a different environment than was trained on to visualize the robustness of the agent. More concretely, a different environment entails a change in simulation parameters, such as mass or friction coefficient. In the paper, it is clearly visible that the algorithms trained with RARL are more robust to changes in the environment than TRPO.

Training both models requires switching between training the protagonist and the adversary. When one agent trains the other, its weights are locked. The complete algorithm is shown in Algorithm 3 recreated from [50]. In this algorithm, μ and υ equal the protagonist and adversarial policies.

Algorithm 3 RARL [50]

function TRAIN(ξ, μ, υ)
  for $i = 1, 2, .., n_{iter}$ **do**
    $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
    for $j = 1, 2, ..., N_\mu$ **do**
      $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\xi, \mu_{\theta_i^\mu}, \upsilon_{\theta_i^\upsilon}, n_{traj})$
      $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$
    $\theta_i^\upsilon \leftarrow \theta_{i-1}^\upsilon$
    for $j = 1, 2, ..., n_\upsilon$ **do**
      $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\xi, \mu_{\theta_i^\mu}, \upsilon_{\theta_i^\upsilon}, n_{traj})$
      $\theta_i^\upsilon \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \upsilon, \theta_i^\upsilon)$
  return $\theta_{n_{iter}}^\mu, \theta_{n_{iter}}^\upsilon$

2.3.2 Domain Randomization

In 2018 Open AI successfully trained a model with which a real world robotic hand has the task of changing the orientation of a block to a desired configuration using five its fingers. [47]. This was achieved by training in a digital twin of the setup, using 3D models and a physics simulator. Here imperfections in the simulator may cause problems when the model trained in simulation is applied in the real world. To handle this issue, Open AI applied domain randomization, where most aspects of the simulated environment were randomized. The goal was to learn both a policy and a vision model that generalizes to reality. In the work, they applied four different randomization effects, shown below:

1. **Observation Noise** - Gaussian noise to mimic the kind of noise expected in reality.
2. **Physics Randomization** - Randomization of physics model parameters including object dimensions, object masses, friction coefficients, dampening coefficients for the robot joins.
3. **Unmodeled Effects** - Adds effects to the existing model that the physics model does not account for. Includes freezing positions of markers on the block to simulate loss of tracking and backlash for the motors.
4. **Visual Appearance Randomization** - Randomization of camera positions, colors, rotations, materials and textures.

To evaluate the effectiveness, a trial was defined. A trial terminates when the object is dropped, 50 rotations are achieved, or a timeout of 80 seconds is reached. In the simulated environment, averaged on 100 trials, the agent performed 43.4 correct rotations on average, and in the real environment, the agent performed 18.8 rotations on average in 10 trials.

Chapter 3

Greenhouse Simulation

In this chapter, the details of the simulator will be elaborated. We start with the structure of the model and give an overview of all its components. Afterwards, we discuss the model validity and the interface including the action and observation space, the objective and the reward function. Finally, the model is validated with the data obtained in the 3rd edition of the Autonomous Greenhouse Challenge [71].

3.1 Simulator Structure

In the standard reinforcement learning setting, as described in subsection 2.2.1, an agent takes actions in a discrete step based environment, hence our simulation model is also a discrete time model. Our goal is to simulate the greenhouse climate control system and its relationship to plant growth and quality. The main design considerations are listed below:

1. There should be a climate simulation model that at least includes the following components: A heater, ventilation, lamps, transparent (energy) screen, blackout screen and carbon dioxide supply. These actuators were selected as they are common in greenhouses as seen in subsection 2.1.2, and were also available during the 3rd edition of the Autonomous Greenhouse Challenge [70].
2. There should be an interchangeable plant model that simulates the growth and quality of a lettuce plant.
3. A weather model should be added that represents the Dutch weather.
4. An economic model should be used to calculate the final balance after a growing cycle. This should be the reward function.
5. The simulation should be encapsulated in an Open AI Gym environment [16], which is the de facto standard interface for reinforcement learning.

We chose not to model anything related to the water and nutrient mixture, crop diseases, and other biological or chemical processes besides the plant growth and quality model. While these processes are important for the plant, they are decoupled from the climate control system to a large extent.

Simulator Components

To organize the simulator implementation, we split the different parts of the simulation into climate model components, a weather model, a plant mode and an economic model. The flow of data within the model, as well as the different simulator components are visualized in Figure 3.1.

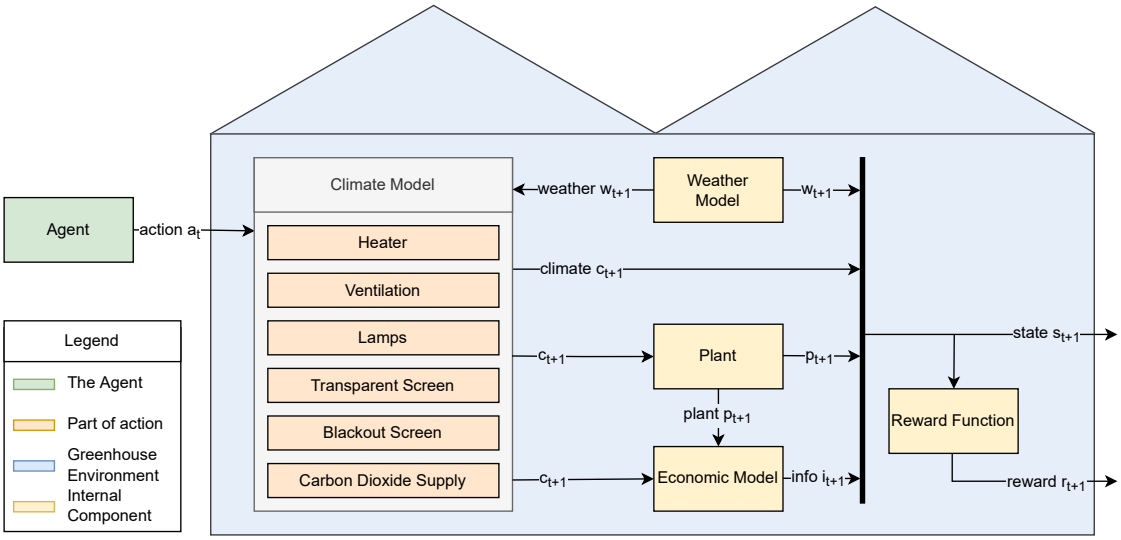


Figure 3.1: High level simulator components. Arrows represent the flow of data in the simulator when a single step is simulated. Everything contained in the Venlo type greenhouse illustration is part of the greenhouse simulator.

The different components of Figure 3.1 are explained in detail in Appendix B. Below we provide a brief summary of its implementation:

1. **Climate Model** - Calculates the greenhouse state using action input and weather. Determines the temperature, humidity, CO_2 level and light level inside the greenhouse.
   (a) **Heater** - Provides a variable amount of heat to the greenhouse.
   (b) **Ventilation** - The window position with values in range $[0, 1]$ where 0 is closed. An open window exchanges heat and air with the outside environment.
   (c) **Lamps** - Provides a variable amount of PAR light to the greenhouse.
   (d) **Transparent Screen** - Screen position with range $[0, 1]$, where 1 is closed. Provides insulation when closed.
   (e) **Blackout Screen** - Screen position with range $[0, 1]$, where 1 is closed. Provides insulation and blocks light when closed.
   (f) **Carbon Dioxide Supply** - Provides a variable amount of CO_2 to the greenhouse.
2. **Weather Model** - Uses real weather data from [34] measured at Hoek van Holland. This area is relevant, as the greenhouse density is high. Starting date can be randomized.
3. **Plant** - Exponential growth model based on limiting factors as described in subsection 2.1.1. Quality loss model based on tipburn, overheating and freezing.
4. **Economic Model** - Calculates cumulative consumption of gas, CO_2 and electricity. Peak and off-peak prices are applied to cost calculation. Plant selling price is modeled to calculate profit.
5. **Reward Function** - Interchangeable function that maps state transition to reward.

In this figure, we can also see how the action of the agent is transformed to a new state and reward tuple. Both the weather and the agent action serve as inputs for the climate model, which updates the climate state of the greenhouse. This climate state is then passed to the plant model, from which the growth speed and plant quality changes are determined. The resources used by the

climate and the current state of the plant are used by the economic model to determine cumulative gas and electricity usage, as well as the selling price of the plant and the current balance. Finally, information from the weather, plant, economic and climate models is concatenated to form the new state s, from which we can also determine the new reward r_{t+1}.

Note that the environment is only partially observable, as each of the aforementioned parts of the simulator maintain an internal state not contained in state s. We hide the internal state for the agent, as it is unrealistic to observe the internal variables of the plants.

As a convention to refer to a specific property from the action a_t or the state a_t we use the following notation: $s_{t,balance}$. Here we refer to the balance at time t, which is a property of the state s_t.

Discrete Time Simulator

To discretize the simulator an appropriate step size needs to be selected. To maintain accuracy in a discrete simulation, the step size should be small enough for the simulated system. In our case, there are multiple subsystems that require different step sizes to be accurately modeled.

For the climate model, a small step size is required to prevent numerical errors. One cause of numerical errors is the presence of differential equations in the simulation model. An example of this is the differential equations for temperature change $dT = Q \cdot dt$ and $Q = h \cdot (T_{out} - T_{in})$. For the agent, the number of actions per episode should be low to reduce the effect of the Credit Assignment Problem Figure 2.2.1. At the same time, the frequency of actions for the agent should be high enough to control the greenhouse properly. Finally, from the plant its perspective, the growth cycle lasts an entire month, while tipburn can damage a plant in a matter of minutes.

We solve this problem by running our simulation at multiple timescales. From the perspective of the agent, a single step takes one hour. The climate and the plant model are updated every 5 minutes. The weather and economic model are updated once every hour. During a single step for the agent, we update the climate model and plant model 12 times, which we refer to with n. We motivate our selection for n in section 3.3.

Simulator Algorithm

A single simulation step for the greenhouse simulator is defined by Algorithm 4.

Algorithm 4 Simulation step from t to $t+1$ using action a_t, returning s_{t+1}, r_{t+1} and d_{t+1}.

Precondition: Simulator is initialized and its current state is s_t

function STEP(a_t)
  $t_{sim}, n, m_{weather}, m_{plant}, m_{climate}, m_{economic} \leftarrow Simulator$
  $dt \leftarrow 1/n$
  $w_{t+1} \leftarrow m_{weather}.step(t+1)$
  for $j \leftarrow 1$ to n **do**
    $t_{sim} \leftarrow t_{sim} + dt$
    $c_{t_{sim}} \leftarrow m_{climate}.step(a_t, w_{t+1}, dt)$
    $p_{t_{sim}} \leftarrow m_{plant}.step(c_{t_{sim}}, dt)$
  $i_{t+1} \leftarrow m_{economic}.step(c_{t_{sim}}, p_{t_{sim}})$

Now t_{sim} equals $t+1$

  $s_{t+1} \leftarrow$ CONCAT($c_{t_{sim}}$, w_{t+1}, $p_{t_{sim}}$, $i_{t_{sim}}$)
  $d_{t+1} \leftarrow$ ISTERMINALSTATE(s_{t+1})
  $r_{t+1} \leftarrow$ REWARDFUNCTION(s_{t+1})
  return $s_{t+1}, r_{t+1}, d_{t+1}$

In Algorithm 4 we define the different model components m on line 2. On line 4 we update the weather. On line 5 to 8 we perform n fine update steps for the climate and plant model. The economic model can now calculate the new financial info i at line 9. After this step, t_{sim} equals $t+1$ and all models are updated. We concatenate the internal model states to form s_{t+1} at line 10, determine if the state is terminal at line 11, and calculate the new reward r_{t+1} at line 12. We have now updated the simulator state from t to $t+1$ and we calculated the $(s_{t+1}, r_{t+1}, d_{t+1})$ tuple.

3.2 The Gym Environment

As described in section 3.1 we wrapped the greenhouse simulator in an Open-AI Gym interface. This interface consists of an action space, an observation space and a reward function. In this section, we will elaborate these properties. Additionally, we define an evaluation criteria to benchmark the performance of trained agents.

3.2.1 Action and Observation Space

In the environment, both the action and observation space consist of normalized continuous values. The shape of the observation space is $(31,)$ with an action space of $(6,)$. We normalize the action and observation space to the [0, 1] range using predefined normalization ranges. The information contained in the observation space can all be measured or estimated in a real greenhouse using standard sensors and a camera to estimate crop weight and quality. In Appendix B.3 we show the full description of the variables and their normalization ranges for both spaces.

3.2.2 Reward Function

The objective in the environment is to maximize the balance in the final step of the simulation. In this step, the final balance in $EUR\ m^{-2}$ represents the profit or loss of the entire growing cycle.

The final step of the simulation is reached when one of the following conditions occurs:

1. The plants reach a mass of 250 grams
2. The duration of the growing cycle exceeds 90 days

Based on these conditions, we can define an *isterminal* function that determines for a given state s_t whether it is a terminal state or not.

$$isterminal(s_t) = \begin{cases} True, & \text{if } s_{t,time} \geq 2160 \text{ or } s_{t,mass} \geq 250 \\ False, & \text{otherwise} \end{cases} \tag{3.1}$$

For the simulation, the standard reward function is the FinalBalanceReward, as the primary objective is to maximize profit. The FinalBalanceReward is defined using the following equation:

$$r_{FinalBalanceReward}(s_t) = \begin{cases} s_{t,balance}, & \text{if } isterminal(s_t) \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

3.2.3 Evaluation Criteria

To evaluate the agent fairly, a deterministic benchmark was developed. In this benchmark, we average the final balance over 10 different runs. These runs all start on March 1st, each in an unique year between 2011 and 2020. The agent will be trained on random weather sequences from the same location, but sampled from 2001 to 2010. This ensures that a high result in the benchmark is not achieved by overfitting on specific weather sequences, and that our agent can generalize to different weather sequences. Secondly, by fixing the start date for the 10 runs, and

because there is no further stochastic in the simulator, we produce a deterministic benchmark that is useful to compare two agents. Finally, by performing 10 runs, the result should represent the Dutch weather well, so that we can expect evaluation in other years to be similar.

3.3 Simulation Validation

In this section, we validate the simulation accuracy. First, we motivate our number of update steps n. We proceed by comparing our model to INTKAM / KASPRO.

We describe and motive all the parameters used in the simulator in section B.2. Most parameters are either retrieved from literature or replicated from the Autonomous Greenhouse Challenge data we obtained during the challenge.

3.3.1 Simulation Step Size

To determine an appropriate value for n we analyze the resulting time series of the simulation when the greenhouse is controlled by a state-of-the-practice policy defined in subsection 4.1.1. In Figure 3.2 we show how the 3 main growth factors and plant mass evolve over 8 days for different values of n.

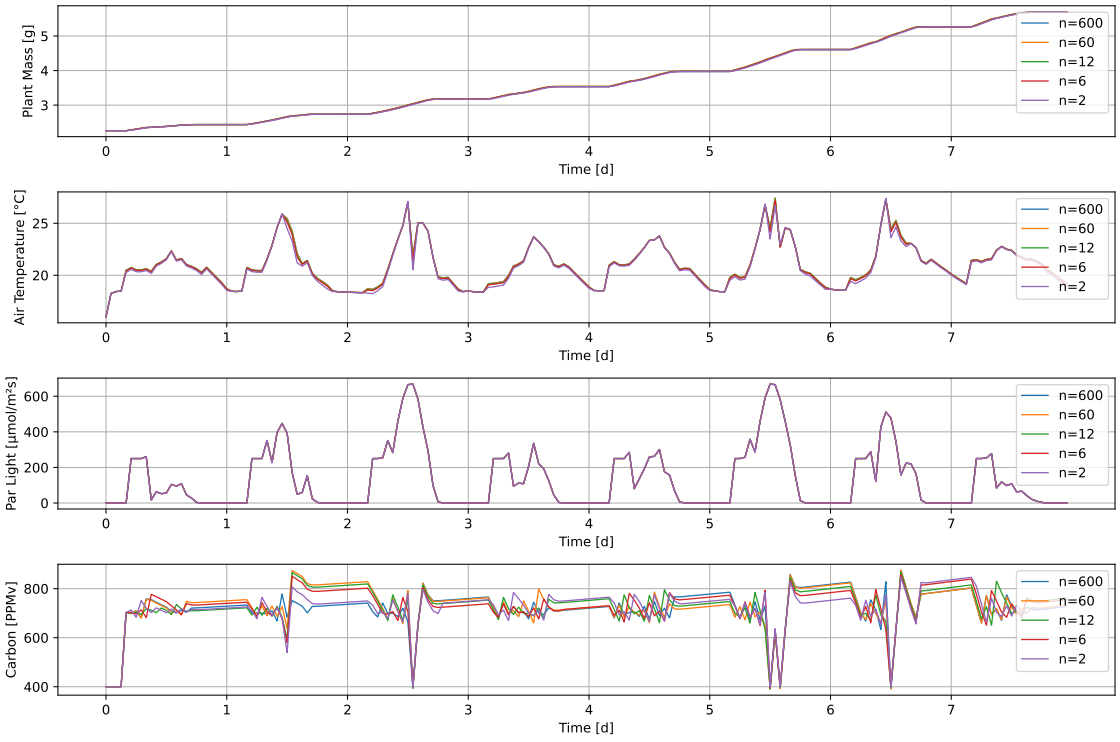


Figure 3.2: Growing cycle for a state-of-the-practice policy defined in subsection 4.1.1. Shows the 3 main growth factors and plant mass over a growing period of 8 days for different values of n.

We notice a clear deviation in the time series between different values for n for the CO_2 state variable, while the other state variables show little deviation. The large deviations are located directly after a drop in CO_2 level. Here, the agent applies a correction to the CO_2 level using the carbon supply. However, the carbon levels in the greenhouse fluctuate at a high frequency due to windows opening, plants growing, and the high CO_2 flow rate from the carbon supply. This shows that the action frequency is not sufficiently high to accurately control the CO_2 level in the

greenhouse. The plant mass and PAR light level do not show large deviations. We can slightly notice the deviation of air temperature for $n = 6$ and $n = 2$.

Although the CO_2 levels differ for different values of n the plant mass is of main concern for the grower. As there are no significant differences in plant mass, and $n = 12$ is the largest n that shows no temperature deviations, we conclude that $n = 12$ yields sufficient numerical accuracy.

3.3.2 Comparison with Other Simulators

We compare our simulator to data from INTKAM / KASPRO obtained during the Autonomous Greenhouse Challenge. We do this by converting this data to inputs for our model and comparing the output of both models.

Lettuce Model

Our lettuce model uses a coarse approximation for plant growth based on data from the 3rd autonomous greenhouse challenge. This data was obtained from the INTKAM / KASPRO model. In Figure 3.3 one can see what shape a typical growing curve has in our model and in the INTKAM / KASPRO model. In INTKAM / KASPRO the plant growth starts exponentially, but it turns into linear growth when the plant becomes larger.

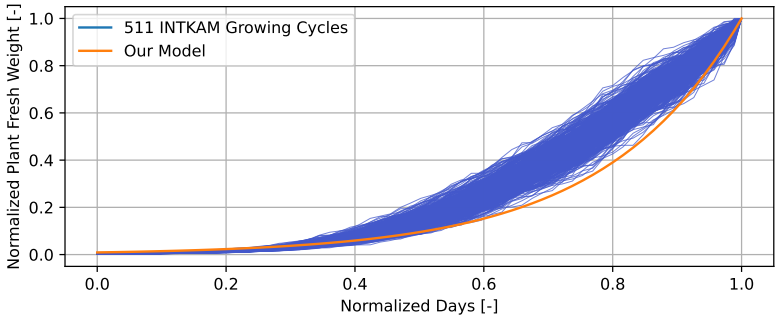


Figure 3.3: Normalized mass versus days curve for 511 growing cycles in INTKAM / KASPRO and the exponential curve of our model.

Under ideal conditions, the lettuce plant in our simulator grows to 250 grams in 27 days. In INTKAM / KASPRO, the best result our team obtained was 32 days. This is caused by our overestimation of growth rate for larger plants.

Lights

Our PAR light model depends on outside illumination, lamp usage and screen position. A comparison was made on our PAR light level compared to the PAR light level in KASPRO, based on identical screen positions, weather and lamp usage. The resulting image can be seen in Figure 3.4.

We observe that under identical conditions, our model for PAR light performs similar to KASPRO in most cases. There are some over and underestimations that occur when sunlight is involved. It could be that KASPRO uses finer grained weather data compared to our hourly values, values not included in the hourly results. To calculate the state at $t + 1$ we apply the weather of $t + 1$ during the entire transition of $t \longrightarrow t + 1$. However, this is only speculation, as we do not have access to the source code of both INTKAM and KASPRO.

Temperature

In Figure 3.5 a comparison on greenhouse air temperature is shown. In this simulation, we compare our model its output under identical outside illumination and temperature levels, heating power

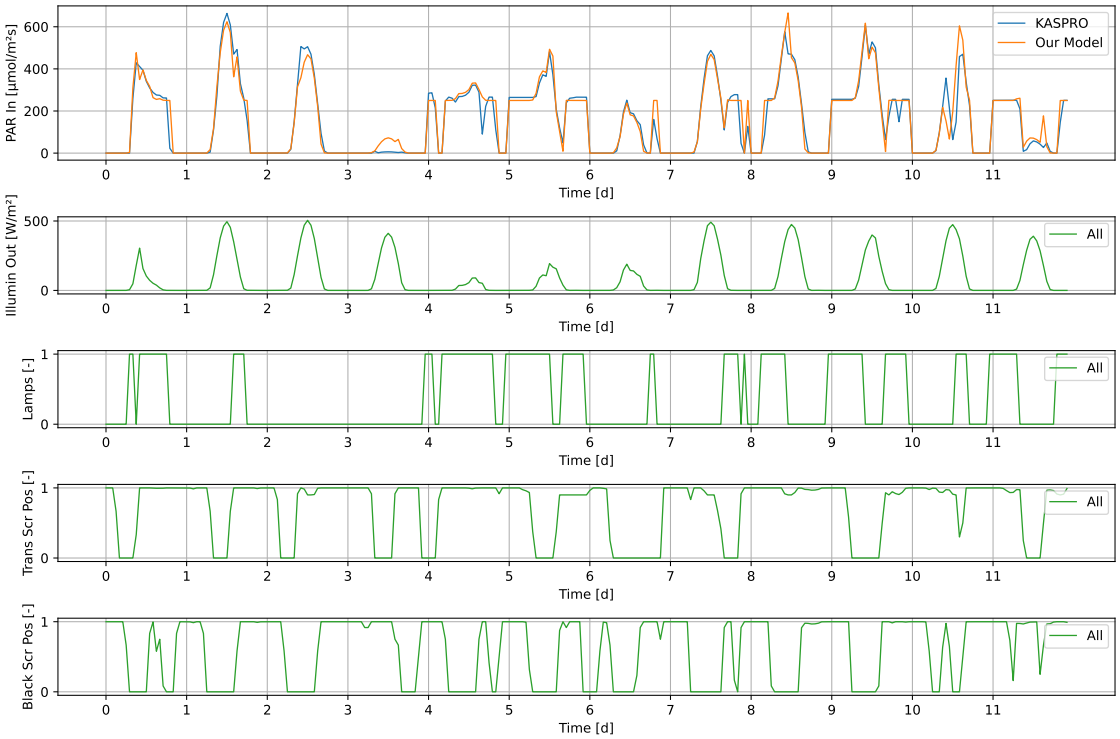


Figure 3.4: Comparison between PAR light level above the plant between KASPRO and our model, given that lamp usage, weather and screen usage are equal.

and ventilation position of the windows. Here we see that our model shows a significant difference with the KASPRO model.

This can be explained by the complexity of the KASPRO [20] model and our model. In our model, we apply heating power directly to the greenhouse air temperature modeled as a single heat buffer, while in KASPRO there are multiple heat buffers involved. The result is that the air temperature in KASPRO is influenced by at least 10 different factors as shown in [20, Figure 5.3]. Evidence for this explanation can be found in day 1-2. Our model shows a clear temperature increase when the sun starts shining, and it strongly decreases when it becomes night and no heating is provided. KASPRO shows opposing behavior, indicating that KASPRO models additional effects.

Secondly, we are not aware of all the temperature related parameters of the simulation set. Our values for heat exchange by the windows, heat capacity, insulation, and the fraction of solar heat absorbed are based on values from different studies, as shown in section B.2. The high gradients can possibly be caused by lower insulation or a higher ventilation capacity.

Both the additional heat buffers and the difference in simulation parameters contribute to the mean absolute error of 6.7 K.

However, our model shows expected behavior. When it is freezing outside, the windows are open and the heater power is low, the greenhouse temperature drops. During daytime, the temperature rises due to sunlight. Finally, when the windows are open, one can see that the greenhouse temperature becomes more equal to the temperature outside.

Carbon Dioxide

In the case of CO_2 we show the comparison between KASPRO and our model in Figure 3.6. Here we show the CO_2 level under effects of identical CO_2 flow rates, plant fresh weight and ventilation position. We observe a large peak between day 2 and 3. During this peak, we also observe that

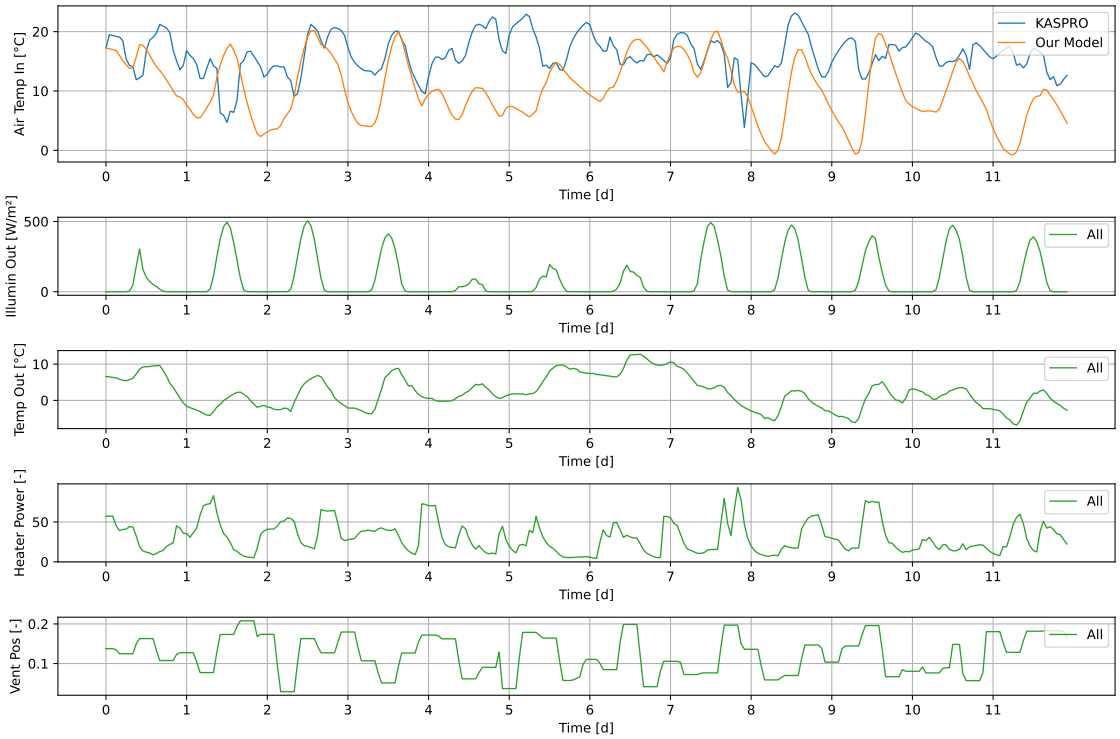


Figure 3.5: Comparison on greenhouse air temperature between KASPRO and our model, given that inputs for outside illumination, temperature outside, heater power and ventilation position are identical.

the windows are closed. This implies that our plant does not absorb an equal amount of CO_2 compared to KASPRO or that KASPRO models an additional effect that removes CO_2 from the greenhouse.

The mean absolute error between traces is 109 $PPMv$. Exclusion of day 2-3 reduces this error to 97 $PPMv$.

Relative Humidity

In Figure 3.7 we compare our model with KASPRO while ventilation position, air temperature inside and relative humidity outside are identical for both simulators.

We observe small differences in relative humidity between day 0 and day 8. After day 8, we observe a drift between our model and KASPRO. Here, our model underestimates the humidity levels in the greenhouse severely. The MAE of relative humidity between both models is 7.0 %. In the period until day 8 the MAE is 4.8 % while after day 8 it is 11.5 %.

A possible explanation for this drift is leaf transpiration, which increases as the lettuce plants grow. In our model, leaf transpiration is not modeled. However, during transpiration, a real plant does emit water vapor [52]. A growing plant will result in additional evaporation over time, raising the relative humidity.

