Policy Search for Automatic Polyculture Farming using Reinforcement Learning

handed in
MASTER'S THESIS

Bachelor of Science Sebastian Oehme

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	Dr. Alejandro Agostini
Start:	01.12.2020
Delivery:	15.01.2022

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

November 24, 2020

M A S T E R ' S T H E S I S
for
Sebastian Oehme
Student ID 3653382, Degree MSEI

Policy Search for Automatic Polyculture Farming using Reinforcement Learning

Problem description:

Polyculture farming constitutes more complex challenges compared to monoculture farming. Like in many real world applications, there is no perfect simulation available to search for possible cultivation policies in large garden configuration spaces [1]. Furthermore, this environment is only partially observable and is governed by complex, non-stationary dynamics. The research of agents that plan decisions in advance in such environments has been a major task of scientists in the field of artificial intelligence. Recently, progress has been made in deep reinforcement learning (deep RL) that permits finding sub-optimal policies in such environments [2] – especially planning algorithms based on look-ahead search in challenging domains [3], which would permit dealing with the gradual and delayed correlations typical of plant caring scenarios [4]. The aim of this work is to tackle the challenges of polyculture farming by first extending the simulator that was developed in the context of the Alpha-Garden project [5] to enable researching robust cultivation policies with a RL approach that is based on a tree-based look-ahead search for planning with a learned model.

Tasks:

- Literature overview on RL approaches suitable for partial observable environments.
- Implement an RL approach for polyculture farming in the Alphagarden scenario.
- Improve the Alphagarden simulator for a better sim2real transfer.
- Performance evaluation of RL approach compared to existing decision-making strategies.

Bibliography:

[1] Y. Avigal et al. Simulating Polyculture Farming to Tune Automation Policies for Plant Diversity and Precision Irrigation, in *CASE* 2020.
[2] J. Schulman et al. Proximal policy optimization algorithms, in *arXiv preprint:1707.06347* 2017.
[3] D. Silver et al. Mastering the game of Go with deep NNs and tree search, in *Nature* 2016.
[4] A. Agostini et al. A cognitive architecture for automatic gardening, in *Computers and Electronics in Agriculture*. 2017.
[5] K. Goldberg et al. AlphaGarden, *[Online]. Available: http://rapid.berkeley.edu.*.

Supervisor:	Dr. Alejandro Agostini
Start:	01.12.2020
Intermediate Report:	15.01.2022
Delivery:	15.01.2022

(D. Lee)
Univ.-Professor

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Munich,

Place, Date	Signature

Abstract

Today monoculture farming is the standard for industrialized agriculture. It allows producing food on a large scale. Nevertheless, a significant drawback is the loss of diversity. Polyculture farming mitigates this issue while benefiting from reducing the demand for resources, such as water. However, polyculture farming is a complex task as it has to consider multiple growth characteristics, and the environment is often only partially observable. This work addresses the challenge of automating polyculture farming using a first-order simulator to identify efficient treatment policies. We utilize randomized environment dynamics combined with a reinforcement-learning-based approach to learn a policy that increases garden coverage and diversity in simulation. We add a network architecture with a memory layer to the utilized Proximal Policy Optimization algorithm to overcome garden configurations that are only partially observable due to occlusion. In simulation experiments, the policy outperforms the previous state-of-the-art heuristic policy. Our simulation experiments serve as a groundwork for real-world application.

Contents

Chapter 1

Introduction

For many centuries cultivation of plants has been essential to feeding humanity. Modern farming made use of industrialized means of production. This industrial form of agriculture has been a significant factor in feeding an ever-growing population. Innovation in agricultural machinery and farming methods supports this. Today, however, about 7.2 billion inhabitants live on earth, and nearly 8.9% of them are affected by hunger [FAO20]. To address this problem, the United Nations state in their Sustainable Development Goal Zero Hunger [1] that

> *[...] a profound change of the global food and agriculture system is needed if we are to nourish the more than 690 million people who are hungry today - and the additional 2 billion people the world will have by 2050. Increasing agricultural productivity and sustainable food production are crucial to help alleviate the perils of hunger.*

Today industrial farming is often growing monocultures. The aim is to maximize yield. However, cultivation of a single crop type requires substantial pest control, high use of fertilizer, and water resources [RLD$^+$14].

Alternative forms of agricultural cultivation that have been used for centuries are polycultures. It can serve as a more sustainable method where multiple crop types that support each other are grown next to each other. Thereby they imitate the diversity of natural ecosystems [GA82]. A typical example is the intercropping of maize, beans, and squash plants [MP16]. Polyculture farming, on the one hand, benefits from resistance to pests and weeds, reduced soil erosion, and increased resource efficiency for water and nutrients [Ris83, Lie87, CCO18]. On the other hand, it is more difficult to service than monoculture. It requires specific farming methods for each plant type, such as individual maintenance and plant treatments. Hence, polyculture farming with plant types with different irrigation and fertilization schedules, varying germination times, and growth rates requires more labor than large-scale monoculture farming.

[1]See https://www.un.org/sustainabledevelopment/hunger/, accessed 2021-11-18

To address these issues, we recently presented AlphaGarden, a robotic system for autonomous polyculture cultivation [PAT$^+$21]. It serves as a research platform to identify sustainable control policies. These aim towards maximizing plant coverage while maintaining diversity and minimizing resource usage. Control of a polyculture system requires a multitude of cues from both plants and their dynamic environment. The process is complicated by model errors and partial observability of the garden environment. For example, soil sensors may have noisy readings, and an overgrowing garden cause parts of the environment to be occluded. Learning treatment policies in nature may require many growth cycles and a high amount of resources. These constraints motivate the use of simulation to speed up the process. Simulators allow making the policy robust before tackling any physical environment [RVR$^+$16, PAZA18].
Like in many real-world applications, there is no perfect simulation available to search for effective cultivation policies. Despite that, AlphaGardenSim [AGW$^+$20] - an parametric, first-order simulator - serves as a first step towards *i)* approximating the problem setting and *ii)* researching automation control policies that at some point may tend to the physical AlphaGarden. The simulator is limited by its ability to model the environment accurately. For example, the simulation depends on realistic parameters for the plant growth and soil water models. Furthermore, it has to account for natural perturbations.

Prior work [AGW$^+$20] presented a heuristic policy that sought to achieve high plant coverage and garden diversity. However, in simulation, this policy struggles to care for plants with substantial differences in plant characteristics. In recent years, research has shown that reinforcement learning agents are able to perform complex tasks better than humans, such as controlling robots [KBP13], playing video games[MKS$^+$13] and treating monoculture or single plant setups [TKBL05, CCW$^+$21, HBDV$^+$21]. Agents show impressive results even without complete knowledge about the underlying environment dynamics or in a partial observable setting [HS15]. In a plant caring scenario, imperfect state information is a challenging problem. Partial state information could arise from occlusion in overgrowing environments or soil moisture information only available for a specific region. Furthermore, past research has shown that reinforcement learning agents trained in simulation on a range of randomized environment dynamics generalize to out-of-range (unseen) dynamics [LKJ01, TS10]. Reinforcement learning in a polyculture farming domain has not received much attention, while it bears the potential to find robust policies that generalize to unseen garden environments.

1.1 Problem Statement

In our recent work [PAT$^+$21], we present the AlphaGarden Autonomous Pipeline, depicted in Fig. 1.1. It shows a pipeline for autonomous decision-making to treat a polyculture garden bed. At its core, it utilizes AlphaGardenSim to simulate future environments and to decide on beneficial treatment actions. The simulator and control policy plays a vital role in succeeding in this domain.

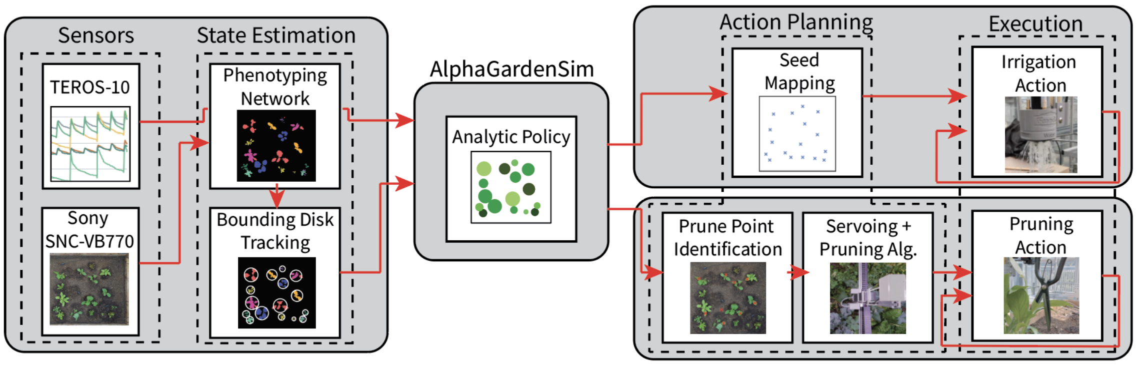


Figure 1.1: AlphaGarden Autonomous Pipeline. Data from soil moisture sensors and an overhead camera is processed to estimate the garden state. The state estimate is passed to AlphaGardenSim to simulate future evolution and determine appropriate treatments with the *analytic policy*. Afterward a control sequence is planned and executed in the real garden. Source [PAT+21].

Given the previous motivation, this work is focused to *i)* improve AlphaGardenSim's dynamics model and *ii)* identify suitable reinforcement learning agents and validate them in a simulated polyculture garden environment.
Therefore, we propose the following research hypotheses:
A reinforcement learning agent can outperform the state-of-the-art policy for polyculture plant treatment within AlphaGardenSim. Furthermore, such an agent can generalize to garden configurations with altered dynamics on which they have not been trained. The performance in these new garden configurations serves as a proxy to assess transferability to reality.

This report is structured as follows: in the next section, we give an overview of the related work for simulations and reinforcement-learning-based approaches for polyculture farming. Chapter 2 introduces the polyculture farming simulator AlphaGardenSim and our policy-gradient-based method with a memory layer PPO_{lstm} for plant care. In Chapter 3, we present our experiments to validate the adapted simulator and our proposed method. Additionally, we discuss the results with respect to our research goal. Finally, we draw our conclusions in Chapter 4.

1.2 Related Works

This section gives an overview of related work in plant and treatment simulations, followed by an analysis of ongoing research in reinforcement learning that applies to the agricultural setup.

Plant and Treatment Simulation Plant simulations aim to accurately model a plant's life cycle or specific characteristics. They are beneficial to speed up research of farming strategies. In practice, they require a lot of time and resources. Historically plant growth simulations have been grouped into three types: process-based models, structural plant models, and functional, structural plant models [VME07]. Process-based models [Heu99] seek to predict plant growth according to endogenous plant properties. Biomass production is shared from a shared pool. Predominant environmental conditions influence this process. These conditions impact structural plant models as well. This type of model aims to simulate a plant's morphology [DL05]. Functional, structural plant models represent a hybrid combination of the two prior models. They incorporate relationships between underlying endogenous processes and a plant's morphology [VME07]. Applications of models that integrate detailed structural information have mainly been on individual plants. This is due to the high number of parameters needed to model plant growth and the surrounding ecosystems, which coincides with computational complexity.

Plants models for single plant types are often tailored towards monoculture farming. They simulate a crop's response to soil and water conditions. AquaCrop [RSHF09] and DSSAT [JHP$^+$03] are widely used in this setting. However, they are not well-suited for a polyculture garden with many plant species as inter-plant competition has a profound impact. In this domain, few simulators have the necessary ability to model the growth of multiple species [SDB$^+$20].