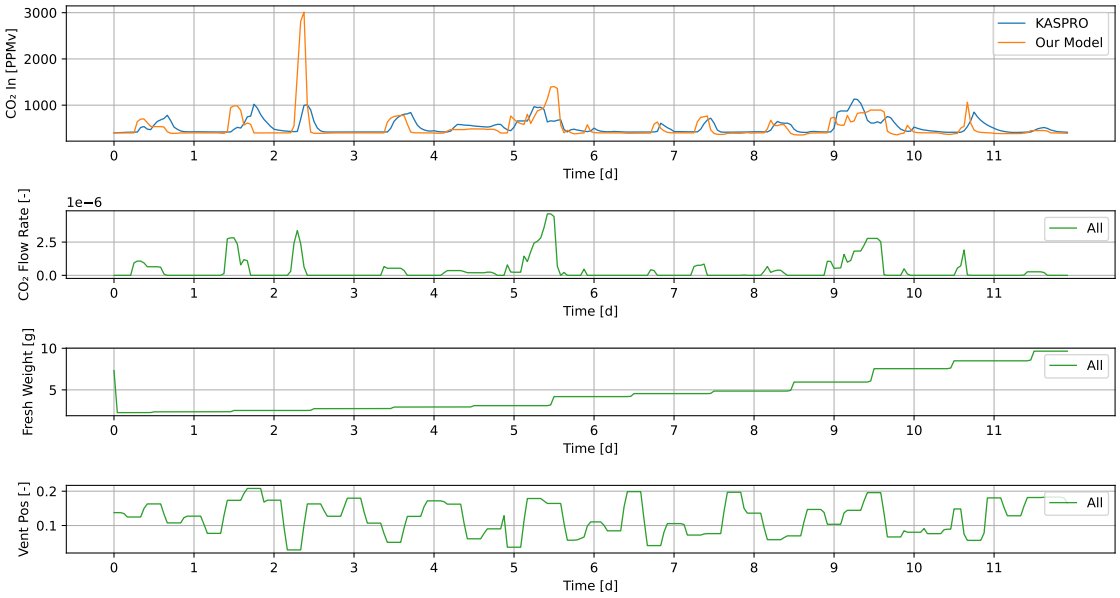


Figure 3.6: Comparison on greenhouse CO_2 levels between KASPRO and our model, given that inputs for CO_2 flow rate and ventilation position are identical.

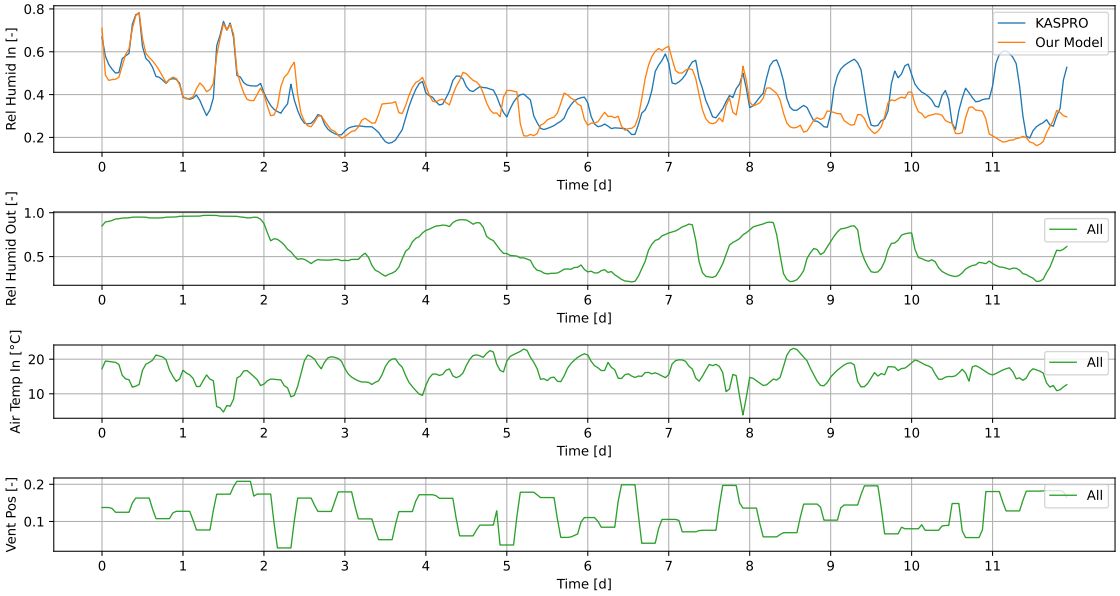


Figure 3.7: Comparison on relative humidity level between KASPRO and our model, given that the air temperature, the ventilation position and the outside relative humidity are identical.

3.4 Conclusions

In this chapter, we presented our greenhouse model that matches the requirements mentioned in 3.1. The result is a gym environment that can be applied to optimize reinforcement learning algorithms.

Additionally, we have shown that the simulator produces comparable results to INTKAM / KASPRO with respect to important state variables for plant growth. Two major points for

improvement include the plant transpiration model and the inclusion of multiple heat buffers in the model.

As mentioned in section 1.2 the simulator is not the main focus of this work. The goal was to design a simulator that resembles the greenhouse environment well. From the figures shown in section 3.3 we conclude that the simulator models many of the effects present in INTKAM / KASPRO with sufficient accuracy for this study.

Chapter 4

Optimized Greenhouse Control

In this chapter, we elaborate our optimization process for both SAC and PPO. Our goal is to maximize the performance on the deterministic benchmark described in subsection 3.2.3. First, we will discuss two baselines to which we compare the SAC and PPO results. Afterwards, we will elaborate which implementation of SAC and PPO we used for training. Finally, we compare the baselines to the trained versions of SAC and PPO.

4.1 Applied Methods

In this section, we will briefly explain the methods used to optimize SAC and PPO, as well as the baselines to which we will compare our result.

4.1.1 State of the Practice Rule Based Policy

In Algorithm 6 we present a baseline policy based on state-of-the-practice climate control systems, as described in section 2.1.3. Here we applied the policy described in [15] and use their set points as a starting point. The policy setpoints are listed in section A.3. Using this policy we obtain a mean final balance on the evaluation benchmark of 0.62 $EUR\ m^{-2}$.

We optimize this algorithm further by searching for optimal parameters. For each of the parameters that are part of the algorithm we define an exploration range, shown in section A.3. Optimization of 19 different parameters creates a large search space. We apply Bayesian Optimization with 1768 initial random search steps and 256 steps guided by the Bayesian Optimizer. The objective is to maximize the mean final balance on evaluation criteria described in subsection 3.2.3. The result is a more optimal set of parameters that yields 0.76 $EUR\ m^{-2}$ on the evaluation benchmark. The optimized policy setpoints are shown in section A.3.

4.1.2 Literature Baseline

As we developed our own greenhouse gym environment, we have no performance baseline from other works. Therefore, we provide a literature reference for the performance of state of the practice Dutch greenhouses. In [55, Table 3.1] we find a typical yearly balance sheet for a Dutch greenhouse of 6.2 ha. The total profit is 475,500 EUR, a correction for the area, and for 12 growing cycles a year yields a profit of $475,500/(62,000 \cdot 12) = 0.639\ EUR\ m^{-2}$.

4.1.3 Deep RL Methods

To train SAC and PPO we use a workstation of which the hardware is shown in section A.1.

We use multiple workers that perform roll-outs to obtain training data. Additionally, we use an evaluation worker that evaluates the performance of the agent on the evaluation benchmark

Algorithm 6 Action policy of the rule based agent. Descriptions of state and action variables are shown in section B.3. All controller settings can be found in section A.3. The **pid** function accepts three values: s, v and p. Here s equals the setpoint, v is the current value, and p is the proportional gain in PID control. When multiple **pid** function calls are made the maximum value is used.

```
1: function COMPUTE_ACTION(ControllerSettings: c, GreenhouseState: s)
2:     a ← Action(), all values are initially 0, max value is 1
3:
4:     if c.day_start < s.hour < c.day_end then
5:
6:         if s.temperature_air < c.t_min_day then
7:             a.heater.pid(s=c.t_min_day, v=s.temperature_air, p=c.p_heater)
8:
9:         if s.illumination_out > c.illum_max then
10:            a.blackout.pid(s=c.illum_max, v=s.illumination_out, p=c.p_blackout)
11:
12:        if s.carbon_ppm < c.carbon_open and s.ventilation > 0 then
13:            a.carbon.pid(s=c.carbon_open, v=s.carbon_ppm , p=c.p_carbon)
14:        else if s.carbon_ppm < c.carbon_closed then
15:            a.carbon.pid(s=c.carbon_closed, v=s.carbon_ppm , p=c.p_carbon)
16:    else
17:        a.transparent ← 1
18:        a.blackout ← 1
19:
20:        if s.temperature_air < c.t_min_night then
21:            a.heater.pid(s=c.t_min_night, v=s.temperature_air, p=c.p_heater)
22:
23:        if s.carbon_ppm < c.carbon_night then
24:            a.carbon.pid(s=c.carbon_night, v=s.carbon_ppm , p=c.p_carbon)
25:
26:    if c.lamps_start < s.hour < c.lamps_end then
27:        a.lamps ← 1
28:
29:    if s.temperature_air > c.t_max then
30:        a.ventilation.pid(s=c.t_max, v=s.temperature_air, p=c.p_temp_ventilation)
31:
32:    if s.relative_humidity > c.rh_max and
33:            s.relative_humidity - s.relative_humidity_out > c.rh_min_diff then
34:        a.ventilation.pid(s=c.rh_max, v=s.relative_humidity, p=c.p_humidity_ventilation)
35:    if s.relative_humidity < c.rh_min and
36:            s.relative_humidity_out - s.relative_humidity > c.rh_min_diff then
37:        a.ventilation.pid(s=c.rh_min, v=s.relative_humidity, p=c.p_humidity_ventilation)
38:    return a
```

after every training step. In this benchmark, we disable all exploitative functions of the agent to show the performance of full exploitation.

We start with the default parameters for SAC and PPO in the Ray implementation [42]. For each of the algorithms, we perform multiple experiments to optimize hyper parameters. In this work, an experiment consists of training n trials, meaning we optimize n agents independently. In each of the experiments, we apply a grid search over a set of predefined hyper parameter combinations. At the end of an experiment, we tune our hyper parameter set using the hyper

parameters of the best performing trials in the experiment.

Proximal Policy Optimization

We apply the clipped version of PPO, in which we also use General Advantage Estimation (GAE) [59]. We start with a model consisting of two fully connected layers with 256 nodes each. The exact implementation used can be found in Ray 1.7.10 [42], which is adapted for using multi continuous action space.

Soft Actor Critic

We apply a version of SAC with the double Q-trick that was described in subsection 2.2.1. This implies we use two state-action value networks, which makes the value estimation and training more stable. As SAC is entropy regularized, it is by default a stochastic policy. However, we use the deterministic version during evaluation, as the deterministic policy performs better in [26].

4.2 Learning Behaviour

Now we will motivate our hyper parameter selection in a series of experiments. For each of the experiments in this section, there is a dedicated page in section A.4.

4.2.1 PPO

We will now elaborate 7 experiments with which we improve the performance of PPO.

Reward Engineering

Initially, we defined the reward function as Equation 3.2, where the agent is only rewarded for the final step. Here we also mentioned that using the termination condition results in episodes with lengths between 414 and 1440 steps. This reward function is a natural example of the credit assignment problem, as mentioned in subsection 2.2.1. If the agent takes at least 414 different actions and receives a single reward on the terminal step, it becomes hard to determine which actions contributed or did not contribute to the reward. For this we designed multiple reward functions using the following criteria:

1. The cumulative sum of rewards should match the profit of the grower
2. The reward should reflect the value of the agent its actions on a high frequency

Using these criteria, the following reward functions were designed:

Balance Reward

The goal of this reward function is to exactly match the value generated in this step. The value of a step is its contribution to the final balance, in which the grower is interested. The final balance itself consists of the plant gains, minus the fixed and variable costs made. The contribution of the costs towards the final balance is trivial, as the costs made at time t are directly subtracted from the final balance. The gains of the plant are more complex, as the gains depend on the mass and quality of the plant. A simple approach would be to scale the reward with the change in mass of the plant. However, this would unjustly reward the agent more for optimal growing conditions for heavier plants compared to smaller plants (as heavier plants grow faster in terms of absolute mass). Therefore, the decision was made to reward the agent for relative growth. As our lettuce growth model uses exponential growth, and we know the starting mass and the final mass, we can determine the value of the current step using the following equation:

$$r_{growth}(t) = G_{max}/log_{g(t)}(g_{req}) \tag{4.1}$$

Here $r_{growth}(t)$ is the growth reward at time t, G_{max} is the maximum turnover by selling the plants at perfect quality, with a value of 5.00 $EUR\ m^{-2}$, $g(t)$ is the growth rate of the plant at time t which equals the relative mass increase, and g_{req} is the relative growth required from shoot to full grown plant. The log term in this equation described the number of hours to grow the plant from shoot to plant, given the growth rate at time t. During a single simulation step, our plant grows at rate $g(t)$ for exactly 1 hour, growing our plant with $1/log_{g(t)}(g_{req})$ of the total relative growth. Hence, we can multiply it by the maximum turnover G_{max} to obtain a reward proportional to the growth rate. Assuming we achieve the selling mass of 250 grams without losing any quality, the cumulative sum of r_{growth} matches the turnover perfectly.

To drop the assumption of perfect plant quality on the previous reward function, we introduce a term to compensate for quality loss. In our model, the quality loss percentage directly reduces the selling price. Therefore, we can define the following reward function for quality loss:

$$r_{quality}(t) = ds_{t,quality} \cdot G_{max} \tag{4.2}$$

Here $r_{quality}(t)$ is the quality reward at time t, $ds_{t,quality}$ is the quality change at time t, for which $ds_{t,quality} \leq 0$ as damage is not reversible in our simulator. This reward function assumes the selling mass of 250 grams is reached during the growing cycle.

Finally, we can account for the costs made by the agent at time t. This reward function is expressed using the following equation:

$$r_{costs}(t) = ds_{t,varcost} + ds_{t,fixcost} \tag{4.3}$$

Here $r_{costs}(t)$ is the cost reward function, denoting costs made during the previous step. $ds_{t,varcost}$ and $ds_{t,fixcost}$ denote the change in total variable costs and fixed costs of the system in $EUR\ m^{-2}$.

Combining the former mentioned reward functions, we can define the balance reward function, shown in Equation 4.4. The cumulative sum of this function equals the final balance after termination of an episode, where the lettuce reaches 250 grams. Additionally, this function gives an immediate reward that represents the value at time t towards the final balance. This ensures the agent can learn the value of its actions effectively with direct feedback.

$$r_{balance}(t) = r_{costs}(t) + r_{quality}(t) + r_{growth}(t) \tag{4.4}$$

One disadvantage is that function assumes the lettuce reaches 250 grams during the growing cycle. During training we need to validate if this condition is reached during the episodes. If it is not, the agent may be rewarded for only partially growing the lettuce plant to a mass that cannot be sold. If the cumulative reward for the optimal policy reaching 250 grams is exceeded by cumulative reward for the optimal policy that does not reach 250 grams, the algorithms learn not to reach the 250 grams. In this case, only costs will be made without any turnover, and performance will be unacceptable. This effect will be validated during the reward function analysis.

Daily Balance Reward

Further analyzing the balance reward function from Equation 4.4 we find that the reward is high during growth. However, if there is no growth, the reward is negative, while costs still need to be made to maintain a desired climate. An example of this is maintaining an appropriate climate during the night. Heating the greenhouse to stay within the optimal lettuce growing conditions will result in costs, resulting in a negative reward, while it is important to grow optimally during the next day. As there is a clear daily cyclic pattern here, we define a reward function that sums the reward to a single daily value awarded at 00:00.

$$r_{daily\_balance}(t) = \begin{cases} \sum_{i=t-23}^{t} r_{balance}(i), & \text{if } s_{t,hour} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Here $r_{daily\_balance}(t)$ is the daily balance reward, $s_{t,hour}$ is the hour of the day at time t.

Effect on Performance

In Figure 4.1 we see a clear advantage in using the BalanceReward and the DailyBalanceReward over the DeltaMassReward and the FinalBalanceReward. The first two reach a profitable score slightly over 1 $EUR\ m^{-2}$ while the latter two are not profitable and do not seem to have an increasing performance. We do not observe any local minimum that withheld the agent from using both functions. Maximization of both functions should lead to maximization of the final balance objective for the grower, which is beneficial. In the experiment DailyBalanceReward performs slightly better than BalanceReward. Therefore, we will use this reward function in the next experiments.

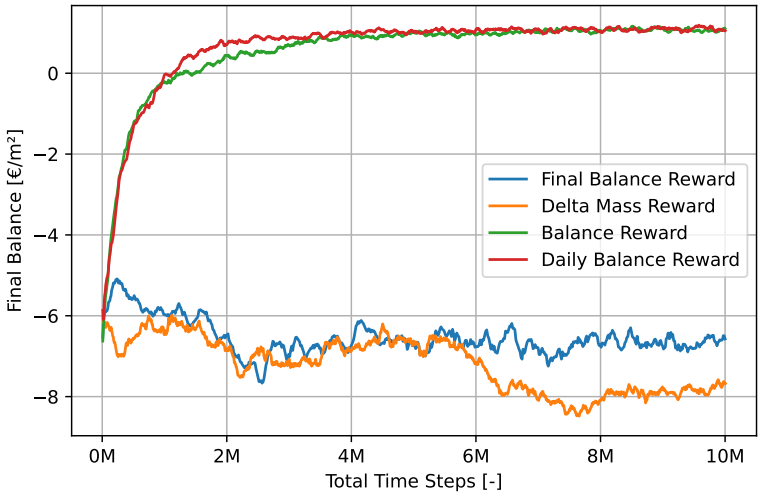


Figure 4.1: Mean final balance on episode ends over the training period, for experiment 1

In Figure 4.2 we can see how the BalanceReward and DailyBalanceReward make significantly smaller costs. The FinalBalanceReward has large fixed costs, which is a daily cost. This implies that the agent using FinalBalanceReward either has longer growing cycles or does not finish growing the plant within the time limit. Its learning ability is low, which can be explained using the credit assignment problem in the following way: A single reward over thousands of steps does not allow the agent to effectively learn what actions were important, therefore it may not learn at all.

Additionally, we find that DeltaMassReward causes the agent to grow the plant, however at the expense of large electricity and heating costs, resulting in bad performance. In this reward function, there is no penalty for using these resources, therefore the agent learns to use them abundantly to grow the plant fast.

Batch Size and Learning Rate

In this experiment, we explore the effect of batch size and learning rate on the performance of the agent. Here we hypothesize that a larger learning rate may cause faster convergence and a higher potential to break through local minima, at the cost of stability in training. A larger batch size may stabilize training and increase the maximum performance at computational expense.

In Figure 4.3 we observe that the larger batch sizes result in higher performance, both for low and high learning rate trials. We hypothesized that the increase in learning rate should make the convergence rate faster, but we observe the opposite in this experiment. A lower learning rate increases convergence rate. It could be that the large batch size causes the gradient to always be in the direction of the optimal policy, and a small learning rate ensures the agent does not step too far.

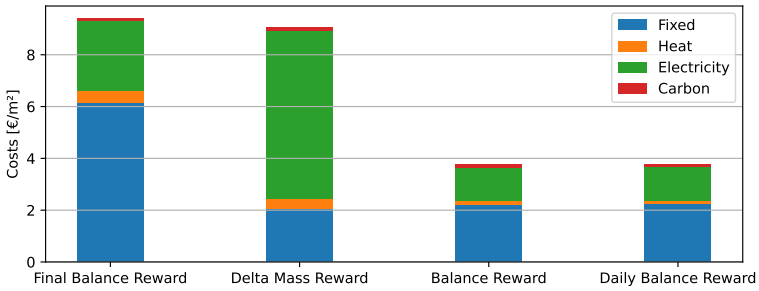


Figure 4.2: Cost breakdown of the 10 benchmark episodes for the trained agents for experiment 1.

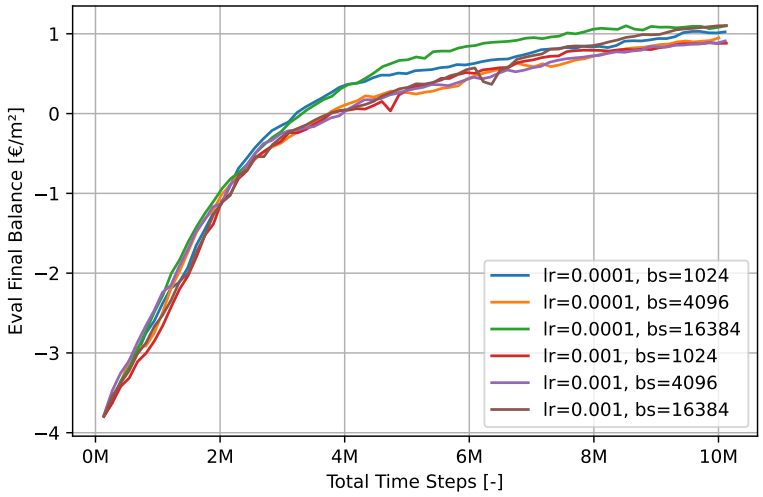


Figure 4.3: Evaluation Final Balance for experiment 2 on the 10 benchmark episodes.

We conclude that a larger batch size and a smaller learning rate improve the rate at which the agent learns to control the environment.

Lambda and Gamma

Lambda and gamma are explored together, as both parameters impact the advantage estimation of PPO. Our reward function gives the agent a reward once every 24 steps. Large values for gamma and lambda increase the long-term importance, a small value directs the agent towards short-term behavior.

In Figure 4.4 we can clearly see that the performance of trials with $\lambda = 0.90$ drop. While $\gamma = 0.990$ performs slightly better in most cases its effect is not as significant as the impact of lambda. From this plot, we conclude that $\lambda = 1.0$ with $\gamma = 0.99$ results in the highest performance, and we will apply this in future experiments.

Entropy

In this experiment, we investigate optional entropy for PPO. Entropy helps the agent explore more, at the cost of exploitation during training.

In Figure 4.5 we observe that for all entropy values, the performance is slightly better than without entropy. This holds for both performance cap and convergence rate. As we cut off our experiments at 10M frames, it may well be that the increased convergence rate allows the

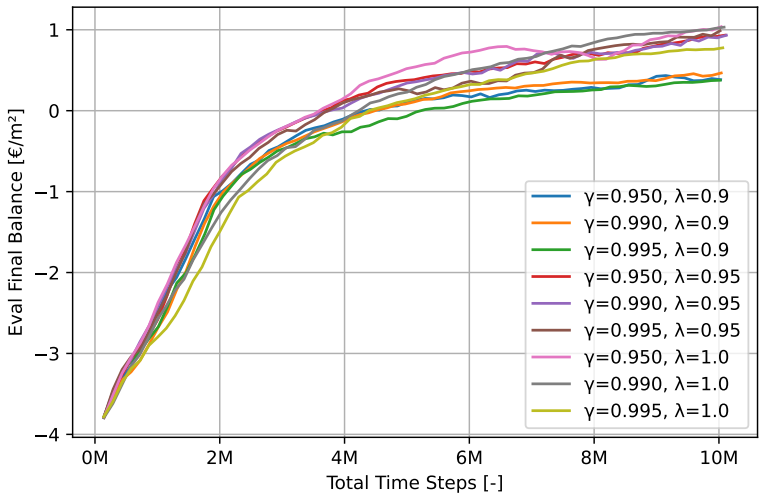


Figure 4.4: Evaluation Final Balance for experiment 3 on the 10 benchmark episodes.

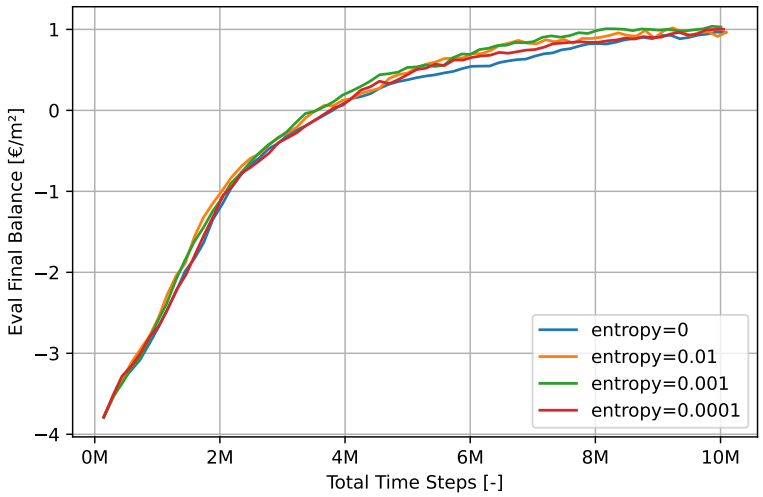


Figure 4.5: Evaluation Final Balance for experiment 4 on the 10 benchmark episodes.

agent to reach a higher performance faster, without actually increasing the performance cap. As $entropy = 0.001$ performs the best, we select this value for former experiments.

Feature Engineering

To further improve performance, we aim to transform the state space into a space where only relevant information is present and where this information is represented in the best possible way. To achieve this we apply two techniques, omitting irrelevant information from the state space and cyclical transformations. We describe the details of these techniques in section A.2.

The resulting feature engineering wrapper creates a modified observation space. Its effect can be seen in Figure 4.6, where starting from 2M frames the trial using the feature engineering wrapper outperforms the trial that does not use it.

We conclude that feature engineering is an effective method to increase performance, and we will apply the feature engineering wrapper in later experiments.

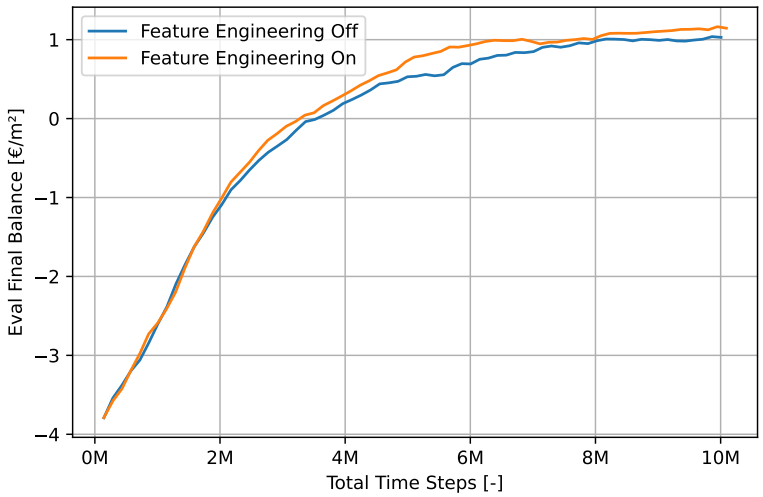


Figure 4.6: Evaluation Final Balance for experiment 5 on the 10 benchmark episodes.

Frame-stacking

In subsection 2.2.1 we mention that the *Markov property* only holds when a state contains all the information required to predict the next state s_{t+1} given A_t. In the context of our greenhouse the growth of our plant model is limited to the maximum daily growth as described in subsection B.1.1. This information is intentionally missing from the state, so our agent is unaware of its current value.

If the growth conditions are met without the plant changing mass in two consecutive states, we know that the plant reached the maximum growth for this day. We know that this holds as this is a result of our lettuce simulation model. This information is valuable as the agent can reduce the amount of gas and electricity used by letting the plant rest. Given that the agent receives two or more states it should be possible for the agent to derive this information.

We hypothesize that the performance of the agent increases if we provide two or more frames to the agent at once. In Figure 4.7 we observe trials with a different number of stacked frames. We observe that our hypothesis does not hold and that the agent that uses a single frame performs best. Therefore, in further experiments we will not apply frame-stacking.

The reason for this could be that the value of the information contained in consecutive frames is not as valuable as previously thought. It could also be that the agent is unable to learn behavior of this complexity with the limited network depth. However, we do not have enough evidence to support either explanation.

Network Architecture

Effects of network architecture were also analyzed. Previously, we used two fully connected layers, each consisting of 256 nodes, followed by a Tanh activation layer each. We first conducted an experiment on the activation layers of the network. In Figure 4.8. We observe that Relu shows a minor improvement in training speed. The reason for this could be that gradients in Relu can become larger than in Tanh, as Tanh binds the output in the range [-1, 1]. Near the end, there is no performance difference. Both functions perform similar after 5 million frames. Therefore, we find no good reason to switch from Tanh to Relu, especially as Relu activation layers can cause the gradient to become zero in extreme cases, which is called the dying Relu problem.

Afterwards, we analyzed the effect of layer size and number of layers. We experiment with layers of size 24 to 128 and with 2 layers or 3 layers. The results are shown in Figure 4.9. While most trials result in similar performance, there are two trials that differ significantly from the rest.

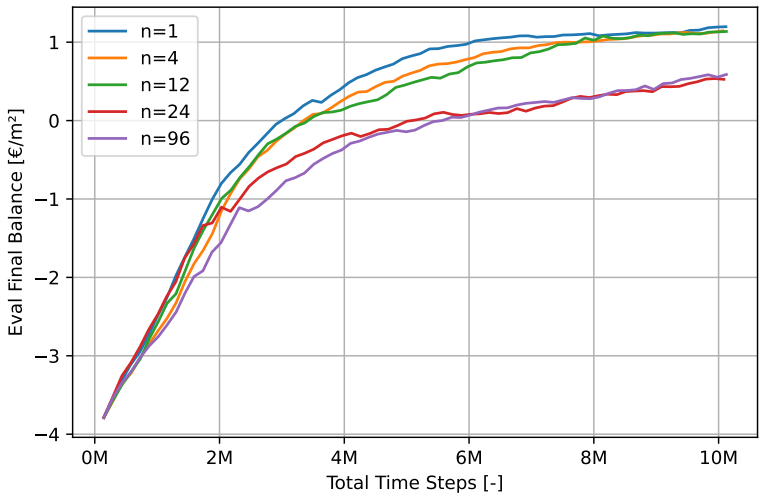


Figure 4.7: Evaluation Final Balance for experiment 6 on the 10 benchmark episodes.