AlphaGardenSim [AGW$^+$20] simulates a polyculture garden using first-order models of individual plant growth. It is a process-based model that incorporates basic structural information - i.e., each plant's height and radius. It simulates each plant with a circular zone-of-influence model [CB90] that is discretized on a coarse grid to reduce computational cost. Plants with overlapping zones compete for water and light. This setup allows to research treatment policies that aim to optimize leaf canopy coverage and diversity at garden level. However, the simulator is not tuned with real-world measurements in this work. The authors intend to do so in future work. Doing so is, on the one hand, a common approach in botany and agriculture [FZS$^+$08, PMV$^+$12], as parameter tuning with actual plants would require a substantial amount of time, possibly years, and resources. On the other hand, the plant growth and irrigation model in AlphagardenSim rely on multiple approximations. They are making it uncertain whether a policy learned in simulation is successful in reality.

In order to identify real-world treatment policies in simulation, its model parameters are usually grounded to the local environmental conditions. Training data is collected by observing the treatment responses of individual plants. Wiggert et al. [WAB$^+$19] presents a garden bed for monitoring water stress of a batch of plants (same type) to automate and optimize plant-level irrigation. Water stress has a profound impact on plant development [Jon07]. Important quantities for water management are the specific Permanent Wilting Point (PWP) [LEL$^+$18] and specific maximal Volumetric Water Content (VWC) [Kel05]. PWP and maximal VWC represent lower and upper bounds of available soil water to the plant, respectively. AlphaGardenSim does not model the latter in [AGW$^+$20] . Agostini et al. [AAF$^+$17] presents an automated robot platform for learning and executing plant treatments. They demonstrate this framework to treat a single plant type with water and nutrients. Furthermore, they used a simulator and cognitive decision-making framework for efficient training data generation. This data is used to extract cause-effects tuples for treatment planning [ATW14]. Plant treatments occur multiple times a day in their simulation. However, observable effects in growth response may vary in time and strength. Their system is designed to learn detrecting past events that impact plant evolution. The simulator adopts a non-linear growth model assuming an asymptotic final plant size proposed by Paine et al. [PMV$^+$12]. The model is validated on multiple plant species.

Reinforcement Learning Policies Recent work aims to overcome challenges - like long delays - in the agricultural domain with reinforcement learning using deep neural networks. Ban & Kim[BK17] and Wang et al. [WHL20] demonstrate superior green-house control in simulation with the Deep Deterministic Policy Gradient (DDPG) algorithm [LHP$^+$15] compared to realistic baselines. DDPG combines the actor-critic method [SB18] and deterministic policy gradient [SLH$^+$14] method. CropGym [OBA21] is a reinforcement learning environment with a process-based crop model to research fertilization treatments. Like AlphaGardenSim it is implemented with Open AI's common reinforcement learning interface Gym [BCP$^+$16]. In CropGym they demonstrate learning an efficient policy with the Proximal Policy Optimization (PPO) [SWD$^+$17] algorithm. PPO again belongs to the policy gradients methods. It incorporates small policy updates inside a Trust Region [SLA$^+$15] that aims to avoid catastrophic outcome when doing Stochastic Gradient Decent (SGD). Chen et al. [CCW$^+$21] present a simulator and Deep Q-Network (DQN) [MKS$^+$15] agent for irrigation management of rice fields. The previous approaches consider different types of treatments for monoculture farming. Each of these problems is set up as a Markov Decision Process (MDP). They rely on full state knowledge, i.e., states encapsulate all aspects of the past that impact the future. Hence, they fulfill the Markov property.

For AlphagardenSim, not all aspects of the environment's state are directly observable. Sutton & Barto [SB18] describe multiple approaches to solve Partially Observable Markov Decision Processes (POMDPs) with reinforcement learning. As with deep neural networks, function approximation has the potential to work in partial observable cases. They have a limited ability to substitute non-present state variables. However, this requires that enough information is embedded in the training data. The problem setup can be rearranged to overcome this issue as an alternative.

A compact, logical sequence of past environment observations and control, called *history*, can restore the Markov property. A *history* is used in combination with a state-update function to perform state estimates incrementally. A simple and effective approach is the *kth-order history* approach, where the last k observations and controls are used for state estimation. As an extension to the previous approach, agents have been enhanced with recurrent neural networks that function as memory. Long Short Term Memory (LSTM) networks [HS97] have been used to solve POMDPs with policy gradient methods [WFPS07], in combination with PPO for drone navigation [HHA21], and in combination with a DQN agent to play computer games [HS15].

It is a challenge to transfer reinforcement learning policies that have been learned in simulation to unseen, realistic environments [TS10]. A technique that deals with this problem is *Sim2Real*. LaValle & Kuffner [LKJ01, PAZA18] demonstrate an approach known as *dynamics randomization*. Agents are trained with different environment configurations on each episode. These parameters are treated as a random variable, and instances get sampled from feasible intervals. AlphaGardenSim has an interface to randomize some of the environment dynamics. In the same work, the authors present a heuristic *analytic automation policy* to control the polyculture setup. However, the policy could not handle plants with significant differences in germination times and growth rates.

Chapter 2

Technical Approach

The first section of this chapter introduces the polyculture simulator AlphaGarden-Sim [AGW+20]. Furthermore, we present our contributions towards model extensions and runtime optimization. Section 2.2 describes the principle of operation for the agent-environment interaction and our reinforcement learning algorithm for plant treatments in a polyculture setup.

2.1 AlphaGardenSim

The original AlphaGardenSim can approximate plant growth at 9000 times the natural growth speedh [AGW+20]. However, many reinforcement learning algorithms need to interact with the environment for many iterations to learn a (near) optimal policy while being limited in computational capacity. Model extensions require additional computation steps, and this affects performance. An increase in simulation efficiency allows expanding the search space, which is beneficial for optimization problems. Therefore, we reimplement AlphaGardenSim. For efficient tensor computations, we use the NumPy programming library for the Python language [HMvdW+20]. Furthermore, we expand the randomization interface of the simulator.
Some of the original notation from AlphaGardenSim [AGW+20] is changed to allow framing the problem in common reinforcement learning notation [SB18]. Parts of the simulator extensions in this section have been published at [ADW+21], as a contribution by the author of this thesis. We state our previous contributions and new changes to the simulator in the following section.

2.1.1 Simulator Quantities

AlphaGardenSim is introduces as a framework to find treatment policies for efficient cultivation of a polyculture garden space in a finite time horizon T. The simulation is updated at each discrete time step $t \in \{1, ..., T\}$, which is specified in units of days. The environment is represented as a discrete $X \times Y$ grid with cell $o_{x,y}$, where x and y

are the row and column coordinates, respectively. For this thesis we set the grid cells to size 0.01m $\times$ 0.01m. The garden grid consists of *soil* and contains I plants of K plant species. The *soil* grid holds at time t a specific VWC $w_{x,y,t}$ in each cell. Each plant $i =\in \{1, ..., I\}$ of plant type $k \in \{1, ..., K\}$ has a cylindrical structure. The plant structures are defined by three I-tuples, these are, the seed locations $(u_i)_{i=1,...,I}$ in the garden, the current plant radii $(r_{t,i})_{i=1,...,I}$ and the current plant heights $(h_{t,i})_{i=1,...,I}$. Their current leaf areas $L_{i,t}$ are calculated for each plant as the sum of the cell area inside its cylinder base. Furthermore, each plant i at time step t is in a growth stage $G_{i,t} \in \{\text{germination}, \text{vegetative}, \text{reproductive}, \text{senescence}, \text{death}\}$, we note the details on the individual stages in the next section. An other plant attribute that is modeled is the plant health level $h_{x,y,t}$ at cell $o_{x,y}$ at time t. The plant health level is determined by water stress, i.e, normal, under or overwatered, which we further define in the next section. With sufficient resources plants will grow over a period of time, which might cause plants to overlap. Hence, multiple plant types or soil could be present in a cell. Therefore, the simulator approximates the canopy cover at each cell $o_{x,y}$ at time t, as $\mathbf{d}_{x,y,t}$, a vector of length $K + 1$. It represents a distribution over possible vegetation and soil in a cell visible to an overhead observer. In this work, the plant type is certain, which makes $\mathbf{d}_{x,y,t}$ a 1-hot vector.

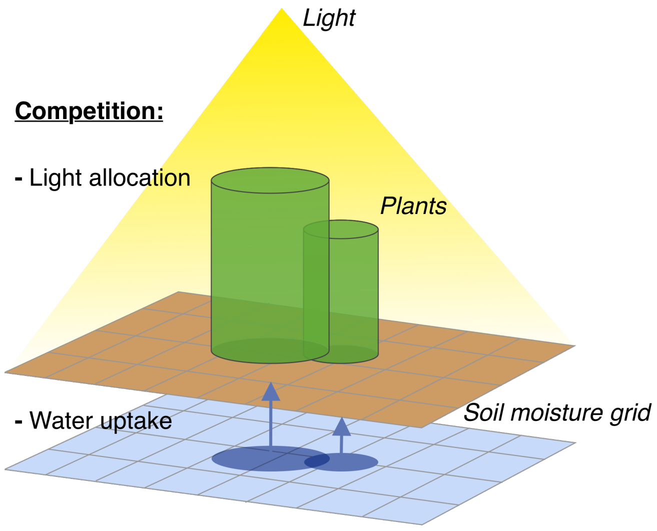
2.1.2 Garden Dynamics

The simulator executes three major updates in each update step: lighting, water use, and plant growth. Nearby plants compete for these resources. We depict this process in Fig. 2.1(a).

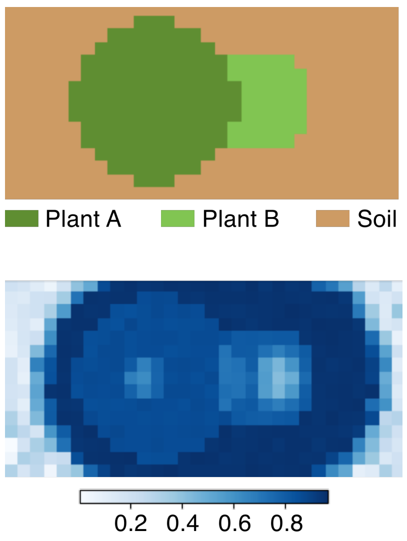
Light model. The light model remains the same as in the original paper. The garden receives a fixed amount of light at each cell $o_{x,y}$ per simulation step. Only the $(n-1)^{\text{th}}$ tallest plants at each grid cell transmit light. With each layer, the available light decays exponentially with $(\frac{1}{2})^n$, with $n \in \{0, 1, 2\}$. Otherwise, no light is available.
Plants seek to allocate light with their current leaf area $L_{i,t}$ to conduct photosynthesis. $L_{i,t}$ of plant i at time t is the total area of cells which are less than distance $r_{t,i} \in (r_{t,i})_{i=1,...,I}$ away from its seed location $u_i \in (u_i)_{i=1,...,I}$. A plant's total amount of accumulated light $l_{\text{alloc},i,t}$ at time t is the sum of light it is able allocate with its current leaf area.

Water model. As described in [AGW+20], the simulator uses a discrete-time linear approximation of Richards equation [TWC+18] for soil water dynamics. The soil water model is defined as:

$$w_{x,y,t} = \max(w_{x,y,t-1} - w_{\text{loss}} + a_{\text{water},x,y,t} - w_{\text{up},x,y,t}, 0) \tag{2.1}$$



(a) Simulator Model



(b) Simulator observation

Figure 2.1: Example of AlphaGardenSim at intermediate simulation time step t. **Left:** Schematic representation of simulator and inter-plant competition for light and water resources. Plants have a circular zone of influence, which is discretized by a grid. Each grid cell where plants overlap has to split water resources equally while taller plants allocate more light. **Right:** Top: Canopy coverage with two plants (types: A, B) and soil. Bottom: Soil grid with present VWC.

AlphaGardenSim uses the previous soil moisture content $w_{x,y,t-1}$, water flux from irrigation $a_{\text{water},x,y,t}$, plant water uptake $w_{\text{up},x,y,t}$, and local water loss w_{loss} to calculate the current soil moisture value for each discrete cell $o_{x,y}$ at time t. Fig. 2.1(b) depicts the canopy view of a simulation with two plants and the accompanying water grid for an intermediate time step t.

In our previous work [ADW+21], we report on experiments in a physical testbed to refine the water model. Irrigation is modeled as a fixed amount of water (in m^3) that is applied to a circular zone with radius 0.09m around some center coordinate inside the soil grid. The amount of water applied to each grid cell is uniform inside a 0.04m radius and decays exponentially as it approaches the edges of the irrigation circle. We incorporate the experiment results into the simulator. In this thesis, the irrigation amount a_{water} is set to 0.0002m^3, and it can only be applied at plant center coordinates.

The local water loss parameter, w_{loss}, and the maximal VWC are also tuned for our findings in [ADW+21]. Water loss is treated as a random variable of univariate Gaussian distribution with mean of $0.042\text{m}^3\,\text{m}^{-3}$ and a standard deviation of $0.0048\text{m}^3\,\text{m}^{-3}$. In this work, w_{loss} is sampled once for each growth episode of length

T. The maximal VWC indicates the water storage capacity of the soil. Based on the experiments from before, we estimate max Volumetric Water Content, w_{max}, to be $0.3\text{m}^3\,\text{m}^{-3}$. This value depends on the soil properties. Thus, we clip both $w_{x,y,t-1}$ and $w_{x,y,t}$ at this value, Eq. 2.1 is then transformed to:

$$w_{x,y,t} = \text{clip}(w_{x,y,t-1} - w_{\text{loss}} + a_{\text{water},x,y,t} - w_{\text{up},x,y,t}, 0, w_{\text{max}})^1 \tag{2.2}$$

Each plant has access to the soil water. The accessible area is constrained to the plant's circular base. In general, allocation of water depends on a plant's available light resources at time t, $l_{\text{alloc},i,t}$, water competition in cells where plants overlap, and the specific PWP.

In AlphaGardenSim the current available light defines the maximal amount of water required by a plant:

$$w_{\text{upmax},i,t} = \frac{e_{\text{light}}}{e_{\text{water}}} \sqrt{l_{\text{alloc},i,t}} \tag{2.3}$$

Where e_{water} and e_{light} correspond to water use efficiency and light use efficiency, respectively, they are plant-specific quantities.
Plants have to compete for resources in cells where they overlap. In this work, available water in these cells is uniformly distributed among competing plants. The water amount plants desired for growth from the soil cells bellow is defined on a per plant basis as:

$$w_{\text{des},i,t} = \frac{w_{\text{upmax},i,t}}{L_{i,t}} \tag{2.4}$$

where $w_{\text{upmax},i,t}$ is a plant's desired water amount and $L_{i,t}$ is a plant's leaf area at time t. The actual current water uptake $w_{\text{act},x,y,i,t}$ from plant i cell $o_{x,y}$ and time t is than limited by the currently available water $w_{x,y,t}$ and a soil-specific PWP, w_{pwp}, as defined by:

$$w_{\text{act},x,y,i,t} = \min\left(w_{\text{des},i,t}, w_{x,y,t} - w_{\text{pwp}}\right) \tag{2.5}$$

Finally, the total actual water uptake of plant i at time t, $w_{\text{uptot},i,t}$, is the sum of all water uptake from individual cells that belong to the current leaf area $L_{i,t}$.

Plant growth The amount of allocated light and the water uptake define the biomass pool available for growth. In our previous work [ADW+21], we adopt a logistic growth profile assuming an asymptotic final size of the plant. [PMV+12]. The growth potential $g_{i,t}$ of plant i at time step t is modeled as:

$$g_{i,t} = e_{\text{water}} \cdot w_{\text{uptot},i,t} \cdot \left(1 - \frac{r_{t,i}}{r_{\text{max},i}}\right)$$

where $r_{t,i}$ is the plant's current radius and $r_{\text{max},i}$ is the plant's growth potential, which controls how large the plant will grow.

[1]We clip values outside the range [min, max] to the closest value inside the range: clip(value, min, max).

As in the original work, the growth potential is strategically distributed to vertical and radial growth to ensure maximum unoccluded leaf area. Avigal et al. [AGW+20] define $l_{\mathrm{o},i,t}$ and $l_{\mathrm{u},i,t}$, as the current number of points where plant i is occluded and unoccluded, respectively. They then model this dynamic as:

$$\begin{aligned} l_{p,t} &= \frac{l_{\mathrm{u},i,t}}{l_{\mathrm{u},i,t} + l_{\mathrm{o},i,t}} \\ b_{i,t} &= \max(k_1, \min(l_{p,t}, k_2)) \\ g_{\text{radial },i,t} &= b_{i,t} g_{i,t} \\ g_{\text{vertical },i,t} &= (1 - b_{i,t}) g_{i,t} \end{aligned} \tag{2.6}$$

Here, k_1 and k_2 are plant-specific parameters that control the ratio of $g_{i,t}$ plants dedicate to radial growth. After executing the three update steps, lighting, water use, and plant growth, the radius and height are incremented according to the computed ratio.

Plant Stages In AlphaGardenSim the plant life cycle consists of five consecutive growth stages: g$G_{i,t}$: germination, vegetative, reproductive, senescence and death. Each stage lasts a number of time steps. Stage duration are treated as random variable sampled from a plant-specific truncated discritized normal distribution, assuming that plants of the same type share transition times between stages. In our work, we truncate the distribution and thereby limit sampling range to a feasible interval. The truncated normal distribution allows to reconfigure the parameter interval before evaluating policies. This configuration allows to assess the policies ability to generalize to unseen garden setups. Details on randomizing the garden environment follow in Sec. 2.2.3.

In AlphaGardenSim the stages are modeled as follows. Germination starts when the seeds are planted. In this stage, the plants have no current radius and height. At the end of the stage, plants germinate with a specific height and radius. During the vegetative stage, the plants allocate resources and grow according to the growth model specified above, unless it experiences water stress, as detailed in the next paragraph. The plants do not change in radii or heights during the reproductive stage unless they experience water stress, as detailed in the paragraph below. Otherwise, the plants behave similarly to the vegetative stage. During the senescence stage, the plants start to wilt. The plants allocate less water than before, and their radii shrink exponentially. However, their heights remain the same. The desired water amount $w_{\mathrm{des},i,t}$ of plant i is adjusted throughout the senescence stage. The adjusted desired water amount $\widetilde{w}_{\text{des },i,t}$ decreases linearly to 0 over time:

$$\widetilde{w}_{\text{des },i,t} = \frac{1 - t'}{t_s} w_{\mathrm{des},i,t} \tag{2.7}$$

where t' is the amount of time the plant has spent in the senescence stage, and t_s is the total duration of the senescence stage. In our work, the plant is removed from the garden, whereas it continues to occupy garden space in the original simulator.

Water Stress Another part that remains unchanged from the original work is the model for water stress. AlphaGardenSim models water stress during the vegetative and reproductive stages. A plant receives sufficient irrigation if the following conditions are satisfied:

$$w_{\mathrm{T},i,t} \geq T_o \cdot w_{\mathrm{des},i,t} \tag{2.8}$$

$$w_{\mathrm{uptot},i,t} \leq T_u \cdot w_{\mathrm{des},i,t} \tag{2.9}$$

Where T_o and T_u are over and underwatering threshold parameters, $w_{\mathrm{T},i,t}$ and $w_{\mathrm{uptot},i,t}$ are the total amount of soil moisture within the current radius of plant i and its actual total water uptake, respectively. Otherwise, the plant enters into a state of water stress. At every time step, the plant experiences stress, the radius decays exponentially. Once the plant is wilted, it dies.

2.2 Policy Search

In the previous section, we give a complete description of AlphaGardenSim and our changes for this thesis. We intend to use AlphaGardenSim to search for an efficient control policy that outperforms the current state-of-the-art *analytical policy*. The PPO algorithm performs comparable or better than state-of-the-art approaches for reinforcement learning problems [SWD$^+$17]. The research community has received the method well and found its way into the agricultural domain. Overweg et al. [OBA21] use PPO to learn efficient fertilization management policies. Therefore, we use PPO to find efficient treatments for our policy culture setup. This section presents fundamentals of reinforcement learning with policy gradient methods, especially PPO, and extensions that we utilize to learn a treatment policy for AlphaGardenSim.

2.2.1 Principle of Operation

Reinforcement learning algorithms are developed for sequential decision-making. Sutton & Barto [SB18] formulate the sequential decision-making as an MDP, as depicted in Fig. 2.2.
At time step t, the environment receives an action $a_t \in \mathcal{A}$ and produces a state vector $s_t \in \mathcal{S}$, where $\mathcal{A}$ and $\mathcal{S}$ are the action and state spaces, respectively. The state is updated according to transition dynamics $P(s_{t+1} \mid s_t, a_t)$. It denotes the probability of transitioning from state s_t to state s_{t+1} by taking an action a_t. After the transition to state s_{t+1}, the agent observes a reward R_t. The Markov property implies that the probability of transition only depends on the current state. Repeating the previous

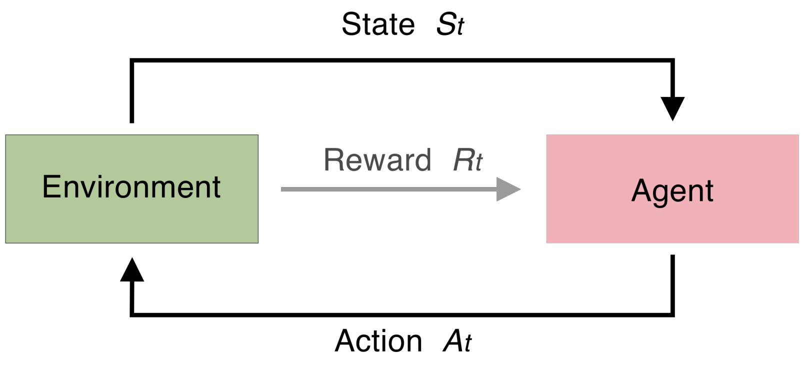


Figure 2.2: Interaction of reinforcement learning agent with environment in a MDP.

interaction leads to a trajectory τ: $(s_t, a_t, s_{t+1}, a_{t+1}, ...)$, a sequence of states and actions. For episodic problems a trajectory is called an episode when a terminal state is reached at time T. Here the environment is reset to initial conditions. The cumulative reward of an episode that the agent observes is referred to as return. The objective of a reinforcement learning agent is to find an optimal policy $\pi^*(a_t \mid s_t)$ that maximizes the expected return J:

$$J = \mathop{\mathbb{E}}_{\tau}[R_\tau] = \sum_t \mathop{\mathbb{E}}_{(s_t,a_t)} \gamma^t \left[R\left(s_t, a_t\right)\right] \tag{2.10}$$

$\mathbb{E}$ denotes the expectation, $R_\tau = \sum_{t=0}^{T} \gamma^t R_t$ is the finite-horizon discounted return, with $\gamma \in [0, 1)$, $R(\cdot, \cdot)$ is the reward function, and actions come from the stochastic policy $\pi(a_t \mid s_t)$. An agent faces the dilemma of either exploring new decisions that might lead to higher rewards or exploiting decisions expected to return high rewards. To make decisions, an agent needs to be able to assess expected return (value) of a state or state-action pair. The value function

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R_\tau \mid s_0 = s] \tag{2.11}$$

measures the expected return of a trajectory τ starting in some state s, which is generated with policy π. The action-value Function

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R_\tau \mid s_0 = s, a_0 = a], \tag{2.12}$$

serves as an alternative. It measures the expected return of a trajectory τ generated with policy π, starting in some state s and taking action a from there.

2.2.2 Policy Gradient Methods

Policy-based methods [SB18] are often used to maximize the expected returns by training a parameterized stochastic policy π_θ. For policy gradient methods, we define the objective as

$$J_\theta = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[R_\tau] \tag{2.13}$$

We seek to find the optimal parameters θ^* that maximizes the objective, i.e.,

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\ J_\theta \tag{2.14}$$

Policy gradient methods use different methods to find the θ^*. These methods optimize the parameters θ either indirectly, by maximizing local estimates of J_{π_θ}, like the PPO algorithm, or directly by gradient ascent on the objective J_{π_θ}, like the REINFORCE algorithm [Wil92], e.g.:

$$\theta_{j+1} = \theta_j + \alpha \nabla_\theta J_{\pi_\theta} \tag{2.15}$$

where θ_{j+1} are the updated parameters, α is the step size, and $\nabla_\theta J_{\pi_\theta}$ is the policy gradient.