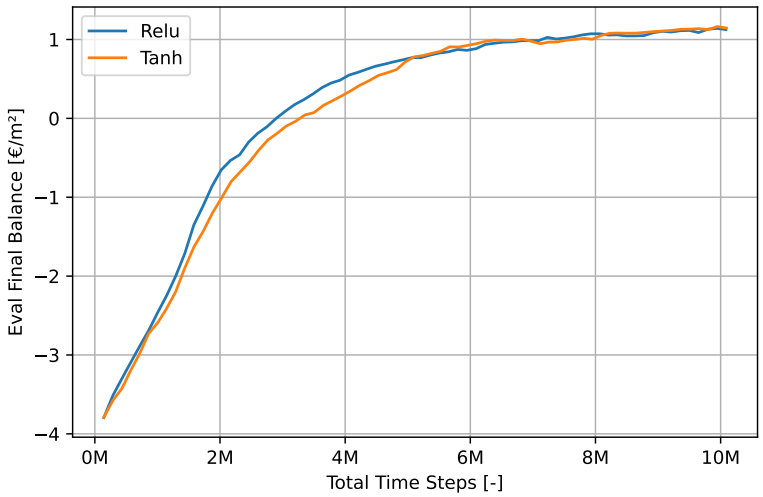


Figure 4.8: Evaluation Final Balance for the reward function experiment on the 10 benchmark episodes.

The 2x24 configuration yields a low performance, as it possibly cannot contain enough logic to control the greenhouse optimally. Our largest network with 3x128 performs best, both in learning speed and final performance. This entry surpasses the 1 $EUR\ m^{-2}$ at 5M frames, which is earlier than our previous architecture of 2x256 (visible in Figure 4.8), while also peaking higher. We observe how our 3 layer trials outperform our 2 layer trials in most cases, and how our larger networks (2x128, 3x96, 3x128) perform best. This gives further evidence that the performance of the 3x128 configuration is not a random result, hence we will use it in further experiments.

4.2.2 SAC

For SAC, we performed 3 optimization experiments, which are elaborated in this section.

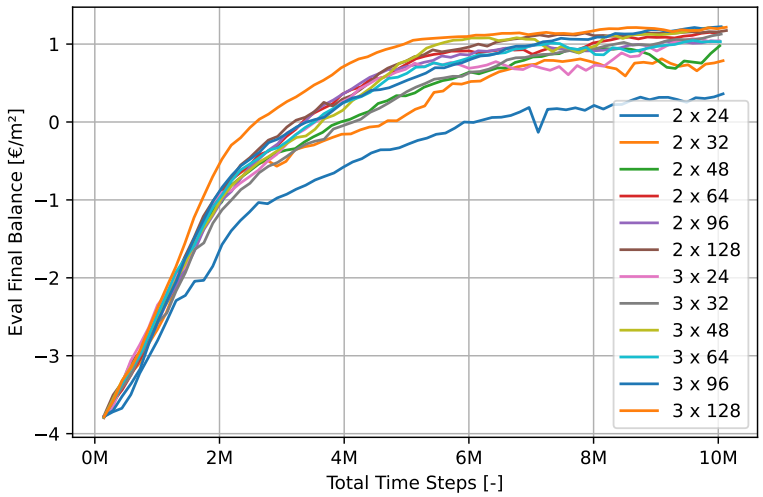
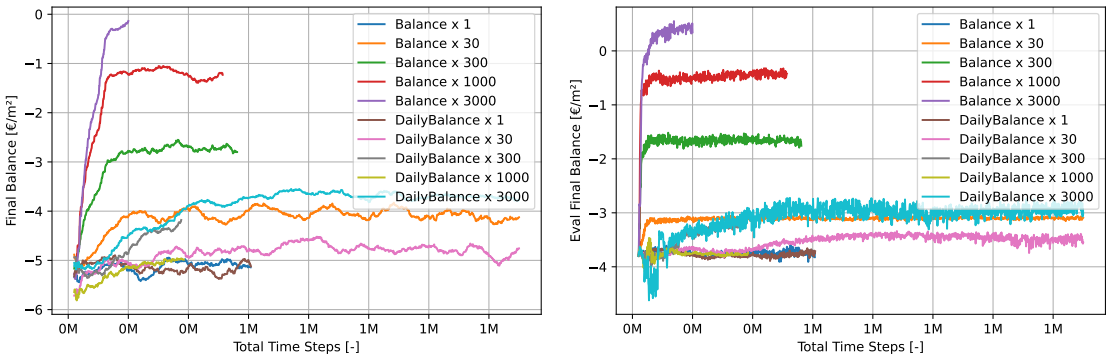


Figure 4.9: Evaluation Final Balance for the layer architecture experiment on the 10 benchmark episodes.

Reward Functions and Reward Scale

In SAC, the objective consists of reward and exploration. As mentioned in subsection 2.2.3 and also found in [26] SAC is sensitive to the reward scale as it determines the balance between exploration and exploitation. Therefore, finding the best reward function also requires the right reward scale for the function. Because of the success of the Balance Reward and Daily Balance reward in the PPO experiments, we will experiment with these reward functions only. Furthermore, in [26], the training environments yielded higher rewards compared to our environment. Therefore, we will experiment only with reward scales greater than 1.



(a) Training

(b) Benchmark evaluation

Figure 4.10: Final balance for the reward function and scale optimization experiment.

In 4.10a and 4.10b we observe that not all trials reach the goal of 1.5M steps. This is because our workstation ran out of RAM. Regardless of this issue we can draw conclusions from this graph.

While 4.10a shows the final balance during training, 4.10b shows the final balance for the deterministic policy during evaluation. Note the noise in the signal of 4.10b. This observation is a typical result for SAC under deterministic evaluation, as found in [26]. We find that the deterministic evaluated agent outperforms the exploring agent.

Additionally, we observe how quickly SAC converges compared to PPO, where most of the

final performance is reached before 200k frames have been reached.

Furthermore, we observe a strong equilibrium for each of the trials, of which the reward scale determines the performance. We can see that the Balance Reward outperforms the Daily Balance Reward when subject to a reward scale of 300 and over. While both reward functions are similar, Balance reward provides more immediate feedback, which appears important for SAC.

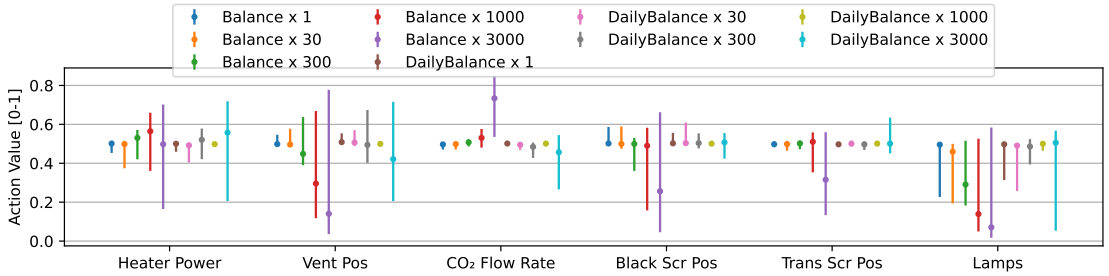
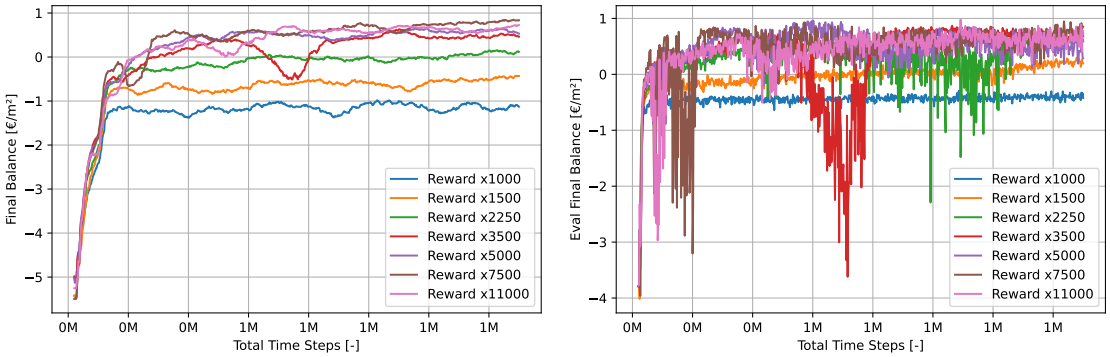


Figure 4.11: Median value and 95% CI for action value selection during the 10 episode evaluation.

The reward scale needs to be sufficient for the agent to exploit the environment properly. This can be seen in Figure 4.11, where we observe that the selected actions for the top 4 exploiting trials in 4.10a select different actions compared to trials that exploit the environment less. As entropy decreases over time, the range of selected actions becomes narrow and near 0.5 for non-exploiting policies, as seen in Figure 4.11. This indicates that less exploiting trials do not attain a policy properly, and instead rely on the exploration part of the objective function.

As BalanceReward with a reward scale of 3000 results in the best performance we will use this reward function in future experiments.

To further improve performance we perform a fine grained experiment to determine the optimal reward scale for the BalanceReward function. In 4.12a and 4.12b we observe that rewards with scales between 5000 and 11000 reach the highest performance, and that a reward scale of 7500 results in the best performance. Therefore, we will apply BalanceReward with a reward scale of 7500 in future experiments.



(a) Training

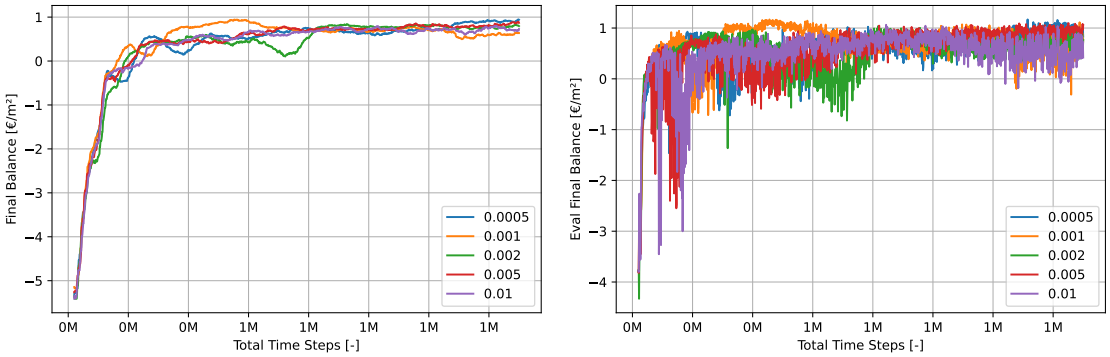
(b) Benchmark evaluation

Figure 4.12: Final balance for the reward scale optimization experiment.

Tau

In [26] the sensitivity of SAC its performance to tau is also mentioned. In 4.13a and 4.13b we observe that tau does not show any significant effect. We proceeded by selecting $\tau = 0.0005$ as

this value obtained the highest performance.



(a) Training (b) Benchmark evaluation

Figure 4.13: Final balance for the τ optimization experiment.

4.3 Result Comparison

In this section, we compare the performance of the rule based algorithm and the policies formed using SAC and PPO. We split this analysis into two parts. In subsection 4.3.1 we analyze the profits and losses for each of the algorithms in the period 2011 to 2020. We compare the cost breakdowns and energy efficiency of the different solutions. In subsection 4.3.2 we show the differences in decision-making for each of the algorithms. We show the weaknesses and strengths of each of the policies and discuss possible room for improvement.

As we have seen in subsection 4.2.2 the deterministic version of SAC shows high variance in performance. For both PPO and SAC we have stored multiple checkpoints. We evaluate the 5 latest checkpoints for both methods and select the best performing checkpoint as the policy for evaluation.

4.3.1 Evaluation Metrics

In this section we start with the main concern of the grower, the balance breakdown. We visualize this breakdown for our baselines and solutions in Figure 4.14.

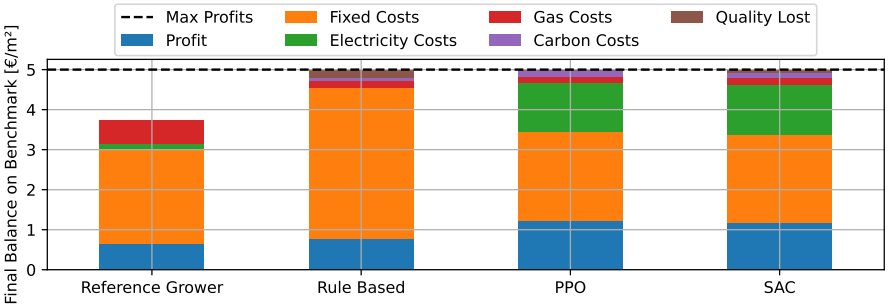


Figure 4.14: Breakdown of balance on the evaluation benchmark. In the simulator, the maximum turnover is 5.00 $EUR\, m^{-2}$. The reference grower values are based on 12 growing cycles a year, and the balance sheet from [55, Table 3.1]. Carbon dioxide costs and quality losses were not recorded.

We observe that the composition of the balance breakdown for PPO and SAC is similar, while the rule based policy and the reference grower are different. Also, the PPO and SAC agents make more profits than the reference grower and the literature baseline. Both the references and solutions are subject to large fixed costs. As the fixed costs are a fixed amount of daily costs, we know that the rule based agent takes more time to grow the lettuce compared to the PPO and SAC agents. Additionally, PPO and SAC agents spent more on electricity than the rule based agent or the literature reference, while using less gas.

We observe that our rule based solution produces similar profit to the reference grower, but the remaining division of the balance is different. A reason for this is that our policy does not match the policy applied in the study from which we constructed the reference grower baseline. Alternatively, our simulator is not accurate enough to result in a similar breakdown for the rule based agent and the reference grower.

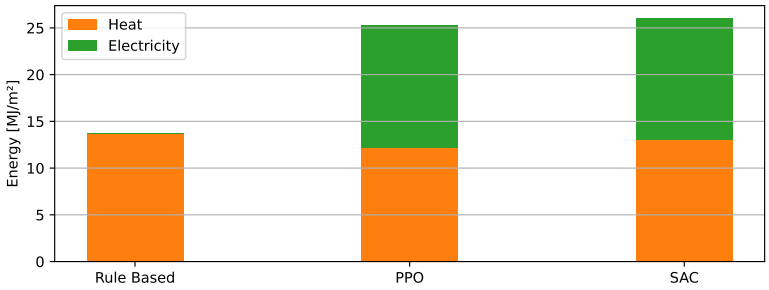


Figure 4.15: Breakdown of energy usage on the evaluation benchmark into its sources

An analysis of the energy usage in Figure 4.15 shows that again PPO and SAC produce similar results, while our tuned rule based agent does not use electricity at all. Energy in the form of heat is roughly equal for all policies.

In Figure 4.16 the rule based agent loses approximately 4% of its potential profits due to quality loss. Most of this is due to high temperature, which shows the policy does not properly cool the greenhouse when needed. The main quality loss reason for PPO and SAC is tipburn, which is caused by mineral deficiencies due to rapid growth. As mentioned before, the rule based agent takes more time to grow the lettuce plant, which makes it less susceptible to tipburn.

In Figure 4.16 we observe that typically a bad year for one algorithm is also a bad year for other algorithms. While PPO and SAC produce profits every year in the benchmark, our rule based agent produces losses in 2017. In subsection 4.3.2 we will observe the decisions made by the rule based agent in this year.

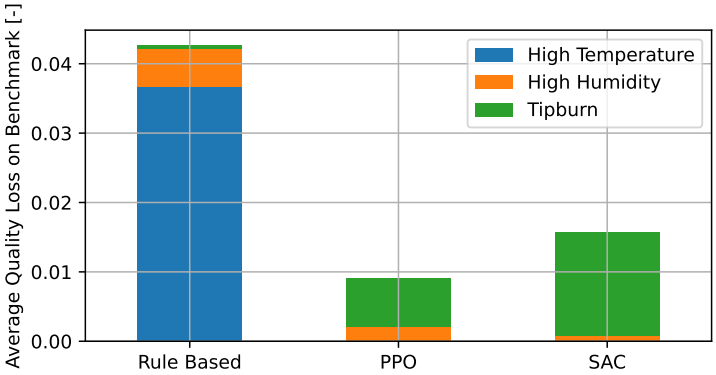


Figure 4.16: Fraction of plant quality loss breakdown into damage categories

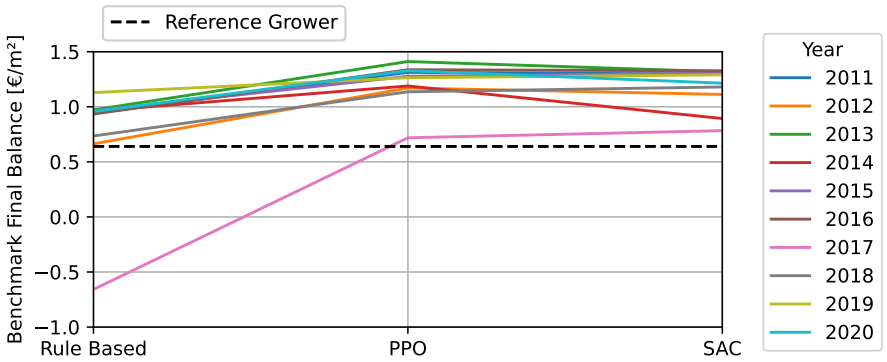


Figure 4.17: Final balance for all benchmark episodes split by year and algorithm.

Finally, in Figure 4.18 we observe the action distributions for all 6 actions for the rule based agent, PPO and SAC. We observe that PPO and SAC are similar, while the rule based agent shows much different distributions for the CO_2 Flow Rate action and the lamps action. In the CO_2 flow rate action our rule based agent only activates the carbon flow when it drops below a certain level. Therefore, the value is often 0. For the lamps, the optimization of the rule based parameters resulted in not using the lamps at all. If we assume that the optimization of the rule based agent indeed resulted in an optimal set of hyperparameters, and that the application of lamps is required for a good result, then we can conclude that our set of rules was not sufficient for the problem.

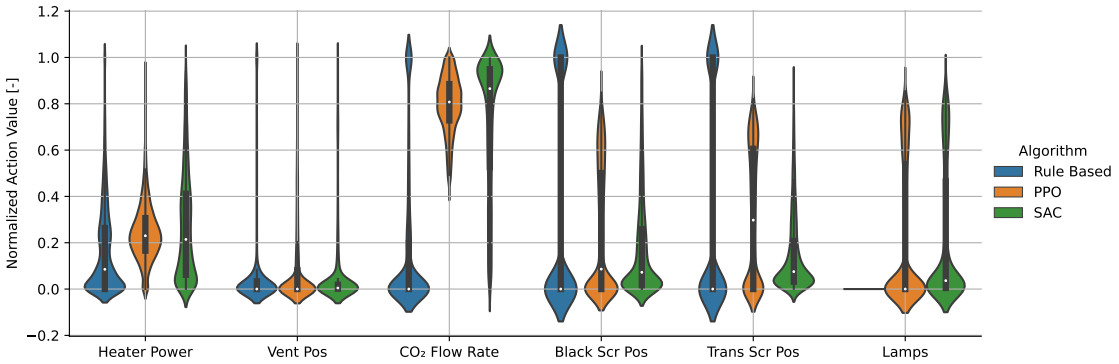


Figure 4.18: Action distribution plot that shows the distribution of selected actions on the benchmark. All action values are clipped in the [0, 1] range.

4.3.2 Strategy Comparison

In this section, we will analyze the decisions of the policies using two time series plots that show growing cycles of two different years that are part of the evaluation benchmark. From the benchmark results, we select the worst year and the best year in average final balance for the different algorithms. 2017 is the worst year with an average final balance of 0.28 $EUR\ m^{-2}$, while 2013 is the best year with an average final balance of 1.23 $EUR\ m^{-2}$.

A time series plot is shown for both the best and the worst year. We show the decisions and state of the policies for the best and worst year in Figure 4.19 and Figure 4.20. In these figures, we show the first 14 days of the growing cycle.

In our greenhouse simulator, we model variable electricity prices. In both Figure 4.19 and Figure 4.20 we observe that near the end of day, PPO and SAC both start using approximately

70% of the lighting, which is approximately 175 $\mu\ mol\ s^{-1}$ until early in the morning for every day. It learned to exploit low energy prices to provide extra light to the plants.

In both figures the rule based agent applies strong ON-OFF control on the transparent and blackout screens, while PPO and SAC activate the screens more gradually over time. The rule based agent applies the screens at night for increased insulation. As the application of screens during the night has no downsides, and PPO and SAC do not fully apply them fully the policies could be improved.

Both PPO and SAC supply more carbon dioxide to the greenhouse than the rule based agent does. When photosynthesis occurs we observe a sharp drop in carbon dioxide level for all agents. As carbon dioxide is a vital part of photosynthesis and it is inexpensive, PPO and SAC learn to supply large quantities of it.

Ventilation is applied in seemingly similar quantities by all algorithms. However, from Figure 4.16 we know that the rule based agent uses it adequately. Insufficient cooling results in temperature damage caused by the rule based policy.

The rule based policy is more effective in keeping the temperature in the greenhouse high, which we can observe from the temperature parameter in both figures. The rule based agent and SAC show more variance in the heater power applied. PPO applies more consistent heating over the day.

In Figure 4.20 we observe that at the start of the growing cycle, the air temperature in the greenhouse is low, while the quantity of gas used is high. Large gas expenses result in high costs, which hampers lettuce growth.

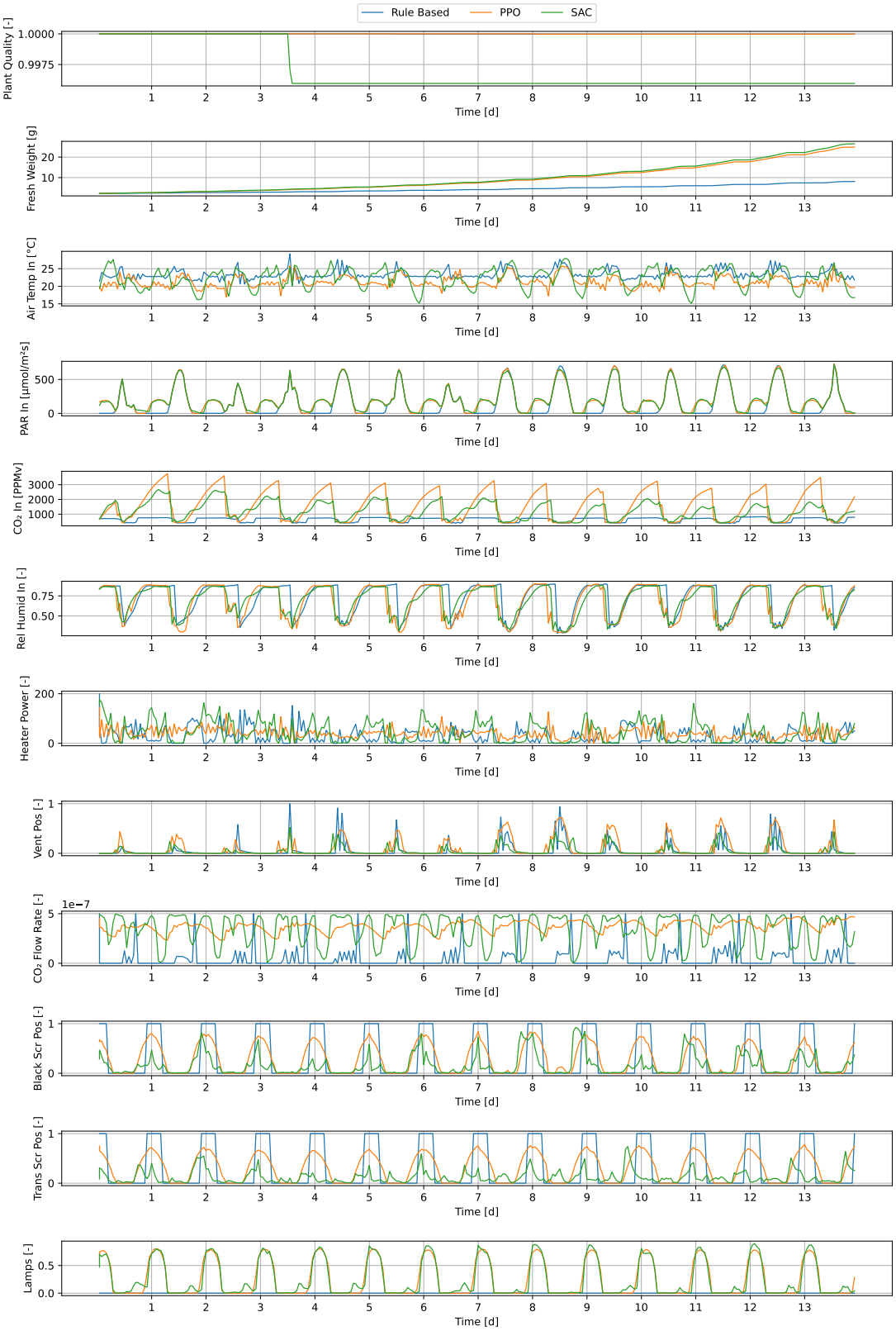


Figure 4.19: First 14 days of the best case benchmark growing cycle in 2013. Shows the evolution of actions and a greenhouse state subset over time for the PPO, SAC and rule based agents.

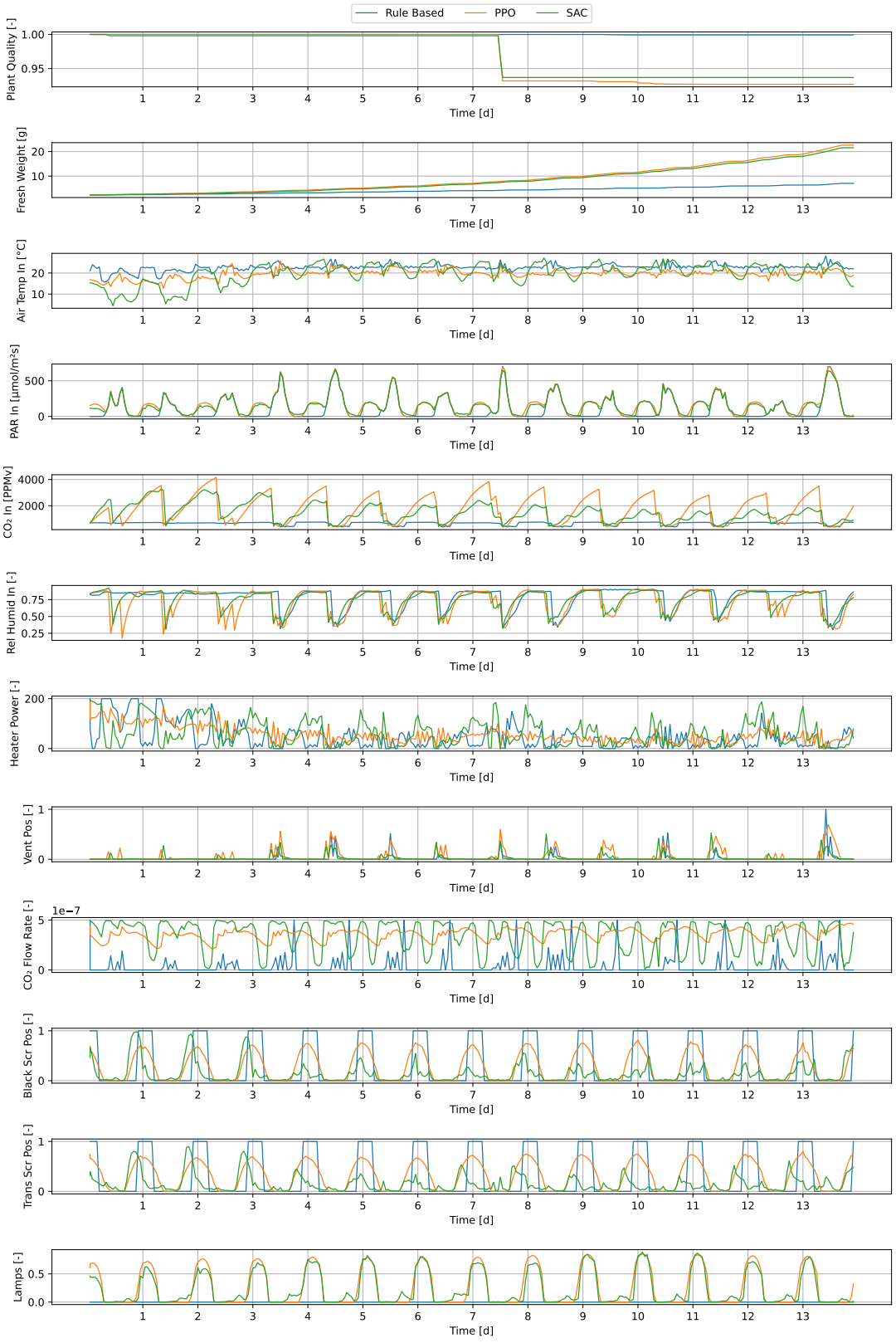


Figure 4.20: First 14 days of the worst case benchmark growing cycle in 2013. Shows the evolution of actions and a greenhouse state subset over time for the PPO, SAC and rule based agents.