We seek an analytical expression for J_{π_θ}, as we need to calculate the policy gradient to do an update step. With the probability of a trajectory $\tau = (s_0, a_0, ..., a_{T-1}, s_T)$ given that actions come from π_θ as

$$p(\tau \mid \theta) = \rho_0(s_0) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \tag{2.16}$$

and Eq. 2.13, we can reformulate the policy gradient:

$$\begin{aligned}
\nabla_\theta J_{\pi_\theta} &= \nabla_\theta \underset{\tau \sim \pi_\theta}{\mathrm{E}} [R_\tau] \\
&= \nabla_\theta \int_\tau p(\tau \mid \theta) R_\tau \\
&= \int_\tau \nabla_\theta p(\tau \mid \theta) R_\tau \\
&= \int_\tau p(\tau \mid \theta) \nabla_\theta \log p(\tau \mid \theta) R_\tau \\
&= \underset{\tau \sim \pi_\theta}{\mathbb{E}} [\nabla_\theta \log p(\tau \mid \theta) R_\tau] \\
&= \underset{\tau \sim \pi_\theta}{\mathbb{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R_\tau \right].
\end{aligned} \tag{2.17}$$

This analytical expression for the policy gradient with an expectation can now be estimated with a sample mean. An agent that interacts with the environment for N episodes, using the policy π_θ, generates a set of trajectories $\mathcal{D} = \{\tau_1, \tau_2, ..., \tau_N\}$. Given $\mathcal{D}$, we can transform the policy gradient to a computable expression; we can estimate the policy gradient with

$$\nabla_\theta J_{\pi_\theta} \approx \frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R_\tau \tag{2.18}$$

Alternative versions of Eq. 2.18 exist that replace R_τ with a cumulative reward

$$R_{\gamma,t} = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}. \tag{2.19}$$

With the cumulative reward, we can transform the policy gradient from Eq. 2.17 to

$$\begin{aligned} \nabla_\theta J_{\pi_\theta} &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}\right] \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_{\gamma,t}\right]. \end{aligned} \tag{2.20}$$

The cumulative reward does not change the expected value but reduces the variance in the policy update. The same holds when subtracting a *baseline* function from the equation above, that further reduces the variance. A common choice for the *baseline* is the value function $V^\pi(s_t)$, which can be approximated, e.g., with a neural network, $V_\phi(s_t)$. In addition, we note that in an MDP, the future only depends on the most current state and action due to the Markov property. This allows to replace the return in above equation with $Q^\pi(s_t, a_t)$. With these changes Eq. 2.20 is transformed to:

$$\nabla_\theta J_{\pi_\theta} = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q^{\pi_\theta}(s_t, a_t) - V_\phi(s_t)\right)\right] \tag{2.21}$$

$$= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right] \tag{2.22}$$

where $A^{\pi_\theta}(s_t, a_t)$ is the advantage function [SML+15].

A problem that might arise from this vanilla policy gradient method is that extensive gradient updates might change π_θ so that the agent performance collapses. Schulman et al. [SLA+15] aim to overcome this problem and propose the Trust-Region Policy Optimization (TRPO) algorithm. Shortly afterwards the PPO [SWD+17] algorithm is introduced. Both algorithms prohibit large deviations of π_θ from the previous policy, $\pi_{\theta_{old}}$, and aim to do the parameter optimization inside a relatively small *Trust Region*. However, PPO offers a more straightforward implementation. Here the *Trust Region* is controlled by probability ratio

$$\sigma_{t,\theta} = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}. \tag{2.23}$$

For similar policies $\sigma_{t,\theta}$ is close to 1.
Policy optimization in PPO is performed *on-policy* [SB18], which means that each policy update uses data generated by the most recent version of the policy. PPO

executes multiple epochs of Stochastic Gradient Ascent (SGA) to perform each policy update.

As stated at the beginning of this section, PPO updates the policy indirectly. The main part for optimization is a clipped surrogate objective function, $\mathcal{L}^{\text{CLIP}}(\theta)$, which is defined as

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathop{\mathbb{E}}_{t}\left[\min\left(\sigma_{t,\theta} A^{\pi_{\theta_{old}}}(s_t, a_t), \operatorname{clip}\left(\sigma_{t,\theta}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_{old}}}(s_t, a_t)\right)\right]. \tag{2.24}$$

Here ϵ is a parameter that determines how far away the new policy deviates from the old, as it discourages for $\sigma_{t,\theta}$ from going outside the interval $[1-\epsilon, 1+\epsilon]$. Taking the minimum in this expression forces the objective to take a conservative lower bound on the unclipped objective.
It is possible to approximate the value function with a neural network. This is done by minimizing the mean-squared-error $\mathcal{L}_t^{\text{VF}}(\theta)$ between value function estimate and the observed cumulative rewards, $\mathcal{L}_t^{\text{VF}}(\theta) = (V_\theta(s_t) - R_\tau)^2$.
However, when using a neural network architecture that shares parameters between the policy and value function, the objective function also needs to account for the error from the value function estimate. Schulman et al. [SWD+17] propose such a combined objective that they maximize:

$$\mathcal{L}_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t\left[\mathcal{L}_t^{\text{CLIP}}(\theta) - c_1 \mathcal{L}_t^{\text{VF}}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right], \tag{2.25}$$

where c_1 and c_2 are coefficients and $S[\pi_\theta](s_t)$ denotes the entropy of π_θ. The entropy term encourages exploration, and c_2 can be annealed throughout training to reduce exploration.

The PPO algorithm with $\mathcal{L}_t^{\text{CLIP+VF+S}}(\theta)$ can be decomposed into two main parts: i) parallel data collection and ii) optimization. For each iteration, N agents can generate data in parallel. They each interact for T time steps with their environment. Afterward, we calculate the surrogate loss $\mathcal{L}$ on these $N \cdot T$ time steps of data, and optimize the policy parameters with minibatch SGA, for F epochs. A version of the PPO-Clip algorithm with a shared neuronal network is summarized in Algo. 1.

2.2.3 Policy Search in AlphaGardenSim

In previous work [AGW+20], the *analytic policy* could not handle plants with significant differences in germination times and growth rates. Invasive plant types serve as an example as they grow pretty quickly. They tend to dominate the garden which causes a low diversity v_t. The *analytic policy* can irrigate and prune plants. We describe the *analytic policy* in Sec. 2.2.4, but essentially it is a deterministic policy that acts based on threshold values. These threshold values are only tuned for a small range of environment configurations, causing the policy to fail in challenging conditions. Real-world dynamics often differ from simulated environment dynamics. Therefore, it is uncertain whether the *analytic policy* will make robust decisions in a real-world environment.

To enable transfer of policies from simulation to reality for robotic control, Peng et al. [PAZA18] train deep reinforcement learning agents in simulation and vary the environment dynamics during training. They parameterize the dynamics of the simulation $\hat{p}\left(s_{t+1} \mid s_t, a_t, \mu_d\right)$, where μ_d is a set of dynamics parameters.By training on a range of dynamics, they can generalize to realistic dynamics. For domain randomization, they modify the objective to maximize the expected return over a distribution of environment dynamics:

$$\underset{\mu_d \sim \rho_\mu}{\mathbb{E}}\left[\underset{\tau \sim p(\tau|\pi,\mu_d)}{\mathbb{E}}\left[R_\tau\right]\right], \tag{2.26}$$

where μ_d is the set of parameters that parameterize the environment dynamics, ρ_μ is the distribution of dynamics, and $p(\tau|\pi, \mu_d)$ is the likelihood of a trajectory $\tau = (s_0, a_0, s_1, ..., a_{T-1}, s_T)$ given a certain policy π and parameters μ_d. We utilize this technique by extending the PPO-Clip algorithm with dynamics randomization. We summarize this modified training procedure in Algo. 1 and visualize in Fig. 2.3 how an agent interacts with multiple randomized environments.

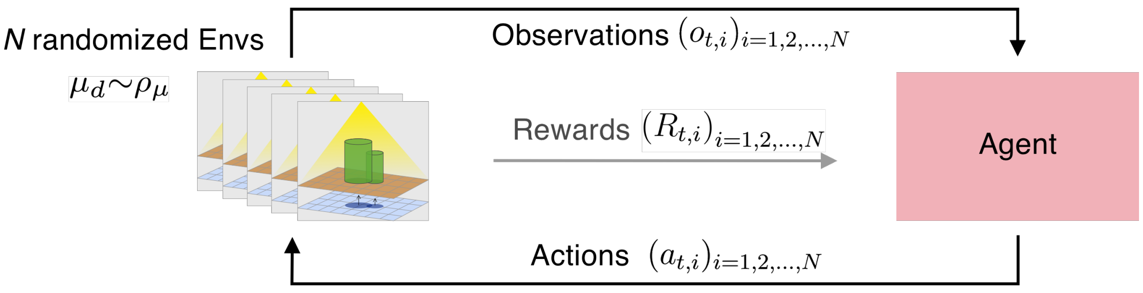

N randomized Envs
$\mu_d \sim \rho_\mu$
Observations $(o_{t,i})_{i=1,2,...,N}$
Rewards $(R_{t,i})_{i=1,2,...,N}$
Agent
Actions $(a_{t,i})_{i=1,2,...,N}$

Figure 2.3: Interaction of agent with a set of N AlphaGardenSim environments, where each environment i as randomized simulation parameters μ_d.

Partial Observability In Section 1.2, we note that in a realistic setup for AlphaGardenSim, the agent does not have access to all state information. Instead, the agent observes present state variables while parts of the underlying system state remain unknown. For example, when plants grow luxuriantly, they occlude the underlying environment. A stationary overhead observer - or even from other perspectives - cannot visually monitor the whole state. Therefore, we frame the general problem as a Partially Observable Markov Decision Process (POMDP). Formally a POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$. The state space, action space, transition dynamics, and reward function remain the same as for the MDP. However, the true system state is no longer exposed to the agent. Instead, the agent

Algorithm 1 PPO-Clip [SWD$^+$17] with random dynamics.

```
 1: Input: initial parameters $\theta_0$ for shared network
 2: for iteration j = 0, 1, 2,... do
 3:     Collect set of trajectories $\mathcal{D}_j = \{\tau_i\}_{i=1,...,N}$:
 4:     for actor = 1,2,...,N do
 5:         $\mu_d \sim \rho_\mu$ sample dynamics
 6:         Run policy $\pi_{\theta_j}$ in environment with dynamics $\mu_d$ for $T$ time steps
 7:         Compute advantage estimates $A^{\pi_{\theta_j}}(s_0, a_0), ...A^{\pi_{\theta_j}}(s_T, a_T)$
 8:     end for
 9:     Update the policy by maximizing the surrogate $\mathcal{L}$ wrt $\theta$, with F epochs and
          minibatch size $M \leq N$ via SGA:
              $$\theta_{j+1} = \arg\max_\theta \frac{1}{M} \sum_{\tau \in \mathcal{D}_M \subseteq \mathcal{D}_j} \mathcal{L}_t^{\text{CLIP+VF+S}}(\theta)$$
10: end for
```

receives an observation $o_t \in \Omega$ at each time step. This observation is generated from the underlying system state $o_t \sim \mathcal{O}(s_t)$, where the probability distribution for the next observation $p(o_{t+1} \mid s_{t+1}, a_t)$ depends on the previous action a_t and the next state s_{t+1}. We don't access the true state to model realistic behavior, rather treat the simulator as a blackbox.

Network Architecture and System The original PPO algorithm has no explicit mechanisms to extract the underlying state of the POMDP. PPO is only effective if the observations are representative of the underlying system states. Hausknecht et al. [HS15] use a recurrent agent [MBM$^+$16] as an alternative to stacking a *history* as input to overcome partial observability. An LSTM is a recurrent neural network [HS97] capable of learning longer-term dependencies between the inputs. It serves as a memory layer for the agent to remember a sequence of information. Hodge et al. [HHA21] extend PPO's policy network with an LSTM layer to apply the agent to a partially observable environment. The previous approaches rely on automatic differentiation software to train these recurrent policy networks.
We use a PPO agent implemented with the RLlib library[LLN$^+$17] and TensorFlow [AAB$^+$16] in the Python language for our work. RLlib can vectorize the simulation environments and enable high performant, scalable training.
To study the effect of partial observability in AlphaGardenSim, we use two PPO agents with different network architectures. Both versions use a fully-connected two-layer perceptron with 128 nodes per layer, with a tanh activation function. For one network, the resulting activations are processed through time by an additional LSTM layer with 64 cells with a sequence length of 20. This output is then passed through another hidden layer with a linear activation function. In both cases, the network layers are shared, and two heads are used to output the value function

approximation and the action logits for the policy. We refer to these agents as PPO$_{\text{full}}$ and PPO$_{\text{lstm}}$. To apply these PPO agents to AlphaGardenSim we specify the POMDP components in the following paragraphs.

Action Space At each time step t, two types of actions can be executed, namely watering and pruning, for each plant i in the garden space.

Watering $a_{\mathrm{w},i,t}$ is a binary decision. Either no water or an irrigation circle with a fixed amount of water, $a_{\text{water}} = 0.0002\text{m}^3$, is applied at the plant center. This process follows the irrigation model described in Section 2.1.2. The water action $\mathbf{a}_{\mathrm{w},t}$ consists of the individual watering decisions for each plant i:

$$\mathbf{a}_{\mathrm{w},t} = [a_{\mathrm{w},1,t}, ..., a_{\mathrm{w},I,t}]^\intercal \tag{2.27}$$

Additionally, the agent decides at each time step t to prune or not. Pruning, denoted as $a_{\mathrm{p},i,t}$, reduces the radius of a plant i by a defined pruning level. Within the scope of this work, the pruning level is set to $\mathcal{P}_{\mathrm{d}} \in \{0\%, 15\%\}$. The prune action $\mathbf{a}_{\mathrm{p},t}$ consists of the individual pruning decisions for each plant i:

$$\mathbf{a}_{\mathrm{p},t} = [a_{\mathrm{p},1,t}, ..., a_{\mathrm{p},I,t}]^\intercal \tag{2.28}$$

State Space We describe the simulator quantities that influence the state $\mathbf{s}_t$ of the environment in Section 2.1.1. These include the health level $h_{x,y,t}$, growth stage $G_{i,t}$, and structural information from each plant i, i.e., their seed locations $(u_i)_{i=1,...,I}$, radii $(r_{t,i})_{i=1,...,I}$ and heights $(h_{t,i})_{i=1,...,I}$. Furthermore, the state includes information on the soil water content $w_{x,y,t}$ for each cell $o_{x,y}$ and the current time step t from the environment.

Observations ($\mathcal{O}$). The observation space consists of two vectors. The first one is the global garden population $\mathbf{z}_t$, at time step t, with elements $z_{k,t}$ as a distribution over the K plant types. The second vector holds soil water readings $\mathbf{w}_{\text{sensor},t}$, with elements $w_{\text{sensor},i,t}$ from each plant i at time step t. Each sensor reading represents the mean VWC, for a 0.09m $\times$ 0.09m window centered around each plants seed location.

Transitions ($\mathcal{T}$). At each time step t, AlphaGardenSim executes a sequence of updates across the garden: irrigation, lighting, water use, and plant growth according to the models described in Section 2.1.2.

Reward Function The objective is to achieve a diverse garden with maximal canopy coverage while minimizing water usage. Avigal et al. [AGW$^+$20] formalize these metrics. They introduces the global garden population vector $\mathbf{z}_t$, at time step t, with elements $z_{k,t}$ as a distribution over the K plant types. This vector defines the global canopy coverage c_t at time step t as the total coverage of plants over the garden area,

$$c_t = \frac{\sum_{k=1}^{K} z_{k,t}}{X \cdot Y} \tag{2.29}$$

and the garden diversity v_t at time t is defined as the normalized entropy H of the global garden population:

$$v_t = \frac{H(\mathbf{z}_t)}{\log K} = \frac{-\sum_{k=1}^{K} z_{k,t} \log z_{k,t}}{\log K} \tag{2.30}$$

The diversity is maximized ($v_t = 1$) for a uniform global population vector $\mathbf{z}_t$. The entropy is minimized ($v_t = 0$) in an unbalanced garden, with one predominant plant type.

In our research [AWP+22], we propose an advanced metric to access both coverage and diversity, called Multi-Modal Entropy (MME). In this work, we define $\tilde{K}$ as the total of the K plant types plus an additional type for soil. With $\tilde{K}$ the global garden population vector $\tilde{\mathbf{z}}_t$ is updated to account the soil coverage. The MME is defined as:

$$mme_t = \frac{H\left(\tilde{\mathbf{z}}_t\right)}{\log \tilde{K}} = \frac{-\sum_{k=1}^{\tilde{K}} \tilde{z}_{k,t} \log \tilde{z}_{k,t}}{\log \tilde{K}} \tag{2.31}$$

Finally, to model the reward for this problem, we combine the mme_t metric with the term that accounts for the irrigation amount. Therefore, AlphaGardenSim emits at time step t a reward $R_t \in [0, 1]$:

$$R_t = \max\left(mme_t - \frac{\beta}{I \cdot a_{\text{water}}} \sum_{i=1}^{I} a_{\text{w},i,t}, 0 \right) \tag{2.32}$$

The motivation for this reward function is as follows. The agent's primary goal is to achieve a high MME. We add the coefficient $\beta \in (0, 1)$ to control the negative influence of irrigation.

2.2.4 Baseline

Avigal et al. [AGW+20], introduce among other policies, an analytic automation policy, *analytic policy*, with hand-tuned parameters. The *analytic policy* has shown the highest performance. Therefore, we use the policy as a baseline to compare it with our agent. The policy observes the local canopy coverage, plant health levels, and soil moisture contents within a masked 0.30m $\times$ 0.15m sector around each plant center within the garden. Additionally, the policy observes the global population $\mathbf{z}_t$. It applies one of four actions at each plant: prune, irrigate, prune and irrigate, or null. The policy waits 20 days before first pruning to allow plants to initiate growth.

Then the policy decides to prune each plant i that contributes to surpassing a higher than uniform threshold in the global population. Before taking an irrigation action, the policy first creates a 0.09m$\times$0.09m window centered around a target plant. The policy irrigates should any plants be underwatered inside the window. Otherwise, the policy irrigates should the total amount of soil moisture inside the sector be less than half of the overwater threshold.

Chapter 3

Evaluation and Discussion

We reimplement the simulator to account for the adaptations described in the previous chapter. Along with these adaptations, we aim to increase simulation throughput. In the first part of this chapter, we evaluate generated growth profiles of the adapted version. We then use the simulator to search for control policies. We aspire that our PPO-based reinforcement learning agents outperform the *analytic policy* for polyculture plant treatments within AlphaGardenSim. Additionally, we aim to show that a PPO-based agent can efficiently control garden configurations with altered dynamics, on which it has not been trained. We assume that the performance in these new garden configurations serves as a proxy to assess transferability to reality.

3.1 Adapted Simulator

In this section, we analyze and validate our implementation of the AlphaGardenSim. We compare the simulator throughput of the previous and current versions. We compare how long the simulation needs to grow a polyculture garden with the analytic policy for our evaluation. The environment consists of 60 plants that are treated for 100 days in a 1.5m $\times$ 1.5m garden space. As stated in Section 2.1, the original AlphaGardenSim can approximate plant growth at 9000 times the speed of natural growth. The adapted version simulates this garden 454000 times faster than the real world. To assess the validity of this result, we analyze the simulation output of the adapted simulator in the following section.

3.1.1 Growth Analysis

In our previous work [ADW$^+$21], we evaluate the radial growth of 120 plants, twelve plants per type, planted on 16-Nov.-2020 inside a garden bed at a greenhouse of UC Berkeley for 46 days. The average radial growth per plant type k at each time step t is used to search for plausible growth parameters.

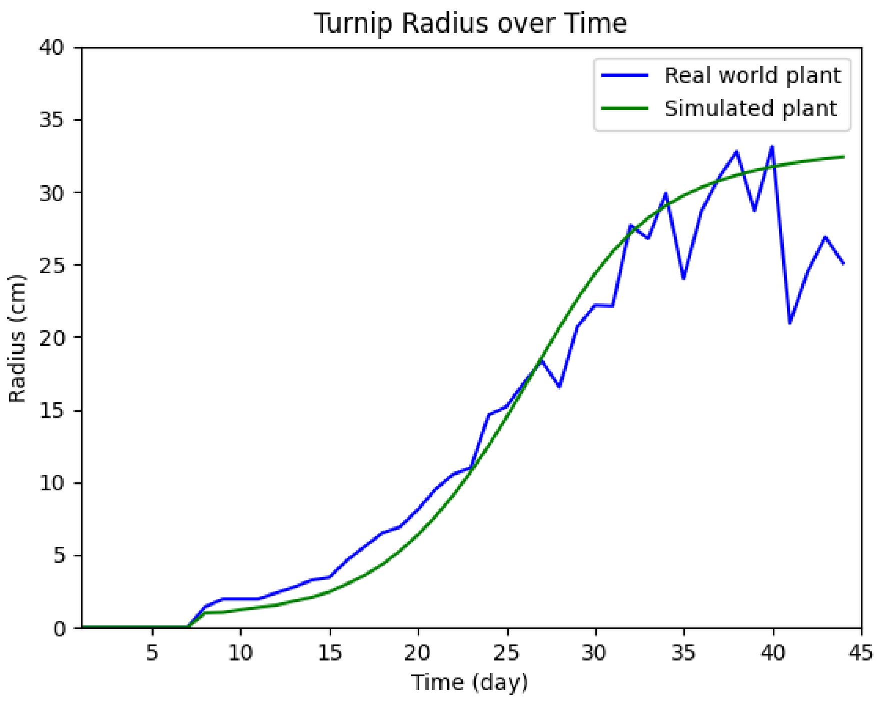
	t_g		$r_{\max}$		e_{water}		$e(35)$	
Plant Type	old	new	old	new	old	new	old	new
Borage	7	4	60	34	0.09	0.16	6.61	**1.11**
Kale	7	6	65	42	0.10	0.16	5.41	**0.87**
Swiss Chard	7	4	47	33	0.11	0.17	9.93	**0.95**
Turnip	7	5	53	35	0.11	0.19	10.04	**1.24**
Green Lettuce	9	8	27	25	0.08	0.14	7.46	**0.69**
Cilantro	10	8	20	20	0.09	0.13	10.76	**4.49**
Red Lettuce	12	20	10	28	0.09	0.14	11.61	**0.25**
Radicchio	9	7	53	22	0.09	0.14	9.28	**4.31**

Table 3.1: Growth analysis of original and adapted simulations, where $t_{\mathrm{g,old}}$ (days) is germination time of the old simulator, $t_{\mathrm{g,new}}$ (days) is tuned germination time for the new simulator, $r_{\mathrm{max,old}}$ is original growth potential, $r_{\mathrm{max,new}}$ is the growth potential for the updated simulator, e_{water} is the water use efficiency parameter, $e(35)$ (cm) is the mean absolute error on day 35 between simulated and average real world radius. Original values were taken from published growth analysis [ADW$^+$21].