4.4 Conclusions

In this chapter, we have shown how PPO and SAC can be optimized to outperform traditional rule based climate control systems. Also, we know that the policies can be improved. An example of this is that the screens were not fully deployed at night, which we have seen in subsection 4.3.2.

While PPO has not shown a high dependency on hyperparameters, while SAC did show this for the reward scaling parameter. Reward scaling affects the balance between entropy and exploitation of the environment, similar to the trade off coefficient α, as seen in subsection 2.2.3. SAC shows more unstable training compared to PPO. When SAC its exploration is disabled, there is a high variety in performance between different checkpoints. We do however use this variant, as the performance is better than the exploring variant.

An upside of SAC is its sample efficiency. While the performance of PPO nearly flattens after 10M frames, in SAC performance flattens after 250K frames.

Furthermore, our BalanceReward and DailyBalanceReward functions have proven to be a great improvement over the FinalBalanceReward. We suspect this is because it applies rewards for its actions directly or almost directly.

We observed how both the PPO and SAC algorithms produce similar policies. The resulting policies cause PPO and SAC to produce similar cost breakdowns, energy breakdowns, and quality loss sources.

The rule based policy shows a different signature, mostly as the decisions are different from PPO and SAC because of the way the rules are set up. Optimization of the rules resulted in not using lamps at all. Also, we observe a mismatch between the literature based grower reference and the rule based agent.

Chapter 5

Policy Robustness

In this chapter, we analyze the robustness of the trained algorithms. Robustness is a broad concept and therefore we will use the following scope: We define robustness as a high worst case performance across a variety of greenhouses. In section 5.1 we will explore the sensitivity and weaknesses of PPO, SAC and our rule based agent. Afterwards we will formalize our robustness criteria used in the experiments. We explain what a variety of greenhouses entails and how we measure worst case performance.

In addition to analyzing the robustness of the resulting agents from chapter 4, we analyze the effect of RARL, physics randomization and Gaussian observation noise. The goal is to show if the methods have a significant impact on the worst case performance and therefore robustness of the agent. As we need to discover the optimal hyper parameters for the solutions we start with a trend analysis. Finally, we select the most promising solutions and the respective parameters, and retrain them 5 times to show if the results are significant.

5.1 Sensitivity Visualized

In the competing solutions of chapter 4 both training and evaluation occurred in the exact same replica of the greenhouse. The only stochastic or randomized aspect was the weather, which was sampled from Dutch weather data in 2001 to 2010. To show the vulnerability to changes, we visualize the relative performance of the agent when evaluated in different greenhouses to the one it was trained on. This is shown for the climate model parameters in Figure 5.2 and for the plant model parameters in Figure 5.1.

In Figure 5.1 we observe that changes in the plant model parameters can lead to large losses. These are clearly explainable. The humidity tolerance upper parameter determines the maximum relative humidity in the greenhouse, above which the plant quality decreases. A 25% decrease of the default value of 90% will result in a tolerance of 67.5% relative humidity. As the greenhouse is a humid environment, the climate almost always exceeds this value, leading to large plant quality degradation. The result is an unsaleable plant.

Additionally, we observe that the greenhouse climate model parameters in Figure 5.2 affect the final balance less than modifications in the plant model parameters in Figure 5.1.

We observe in both plots that the sensitivity of parameter changes for trained agents is not necessarily worse than for the rule based agent. For "heat exchange open window", "greenhouse heat capacity", "fraction sun heat absorbed", "price electricity on peak", "greenhouse height" and "growth carbon scalar", we find that SAC and PPO result in similar sensitivities, while the Rule Based agent shows a significant difference. This could be due to the similarity of the trained policies of PPO and SAC, as seen in subsection 4.3.2. We suspect that the sensitivity is explained by the policy to a large extent.

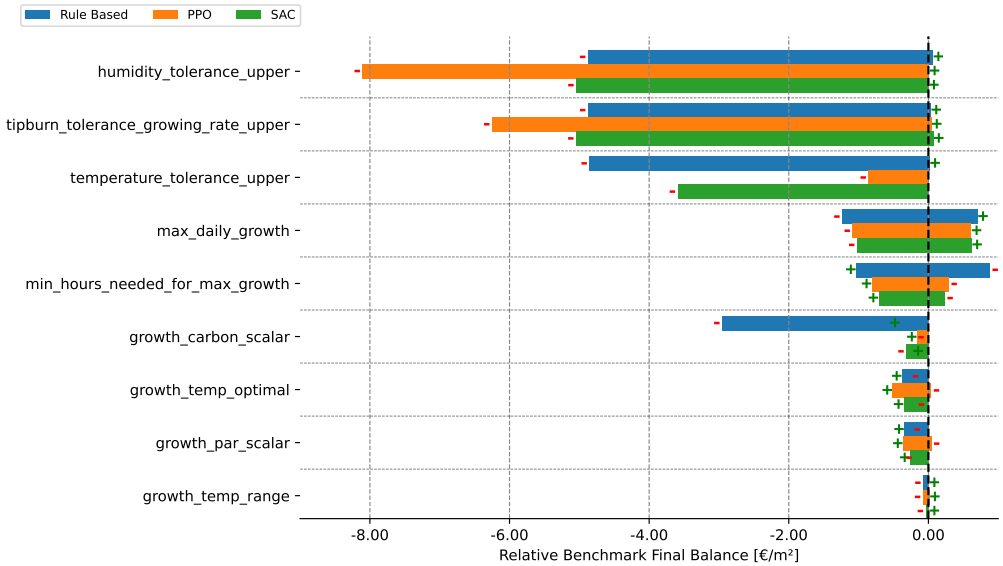


Figure 5.1: Tornado diagram where the agent is evaluated on variations of the training greenhouse. The black line indicates performance on the training greenhouse. The + sign denotes performance when increasing the parameter by 25% and the - sign denotes performance when decreasing the parameter by 25%. This plot shows only plant model parameters.

5.1.1 Worst Case Performance Criteria

At the start of this chapter, we defined robustness as a high worst case performance across a variety of greenhouses, as in [50]. In the heatmaps of [50, Figure 7] robustness is analyzed using various training environments with different simulation parameters. The MuJoCo environments [72] used for evaluation are varied using different friction coefficients and torso masses. To limit the scope, we decided to select 3 parameter pairs, of which each pair is from a different greenhouse subsystem. Heatmaps are effective visualizations for a multivariate analysis with two parameters at a time. The multivariate analysis is most interesting if the parameters in the pair are dependent. Using these criteria we selected the following pairs that describe a different subsystem each:

1. **Lights** - "lamp intensity" and "reflectance greenhouse"
2. **Heat** - "roof heat transfer rate" and "heat exchange open window"
3. **Growth** - "growth temp optimal" and "growth par scalar"

For each of the parameters we defined ranges shown in Table 5.1 from which we sampled 11 evenly spaced values. The center of each range is equal to the training greenhouse used to train PPO and SAC in 4. For each pair, this results in $11 \cdot 11 = 121$ different evaluation greenhouses, yielding 363 greenhouses in total. On each of the greenhouses, we will evaluate the agent using the evaluation benchmark described in subsection 3.2.3.

To express robustness in a single value, we will use the 5th percentile of the 363 greenhouse benchmark results. This value can be interpreted as follows: In 95% of the greenhouses the agent will deliver a final balance of at least this value. To maximize the robustness we are interested in increasing this value for worst case performance.

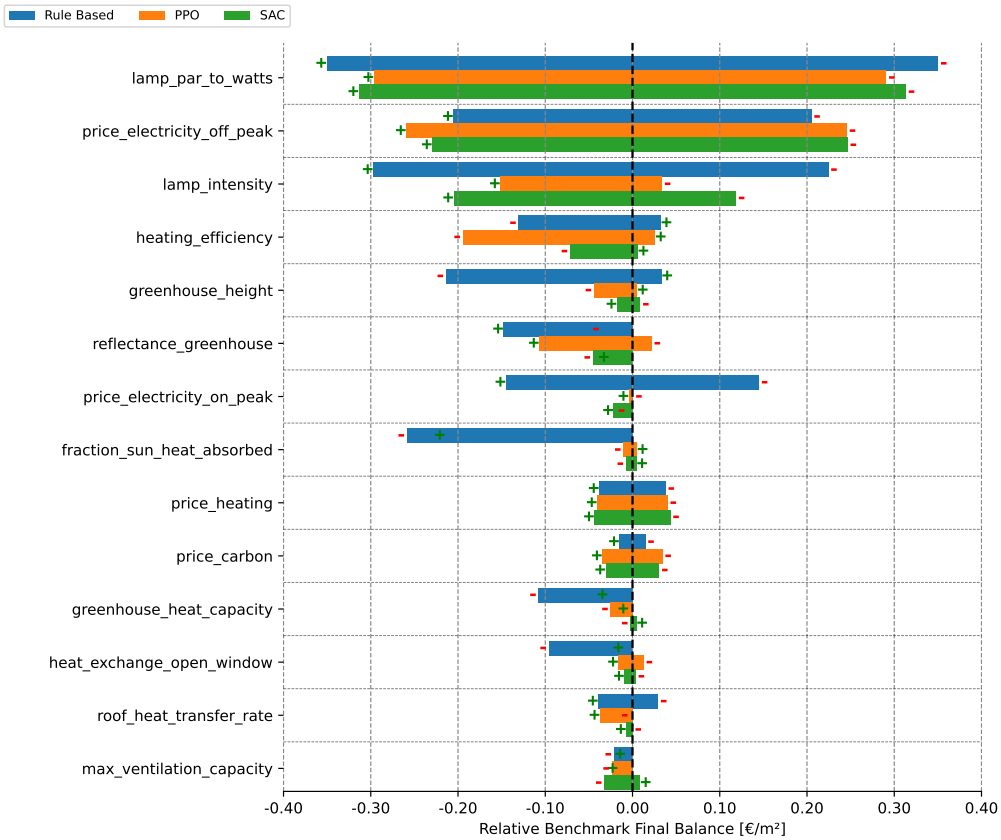


Figure 5.2: Tornado diagram where the agent is evaluated on variations of the training greenhouse. The black line shows performance on the training greenhouse. The + sign denotes performance when increasing the parameter by 25% and the - sign denotes performance when decreasing the parameter by 25%. This plot shows only climate model parameters.

Pair	Parameter	Range	Unit
Lights	lamp_intensity	[0, 500]	$\mu\ mol\ s^{-1}$
Lights	reflectance_greenhouse	[0, 0.800]	–
Heat	roof_heat_transfer_rate	[0.568, 1.704]	WK^{-1}
Heat	heat_exchange_open_window	[5.00, 15.0]	W
Growth	growth_par_scalar	[150, 250]	–
Growth	growth_temp_optimal	[18.75, 31.25]	K

Table 5.1: Parameter ranges used to sample 363 greenhouses for the multiple greenhouse benchmark.

5.2 Applied Algorithms and Methods

In this section we present the methods used in the worst case performance analysis. Our baseline consists of the rule based agent, the PPO agent, and the SAC agent trained in chapter 4. We apply two domain randomization methods from [47] and an adversarial method from [50]. Each of the methods is applied to both SAC and PPO. All of these methods have shown significant performance increases in the robotics scene, and therefore we hypothesize that it will yield similar improvements in our greenhouse environment.

5.2.1 Adversarial Attacks (RARL)

RARL, which we described in subsection 2.3.1, is the first solution applied. We created a wrapper that allows a second agent to participate in the greenhouse simulation environment, the adversarial agent. The adversarial agent has access to the adversarial action space, which is a modified action space that allows the agent to add or remove heat, humidity, par light and carbon dioxide from the environment. As described in subsection 2.3.1 both agents participate in the environment simultaneously using a different action space.

We need to tune the strength of the adversarial, which scales the amount of disturbances it can apply. An adversarial agent with n% strength can select actions with intervals scaled by n% of the default adversarial action space defined in Appendix section B.3. In section 5.3 we will show how different strengths for the adversary affect its performance. We run 5 trials, one at 100% strength, but also at 25%, 50%, 75% and 125% strength. This percentage is applied to the allowed action interval, meaning a 125% strength adversarial agent can remove or add 25% more heat than the 100% strength agent.

After training, we select only the protagonist agent for evaluation. We hypothesize that if the protagonist learns to obtain a high reward when the adversary is present, it can also handle other disturbances. We expect it will perform better on the multiple greenhouse benchmark.

5.2.2 Physics Randomization

The second solution is physics randomization of the greenhouse, as described in subsection 2.3.2. In this solution, we reinitialize the training greenhouse with randomly sampled parameters every time a training episode is finished. With over 1500 episodes of experience this should result in experience obtained from various greenhouses. Therefore, our hypothesis is that it learns how to perform well in multiple greenhouses instead of overfitting on a single greenhouse, and therefore our worst case performance should increase.

We will apply physics randomization to a subset of parameters, and we specify ranges from which the parameters are uniformly sampled. We search for the optimal set of parameters and ranges as defined in Table 5.2. In addition to the three options PR Low, PR Mid and PR High we create 3 more options by removing the growth parameters (growth_temp_optimal and growth_par_scalar) from the randomization. This creates 6 options: PR Low, PR Mid, PR High, PR Low NP (No Plant), PR Mid NP and PR High NP.

Parameter	PR Low		PR Mid		PR High	
reflectance_greenhouse	[0.360,	0.440]	[0.320,	0.480]	[0.200,	0.600]
lamp_intensity	[225,	275]	[200,	300]	[125,	375]
roof_heat_transfer_rate	[1.02,	1.25]	[0.909,	1.36]	[0.570,	1.700]
heat_exchange_open_window	[9.00,	11.0]	[8.00,	12.0]	[5.0,	15]
growth_temp_optimal	[22.5,	27.5]	[20.0,	30.0]	[12.5,	37.5]
growth_par_scalar	[180,	220]	[160,	240]	[100,	300]

Table 5.2: Physics randomization parameter sampling ranges for re-initializing the greenhouse. We show 3 settings for increasing amounts of physics randomization, PR Low, PR Mid, PR High.

5.2.3 Gaussian Noise

The last solution is the application of Gaussian noise on the observation during training as described in subsection 2.3.2. Here we apply noise to the normalized observation where $N \sim \mathbb{N}(0, \sigma)$. The observation is then clipped to prevent forming values outside the $[0, 1]$ bounds. We optimize the standard deviation by performing a grid search on the following values $\sigma = 0.001, 0.005, 0.01, 0.02, 0.05$. We hypothesize that the noise should put the agent in a seemingly

new situation during training. If it can react to this appropriately it can map a wider range of observations to the appropriate action, which increases robustness.

5.3 Trends in Results

For each of the proposed methods we trained the agents, and afterwards we evaluated the resulting agents on variations on the three parameter pairs described in section 5.1. For each greenhouse in the multiple greenhouse benchmark we obtained a single final balance value, yielding 363 data points per trained agent. The performance of the resulting agents is shown in Figure 5.3.

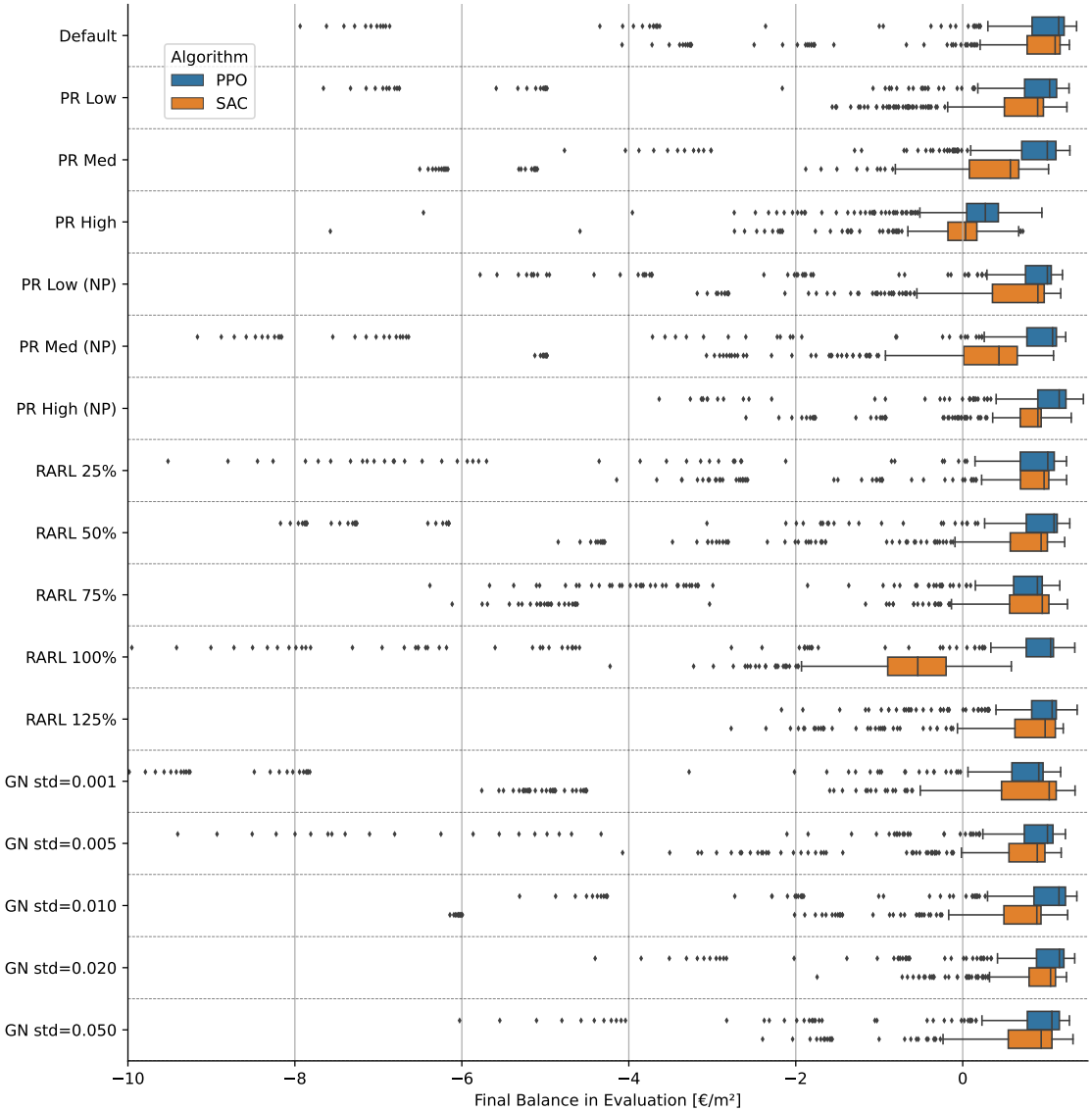


Figure 5.3: Performance for the SAC and PPO agent the applied adaptations. Each box plot is formed by evaluation on the multiple greenhouses benchmark, yielding 363 results. In the plot, GN stands for Gaussian noise and PR for physics randomization.

In Figure 5.3 we observe multiple effects between PPO and SAC. In 15 out of 17 cases the median performance of PPO is better than SAC while in the original work of [26] SAC outperforms

PPO. A reason for this difference could be that we performed almost twice as many experiments to optimize PPO compared to SAC.

In 14 out of 17 cases the worst case performance of SAC is better than the worst case performance of PPO. We suspect this is caused by two effects in SAC: its state-action value function in combination with its replay buffer and its entropy term in the objective function that leads to exploration. This gives SAC a more diverse experience over PPO, and therefore it may perform better in unexpected situations.

For physics randomization we observe mixed results. For the case where also plant parameters are randomized the median performance of both agents is significantly affected. For the no plant case results appear to improve when we apply high randomization levels. However, as medium randomization levels show worse results, we are not convinced of its effectiveness.

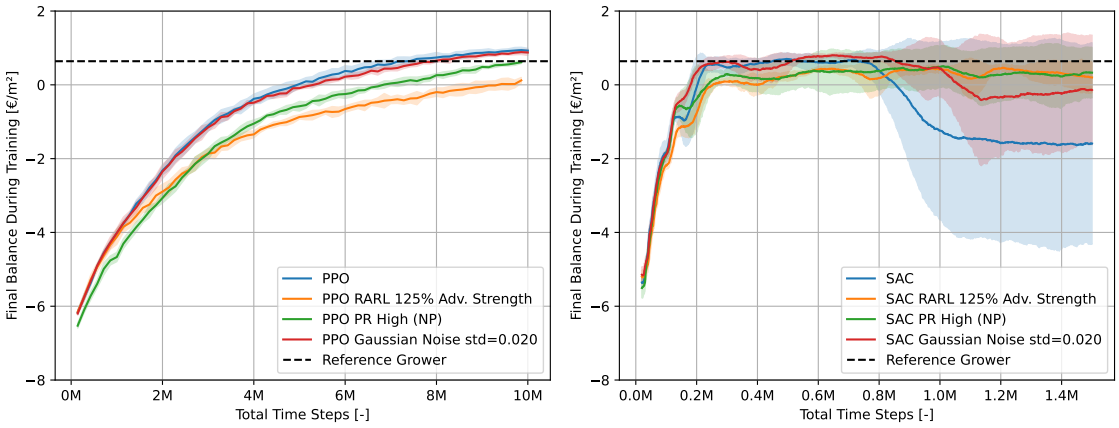
RARL shows a decreasing worst case performance similar to the results presented in the original paper [50]. We do notice that at a strength level of 100% both the SAC model shows worse median performance compared to the default SAC. As all the other RARL agents do not show this behavior we suspect this is an outlier. All other entries show no decrease of median performance, while an increase in worst case performance is visible. RARL at 125% strength shows the highest worst case performance.

For Gaussian noise we observe a parabolic trend in the worst case performance for both SAC and PPO, while medians values remain roughly equal. The optimal worst case performance uses a standard deviation of $\sigma = 0.020$.

From this trend analysis we conclude that both RARL and Gaussian noise show convincing evidence that the worst case performance is positively affected. For physics randomization we observe no clear trend, but we observe that high physics randomization without modifying the plant model performs best (PR High (NP)). We select Gaussian noise with $\sigma = 0.020$, RARL with strength 125% and physics randomization PR High NP for further analysis.

5.4 Result Comparison

In this section we compare PPO and SAC to their adaptations, as well as to the rule based agent. To show the results are of statistical significance we train 5 independent agents which we refer to as trials. We first trained the agents, of which we show the obtained reward during training in Figure 5.4. Note that this is not the final balance on the evaluation benchmark, as obtaining this metric doubles the training time.



(a) PPO (b) SAC

Figure 5.4: Comparison of training final balance versus training time steps for all trained solutions.

We trained PPO for 10M frames, after which the default implementation, the Gaussian noise

adaptation and RARL nearly reach a plateau. The physics randomization variant shows the highest slope at 10M frames, which shows that more performance could be gained with prolonged training. As physics randomization adds stochasticity to the environment the required increase in time steps for optimal performance is not unexpected. Note that the performance of RARL is affected by the adversarial agent during training, which leads to a lower reward.

We observe stable training for PPO and sometimes unstable training for SAC. We could not think of a reason for the instability, but it typically appears when training after reaching the performance plateau.

5.4.1 Heatmap Analysis

In this section we perform a qualitative analysis in the form of heatmaps, similar to [50, Figure 7]. For this, we use a single agent trial, as averaging over multiple trials could make the shapes of the heatmaps less explainable due to smoothing.

Baselines

We show the performance of our baselines in heatmaps in Figure 5.5.

In Figure 5.5 all agents experience performance loss for some parameter values. We observe how the shapes of the heatmaps for PPO and SAC are more similar in shape compared to the Rule Based agent, which can be explained by the similarity of the policies. PPO and SAC experience sudden performance loss for lamp intensities above 400 $\mu mols^{-1}$. In Figure 4.16 we observed how the SAC and PPO policies suffered from tipburn, which is caused by rapid growth. Increasing lamp intensities will increase the tipburn rate, hence the performance drop. Tipburn does not affect the rule based agent, which explains why we do not observe a similar sharp performance drop.

The rule based policy shows plant damage caused by high temperatures. This shows that our rule based policy does not use sufficient ventilation. This should be stronger for greenhouses with a low window heat exchange, and in Figure 5.5 we observe that for these greenhouses the rule based agent indeed performs worse.

The worst case performance for PPO and SAC approaches the -4.00 $EUR\ m^{-2}$ while this is only -0.25 $EUR\ m^{-2}$ for the rule based agent. However, in the best case scenario we observe that the profits for the PPO and SAC agents can reach 1.25 $EUR\ m^{-2}$ whereas the rule based agent only results in approximately 0.80 $EUR\ m^{-2}$.

Note for each greenhouse the maximum achievable final balance differs. A noteworthy example is the greenhouse at the bottom right for the lights parameter pair heatmap. This greenhouse has a maximum lamp intensity of 0, while it passively reflects 80% of the incoming light.

Adaptations

In this section we show heatmaps for our PPO and SAC baselines as well as their adaptations. In Figure 5.6 and Figure 5.7 we show the comparison for PPO and SAC respectively.

For both SAC and PPO, we observe that all adaptations reduce the sharp performance drop for the plots related to lights.

Most adaptations show similar shapes in the heatmaps compared to the baselines, except on the growth plot for SAC PR High (NP), which shows a different pattern. As most plots retain the same shape, we suspect the policies have not changed much. However, the policies show stronger robustness, as the performance drops reduces in sharpness.

RARL improves the results especially in the lights plots, and does not affect the heat plots by a large amount. In the growth plot we see a small increase in sensitivity. However, this effect is outweighed by the large benefits gained in the sensitivity against the lights.

From the heatmaps we observe how Gaussian noise is the least improving solution for PPO, while it is the best solution for SAC. A reason for this could be that the plots show a single trained agent, which is susceptible to outliers.

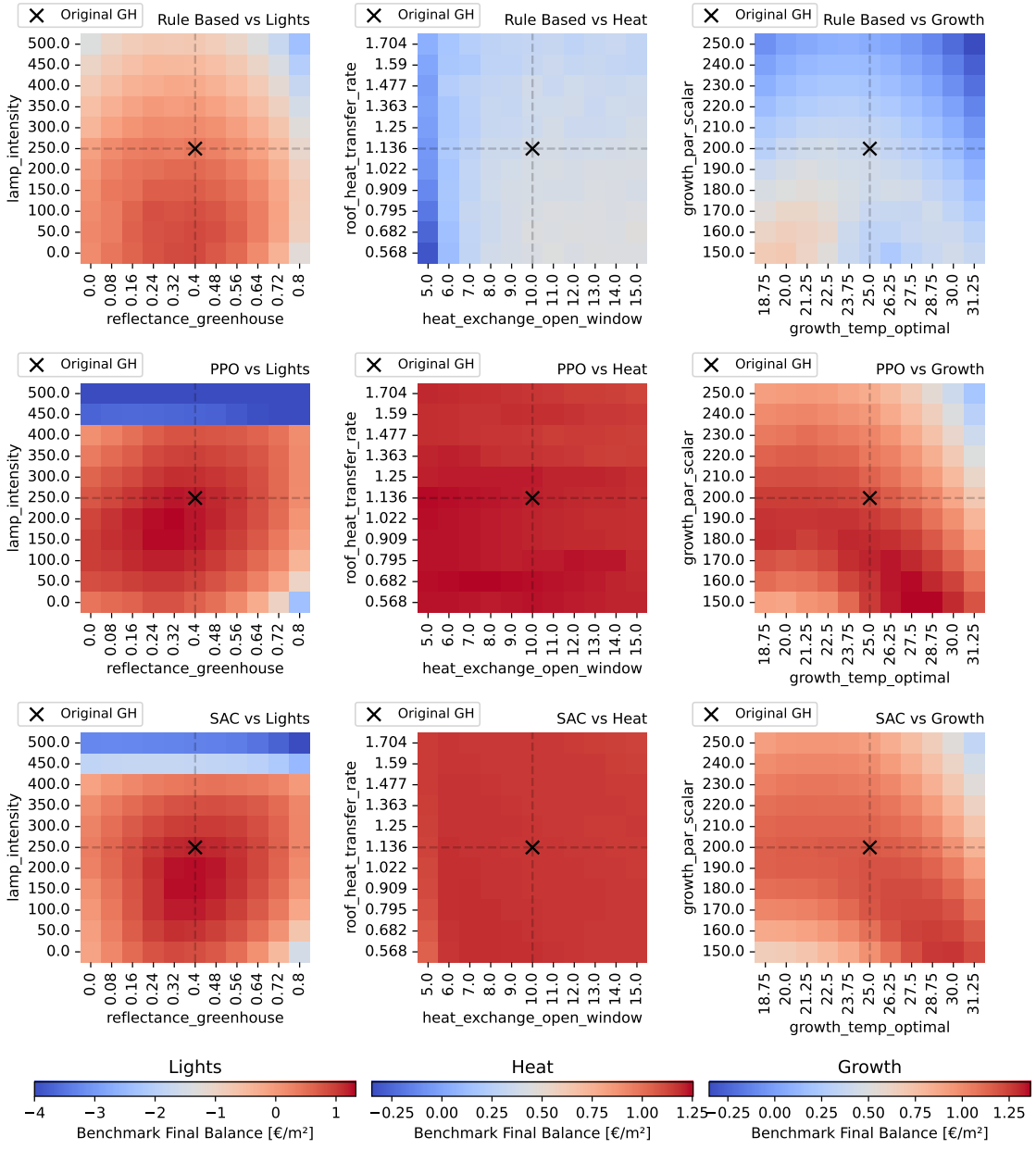


Figure 5.5: Results of the multiple greenhouse benchmark for the rule based, SAC and PPO agents visualized in heatmaps. On each axis, we show the parameters for the evaluation greenhouses. Each small square shows the evaluation performance in a single greenhouse. Each column shows one parameter pair and has its own color bar. Each row shows a different algorithm. In each heatmap a cross denotes the training greenhouse.

5.4.2 Worst Case Performance Analysis

For each of the 5 agents trained per PPO and SAC adaptation, we evaluate the agent on the benchmark for each of the 363 greenhouse variations shown in section 5.1. We then calculate the 5th percentile of all final balance results. We will apply the Mann Whitney U test to show if the sample mean of the worst case performance for each of the adaptations results in an improvement over the default PPO and SAC algorithms. We also considered ANOVA, but we cannot show that

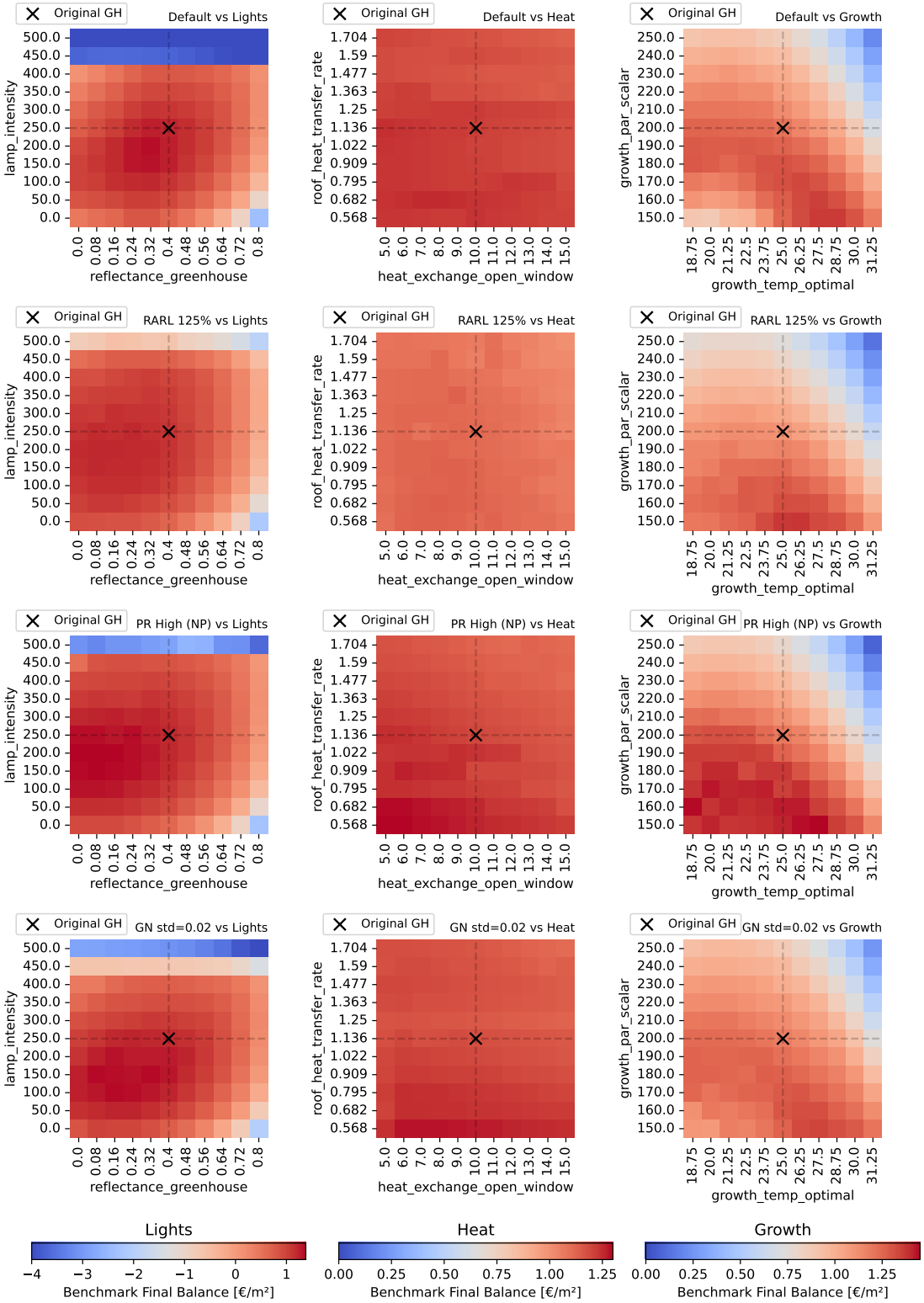


Figure 5.6: Heatmap of the multiple greenhouse benchmark result for PPO and its adaptations. The axes show the evaluation greenhouse parameters and each small square shows the evaluation result of that greenhouse. The columns show a parameter pair and a color bar. The rows show different algorithms. Each black cross denotes the training greenhouse.

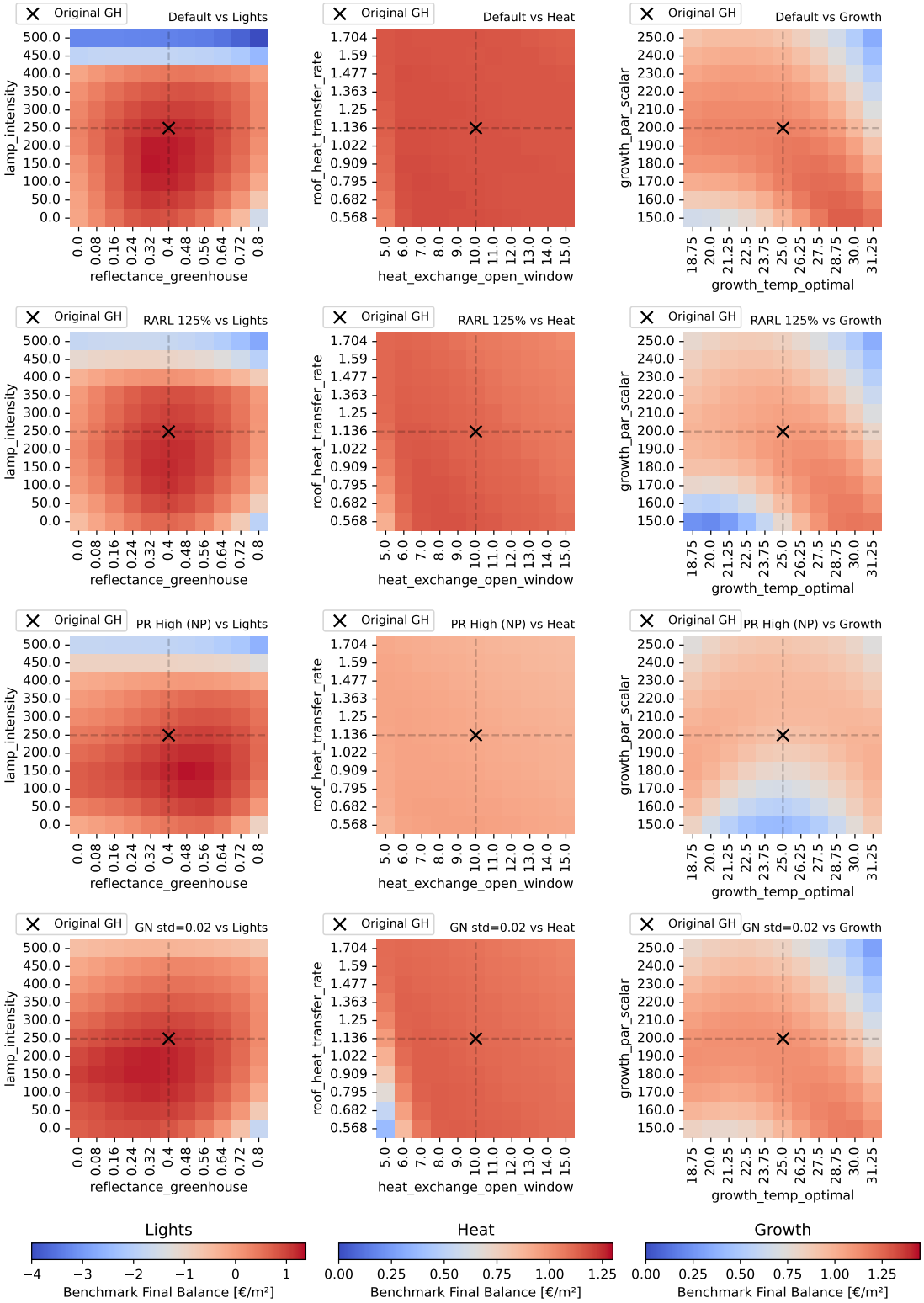


Figure 5.7: Heatmap of the multiple greenhouse benchmark results for SAC and its adaptations. The axis show the evaluation greenhouse parameters and each small square shows the evaluation result of that greenhouse. The columns show a parameter pair and a color bar. The rows show different algorithms. Each black cross denotes the training greenhouse.

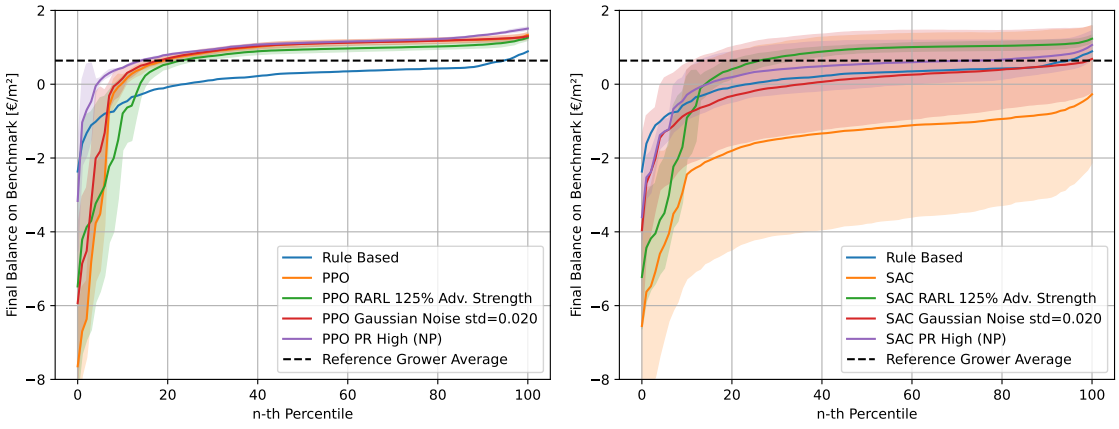
the samples of size 5 are normally distributed with significant statistical power. If the samples are not normally distributed, we violate an assumption, which makes the conclusion unreliable. The Mann Whitney U Test does not require normally distributed samples, and therefore we do not violate its assumptions.

The results for the Mann Whitney U Tests for $\alpha = 0.05$ are shown in Table 5.3. Using $\alpha = 0.05$ we find that physics randomization significantly increases the worst case performance of SAC and PPO, and that Gaussian noise increases the worst case performance of SAC. Note that Gaussian Noise for PPO nearly shows a significant improvement too. We also observe how the sample mean for worst case value of all solutions is less than the sample mean for the baselines. Overall physics randomization works best, followed by Gaussian noise. RARL does not significantly affect the worst case performance.

Solution	Baseline Mean n=5	Solution Mean n=5	U	P
SAC RARL 125% Adv. Strength	-4.35	-3.49	8	0.2017
SAC Gaussian Noise std=0.020	-4.35	-1.29	4	0.0473
SAC PR High (NP)	-4.35	-1.28	2	0.0184
PPO RARL 125% Adv. Strength	-3.51	-3.00	12	0.5000
PPO Gaussian Noise std=0.020	-3.51	-1.82	5	0.0718
PPO PR High (NP)	-3.51	0.11	0	0.0061

Table 5.3: Results for an one-sided Mann Whitney U Test comparing the sample mean 5th percentile performance for both PPO and SAC to their respective adaptations.

A more detailed view of all nth performance percentiles is shown in Figure 5.8. In this plot, we observe that PPO, PPO RARL and PPO Gaussian noise perform badly in the 0 to 10th percentile, while the physics randomization adaptation and the rule based agent show better performance. From the 15th percentile upwards, all PPO solutions outperform the rule based agent by approximately 1 $EUR\ m^{-2}$. For SAC, we observe that the worst case performance is less severe than PPO. Also, we observe how its adaptations show more stability in the result. Physics randomization shows the best worst case performance of all SAC adaptations, while also performing well from the 15th percentile upwards.

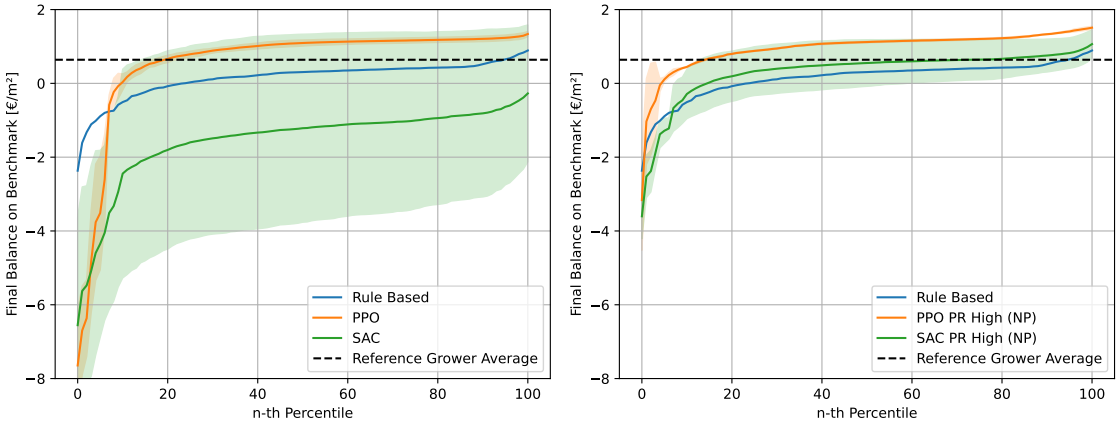


(a) PPO (b) SAC

Figure 5.8: Comparison of the n-th percentile performance on the benchmark for the PPO and SAC baselines as well as the proposed solutions. In both plots, we show a single sample standard deviation.

In Figure 5.9 we observe how physics randomization shows a significant increase in robustness

of PPO and SAC in the worst case, while also increasing the performance in the average and best case. While our rule based solution shows much better worst case performance than the original PPO and SAC agents, our adaptation shows much better worst case performance for PPO and comparable worst case performance for SAC.



(a) Original performance

(b) Application of physics randomization

Figure 5.9: Comparison of the n-th percentile performance on the benchmark, before and after application of physics randomization. In both plots we show a single sample standard deviation for PPO and SAC. The Rule Based agent is deterministic and therefore has no standard deviation.

5.5 Conclusions

In this chapter, we have shown that our trained agents of chapter 4 are less robust than the rule based agent. To achieve this, we defined a metric for robustness using multiple evaluation greenhouses, and a metric for worst case performance. We proposed three methods to address the robustness deficiency of the agent: RARL, Gaussian observation noise and physics randomization. Each of these methods is independent of the reinforcement learning algorithm used. Out of the 3 methods, we have shown that physics randomization is an effective way to significantly boost robustness. Additionally, Gaussian noise significantly improved robustness for the SAC agent. While all other techniques have shown an improvement in robustness over the robustness over the original versions of PPO and SAC, they did not show significant improvement.

Chapter 6

Conclusions

In this work we have shown how reinforcement learning can be applied to robustly optimize grower profits. In chapter 3 we presented our simulator design, which produces results that in some aspects are reasonable close to the reputable INTKAM / KASPRO simulator combination. We applied this simulator model to train agents to optimize the profits of the grower. We have shown how our solutions outperform the baselines in terms of grower profits on the evaluation benchmarks. We have also shown empirically that reinforcement learning agents are less robust in terms of worst case performance. We implemented 3 methods to improve the worst case performance of PPO and SAC. Of these 3 methods, physics randomization in the greenhouse environment has shown significant improvement of the worst case performance for both PPO and SAC, while Gaussian noise on the observation has shown an improved worst case performance on SAC. While average case performance is unaffected, best case performance is also improved by applying physics randomization. We will now answer our three research questions:

R1. How can reinforcement learning be applied to maximize grower profit?
In chapter 4 we have shown how PPO and SAC can be applied to outperform our rule based baseline based on state-of-the-practice grower techniques, as well as the average grower profit baseline that we derived from literature. Both SAC and PPO produce similar policies and both learn how to grow the lettuce optimally while minimizing costs. On our standardized benchmark, our rule based baseline and the literature reference earn an average of 0.76 $EUR\ m^{-2}$ and 0.64 $EUR\ m^{-2}$, while our PPO and SAC agents earn 1.21 $EUR\ m^{-2}$ and 1.17 $EUR\ m^{-2}$ respectively. Additionally, we provide the full configuration and implementation of the reinforcement learning algorithms such that the results can be reproduced.

R2. Is there a difference in robustness of reinforcement learning algorithms compared to traditional control methods?
In chapter 5 we have shown how policies learned by PPO and SAC in one greenhouse can lead to high monetary losses in a different greenhouse. Both reinforcement learning policies show higher sensitivity to greenhouse changes than our state-of-the-practice rule based baseline.

R3. How can we make the performance of a reinforcement learning agent robust to differences between the training and evaluation environment?
To improve this performance we applied 3 different methods: Robust Adversarial Reinforcement Learning (RARL), physics randomization and Gaussian observation noise. We have not only demonstrated how physics randomization significantly improves the worst case performance for both PPO and SAC but also the best case performance. Additionally, Gaussian noise on the observations has shown a significant improvement in performance for SAC.

6.1 Limitations and Future Research

We have shown how robustness can be improved against greenhouses different from the training greenhouse. In this work we have defined robustness as worst case performance across various greenhouses. In reality, a reinforcement learning agent needs to be robust against sensor noise and bias, failure of sensors, and unmodeled effects.

As open source greenhouse simulator software is not publicly available to the best of our knowledge, we have designed and implemented our own greenhouse environment. While the environment produces results reasonably close to the reputable simulator pair INTKAM / KASPRO, it remains a simplified version that has not been validated using real data. Therefore, we would advise against applying agents trained in this environment in the real world, regardless of our promising worst case improvements.

Additionally, our simulator models a separate heater, carbon supply and electricity price, while modern greenhouses are equipped with combined heat and power systems. Such systems produce electricity, use the waste heat to warm the greenhouse, and use the carbon dioxide produced to provide carbon dioxide to the greenhouse in a sustainable way. To apply our methods in a modern greenhouse the simulator should be adapted.

Furthermore, we applied both PPO and SAC. However, we have performed more optimization experiments for PPO than for SAC. Therefore, one should be cautious when making a comparison between the performance of both algorithms in this work. Further optimization of SAC may improve its performance.

We observed instability in the training of SAC, while no instability was observed in the adaptations of SAC. This may have an impact on the validity of the robustness analysis. As PPO has shown no such instabilities, and we have drawn similar conclusions about the 3 robustness solutions, we see no reason to doubt the effectiveness.

We recommend investigating 3 potential issues before applying reinforcement learning algorithms. These issues have been listed below:

1. **Robustness against other sources of instability**
   We recommend investigating the other aforementioned potential sources that may destabilize the agent. We mentioned sensor noise and bias, failure of sensors and unmodeled effects. Chances of successful real world application increase when all these potential issues have been addressed.

2. **Safety guarantees**
   While we have shown how to increase the worst case performance of the agent we have no guarantees. Guarantees on safety of a black box algorithm may not only prevent issues, but also increase the trust of the grower.

3. **Simulator Realism**
   An increase in robustness is useful to maintain performance in the real world when subject to a small simulation to reality gap. However, we have not validated our simulator with respect to the real world, and therefore we cannot tell how large the gap to reality is. Improvement of the simulator and validation of its accuracy are critical steps towards real world application.

6.2 Recommendations

For any real world application of reinforcement learning, we recommend to apply Gaussian observation noise and physics randomization during training, as this significantly improves robustness against differences between the training and evaluation greenhouse. Both solutions require little implementation time and are independent of the reinforcement learning algorithm applied.

Furthermore, we recommend to investigate the 3 remaining points of research in item 3 before attempting to apply any reinforcement learning algorithm in reality.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. *ArXiv*, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/. 15

[2] Open AI. Introduction to rl: Part 2 kinds of rl algorithms [online]. 2022. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. 13, 14, 17

[3] Open AI. Open ai baselines [online]. 2022. URL: https://github.com/openai/baselines. 14

[4] Open AI, A. Hill, A. Raffin, M. Ernestus, A. Gleave, and A. Kanervisto. Stable baselines [online]. 2022. URL: https://github.com/hill-a/stable-baselines. 14

[5] Vyacheslav Alipov, Riley Simmons-Edler, Nikita Putintsev, Pavel Kalinin, and Dmitry Vetrov. Towards practical credit assignment for deep reinforcement learning, 2021. URL: https://arxiv.org/abs/2106.04499, doi:10.48550/ARXIV.2106.04499. 12

[6] Zhicheng An, Xiaoyan Cao, Yao Yao, Wanpeng Zhang, Lanqing Li, Yue Wang, Shihui Guo, and Dijun Luo. A simulator-based planning framework for optimizing autonomous greenhouse control strategy. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1):436–444, May 2021. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15989. 10

[7] Anne-Peter Alberda, Arjen Berkenbos (DNB), Chris de Blois, Timon Bohn, Sarah Creemers, Hans Draper, Eva Hagendoorn (DNB), Marjolijn Jaarsma, Bart Loog, Tom Notten, Tim Peeters, Leen Prenen, Janneke Rooyakkers, and Khee Fung Wong. Dutch Trade in Facts and Figures 2021: Exports, imports and investment - An introduction. Technical report, Centraal Bureau voor de Statistiek, The Hague, Netherlands, November 2021. URL: https://longreads.cbs.nl/dutch-trade-in-facts-and-figures-2021/dutch-trade-in-facts-and-figures-2021-exports-imports-and-investment-an-introduction/.

[8] Alan A. Author, Bill B. Author, and Cathy Author. Title of article. *Title of Journal*, 10(2):49–53, 2005. 1

[9] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning*

Research, pages 507–517. PMLR, 13–18 Jul 2020. URL: https://proceedings.mlr.press/v119/badia20a.html. 2, 14

[10] Byunghyun Ban and Soobin Kim. Control of nonlinear, complex and black-boxed greenhouse system with reinforcement learning. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 913–918, Jeju, October 2017. IEEE. URL: http://ieeexplore.ieee.org/document/8190813/, doi:10.1109/ICTC.2017.8190813. 2, 10

[11] BBC. Factors affecting photosynthesis - photosynthesis - aqa - gcse combined science revision - aqa trilogy - bbc bitesize [online]. URL: https://www.bbc.co.uk/bitesize/guides/zs4mk2p/revision/2. 6, 7, 83

[12] BBC. Gas exchange in plants - gcse - bbc bitesize [online]. URL: https://www.bbc.co.uk/bitesize/guides/zs4mk2p/revision/2. 6

[13] Marc Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, 07 2012. doi:10.1613/jair.3912. 2

[14] C. Borgnakke and R.E. Sonntag. *Fundamentals of Thermodynamics, 8th Edition*. Wiley, 2012. URL: https://books.google.nl/books?id=HUUcAAAAQBAJ. 85, 86

[15] M. Brechner and A.J. Both. Hydroponic Lettuce Handbook, 2013. 9, 30

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. arXiv:1606.01540. 4, 20

[17] J.C.A.M. Buijs. Mapping Data Sources to XES in a Generic Way. Master's thesis, Eindhoven University of Technology, 2010.

[18] Graham Collier and Theodore Tibbitts. *Tipburn of Lettuce*, volume IV, pages 49 – 65. The AVI Publishing Company Inc., 02 1982. doi:10.1002/9781118060773.ch2. 7

[19] FH de Zwart. Simulatiemodel kaspro, 2022. URL: https://www.wur.nl/nl/show/Simulatiemodel-KASPRO-1.htm. 10

[20] H.F. de Zwart. *Analyzing energy-saving options in greenhouse cultivation using a simulation model*. PhD thesis, Wageningen University & Research, 1996. WU thesis 2071 Proefschrift Wageningen. 26

[21] Piotr Dąbrowski, Magdalena Kusaka, Izabela Samborska, and Hazem Kalaji. Measuring light spectrum as a main indicator of artificial sources quality. *Journal of Coastal Life Medicine*, 3:400–406, 05 2015. doi:10.12980/JCLM.3.2015J5-25. 6

[22] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL: https://proceedings.mlr.press/v80/fujimoto18a.html. 14

[23] Vicki Funk, R. Bayer, Sterling Keeley, Raul Chan, Linda Watson, Birgit Gemeinholzer, Edward Schilling, J. Panero, Bruce Baldwin, Núria Garcia-Jacas, Alfonso Susanna, and Robert Jansen. Everywhere but antarctica: Using a supertree to understand the diversity and distribution of the compositae. *Proceedings of a Symposium on Plant Diversity and Complexity Patterns*, 55:343–373, 01 2005. 4

[24] Inc. GitHub. Github: Where the world builds software [online]. 2022. URL: http://www.github.com. 2

[25] Luuk Graamans, Esteban Baeza, Andy van den Dobbelsteen, Ilias Tsafaras, and Cecilia Stanghellini. Plant factories versus greenhouses: Comparison of resource use efficiency. *Agricultural Systems*, 160:31–43, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0308521X17307151, doi:https://doi.org/10.1016/j.agsy.2017.11.003. 86

[26] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018. URL: http://arxiv.org/abs/1801.01290. 4, 13, 14, 15, 17, 32, 39, 40, 52

[27] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016. 13

[28] Silke Hemming, Feije de Zwart, Anne Elings, Isabella Righini, and Anna Petropoulou. Remote Control of Greenhouse Vegetable Production with Artificial Intelligence-Greenhouse Climate, Irrigation, and Crop Production. *Sensors*, 19(8), April 2019. URL: https://www.mdpi.com/1424-8220/19/8/1807, doi:10.3390/s19081807. 2, 10

[29] Silke Hemming, Feije de Zwart, Anne Elings, Anna Petropoulou, and Isabella Righini. Cherry Tomato Production in Intelligent Greenhouses-Sensors and AI for Control of Climate, Irrigation, Crop Yield, and Quality. *Sensors*, 20(22):6430, November 2020. URL: https://www.mdpi.com/1424-8220/20/22/6430, doi:10.3390/s20226430. 2, 10

[30] JR JONES. *Complete guide for growing plants hydroponically*. CRC Press, 2017. 7, 10

[31] David Katzin, Leo F.M. Marcelis, and Simon van Mourik. Energy savings in greenhouses by transition from high-pressure sodium to led lighting. *Applied Energy*, 281:116019, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0306261920314628, doi:https://doi.org/10.1016/j.apenergy.2020.116019. 9

[32] Robiul Kawser, Md Hossain, and MST Yeasmin. Optimizing planting density of lettuce (lactuca sativa) with tilapia (oreochromis niloticus). https://www.researchgate.net/publication/316715713_Optimizing_planting_density_of_lettuce_Lactuca_sativa_with_tilapia_Oreochromis_niloticus, 01 2016. 8

[33] Beom-Su Kim, Sungkwan Youm, and Yong-Kab Kim. Sterilization of harmful microorganisms in hydroponic cultivation using an ultraviolet led light source. *Sensors and Materials*, 32:3773, 11 2020. doi:10.18494/SAM.2020.2979. 8

[34] KNMI. Daggegevens van het weer in nederland [online]. 2022. URL: https://www.knmi.nl/nederland-nu/klimatologie/daggegevens. 21, 88

[35] T. Kozai, J. Goudriaan, and T. Kimura. *Light transmission and photosynthesis in greenhouses*. Simulation monographs. Pudoc, 1978. 85

[36] Wouter Jacobus Peter Kuijpers. *Model Selection and Optimal Control Design for Automatic Greenhouse Climate Control*. PhD thesis, Mechanical Engineering, March 2021. Proefschrift. 2, 10

[37] Robert W. Langhans and T. W. Tibbitts. *Plant Growth Chamber Handbook*. Iowa Agricultural and Home Economics Experiment Station, 1997. 85

[38] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. *arXiv:1712.09381 [cs]*, June 2018. arXiv: 1712.09381. URL: http://arxiv.org/abs/1712.09381. 14

[39] Cristian C. Millan-Arias, Bruno J. T. Fernandes, Francisco Cruz, Richard Dazeley, and Sergio Fernandes. A Robust Approach for Continuous Interactive Actor-Critic Algorithms. *IEEE Access*, 9, 2021. URL: https://ieeexplore.ieee.org/document/9493212/, doi:10.1109/ACCESS.2021.3099071.