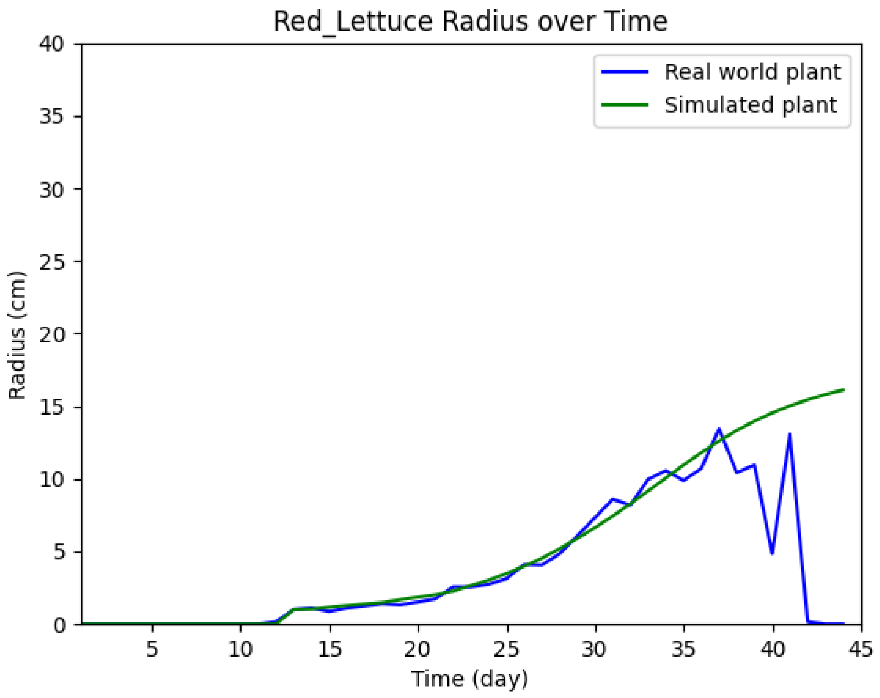
For the scope of this thesis, we use data from eight plant types to identify parameters for our adapted simulator. We conduct a grid search to minimize the mean absolute error between the simulated and observed plant radii on day 35. After day 35, parts of the plants start to occlude each other, and precise data is not available anymore. As part of the grid search, we vary the germination time, water use efficiency, and maximal radius parameters to influence the growth profiles. For our experiments, the irrigation amount is fixed to 0.0002m^3. The water demand of each plant is met by irrigating daily. During the parameter search pruning actions are deactivated and stochastic plant parameters are set to deterministic values. The simulator configuration is listed in Appendix A.1.

We plot the growth curves, radius (cm) over time (days), for the simulated and real-world plants to validate the adapted simulator. Figure 3.1 shows the growth curves for two plant types. The remaining plots for the other six plant types are depicted in Figure A.1. The blue curve shows the real-world radius. The red logistic curve is a simulated radius from AlphaGardenSim. All figures show that the logistic growth profiles of the simulated plants match up with the observed data for the specified time horizon. We observe that the measurements for germination coincide.

Table 3.1 compares parameters settings for the old and new simulators. These listed parameters are the germination times, maximal radii, and water use efficiencies. Furthermore, the table shows the mean absolute error on day 35 between the simulated and the real-world plant radii. For all plants, the mean absolute error is lower for the adapted simulator than the original one.



(a) Growth analysis of turnip.



(b) Growth analysis of red lettuce.

Figure 3.1: Radial growth profile for two plant types. Green curve shows the simulated plant with the re-implemented simulator. Blue curve shows the real world mean radii from our previous work [ADW+21].

3.1.2 Discussion

Our reimplemented simulator has a higher throughput compared to the original work. Due to the improved performance of factor 50, the adapted simulator can conduct experiments in larger problem spaces when given the same amount of computation resources as the original simulator. The improved simulation time is beneficial for optimization problems like reinforcement learning. While we do not have new data to tune other simulator parameters, the restricted growth analysis shows that growth curves for unoccluded growth stages match the logistic biological model. The analysis shows that the modeling capability of the reimplemented simulator is on par with the original AlphaGardenSim.

Nevertheless, the simulator remains limited by the expressiveness of the implemented models. Additional data and more extensive experiment trials are needed to assess the simulator for different growth stages or environmental conditions. For example, to determine the maximal radial growth potential for plants that grow in competition and are subject to occlusion. Therefore the real-world validity of the simulation output remains limited. However, it is a common approach in botany and agriculture to use these approximations. Hence, we assume that the simulator can generate realistic enough data to learn treatment policies.

3.2 PPO Policy Search

In this section, we use the adapted simulator to search for treatment policies. We aim to find a policy that can outperform the *analytic policy* in a set of environments. Additionally, we examine how the agents perform in environment configurations that they have not been trained on. To validate our research hypothesis, we train and evaluate the PPO-based policies on different garden configurations. For these experiments, we generate versatile environments with randomization dynamics. We control this process through a subset of the plant parameters modeled as random variables. Plant parameters are sampled according to the distributions listed in Table A.1. We fix the random seeds for Numpy's random number generator to reproduce the simulation outputs for all policies within an experiment. For validation experiments the generator is reseeded with new values. With the help of the RLlib package, we concurrently collect experience from each randomized garden environment. Our simulation experiments are split into two general garden configurations, *A* and *B*.

In all experiments, plants germinate at fixed locations $(u_i)_{i=1,\dots,I}$. As in previous research [AGW+20, PAT+21, AWP+22], all simulation episodes are grown for a 100-day horizon. Within this period, the entire plant life cycle is simulated. We configure the simulator according to the models and settings we state in Section 2.1. Furthermore, we set the overwatering threshold to $T_o = 10$. Hence, plants are overwatered should they get more than ten times the water amount they desire. The underwatering threshold is set to $T_u = 0.1$. We randomize each cell's initial VWC (%) in the water grid. Instances are sampled from a normal distribution with mean of 0.2 and a standard deviation of 0.04. The PWP and max VWC are set to 0% and 30%, respectively. A fixed value for evaporation (%) is sampled for each episode. Evaporation is modeled as a normal distribution with a mean of 0.04 and a standard deviation of 0.0054.

Garden Environment Configuration *A* The first set of experiments run in 1m $\times$ 1m garden spaces, each with $I = 3$ plants, as depicted in Figure 3.2. The policies are trained on three plant types, i.e., green lettuce, red lettuce, and an invasive model plant. On average, this invasive model plant germinates earlier and occupies a larger space than other plant types used in this project. Invasive model plants have the potential to inflict substantial occlusion. The other two plants are modeled with the tuned growth parameters described in Section 3.1.1. After training an agent, it is validated in two setups. One setup consists of the same plant types. The other validation is with an altered garden environment. To create a new garden environment for the agent, we replace the invasive plant type with turnip. The policy has not been trained on such a plant. We adapt the truncated normal distributions for germination time and radial growth potential such that they are out of range from the parameter interval that was used for training. We want to evaluate whether the policy generalizes to unseen garden setups with this garden configuration.

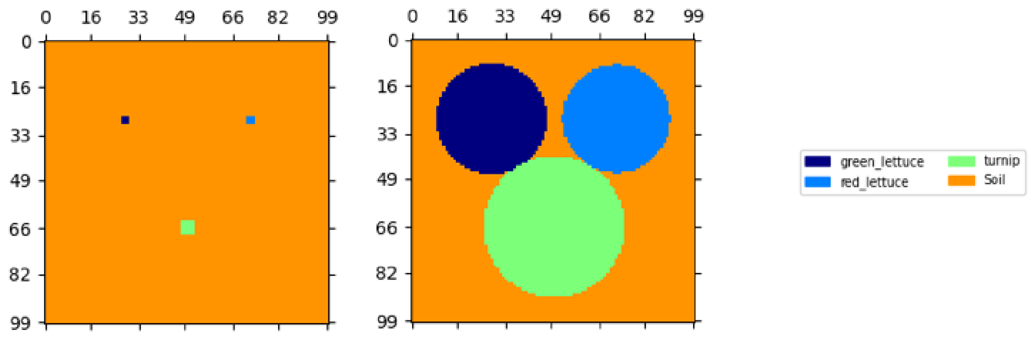


Figure 3.2: Canopy view of the plant setup for experiment configuration A. Green lettuce seed coordinate is (28,28), red lettuce is located at (72,28) and turnip or invasive grow at (50, 66). **Left**: Canopy view shortly after germination. **Right**: Canopy coverage without occlusion.

Garden Environment Configuration *B* In our second set of experiments, we scale the problem to 1.5m $\times$ 1.5m garden spaces, each with $I = 16$ plants, as depicted in Figure 3.3. The training and evaluation procedure does not differ from the smaller setup. Two invasive model plants and two of each plant types listed in Table 3.1 are used in this experiment configuration. The agents are trained on all plant types, excluding turnip. The search space of this problem setup is considerably larger. Furthermore, these many plants in a limited space generate garden configurations with a dense canopy cover. Here, vital information about the underlying plants might not be directly observable. Hence, an agent is likely to act based on partial state information in many environment steps.

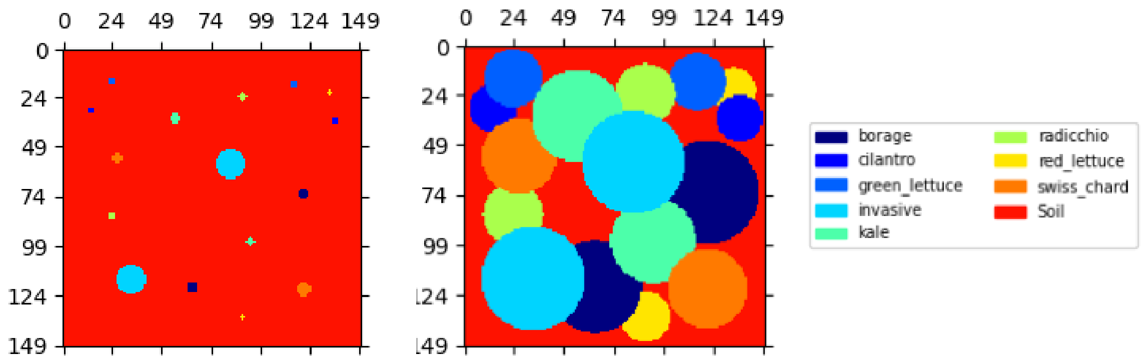


Figure 3.3: Canopy view of the plant setup for experiment configuration B. The present plant types and seed coordinates are: borage at (65, 120), (121, 73), cilantro at (137 36), (14 31), radicchio at (90 24), (24 84), kale at (56 35), (94 97), green lettuce at (24, 16), (116, 18), red lettuce at (90 135), (134 22), swiss chard at (27 55), (121 121) and invasive or turnip at (84 58), (34 116). **Left**: Canopy view shortly after germination. **Right**: Canopy view of dense garden with partial occlusion.

Agent Configuration We evaluate the learning capacity of the PPO_{full} and PPO_{lstm} agents described in Section 2.2. For experiment setup A, the agents are trained for 240000 time steps. The episodes are generated from 2400 environment configurations. As the search space for the sixteen plant setup is much larger, we train our agents for over 5 million time steps in experiment setup B. The episodes are generated from 50064 environment configurations. We manually tune the hyperparameters of the PPO algorithm to increase the learning capacity of the agents. The hyperparameters of the PPO algorithm used for training are listed in Table A.2. Multiple instances of an agent (actors) concurrently collect experience within a training batch. We adapt the number of actors and the batch size for training, considering the search space for both environment configurations. All agents trained in experiment setup A are allocated one virtual core per actor from a 2.8 GHz Quad-Core Intel Core i7 CPU with 16 GB RAM. Furthermore, allocate one virtual core per actor for experiment setup B from a 2.2GHz Intel Xeon CPU E5-2698 v4 with 256 GB RAM. An Nvidia Tesla V100 GPU with 32GB RAM is used only in setup B to train an agent's neural network parameters.

3.2.1 Performance Analysis

For our experiments we use our reward function (Eq. 2.32). We set $\beta = 0.2$, to reduce the negative impact of watering. During training we analyze the agent performance with the average episode reward across the batch collected at each time step. For environment configuration A, each batch is generated from twelve unique environment configurations. For environment configuration B, the data is generated by 82 environments.