[40] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL: https://proceedings.mlr.press/v48/mniha16.html. 14

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. doi:10.1038/nature14236. 13

[42] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017. URL: http://arxiv.org/abs/1712.05889, arXiv:1712.05889. 14, 31, 32

[43] Pere Muñoz, Assumpció Antón, M. Nuñez, Ashwin Paranjpe, J. Ariño, X. Castells, J.I. Montera, and Joan Rieradevall. Comparing the environmental impacts of greenhouse versus open-field tomato production in the mediterranean region. *Acta Horticulturae*, 801:1591–1596, 11 2008. doi:10.17660/ActaHortic.2008.801.197.

[44] United Nations. Paris agreement [online]. URL: https://treaties.un.org/pages/ViewDetails.aspx?src=TREATY&mtdsg_no=XXVII-7-d&chapter=27&clang=_en. 1

[45] Elly Nederhoff. Effects of co2 concentration on photosynthesis, transpiration and production of greenhouse fruit vegetable crops. *Prof.dr.ir. H. Challa (promotor). Landbouwuniversiteit Wageningen (1994) 213 pp.*, 01 1994. 6, 83

[46] G.M. Nugteren. Process Model Simplification. Master's thesis, Eindhoven University of Technology, 2010. URL: http://alexandria.tue.nl/extra1/afstversl/wsk-i/nugteren2010.pdf.

[47] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. *arXiv:1808.00177 [cs, stat]*, January 2019. URL: http://arxiv.org/abs/1808.00177. 4, 17, 18, 50

[48] Hiske Overweg, Herman N. C. Berghuijs, and Ioannis N. Athanasiadis. CropGym: a Reinforcement Learning Environment for Crop Management. *arXiv:2104.04326 [cs]*, April 2021. arXiv: 2104.04326. URL: http://arxiv.org/abs/2104.04326. 10

[49] H. Ozturk and Kaan Kucukerdem. Determination of heating requirements and energy consumption of greenhouses in adana region of turkey. *AgroLife Scientific Journal*, 5:157–160, 06 2016. 86

[50] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 06–11 Aug 2017. URL: https://proceedings.mlr.press/v70/pinto17a.html. 4, 17, 18, 49, 50, 53, 54

[51] Pedro Ponce, Arturo Molina, Paul Cepeda, Esther Lugo, and Brian MacCleery. *Greenhouse Design and Control*. Taylor & Francis Ltd, 09 2014. doi:10.1201/b17391. 3, 8, 9, 10

[52] John E. Preece and Paul E. Read. *The Biology of Horticulture: An introductory textbook*. John Wiley & Sons, 2005. 5, 6, 7, 27, 83

[53] Nederlandse Rijksoverheid. Klimaatakkoord [online]. URL: https://www.rijksoverheid.nl/documenten/rapporten/2019/06/28/klimaatakkoord. 1

[54] Aurko Roy, Huan Xu, and Sebastian Pokutta. Reinforcement Learning under Model Mismatch. *arXiv:1706.04711 [cs, stat]*, November 2017. URL: http://arxiv.org/abs/1706.04711.

[55] Marc Ruijs and Jan Benninga. *Market potential and investment opportunities of high-tech greenhouse vegetable production in the USA: An exploratory study for Midwest and East Coast regions and the state of California*. Number 2020-064 in Report / Wageningen Economic Research. Wageningen Economic Research, 2020. Project code 2282200573. doi:10.18174/526843. 1, 9, 30, 41

[56] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153187, 2021. doi:10.1109/ACCESS.2021.3126658. 2

[57] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020. doi:10.1038/s41586-020-03051-4. 14

[58] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL: https://proceedings.mlr.press/v37/schulman15.html. 14

[59] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016. arXiv:1506.02438. 32

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347. 4, 14, 15, 16

[61] Nisha Sharma, Somen Acharya, Kaushal Kumar, Narendra Singh, and Om Chaurasia. Hydroponics as an advanced technique for vegetable production: An overview. *Journal of Soil and Water Conservation*, 17:364–371, 01 2019. doi:10.5958/2455-7145.2018.00056.5. 3

[62] Nisha Sharma, Somen Acharya, Kaushal Kumar, Narendra Singh, and Om Chaurasia. Hydroponics as an advanced technique for vegetable production: An overview. *Journal of Soil and Water Conservation*, 17:364–371, 01 2019. doi:10.5958/2455-7145.2018.00056.5. 8

[63] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi:10.1038/nature16961. 2, 13

[64] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. URL: https://www.science.org/doi/abs/10.1126/science.aar6404, arXiv:https://www.science.org/doi/pdf/10.1126/science.aar6404, doi:10.1126/science.aar6404. 2, 14

[65] Pepijn Smit and Nico van der Velden. *Energiemonitor van de Nederlandse glastuinbouw 2020*. Number 2021-127 in Rapport / Wageningen Economic Research. Wageningen Economic Research, 2021. doi:10.18174/555540. 1

[66] David Still. *Lettuce*, volume 5, pages 127–140. Springer, 07 2007. doi:10.1007/978-3-540-34536-7_2. 4

[67] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, volume 12, pages 1057–1063. MIT Press, 2000.

[68] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2020. 10, 11

[69] Makiko Taguchi, Fenton Beed, Bruno Telemans, and Sara Hassan. Fruit and vegetables - your dietary essentials. the international year of fruits and vegetables, 2021 background paper. *FRUIT AND VEGETABLES - your dietary essentials.*, 12 2020. doi:10.4060/cb2395en. 1

[70] Tencent, Wageningen University, and Research. Autonomous greenhouses international challenge 3rd [online]. 2022. URL: http://www.autonomousgreenhouses.com/. 20

[71] WUR Tencent. Autonomous greenhouses international challenge 3rd [online]. 2022. URL: http://www.autonomousgreenhouses.com/. 20

[72] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109. 18, 49

[73] Engineering ToolBox. Moist air - relative humidity [online]. 2004. URL: https://www.engineeringtoolbox.com/relative-humidity-air-d_687.html.

[74] Engineering ToolBox. Moist air - relative humidity [online]. 2004. URL: https://www.engineeringtoolbox.com/evaporation-water-surface-d_690.html. 88

[75] Michiel van Dijk, Tom Morley, Marie-Luise Rau, and Yashar Saghai. A meta-analysis of projected global food demand and population at risk of hunger for the period 2010–2050. *Nature Food*, 2:494–501, 07 2021. doi:10.1038/s43016-021-00322-9. 1

[76] Peter Vermeulen and Wageningen Glastuinbouw. *Kwantitatieve Informatie voor de Glastuinbouw 2008: Groenten - Snijbloemen - Potplanten*. Wageningen University & Research, 01 2008. 10

[77] Centraal Bureau voor Statistiek. Cbs statline: Energiebalans; aanbod, omzetting en verbruik [online]. 2021. URL: https://opendata.cbs.nl/#/CBS/nl/dataset/83140NED/table?defaultview. 1

[78] Centraal Bureau voor Statistiek. Cbs statline: Groenteteelt; oogst en teeltoppervlakte per groentesoort [online]. 2021. URL: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/37738/table?fromstatweb. 1

[79] Karina Winkler, Richard Fuchs, Mark Rounsevell, and Martin Herold. Global land use changes are four times greater than previously estimated. *Nature Communications*, 12(1), may 2021. doi:10.1038/s41467-021-22702-2. 1

[80] Ismail şahin, M. Hanefi CALP, and Atacan Özkan. An expert system design and application for hydroponics greenhouse systems. *Gazi University Journal of Science*, 27:809–822, 01 2014. 8

Appendix A

Model Training

A.1 Training Hardware

For training we use a workstation referred to as Blobfish, that contains the following hardware:

Part	Name
CPU	AMD Ryzen Threadripper 3970X 32 Core
GPU	NVIDIA GeForce 2080 Ti
RAM	64 GB

Table A.1: Hardware specification for training setup

A.2 Feature Engineering

We applied two feature engineering methods, omitting variables and cyclic transformations. These methods are elaborated below.

Omitting Variables

In the state space defined in section B.3 we find several values that contain little to no information that is relevant for growing the plant. As the costs made are already included in the agent reward function we can omit all variables related to cumulative resource usage and cumulative costs. This includes fixed costs, number of plants per square meter and the gains made from selling the plant.

We exclude the current heating price and carbon price as well, but we keep the electricity price. The information our these prices is important for our agent to know. However, in our model the heating and carbon prices are constant while only the electricity price fluctuates. As constant input values add computational expense without adding information to the network we chose to omit these fields.

Finally we need to consider removing variables that can cause over fitting. We chose to remove the time variable from this simulation as it may cause the agent to learn to rely on the current time since the start of the growing cycle, while it should rely on the current state of the plant and the greenhouse to determine its optimal strategy.

Cyclic transformations

Variables that are cyclic by nature such as hour of day or week number of year can be represented better with a cosine / sine transformation. By applying this transformation on the hour value we

form two values, one using the cosine and the other using the sine function. As these values are normalized in the [0, 1] range we can apply the following transformation.

$$v_{cyclic} = \frac{1}{2} * \begin{bmatrix} 1 + cos(2 * \pi * v_{ori}) \\ 1 + sin(2 * \pi * v_{ori}) \end{bmatrix} \tag{A.1}$$

After transforming the variables the euclidean distance between any pair of v_{cyclic} will be equal as long as these pairs have an equal difference in real world time. As opposed to the initial situation 23:00 will now have the same euclidean distance to 00:00 as 14:00 has to 15:00.

A.3 Rule Based Policy Optimization

Parameter	Initial Setpoints	Optimized Setpoints	Search Space	
heater_minimal_temp_day	24	24	[18,	24]
heater_minimal_temp_night	19	23.03	[15,	24]
p_control_heater	0.2	0.2	[0.01,	0.2]
illumination_max_watts	1048.7	1048.7	[800,	1200]
p_control_blackout	0.017092	0.017092	[0.005,	0.05]
ventilation_max_temp	25	25.292	[24,	28]
ventilation_p_action	0.39137	0.39137	[0.1,	0.8]
ventilation_max_humidity	0.7	0.8887	[0.85,	0.95]
ventilation_min_humidity	0.5	0.6	[0.1,	0.6]
minimal_difference_to_open	0	0.086	[0.05,	0.15]
ventilation_humidity_p_action	0.5	0.25	[0.25,	0.75]
carbon_level_closed	1500	676.07	[600,	1200]
carbon_level_open	500	423.5	[400,	500]
carbon_level_night	390	555.08	[400,	600]
carbon_p_action	0.01	0.01	[0.05	0.1]
light_start	6	7	[1,	7]
light_stop	7	7	[7,	10]
night_start_hour	20	22	[19,	24]
day_start_hour	5	4	[1,	7]

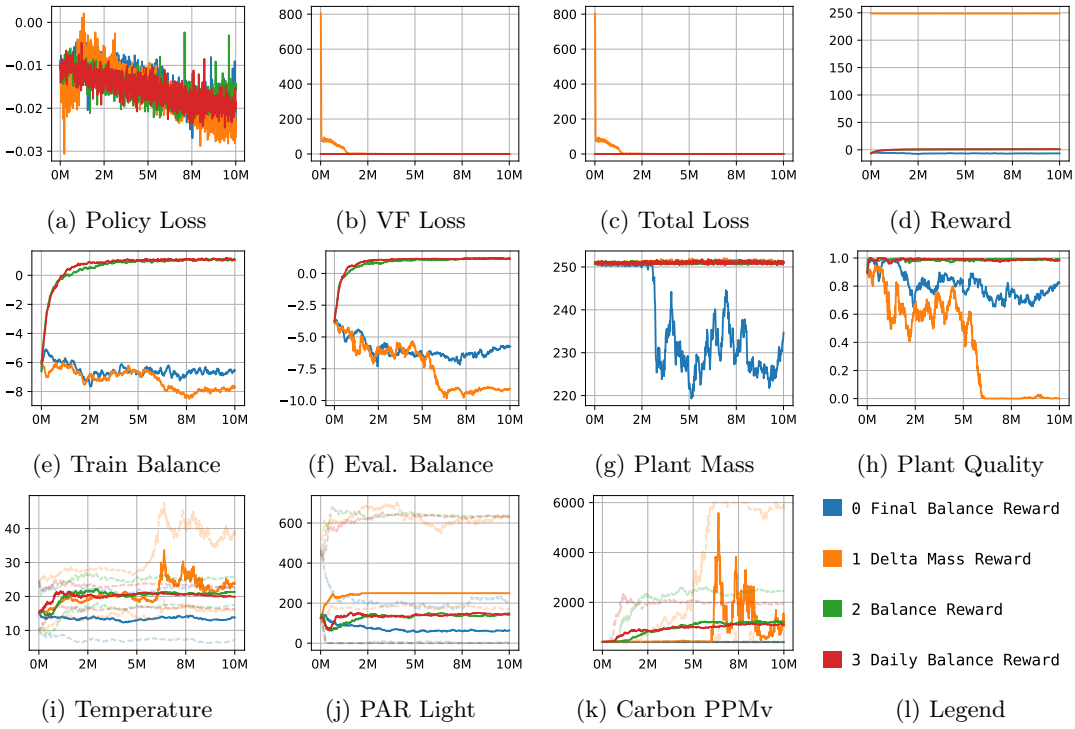
Table A.2: Rule based agent parameters for the initial setpoints, the optimized setpoints and its search space.

A.4 Training Results

For each of the performed experiments in chapter 4 we include a one page summary that describes agent performance on the benchmark as well as the training procedure itself. Additionally, we mention the config file name, which is included in the Git repository that contains all scripts for this work.

PPO 1: Reward Functions

Config File Name: `ppo_1_rf.yaml`
Determine the best reward function.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.1: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

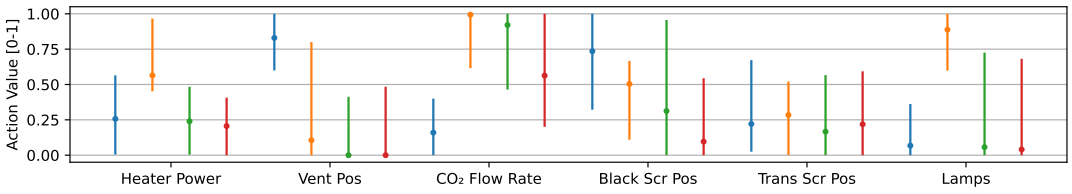
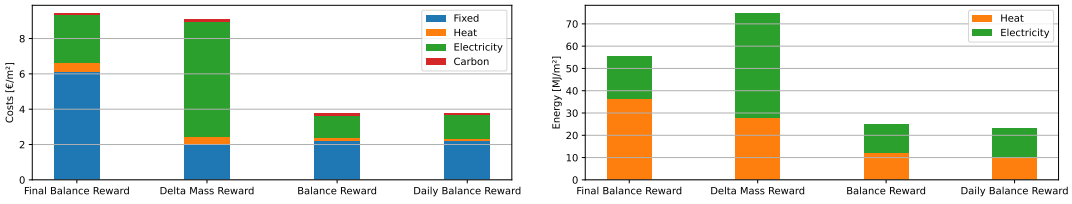


Figure A.2: Normalized action distribution for the final evaluation on the benchmark.

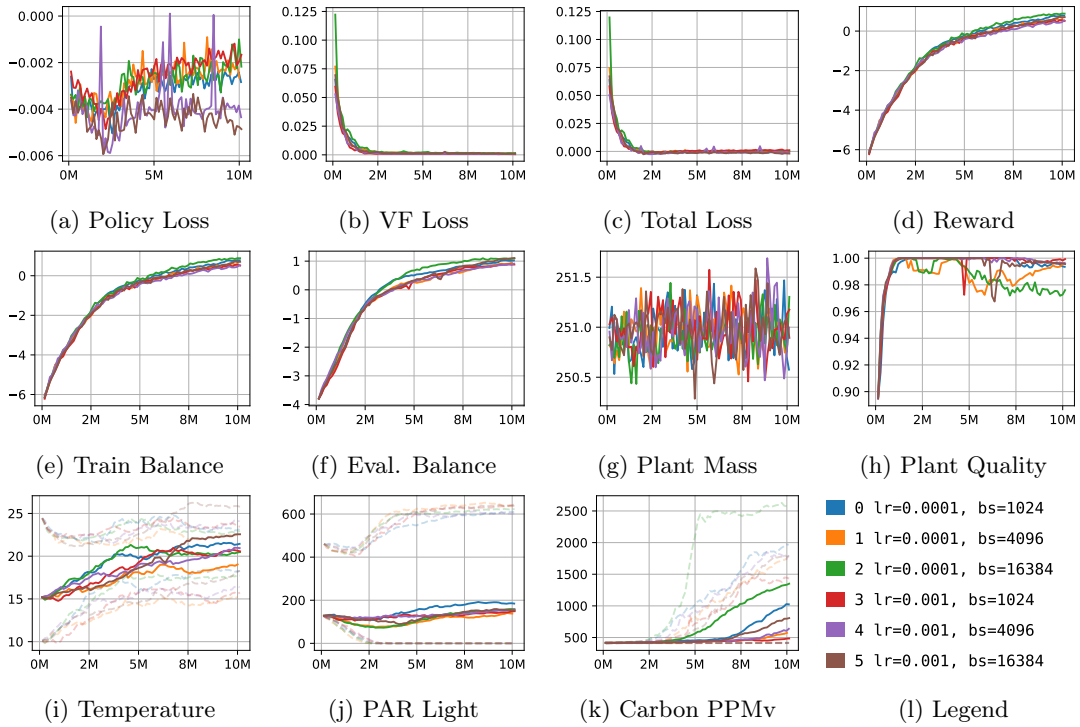


(a) Costs (b) Energy Usage

Figure A.3: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 2: Batch Size and Learning Rate

Config File Name: `ppo_2_bslr.yaml`
Find the optimal batch size and learning rate.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.4: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

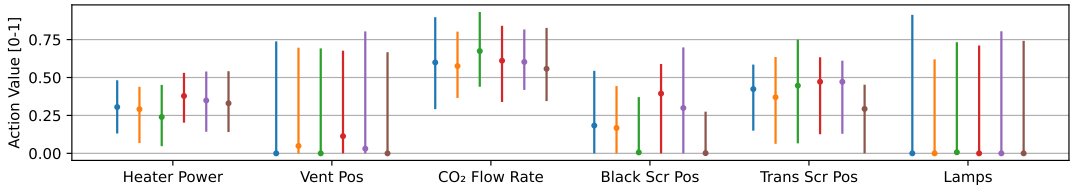
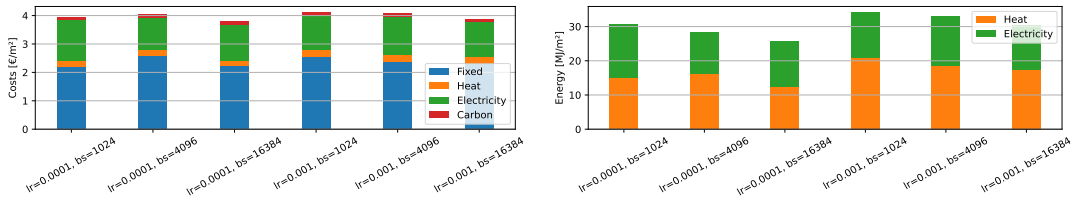


Figure A.5: Normalized action distribution for the final evaluation on the benchmark.

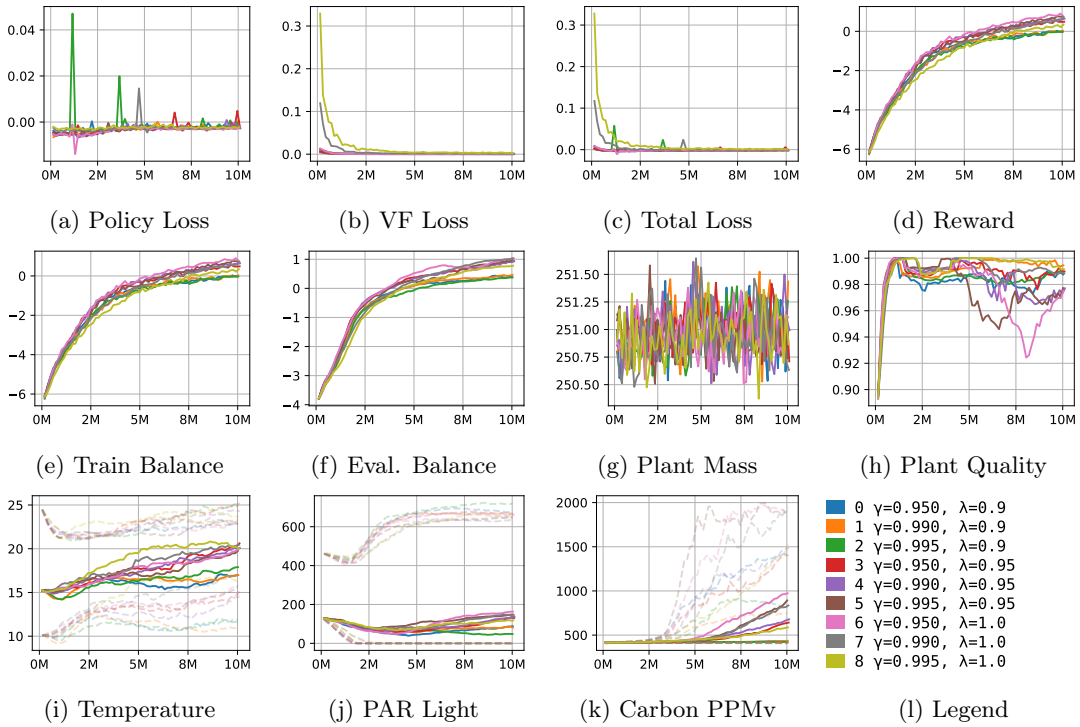


(a) Costs (b) Energy Usage

Figure A.6: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 3: Lambda and Gamma

Config File Name: `ppo_3_lambda_gamma.yaml`
Find the optimal values for lambda and gamma.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.7: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

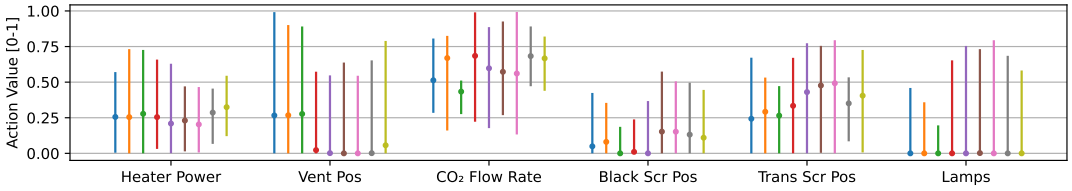
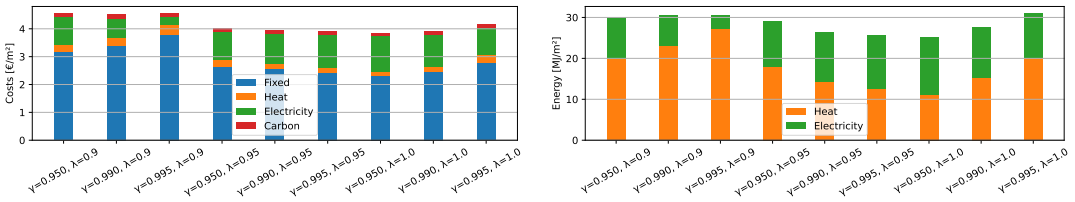


Figure A.8: Normalized action distribution for the final evaluation on the benchmark.

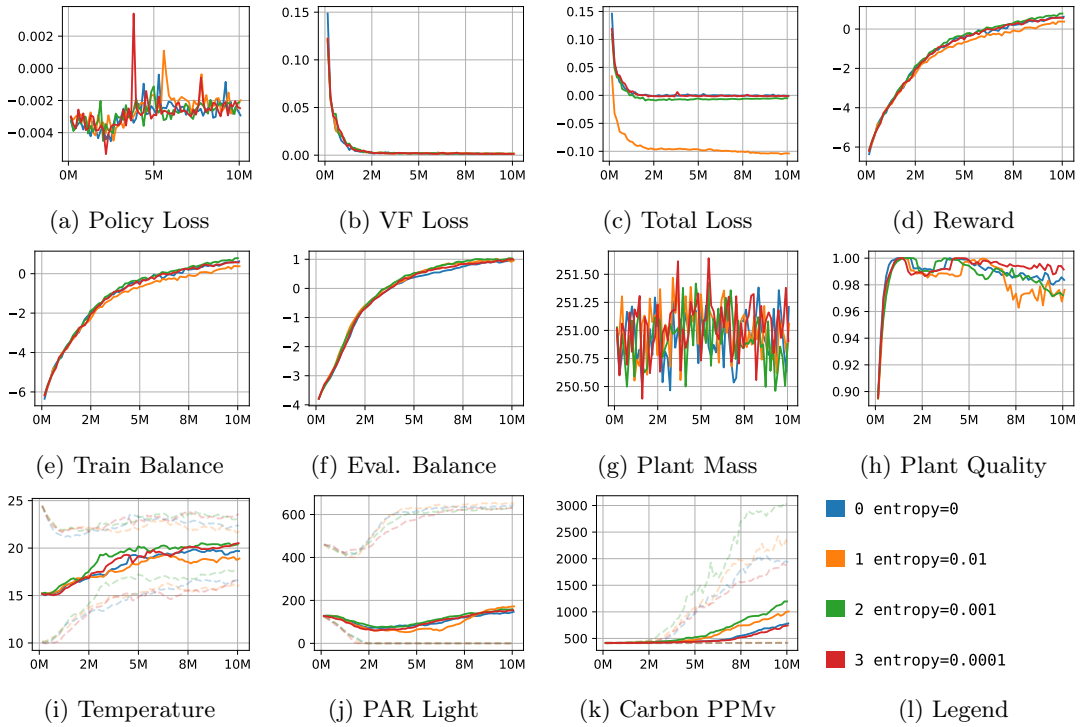


(a) Costs (b) Energy Usage

Figure A.9: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 4: Entropy

Config File Name: `ppo_4_entropy.yaml`
Find the optimal value for entropy.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.10: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

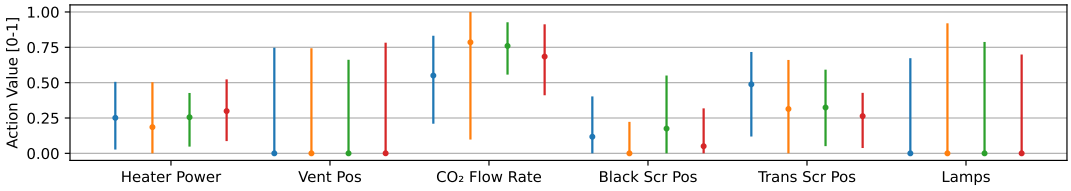
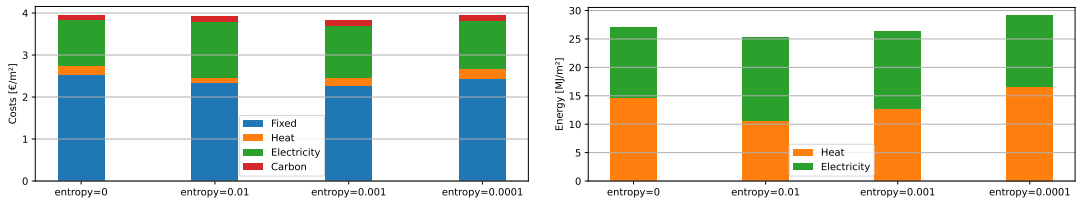


Figure A.11: Normalized action distribution for the final evaluation on the benchmark.

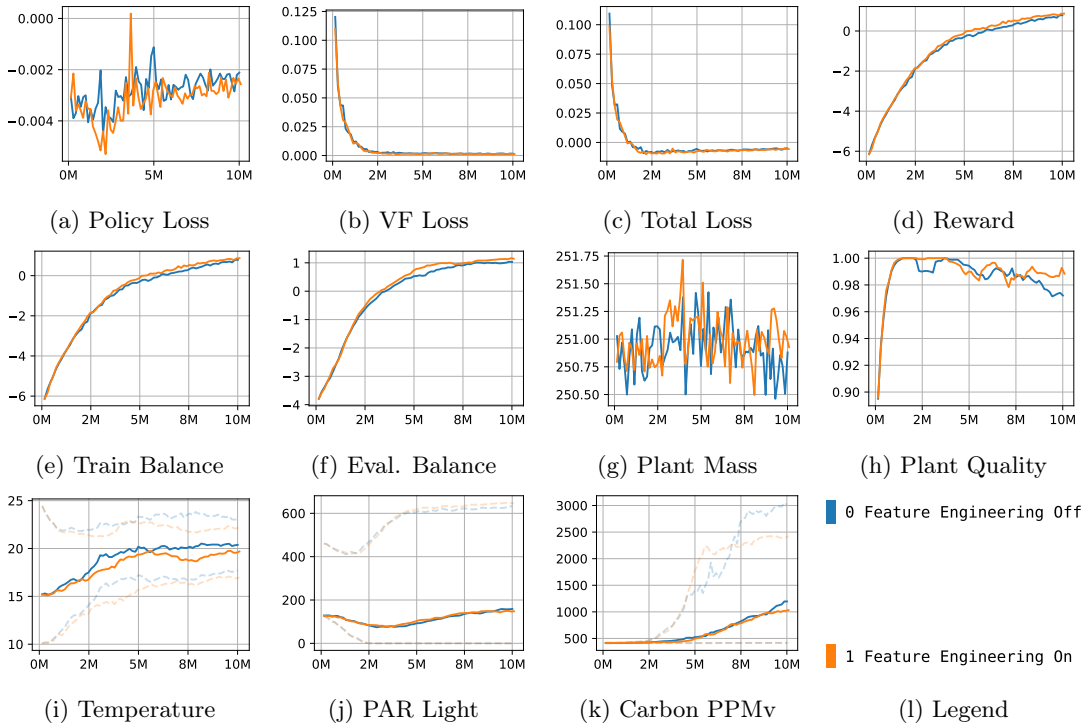


(a) Costs (b) Energy Usage

Figure A.12: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 5: Feature Engineering

Config File Name: `ppo_5_feats.yaml`
Determine if feature engineering improves PPO performance.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.13: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

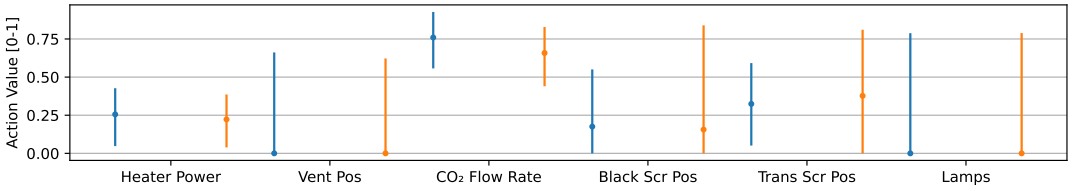
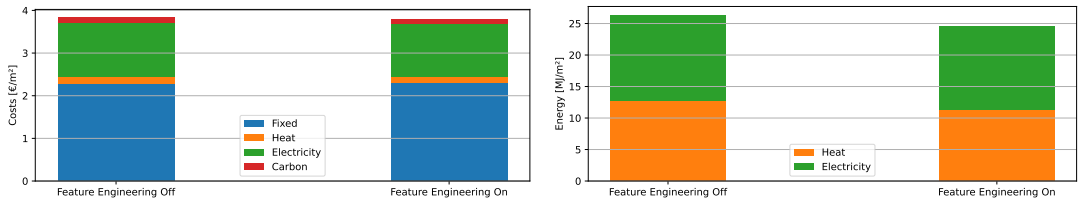


Figure A.14: Normalized action distribution for the final evaluation on the benchmark.

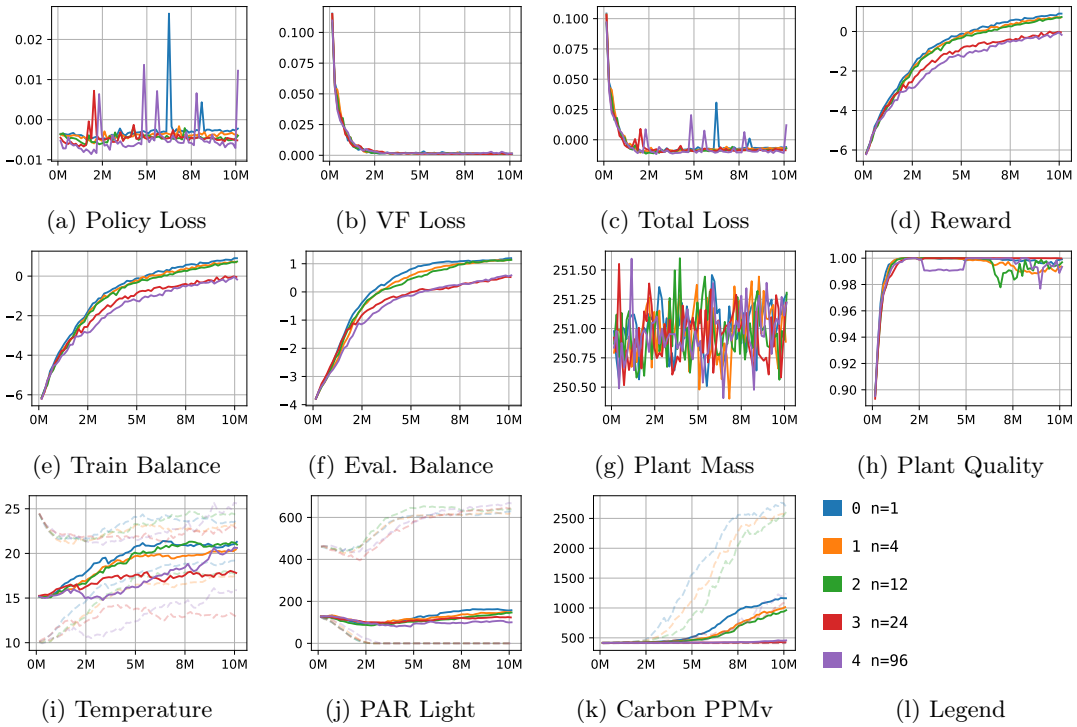


(a) Costs (b) Energy Usage

Figure A.15: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 6: Framestacking

Config File Name: `ppo_6_fs.yaml`
Determine if frame stacking improves performance.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.16: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

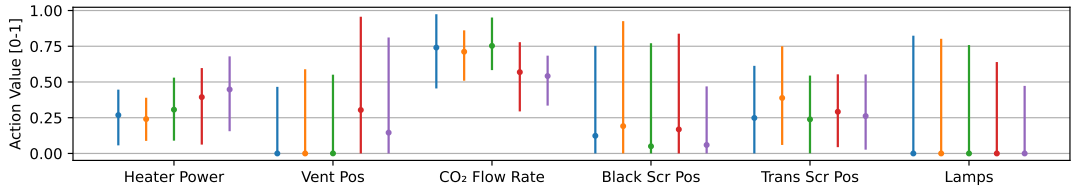
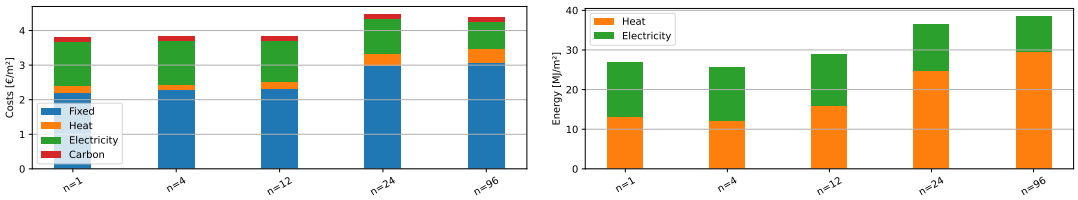


Figure A.17: Normalized action distribution for the final evaluation on the benchmark.

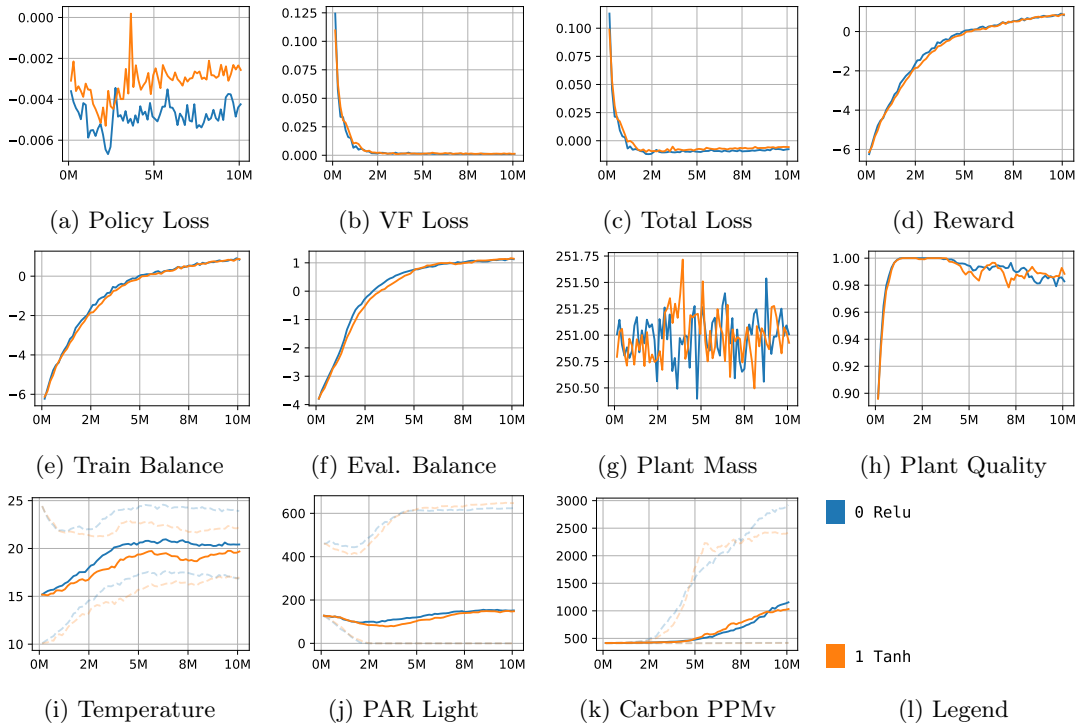


(a) Costs (b) Energy Usage

Figure A.18: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 7: Activation Layers

Config File Name: `ppo_7_act.yaml`
Determine what activation layer provides the best performance.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.19: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

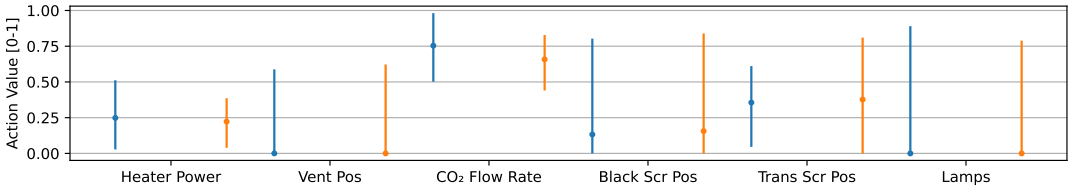
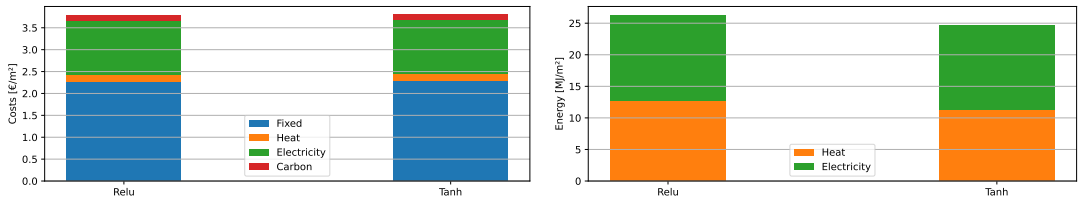


Figure A.20: Normalized action distribution for the final evaluation on the benchmark.

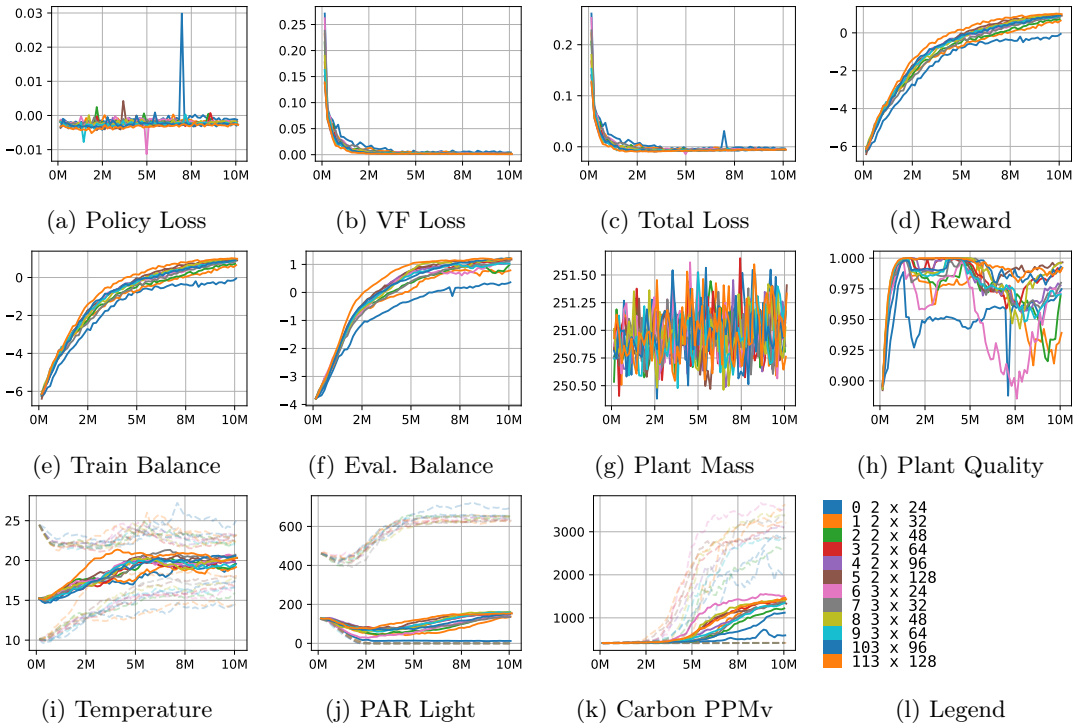


(a) Costs (b) Energy Usage

Figure A.21: Breakdown of costs and energy usage for the final evaluation on the benchmark.

PPO 8: Network Layers and Depth

Config File Name: `ppo_8_ns.yaml`
Determine the optimal amount of layers and layer size.



(a) Policy Loss (b) VF Loss (c) Total Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.22: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

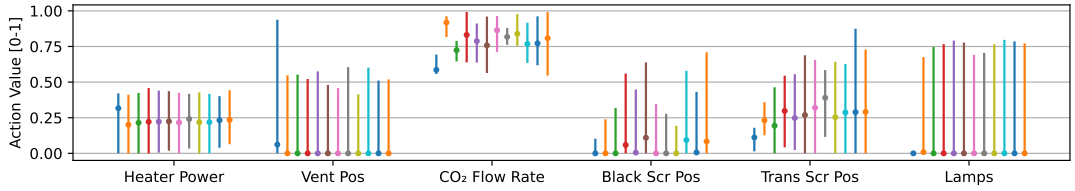
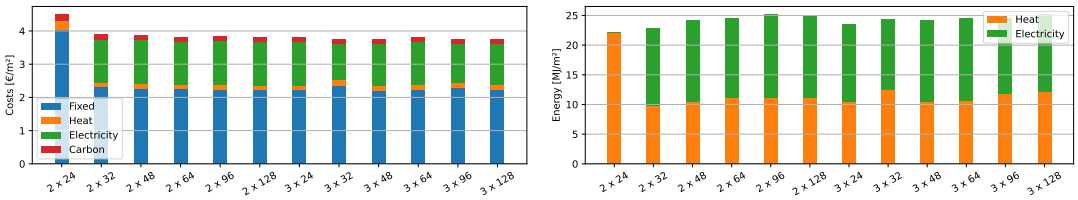


Figure A.23: Normalized action distribution for the final evaluation on the benchmark.

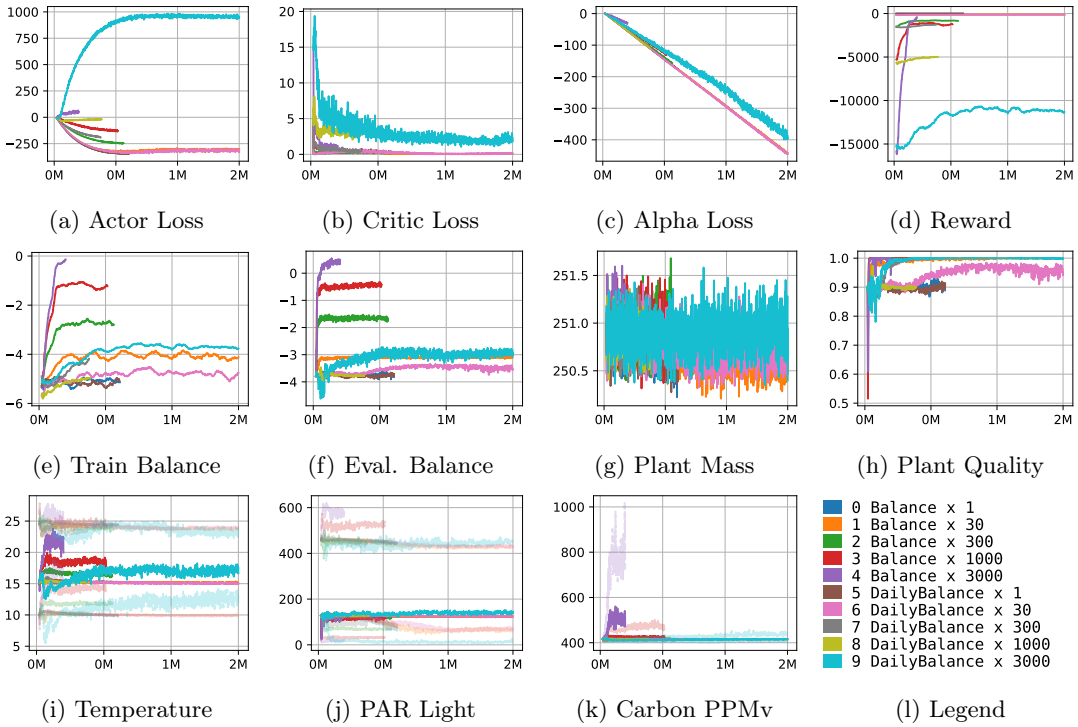


(a) Costs (b) Energy Usage

Figure A.24: Breakdown of costs and energy usage for the final evaluation on the benchmark.

SAC 1: Reward Functions and Scale

Config File Name: `sac_1_rf.yaml`
Determine the optimal reward function and matching coarse value for reward scale.



(a) Actor Loss (b) Critic Loss (c) Alpha Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.25: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

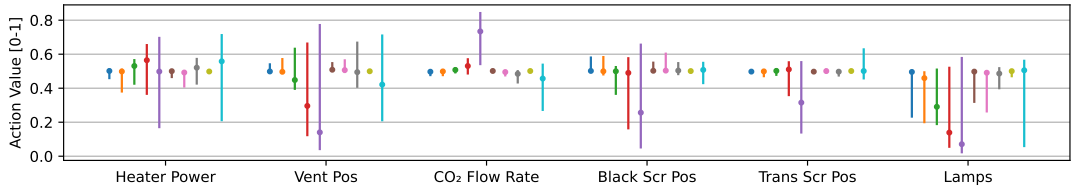
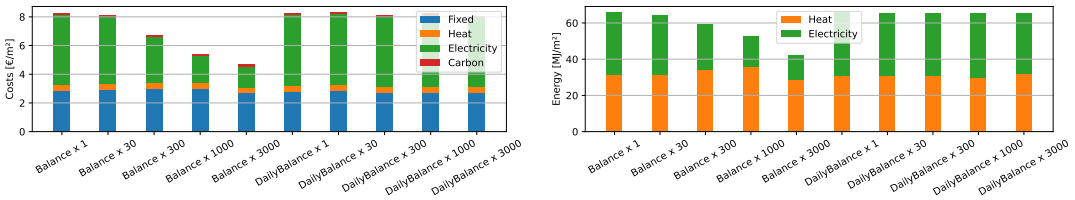


Figure A.26: Normalized action distribution for the final evaluation on the benchmark.

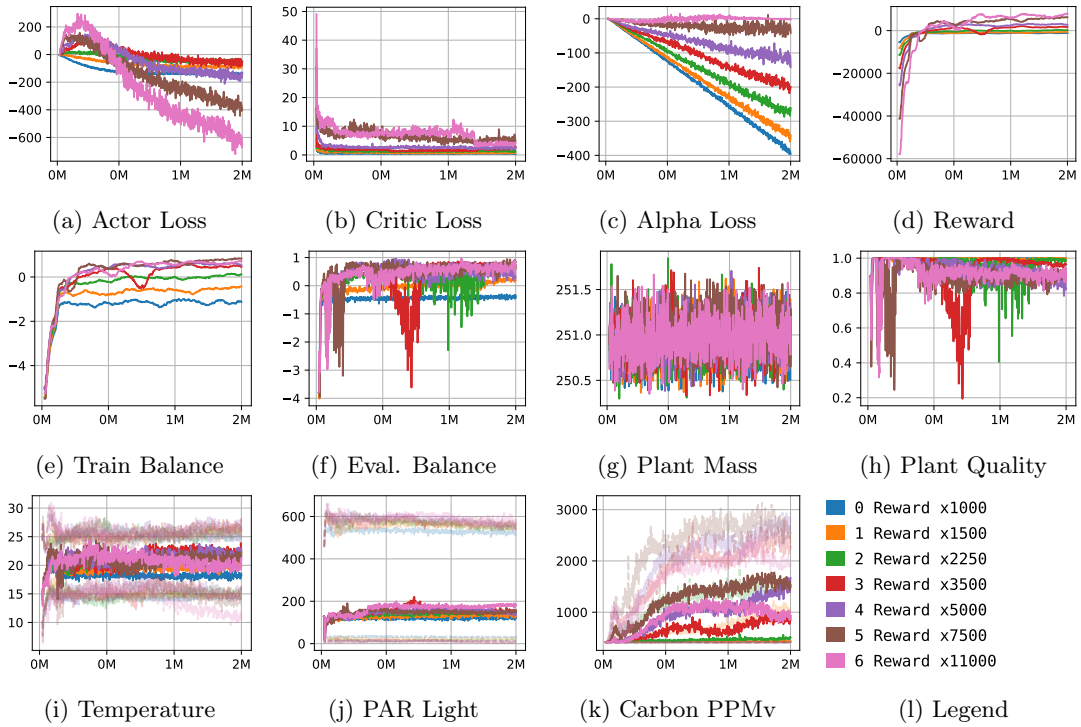


(a) Costs (b) Energy Usage

Figure A.27: Breakdown of costs and energy usage for the final evaluation on the benchmark.

SAC 2: Reward Scale

Config File Name: `sac_2_rs.yaml`
Determine the optimal reward scale for the selected reward function.



(a) Actor Loss (b) Critic Loss (c) Alpha Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.28: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

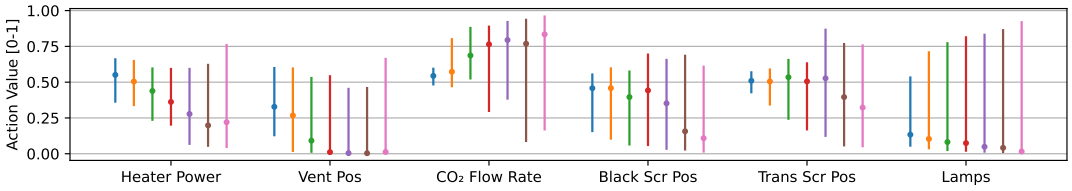
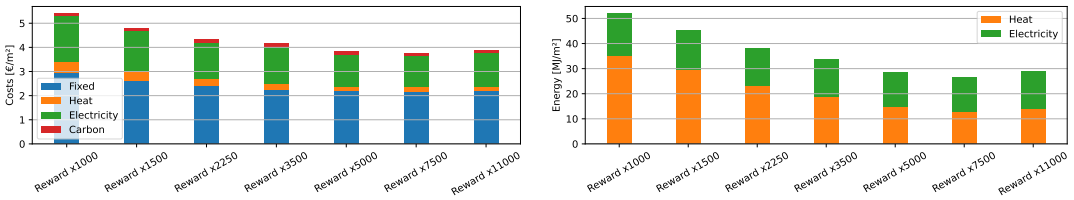


Figure A.29: Normalized action distribution for the final evaluation on the benchmark.

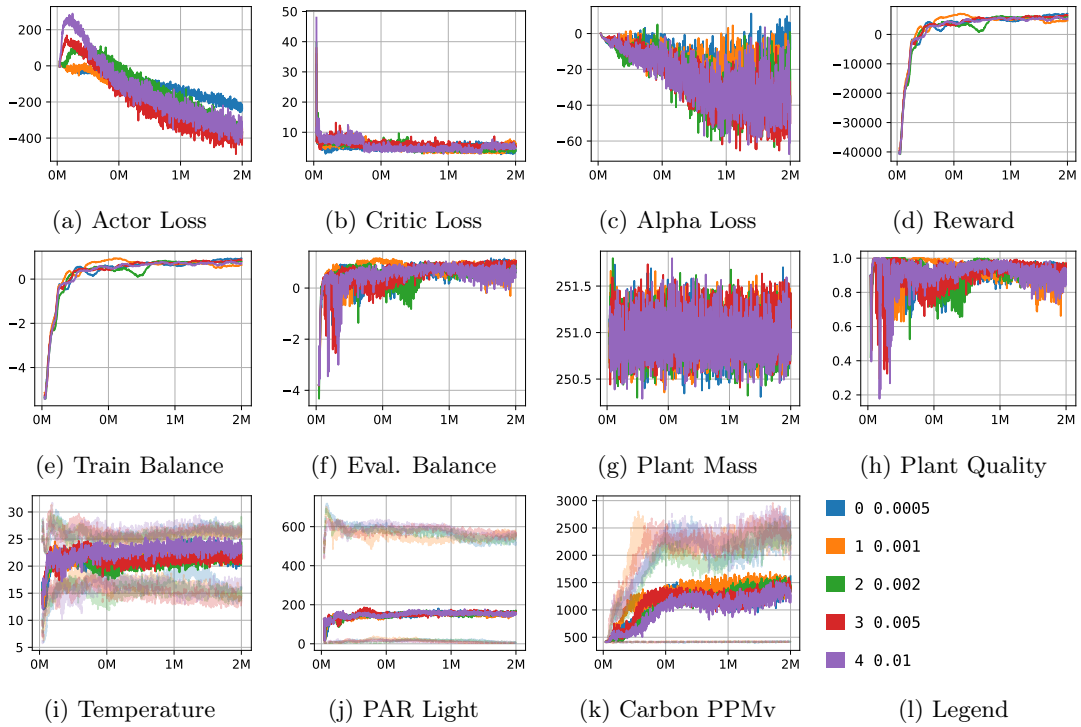


(a) Costs (b) Energy Usage

Figure A.30: Breakdown of costs and energy usage for the final evaluation on the benchmark.

SAC 3: Tau

Config File Name: `sac_3_tau.yaml`
Determine the optimal value for tau.



(a) Actor Loss (b) Critic Loss (c) Alpha Loss (d) Reward

(e) Train Balance (f) Eval. Balance (g) Plant Mass (h) Plant Quality

(i) Temperature (j) PAR Light (k) Carbon PPMv (l) Legend

Figure A.31: Time step based charts of various training performance metrics. Charts a-d: training info, charts e-h: final state info, charts i-k greenhouse state info with 95% confidence interval.

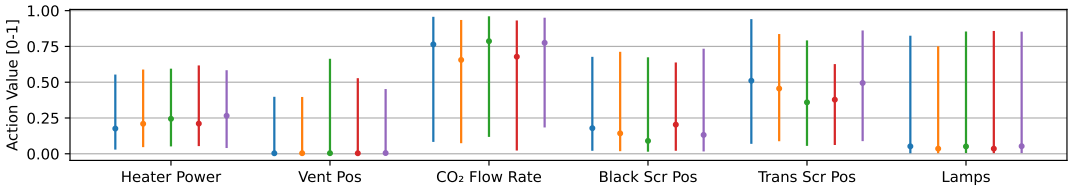
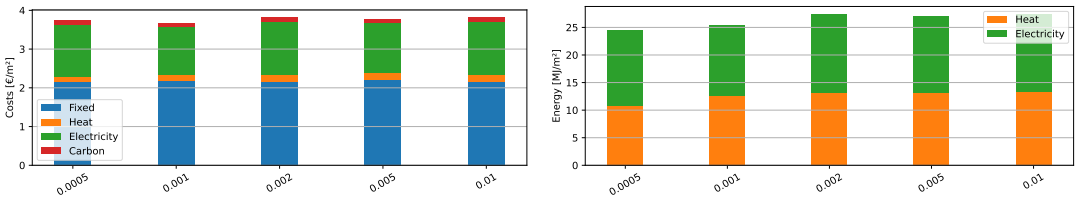


Figure A.32: Normalized action distribution for the final evaluation on the benchmark.



(a) Costs (b) Energy Usage

Figure A.33: Breakdown of costs and energy usage for the final evaluation on the benchmark.

Appendix B

Greenhouse Simulation

B.1 Simulation Components

In this section we will explain the different simulation models that together form the greenhouse simulator. As a convention to refer to a specific property from the action a_t or the state a_t we define the following notation: $s_{t,balance}$. Here we refer to the balance at time t, which is a property of the state s_t. A complete overview of all action and state variables can be found in section B.3.

B.1.1 Lettuce Plant Model

At the heart of the simulation we have the plant model. We model the quality and mass of the plant as a function of the greenhouse climate that our agent attempts to control.