Figure 3.4 shows the training performance of the two agents against the baseline in environment configuration A. Solid lines are the average episode returns of the agents across the batches collected at each epoch and their 99%-confidence band (shaded area). The *analytic policy* is shown as a green dashed line. The figure indicates that both agents perform better than the baseline policy. However, our PPO_{lstm} agent converges with fewer training iterations and reaches the highest performance.

Figure 3.5 shows the training performance of the PPO_{full} and PPO_{lstm} agents in environment configuration B. Solid lines represent the average episode returns across collected batches. The shaded areas depict the accompanying 99%-confidence band for the reward observed in the environments the agent acts in parallel. The *analytic policy* is shown as a green dashed line. In this configuration, only the PPO_{lstm} agent is able to outperform the baseline. After about 1.8 million environment interactions it surpasses the baseline policy. The PPO_{full} shows worse performance than the baseline.

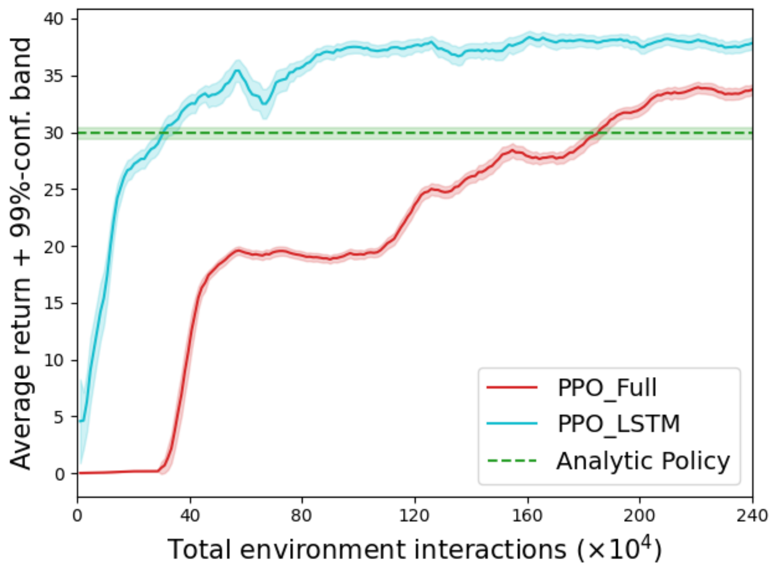


Figure 3.4: Training performance of policies in Environment Configuration A.

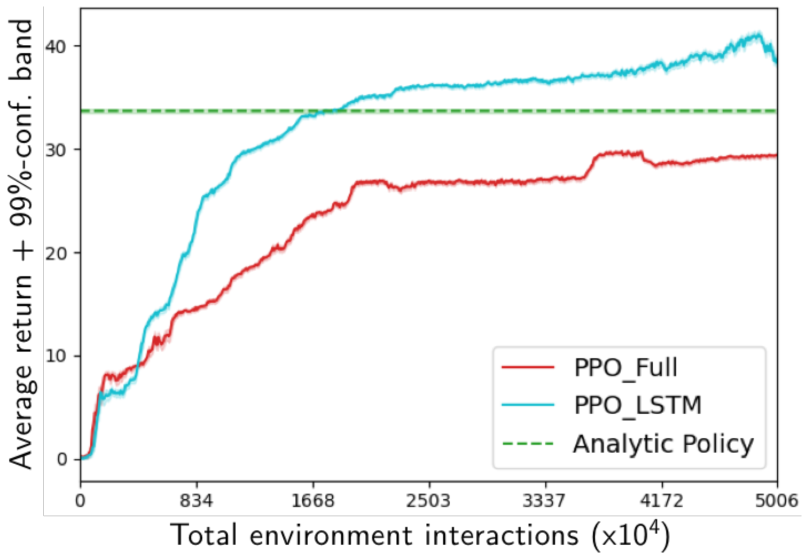


Figure 3.5: Training performance of policies in Environment Configuration B.

Policy Validation After the training procedure, we validate the agents with a set of metrics. Policy performance is validated over days 20 to 70, with a growing period $T = 50$, using the same metrics from our previous work [AWP+22]:

1. Average total plant coverage - We average the total canopy coverage c_t for the growing period per experiment:
$$M_{\mathrm{C}} = \frac{\sum_t c_t}{T} \tag{3.1}$$
2. Average diversity - We average the garden diversity v_t at time t for the growing period per experiment:
$$M_{\mathrm{D}} = \frac{\sum_t v_t}{T} \tag{3.2}$$
3. Water usage (m^3) - We sum the water used in a single experiment over the entire growing period (100 days):
$$M_{\mathrm{W}} = \sum_t -\mathbf{a}_{\mathrm{w},t} \tag{3.3}$$

Before and after the growing period, the diversity measurements do not accurately reflect the policy's performance. This is due to the small plant sizes at these time steps. Therefore, we exclude these data points from the M_{C} and M_{D} metrics.

In the first set of validation experiments, we analyze the performance of the trained agents in the environment they have been trained. Table 3.2 gives an overview on the experiment outcome. The $\mathrm{PPO}_{\mathrm{lstm}}$ agent revives the most reward for both configurations. However, no agent has an outstanding overall performance for the other metrics. Both PPO-based agents use less water than the baseline for irrigation. The *analytic policy* grows gardens with a higher diversity than our agents. In experiment configuration A the reinforcement-learning-based agents grow gardens with a significantly higher canopy coverage.

We replace all invasive plant types with turnip plants for the second part of the validation experiments. Table 3.3 gives an overview of the agents' performance in this altered environment. For experiment configuration A, no agent receives significantly more rewards than the others. Our $\mathrm{PPO}_{\mathrm{full}}$ agent grows gardens with a high canopy coverage. Both the PPO-based agents have lower water consumption than the *analytic policy.* For experiment configuration B, do not beat the baseline. The *analytic policy* achieves a higher reward than our proposed methods. Furthermore, the baseline has a higher canopy coverage and diversity.

	Experiment configuration A			Experiment configuration B		
Metrics	$\text{PPO}_{\text{full}} \pm \sigma$	$\text{PPO}_{\text{lstm}} \pm \sigma$	Baseline $\pm\sigma$	$\text{PPO}_{\text{full}} \pm \sigma$	$\text{PPO}_{\text{lstm}} \pm \sigma$	Baseline $\pm\sigma$
Total reward	34.15 ± 1.98	$\mathbf{38.54 \pm 1.62}$	31.44 ± 1.92	30.08 ± 0.69	$\mathbf{39.06 \pm 1.46}$	35.28 ± 0.72
M_{C}	0.39 ± 0.03	$\mathbf{0.41 \pm 0.04}$	0.28 ± 0.02	$\mathbf{0.61 \pm 0.02}$	$\mathbf{0.60 \pm 0.02}$	$\mathbf{0.60 \pm 0.02}$
M_{D}	0.55 ± 0.05	0.69 ± 0.05	$\mathbf{0.86 \pm 0.02}$	0.70 ± 0.01	0.75 ± 0.01	$\mathbf{0.90 \pm 0.01}$
M_{W} (m^3)	$\mathbf{-0.05 \pm 0.00}$	-0.06 ± 0.00	-0.06 ± 0.00	-0.27 ± 0.00	$\mathbf{-0.07 \pm 0.0}$	-0.32 ± 0.00

Table 3.2: Results of validation experiments with original plant configurations.

	Experiment configuration A			Experiment configuration B		
Metrics	$\text{PPO}_{\text{full}} \pm \sigma$	$\text{PPO}_{\text{lstm}} \pm \sigma$	Baseline $\pm\sigma$	$\text{PPO}_{\text{full}} \pm \sigma$	$\text{PPO}_{\text{lstm}} \pm \sigma$	Baseline $\pm\sigma$
Total Reward	$\mathbf{38.03 \pm 2.16}$	$\mathbf{38.57 \pm 1.74}$	$\mathbf{36.44 \pm 2.40}$	29.17 ± 0.8	28.5 ± 1.81	$\mathbf{35.83 \pm 0.66}$
M_{C}	$\mathbf{0.40 \pm 0.03}$	0.33 ± 0.02	0.30 ± 0.03	$\mathbf{0.59 \pm 0.02}$	0.54 ± 0.03	$\mathbf{0.59 \pm 0.01}$
M_{D}	0.54 ± 0.07	0.77 ± 0.05	$\mathbf{0.93 \pm 0.02}$	0.71 ± 0.02	0.71 ± 0.03	$\mathbf{0.94 \pm 0.00}$
M_{W} (m^3)	$\mathbf{-0.05 \pm 0.00}$	$\mathbf{-0.05 \pm 0.00}$	-0.06 ± 0.00	-0.26 ± 0.0	$\mathbf{-0.10 \pm 0.02}$	-0.32 ± 0.00

Table 3.3: Results of validation experiments with new plant type.

3.2.2 Discussion

Training performance Our proposed reinforcement-learning-based agents all learn to care for plants in a randomized polyculture environment. The PPO_{lstm} agent shows state-of-the-art performance, beating the analytic policy in both training environments. The LSTM memory layer improves the learning capacity of the PPO algorithm for this domain. The PPO_{full} performance falls short of the analytic policy for the sixteen plant environment but is able to outperform the baseline in the three plant setup.

We observe in Figure 3.4 that the average reward of PPO_{full} is below 20 for the first 100000 training iterations. The PPO_{full} agent lets the invasive plant grows larger than the baseline and PPO_{lstm} agent. The causes invasive plants occlude other plants in the environment. This has a negative effect on diversity and therefore on the overall performance. The invasive plant has the characteristic to germinate earlier than the other present plant types. The PPO_{lstm} agent learns to time the pruning actions in a way that the invasive plants is allowed to grow just large enough to not hinder the other plants during their development. The memory layer of the PPO_{lstm} agent helps to understand the temporal correlations within the environment. The plain fully connected network does not have this ability. It needs more training iterations to improve learning in the small setup. However, for experiment configuration B the PPO_{full} agent falls short of the baseline, while the PPO_{lstm} agent seems to also generalize to larger environments. We see in Figure 3.5 that it outperforms the analytic policy. In this garden environment many plants occlude each other. Hence, the agent has to act based on partial state information. The PPO_{full} agent lacks an memory layer to keep track of the temporal correlations.

We observe for both agents that a larger search space slows down the learning process. In this setup, the PPO_{lstm} agent needs about factor 10 more environment interactions are to surpass the baseline than for experiment configuration A. Along with the manual hyperparameter search tuning such a model requires a significant amount of computation resources and energy. To further scale the experiment might be intractable. However, the small scale configurations show good performance and do not require as many resources. Future research could analyze whether learning to treat local plants can be utilized to control large scale environments.

Sim2Real Transfer We observe in Table 3.2 that our trained PPO_{lstm} agent can deliver better performance than the baseline when tested with new data from the same configuration it has been trained on. However, this might not be enough to evaluate how the agent performs in a real-world garden. We test whether our agents can generalize to new garden configurations with the validation experiments. This configuration represents a scenario in which an agent has to control a plant with very different growth characteristics. As shown in Table 3.3 the PPO-based agents are

able to outperform on par the baseline for the experiment configuration A. However, the performance drops in the validation experiments with configuration B. Hence, we are not able to show that a PPO-based agent generalizes to treating plants with different growth characteristics. In our recent work [PAT^{+}21], we present the AlphaGarden Autonomous Pipeline, as depicted in Fig. 1.1. For this pipeline, we use the *analytic policy* to determine appropriate treatments with AlphaGardenSim that are executed in a real physical garden. It was not possible to validate the performance of our PPO-based agents in reality within the scope of this work. We leave this for future work.
Our analysis to assess transferability to reality is limited as we only randomize a subset of environment parameters. It is uncertain how the agent behaves when changing other simulator parameters. In a physical garden, even in a controlled environment like a green house, an agent is likely to observe a even more versatile environment and plant behavior. For future work it is of interest to analyze the policy behavior in a real garden space.