Growth

As mentioned in 2.1.1 photosynthesis is vital for the plant to generate energy for growing. We decide to base the growth of the lettuce crop solely on photosynthesis rate, as the scope of this project is controlling the greenhouse climate excluding other effects such as the nutrient mixture. From subsection 2.1.1 we find that the most important parameters for photosynthesis include:

1. Temperature in the greenhouse in K
2. PAR Light level above the plant in $\mu mols^{-1}$
3. Carbon dioxide level in the greenhouse in ppm_v

In our model we assume that the plant grows exponentially, with a maximum daily growth rate. The maximum hourly grow rate can be approximated as a fraction of the maximum grow rate depending on limiting factors in the mentioned three main growth factors subsection 2.1.1. Not only should the climate be optimal, it must also be sustained to achieve maximum growth. To model this effect we define a minimum photosynthesis period to obtain maximum growth in a day. Based on the maximum daily growth rate $g_{d,max}$ and the minimum number of hours for maximum growth h_{min} we can determine the maximum hourly growth rate $g_{h,max}(t)$. The equation for $g_{h,max}(t)$ is shown below:

$$g_{h,max}(t) = min\left(g_{d,max}^{\frac{1}{h_{min}}}, \frac{g_{d,max}}{g_{d,cum}(t)}\right) \tag{B.1}$$

Here $g_{h,max}$ describes the maximum hourly growth at time t and $g_{d,max}$ describes the maximum daily growth, $g_{d,cum}$ is the daily cumulative growth which is ratio between the plant mass at time step t and the mass of the plant at the start of the day. The maximum growth rate can be achieved

if all growth factors for the lettuce plant are optimal. To approximate this effect we approximate the maximal growth rate at time t using the following equation.

$$g(t) = g_{h,max}(t) \quad \cdot \quad f_{CO_2}(s_{t,carbon\_ppm}) \quad \cdot \quad f_{PAR}(s_{t,par\_light}) \quad \cdot \quad f_T(s_{t,T_{temperature\_air}}) \tag{B.2}$$

Here $g(t)$ describes the growth rate at time step t, $g_{h,max}$ is the maximum growth rate for hour t, f_{CO_2}, f_{PAR} and f_T model the limiting factors of carbon dioxide, PAR light and temperature respectively, all in the range [0, 1]. A multiplication is chosen to ensure that if at least one factor is limiting, it will cause g to drop.

In [45] and [11] we find curves that describes the relative photosynthesis rate as a function of the carbon dioxide level in the greenhouse. We observe that the photosynthesis rate of the plant increases when the carbon dioxide level increases until at least 1200 ppm_v. This effect is limited by diminishing returns. To approximate this effect we use the following equation:

$$f_{CO_2}(CO_2) = 1 - e^{-CO_2/\beta_{CO_2}} \tag{B.3}$$

Here f_{CO_2} is the dimensionless carbon growth factor between 0 and 1, C_{carbon} is the carbon level in the greenhouse in ppm_v and β_{CO_2} is a dimensionless tunable parameter.

In [11] we find that the relationship between PAR light intensity and photosynthesis rate is similar to the carbon dioxide and photosynthesis relationship. Therefore the decision was made to approximate it using a similar equation:

$$f_{PAR}(PAR) = 1 - e^{-PAR/\beta_{PAR}} \tag{B.4}$$

Here f_{PAR} is the dimensionless PAR light growth factor between 0 and 1, PAR is the PAR light level in the greenhouse in $\mu mols^{-1}$ and β_{PAR} is tunable parameter in $(\mu mol/s)^{-1}$.

The relationship between temperature and plant growth rate is different to the former two growth factors. For plants in general there exists a certain optimal temperature for which the plant growths optimally [52]. Both hotter and colder temperatures than the optimal temperature will have a negative impact on the plant growth rate. Therefore we model the temperature growth factor as a clipped parabola using the equation below:

$$T_{diff}(T) = |T - T_{opt}|$$

$$f_T(T) = \begin{cases} 1 - \left(T_{diff}(T)/T_{rng}\right)^2, & \text{if } T_{diff}(T) < T_{rng} \\ 0, & \text{otherwise} \end{cases} \tag{B.5}$$

Here f_T is the dimensionless temperature growth factor between 0 and 1. T is the temperature in the greenhouse in K. T_{opt} is the tunable optimal growth temperature in K for the lettuce plant. T_{rng} is the one-way maximum difference from the optimal growth temperature in K at which a plant will stop growing.

Carbon Respiration and Absorption

In subsection 2.1.1 we discussed that there are two chemical reactions involved in the gas exchange of a lettuce plant, photosynthesis and respiration. As the plant growths, part of the carbon absorbed in assimilated into structural weight. This structural carbon can not be respired by the plant.

We model this system is a simplified way:

1. When the plant gains weight due to photosynthesis, we assume that a constant fraction of the weight gained comes from the carbon dioxide absorbed.
2. Half of the absorbed carbon turns into structural weight while the other half is converted stored in a buffer for respiration.

3. The plant respires carbon in the buffer unless it is empty.

We define the buffer as C_b that denotes the internal carbon for respiration in kg. At the initial time step we start with an empty buffer C_b. We can determine the change

$$dC_b(t) = \left(\frac{1}{2}\dot{m}_{pho}(t) - \dot{m}_{res}(t)\right) \cdot dt \tag{B.6}$$

$$\dot{m}_{pho}(t) = \omega_{abs} ds_{t,mass}(t) \quad and \quad \dot{m}_{res}(t) = min(dt * \dot{m}_{res,max}, C_b(t)) \tag{B.7}$$

Here $\dot{m}_{res}$ and $\dot{m}_pho$ denote the respiration and photosynthesis rate respectively in kg/s. $\dot{m}_{res,max}$ denotes the maximum respiration rate in kg/s. ω_{abs} is the dimensionless mass gain to CO_2 absorption factor. $dC_b(t)$ denotes the change in the carbon buffer at time step t.

We can now calculate the change in carbon within the greenhouse using the following equation.

$$ds_{t,carbon\_ppm,p}(t) = s_{t,ppsm}(t) \cdot \left(\dot{m}_{pho}(t) - \dot{m}_{res}(t)\right) \cdot dt \tag{B.8}$$

Here $ds_{t,carbon\_ppm,p}(t)$ denotes the change in CO_2 level in the greenhouse due to plant respiration and photosynthesis and $s_{t,ppsm}(t)$ is the number of plants per unit area m^{-2}.

Quality Degradation

In extreme conditions our plant will experience quality degradation. We model this as a quality value, that starts with a value of 1. If damage occurs the quality will be reduced. We apply quality reductions as negative exponential growth. This can be interpreted as: Damage causes a percentage of the remaining plants to die. In reality it may not only cause plant death, but also visual damage which can lower the sales category of the plant, affecting the selling price. Formally we model quality degradation using Equation B.9.

$$s_{t,quality} = s_{t-dt,quality} * g_{quality\_loss}^{\frac{dt}{3600}} \tag{B.9}$$

Here $g_{quality_loss}$ is the quality loss rate. We model the quality loss rate as a sum of multiple quality loss sources using the equation below.

$$g_{quality\_loss} = 1 - \sum_{L \in L_{sources}} L \tag{B.10}$$

Here $L_{sources}$ is the set of quality loss rates for each source. We apply 5 types of damage: Low and high temperatures, low and high humidity levels and tipburn. As described in subsection B.1.1 tipburn is caused by calcium deficiency due to rapid growth. To model this we keep a calcium deficiency counter $C(t)$. We first define the relative growth rate with respect to the maximum tipburn growth rate.

$$r_{growth} = \frac{g - g_{max,tipburn}}{g_{max,tipburn} - 1}$$

Here r_{growth} is the difference between current growth rate and the maximum growth rate before tipburn $g_{max,tipburn}$ as a fraction. If $r > 0$ we exceed the maximum growth rate. We modify the carbon deficiency using the equation below:

$$C(t) = C(t - dt) + r_{growth} \cdot dt$$

We calculate the loss L for all quality loss sources using the equation below:

$$L_{source} = \begin{cases} \epsilon \cdot R_{quality,source}, & \text{if } \epsilon > 0 \\ 0, & \text{otherwise} \end{cases} \tag{B.11}$$

Here ϵ is the amount by which either the low or high values are exceeded and L_{source} is any loss in $L_{sources}$ with its respective quality loss rate $R_{quality,source}$. For both temperatures and

humidity we calculate ϵ with respect to the temperature and humidity limits. For tipburn we use $\epsilon = C(t)$.

B.1.2 Climate Model

We explain the climate model in 5 different sections. We start explain the submodels in the following order: PAR light Model, Temperature Model, Ventilation Model, Carbon Dioxide and Humidity.

PAR Light Model

The PAR light level just above the plant is modeled as the addition of PAR light from the sun and the PAR light from the lamps. Here the PAR light from the sun is affected by the current position of the blackout and transparent screens. We start of with the main equation:

$$s_{t,PAR}(a_t) = t_{solar} PAR_{solar} + a_{t,lamps} PAR_{lamps} \tag{B.12}$$

Here PAR equals the PAR light level above the plant, PAR_{solar} is the PAR level of the sunlight outside and PAR_{lamps} is a constant of 350 $\mu\ mol\ s^{-1}$ that equals the PAR level provided when all the lamps are enabled. All former mentioned parameters are in $\mu\ mol\ s^{-1}$. PAR_{solar} can be approximated using outside illumination using a conversion factor where 1 $W \approx 4.6\ \mu\ mol\ s^{-1}$ as seen in [37]. T_{solar} is the total transmittance of sunlight outside to the light above the plant and $a_{t,lamps}$ is the fraction of the lamps that are currently on.

T_{solar} is a function of the current position of the blackout and transparent screen.

$$T_{solar} = u_{roof} \cdot u_{blackout} \cdot u_{transparent} \tag{B.13}$$

Here T_{roof} is a constant dimensionless value of 0.60 as found in [35]. The blackout and transparent transmittance are approximated using the following equations:

$$T_{blackout} = 1 - r_{blackout} \cdot a_{t,blackout} \tag{B.14}$$

$$T_{transparent} = 1 - r_{transparent} \cdot a_{t,transparent} \tag{B.15}$$

Here the r values denote the dimensionless fraction of light reflected by the screen if fully enabled. The $a_{t,blackout}$ and $a_{t,transparent}$ values denote the blackout and transparent screen position set by the agent respectively, with a range of [0, 1]. The amount of light blocked by the screens has a linear relation to the screen position. We fit the r values by querying timeseries from the greenhouse dataset for which the lamps were off, there was sunlight and only its respective screen was enabled. We found that a value for $r_{blackout}$ of 0.86 and a value for $r_{transparent}$ of 0.27 minimizes the model error.

Temperature Model

We model the temperature of the greenhouse as a single heat buffer for simplicity. The temperature change for a single heat buffer model is modelled using the following equation from [14].

$$dT = \frac{du}{C} \tag{B.16}$$

Here T denotes the temperature of the body, du denotes the change in internal energy and C denotes the heat capacity constant in $JK^{-1}m^{-2}$. The change in internal energy is a sum of all incoming and outgoing flows of energy. In the case of our simulation it is described using the following equation, which is derived directly from the first law of thermodynamics:

$$du = \dot{Q}_{heater} + \dot{Q}_{conduction} + \dot{Q}_{solar} + \dot{Q}_{evaporation} + \dot{Q}_{ventilation} \tag{B.17}$$

Here all Q terms describes the amount of heat that is added to the system in Jm^{-2}. In case of $Q_{conduction}$ this value is typically negative, meaning that heat is removed from the system by means of conduction. Now we will briefly elaborate each of the terms.

The first term $\dot{Q}_{heater}$ is defined by the equation below:

$$\dot{Q}_{heater}(t) = \eta \cdot a_{t,heater} \cdot dt \tag{B.18}$$

Here η is the heating efficiency of 0.8 , $a_{t,heater}$ is part of the agent action and dt is the simulation step size in s. A typical greenhouse consumes up to 180 Wm^{-2} [49]. We slightly increase this range in our action space to [0, 200] Wm^{-2} to allow the agent to experience the effects of overheating.

The second term $\dot{Q}_{conduction}$ is derived from Newton's law of cooling [14]:

$$\dot{Q}_{conduction} = h\Delta T = h(t)(T_{greenhouse\_air} - T_{outside}) \tag{B.19}$$

Here $h(t)$ is the heat transfer coefficient at time t in $Wm^{-2}K^{-1}$ and ΔT is the difference in temperature between the greenhouse air and the outside air in K. This coefficient is dependent on the different insulating layers in the greenhouse. We can determine this coefficient using the following equation:

$$h = \frac{1}{r_{roof} + r_{blackout} + r_{transparent}} \tag{B.20}$$

Here r_{roof}, $r_{blackout}$ and $r_{transparent}$ are insulation values of the roof, the blackout screen and the transparent screen respectively. Single pane glass has an r of 0.88 m^2KW^{-1} as found in [25]. Here we also find that a typical material for greenhouse screens is Polyethylene where a single layer has an r-value of 0.87 $m^{-2}KW^{-1}$. This value only occurs when the screens are completely closed. In practice, a partially open screen will quickly lose its insulating effect as air can pass through the gap. To model this we modify equation B.20 to the following approximation:

$$h(t) = \frac{1}{r_{roof} + r_{blackout} \cdot (a_{t,blackout})^{10} + r_{transparent} \cdot (a_{t,transparent})^{10}} \tag{B.21}$$

Here the *Action* parameters are part of the agent action, with values in the range [0, 1]. We add a power of 10 to approximate the rapid decline of insulation value as the screen opens. A closed screen results in 100% of the insulation value, while a 10% opened screen only has 35% of the insulation value and a 50% opened screen has less than 0.1% of the insulation value.

The third term $\dot{Q}_{solar}$ is approximated by the following equation:

$$\dot{Q}_{solar} = AbsorptionFraction_{heat} \cdot T_{solar} \cdot s_{t,illumination_out} \tag{B.22}$$

Here $AbsorptionFraction_{heat}$ is the fraction of solar heat absorbed. We calculate $\dot{Q}_{evaporation}$ using the following equation:

$$\dot{Q}_{evaporation} = -\Delta H_{vap,water} \cdot m_{evaporated}$$

Here $\Delta H_{vap,water}$ is the evaporation heat for water which equals $2_4 53_5 00$ $J\ kg^{-1}$, $m_{evaporated}$ is the amount of evaporated water which is described in section B.1.2. The final term $\dot{Q}_{ventilation}$ is described using the following equation.

$$\dot{Q}_{ventilation} = \dot{Q}_{ventilation,max} \cdot a_{t,ventilation}$$

Ventilation Model

To model the agents ability to open the roof windows we introduce a gas exchange model. We model the air exchange as a volume flow rate that replaces air in the greenhouse with air from the outside environment. We model this volume flow of the air as a linear dependence on the ventilation action using the following equation.

$$\dot{V}(t) = \dot{V_{max}} \cdot a_{t,ventilation} \tag{B.23}$$

Here $\dot{V}$ is the volume flow of air intro the greenhouse in $m^3\ s^{-1}\ m^{-2}$, $\dot{V_{max}}$ is the maximum ventilation capacity in $m^3\ s^{-1}\ m^{-2}$ and $a_{t,ventilation}$ is the ventilation action.

$$R(t) = \frac{h_{gh}}{\dot{V}(t) \cdot dt} \tag{B.24}$$

Here $R(t)$ is the fraction of air replaced, h_{gh} is the height of the greenhouse in m. We apply this fraction to calculate the new temperature and carbon levels using the following equation:

$$s_{t,temperature\_air} = (1 - R(t))s_{t-dt,temperature\_air} + R(t)s_{t-dt,temperature\_out}$$

$$s_{t,carbon\_ppm} = (1 - R(t))s_{t-dt,carbon\_ppm} + R(t) \cdot 400$$

Here 400 is the atmospheric carbon level in ppm_v. For the relative humidity level we calculate both absolute humidity levels for the incoming air and air existing air. We add these levels together and calculate the new relative humidity value, depending on the temperature of the mix.

Carbon Model

In addition to the air mixture effects on the carbon level explained in section B.1.2 we need to model the effects of the plant and the carbon supply on the carbon level in the greenhouse. First, we calculate the existing

$$V_{existing} = \frac{s_{t,carbon\_ppm} \cdot greenhouse\_height}{1e6}$$

$$V_{injected} = \dot{V}_{carbon\_max} \cdot a_{t,carbon} \cdot dt$$

$$ds_{t,carbon\_ppm,s}(t) = 1e6\big(\frac{V_{existing} + V_{injected}}{V_{injected} + greenhouse\_height} - s_{t-dt,carbon\_ppm}\big)$$

Here $V_{existing}$ is the existing carbon dioxide volume in the greenhouse and $V_{injected}$ is the injected volume into the greenhouse, both in m^3. We apply these to calculate $ds_{t,carbon\_ppm,s}(t)$, which is the carbon ppm_v change. Our carbon dioxide model depends on the carbon injections from the carbon supply, as well as the changes in carbon level caused by plant respiration and photosynthesis. This is shown in the equation below, which determines the new carbon level for time t.

$$s_{t,carbon\_ppm} = s_{t-dt,carbon\_ppm} + ds_{t,carbon\_ppm,p}(t) + ds_{t,carbon\_ppm,s}(t) \tag{B.25}$$

Humidity Model

In this model we calculate the change in evaporated water, which in turn affects the humidity level and air temperature in the greenhouse. The change in evaporated water is defined by the equation below.

$$dm_{humidity} = m_{evaporated} - m_{condensed}$$

Here $dm_{humidity}$ is the absolute change in humidity, $m_{evaporated}$ is the amount of evaporated water and $m_{condensed}$ is the amount of condensed water, all in kg. To continue, we first describe two ratios:

$$R_{air,dry} = \frac{abs\_vapor\_density(s_{t,temperature\_air}, \quad s_{t,relative\_humidity})}{\rho_{air}}$$

$$R_{air,max} = \frac{saturated\_vapor\_density(s_{t,temperature\_air})}{\rho_{air}}$$

Here $R_{air,max}$ and $R_{air,dry}$ denote the humidity to dry air ratio and the maximum humidity to dry air ratio respectively, in $kg\ kg^{-1}$. We applied two functions $abs\_vapor\_density$ and $saturated\_vapor\_density$ that calculate absolute vapor density and maximum vapor density respectively, based on [74]. Furthermore we use an air density of $\rho_{air} = 1.204\ kg\ m^{-3}$. We calculate the evaporated water using the Equation B.26.

$$m_{evaporated} = \frac{dt}{3600} \dot{A}_{exposed} \Theta (R_{air,max} - R_{air,dry}) \tag{B.26}$$

Here $A_{exposed}$ is the fraction of area where water is directly exposed to the air and Θ is the evaporation coefficient of 26 [74]. In our greenhouse we have walkways and there are also trays on top of the water, therefore we assume $A_{exposed} = 0.04$. For the condensation model we assume a constant airflow near the greenhouse roof. Here the air cools down and the maximum humidity reduces, causing condensation on the windows. We approximate the glass temperature inside with a simple approximation.

$$T_{glass} = \frac{1}{3} \cdot s_{t,temperature\_air} + \frac{2}{3} \cdot s_{t,temperature\_out}$$

Here T_{glass} is the glass temperature in K. We can now estimate the maximum and current absolute vapor density near the wall.

$$\rho_{near\_wall} = abs\_vapor\_density(T_{glass}, \quad s_{t,relative\_humidity})$$

$$\rho_{near\_wall,max} = saturated\_vapor\_density(T_{glass})$$

Here ρ_{near_wall} and $rho_{near\_wall,max}$ are the current and maximum absolute vapor density near the wall in $kg\ m^{-3}$. We can now calculate the amount of condensed water using the equation below:

$$m_{condensed} = (\rho_{near\_wall} - \rho_{near\_wall,max}) \cdot dotV_{near\_wall} \cdot dt$$

Here $dotV_{near\_wall}$ is the volume flow of air near the wall, we assume this to be approximately 2 liters a second which is $dotV_{near\_wall} = 0.002\ m^3\ s^{-1}$. We can now update the relative humidity using the change in absolute humidity from Equation B.26.

B.1.3 Weather Model

To create a truthful and accurate representation of the Dutch weather we decided to use historical data for the weather model. As the province of South-Holland contains the highest density of greenhouses in the Netherlands we selected historical weather dataset measured at Hoek van Holland by the Royal Dutch Weather Institute KNMI [34]. We use hourly weather data measured between 2001 and 2020. The weather data contains the average wind speed, the air temperature, the illumination level and the relative humidity.

To prevent overfitting on the weather two modes for running the weather model have been developed, one for training and one for evaluation. Both modes select a starting date for the model. For any given moment in the growing cycle the weather model will lookup the weather relative to this starting date. The selection of the starting dates of both modes is motivated below:

1. **Training**
   Simulates the weather starting at an uniformly sampled day between the 1st of February and the 31st of March, for an uniformly sampled year between 2001 and 2010. Here we randomize the weather the agent experiences to prevent overfitting on a particular weather sequence.

2. **Evaluation**
   Simulates the weather at the 1st of March for each year between 2011 and 2020. As the 1st of March is in the same period of the year but in a different year we will evaluate the agent on weather that it has not seen before. This allows us to analyze the capability of the agent to generalize to future weather in the area.

B.1.4 Economic Model

The economic model calculates the balance for the episode. It keeps track of cumulative heat, carbon and electricity usage. In our model, we apply a variable electricity price:

$$s_{t,price\_elec} = \begin{cases} price_{high,elec}, & \text{if } 7 \leq s_{t,hour} \leq 23 \\ price_{low,elec}, & \text{otherwise} \end{cases}$$

Furthermore, in our model the gas and carbon prices are constant. Simple multiplication yields the variable cumulative costs for heat, carbon and electricity. Additionally, we apply a fixed cost per day for greenhouse deprecation equipment and rents. When the lettuce reaches the selling mass, we sell it. This results in the following equation for final balance:

$$s_{t,balance} = s_{t,gains\_plant} - s_{t,costs\_fix\_total} - s_{t,costs\_var\_total}$$

Where:

$$s_{t,gains\_plant} = \begin{cases} s_{t,avg\_plant\_per\_m2} \cdot price_{plant} \cdot s_{t,quality}, & \text{if } s_{t,mass} \geq 250 \\ 0, & \text{otherwise} \end{cases}$$

$$s_{t,costs\_fix\_total} = price_{fix} \cdot s_{t,time}$$

$$s_{t,costs\_var\_total} = s_{t,costs\_var\_elec} + s_{t,costs\_var\_heat} + s_{t,costs\_var\_carbon}$$

The prices in this model are described in section B.2 and the descriptions of the state variables are given in section B.3.

B.2 Simulator Parameters

Parameter	Description	Value	Unit
greenhouse_height	Greenhouse height	3	m
heating_efficiency	Thermal heater efficiency	0.8	
reflectance_greenhouse	Fraction of outside light reflected	0.4	
lamp_intensity	Brightness of lamps	250	$\mu\ mol\ s^{-1}$
price_electricity_on_peak	On peak price for electricity	0.168	$EUR\ kWh^{-1}$
price_electricity_off_peak	Off peak price for electricity	0.088	$EUR\ kWh^{-1}$
price_heating	Constant gas price	0.0134	$EUR\ MJ^{-1}$
price_carbon	Constant carbon price	0.14	$EUR\ kg^{-1}$
price_fixed_costs	Constant hourly deprecation price	0.0032	$EUR\ h^{-1}$
greenhouse_heat_capacity	Heat capacity for the greenhouse	60000	$J\ K^{-1}$
heat_exchange_open_window	Heat exchange when the window is open	10	$W\ K^{-1}$
roof_heat_transfer_rate	Heat exchange for the roof	1.136	$W\ K^{-1}$
max_ventilation_capacity	Maximum airflow with open windows	50	$m^3\ s^{-1}$

Table B.1: Greenhouse climate simulation parameters used in this work

Parameter	Description	Value	Unit
growth_temp_optimal	Optimal plant growing temperature	25	
growth_temp_range	Maximum distance from optimal for growth	25	
growth_carbon_scalar	Scalar value to shape the carbon growth curve	400	
growth_par_scalar	Scalar value to shape the par growth curve	200	
max_daily_growth	Maximum relative growth in a day	1.194	
temperature_tolerance_upper	Maximum temperature before damage starts	30	$°C$
temperature_tolerance_lower	Minimum temperature before damage starts	0	$°C$
tipburn_tolerance_growing_rate_upper	Maximum growing rate before tipburn starts	0.8	
humidity_tolerance_upper	Maximum humidity before damage starts	0.9	
humidity_tolerance_lower	Minimum humidity before damage starts	0.1	

Table B.2: Greenhouse climate simulation parameters used in this work

B.3 Environment Spaces

Normalization Method

For each value in both spaces a normalization range has been defined that is used to map the values to the $[0, 1]$ domain using the equation below.

$$X_{norm} = \frac{X_{ori} - R_{low}}{R_{high} - R_{low}} \tag{B.27}$$

Here X_{ori} is the original value, X_{norm} is the normalized value and R is the normalization range as found in Appendix B.3. The normalization range R has a lower and upper bound denoted as R_{low} and R_{high}.

The boundaries for the normalization range have been carefully selected to ensure that both normal working conditions and extreme conditions fall within the interval. An example of this is the greenhouse air temperature which is typically in the range 18 to 28 $°C$. By analyzing the Dutch weather in the period of 2011-2020 we find that the minimal and maximal outside temperature were -11.4 $°C$ and 38.3 $°C$ respectively. The normalization range $R_{tair} : [-20, 60]$ should cover all operation ranges. Note that the normalization range is applied only to the state the agent observes, not the internal state of the greenhouse.

Spaces

Variable	Description	Range	Unit
heater	Power of the greenhouse heater	[0, 200]	$W\ m^{-2}$
ventilation	Ventilation window position (1=open)	[0, 1]	
carbon	Mass flow rate from the CO_2 supply tank	[0, 1.0e-6]	$kg\ s^{-1}\ m^{-2}$
blackout	Position of the blackout screen (1=open)	[0, 1]	
transparent	Position of the transparent screen (1=open)	[0, 1]	
lamps	Fraction of lamps that is on (1=all)	[0, 1]	

Table B.3: Action space description for the greenhouse environment

Variable	Description	Range	Unit
heat	Heater power loss or increase	[-10, 10]	W
light	PAR light loss or increase	[-25, 25]	$\mu\ mol\ s^{-1}$
carbon	Carbon mass flow rate from inside	[-1.0e-8, 1.0e-8]	$kg\ s^{-1}\ m^{-2}$
humidity	Water evaporation or condensation rate	[-5.0e-7, 5.0e-7]	$kg\ s^{-1}\ m^{-3}$

Table B.4: Adversarial action space description for the adversarial greenhouse wrapper

Variable	Description	Range		Unit
time	Hours since start of growing cycle	[0,	2400]	h
week	Week number since 1st of Jan	[0,	52]	w
hour	Current hour of the day	[0,	24]	h
cum_elec_usage	Cumulative electricity usage since the start of simulation	[0,	2400]	$kWh\ m^{-2}$
cum_heat_usage	Cumulative heat usage since the start of simulation	[0,	2400]	m^{-2}
cum_carbon_usage	Cumulative carbon usage since the start of simulation	[0,	10]	m^{-2}
price_elec	Current electricity price	[0,	1]	$EUR\ kWh^{-1}$
price_heat	Current heat price	[0,	10]	$EUR\ MJ^{-1}$
price_carbon	Current carbon price	[0,	10]	$EUR\ kg^{-1}$
costs_var_elec	Cumulative electricity costs	[0,	10]	$EUR\ m^{-2}$
costs_var_heat	Cumulative heating costs	[0,	10]	$EUR\ m^{-2}$
costs_var_carbon	Cumulative carbon costs	[0,	10]	$EUR\ m^{-2}$
avg_plant_per_m2	Average number of plants per square meter	[0,	100]	m^{-2}
gains_plant	Turnover when plants are sold	[0,	50]	$EUR\ m^{-2}$
costs_var_total	Cumulative costs for heating, electricity and carbon	[0,	50]	$EUR\ m^{-2}$
costs_fix_total	Cumulative fixed costs for running the greenhouse	[0,	50]	$EUR\ m^{-2}$
balance	Current final balance	[-50,	50]	$EUR\ m^{-2}$
mass	Mass of a single plant	[0,	400]	g
quality	Quality of the plants (1=perfect)	[0,	1]	
plants_per_m2	Current number of plants per area	[1,	80]	m^{-2}
temperature_air	Temperature of the greenhouse air	[-20,	60]	$°C$
pipe_temperature	Temperature of the heating pipe	[0,	100]	$°C$
relative_humidity	Relative humidity (0=dry)	[0,	1]	
par_light	PAR light level just above the plant	[0,	2000]	$\mu\ mol\ s^{-1}$
carbon_ppm	Carbon level inside in volumetric parts per million	[0,	6000]	ppm_v
wind_speed	Wind speed inside the greenhouse	[0,	20]	$m\ s^{-1}$
electric_power	Electric power consumed	[0,	1000]	W
illumination_out	Illumination level outside	[0,	1000]	$W\ m^{-2}$
relative_humidity_out	Relative humidity outside (0=dry)	[0,	1]	
temperature_out	Temperature outside	[-20,	60]	$°C$
wind_out	Average wind speed outside	[0,	20]	$m\ s^{-1}$

Table B.5: Observation space description for the greenhouse environment