Chapter 4

Conclusion

Several ideas for learning to treat polyculture garden environments have been presented in this work. We identify whether a reinforcement learning agent can outperform the state-of-the-art policy for polyculture plant treatment within AlphaGardenSim. Furthermore, we analyze whether such an agent can generalize to garden configurations with altered dynamics that have not been present during training.
We give an overview of AlphaGardenSim and introduce extensions we add to improve the simulator's modeling capabilities. We reimplement the simulator and add an interface to apply domain randomization to the environment. In experiments, we demonstrate that the simulation throughput is increased significantly.
We recapitulate the fundamentals for reinforcement learning and policy gradient methods. Furthermore, we assess methods to overcome the challenges posed to an agent by partial observability. We combine the PPO algorithm with a model architecture that uses a LSTM network. This setup is tested in two stochastic environments, one with three plants and the other with sixteen plants. We observe for both configurations that a PPO_{lstm} agent grows environments with higher garden coverage, diversity and has a lower water usage than the previous state-of-the-art policy. A fully connected network architecture without a memory layer was only successful in the small garden setup. The PPO_{lstm} agent's memory layer is beneficial to control environments with many occluded plants. Furthermore, we observe that both PPO-based agents are not able to beat the baseline when they are deployed in a new garden environment that they did not interact with during training.

Appendix A

Environment Configuration

Listing A.1: Simulator configuration file for tuning growth curves in the human-readable data-serialization language YAML.

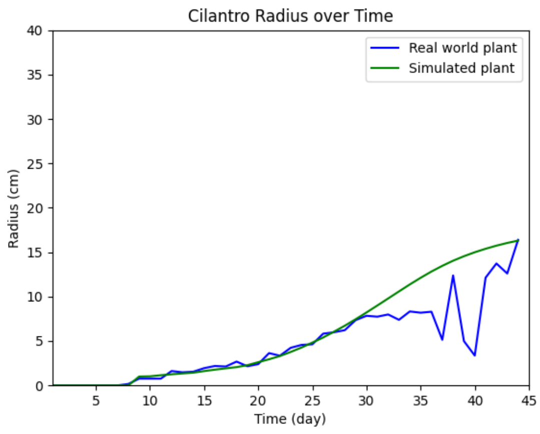
```
---
# Garden Parameters
amount_plants:
  default_value: 8
amount_plant_types:
  default_value: 8
garden_length:
  default_value: 150 # cm
garden_width:
  default_value: 150 # cm
garden_days:
    default_value: 100
max_water_content:
  default_value: 0.3
permanent_wilting_point:
  default_value: 0.0
init_water_mean:
  default_value: 0.2
init_water_scale:
  default_value: 0.04
evaporation_percent_mean:
  default_value: 0.04
evaporation_percent_scale:
  default_value: 0.0054
irrigation_amount:
  default_value: 0.0002

# Policy Parameters
```

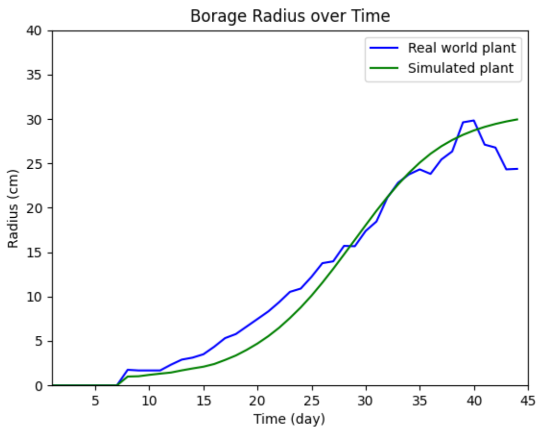
```
water_threshold:
  default_value: 1.0
irr_health_window_width:
  default_value: 9  # cm
prune_window_rows:
  default_value: 5
prune_window_cols:
  default_value: 5
prune_rate:
  default_value: 0.0 # disable pruning
prune_delay:
  default_value: 20
sector_rows:
  default_value: 15 # cm
sector_cols:
  default_value: 30 # cm

# Plant Parameters
reference_outer_radii: # cm
  default_value: [25, 20, 34, 33, 42, 20, 22, 35]
  dtype: np.int
common_names:
  default_value:
  - Green Lettuce
  - Red_Lettuce
  - Borage
  - Swiss Chard
  - Kale
  - Cilantro
  - Radicchio
  - Turnip
x_coordinates:
  default_value: [37, 37, 37, 75, 75, 75, 113, 113]
  dtype: np.int
y_coordinates:
  default_value: [75, 112, 37, 75, 112, 37, 75, 112]
  dtype: np.int
current_outer_radii: # germination radius (cm)
  default_value: [1, 1, 1, 1, 1, 1, 1, 1]
  dtype: np.int
germination_times:
  default_value: [8, 10, 4, 4, 6, 8, 7, 5]
  dtype: np.int
```

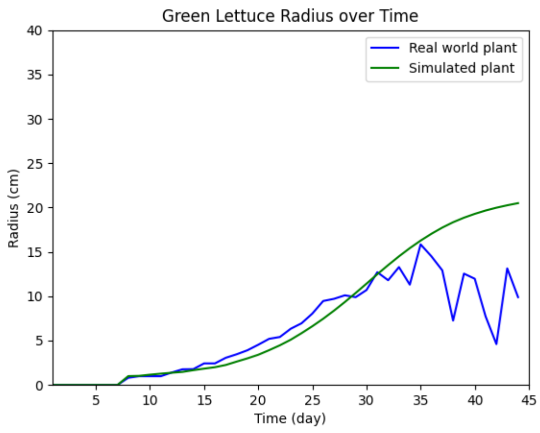
```yaml
# vegetative stage = maturation - germination (time)
maturation_times:
  default_value: [60, 70, 60, 60, 70, 70, 55, 70]
  dtype: np.int
# reproductive stage
waiting_times:
  default_value: [5, 3, 5, 8, 3, 3, 10, 3]
  dtype: np.int
# senescence stage
wilting_times:
  default_value: [18, 16, 18, 18, 16, 16, 18, 16]
  dtype: np.int
light_use_efficiencies:
  default_value: [1, 1, 1, 1, 1, 1, 1, 1]
  dtype: np.float32
water_use_efficiencies:
  default_value:
  - 0.14
  - 0.145
  - 0.165
  - 0.1699
  - 0.165
  - 0.13
  - 0.145
  - 0.195
  dtype: np.float32
overwatered_time_threshold:
  default_value: 5
underwatered_time_threshold:
  default_value: 5
overwatered_threshold:
  default_value: 10
underwaterd_threshold:
  default_value: 0.01
---
```



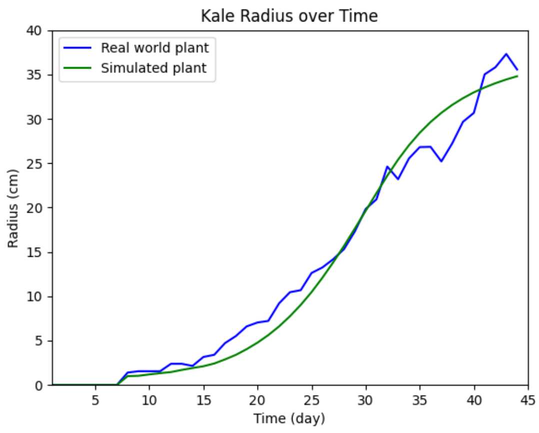
(a) Growth analysis of cilantro.



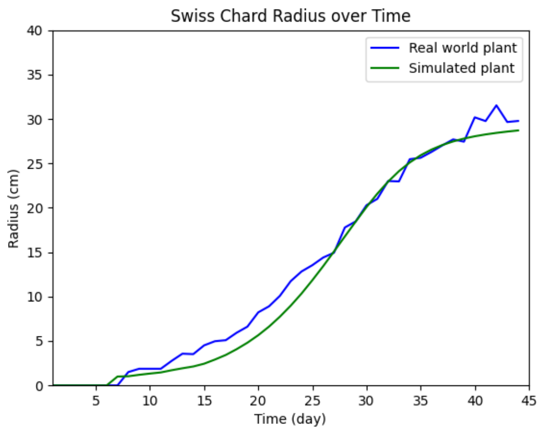
(b) Growth analysis of borage.



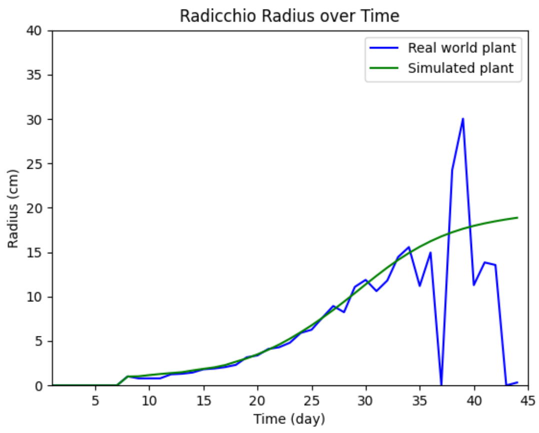
(c) Growth analysis of green lettuce.



(d) Growth analysis of kale.



(e) Growth analysis of Swiss chard.



(f) Growth analysis of radicchio.

Figure A.1: Growth profile for six plant types. Green curve shows the simulated plant. Blue curve shows the real world measurements from our previous work [ADW+21].

Plant type	$r_{\max}$ (cm)			t_g (days)			t_m (days)			t_{wa} (days)			t_{wi} (days)		
	μ	σ	$[a,b]$	μ	σ	$[a,b]$	μ	σ	$[a,b]$	μ	σ	$[a,b]$	μ	σ	$[a,b]$
Borage	34	2	$[30,38]$	4	1	[1,4]	60	2	[1,63]	5	2	[1,100]	18	2	[1,20]
Cilantro	20	2	$[16,24]$	8	1	[1,13]	70	2	[1,73]	3	2	[1,100]	16	2	[1,18]
Radicchio	22	2	$[18,26]$	7	1	[1,12]	55	2	[1,58]	10	2	[1,100]	18	2	[1,20]
Kale	42	2	$[38,46]$	6	1	[1,11]	70	2	[1,73]	3	2	[1,100]	3	2	[1,5]
Green lettuce	25	2	$[21,29]$	8	1	[1,13]	60	2	[1,63]	5	2	[1,100]	18	2	[1,20]
Red lettuce	20	2	$[16,24]$	10	1	[1,15]	70	2	[1,73]	3	2	[1,100]	16	2	[1,18]
Swiss chard	33	2	$[29,37]$	4	1	[1,9]	60	2	[1,63]	8	2	[1,100]	18	2	[1,20]
Turnip	35	2	$[31,39]$	5	1	[1,10]	70	2	[1,73]	3	2	[1,100]	16	2	[1,18]
Invasive	55	2	$[51,59]$	2	1	[1,7]	55	2	[1,58]	3	2	[1,100]	10	2	[1,12]

Table A.1: Plant parameters from AlphaGardenSim modeled as truncated normal distributions, with mean μ, variance σ and interval $[a,b]$. The randomized parameters are the maximal growth potential $r_{\max}$, the germination time t_{g}, the maturation time t_{m}, the waiting duration t_{wa} and the wilting stage duration t_{wi}.

	Environment configuration	
Hyperparameters	*A*	*B*
Train batch size	1200	7600
Entropy coeff. c_2	0.001	0.001
Discount γ	0.99	0.9
GAE parameter λ	0.95	0.95
Learning rate	0.001	0.001
SGD minibatch size	300	513
VF coeff. c_1	1.0	1.0
Number of actors	3	14
Clipping ϵ	0.3	0.3

Table A.2: Hyperparameters for PPO algorithm for each environment configuration.

List of Figures

Acronyms

DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
LSTM	Long Short Term Memory
MDP	Markov Decision Process
MME	Multi-Modal Entropy
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PWP	Permanent Wilting Point
SGA	Stochastic Gradient Ascent
SGD	Stochastic Gradient Decent
TRPO	Trust-Region Policy Optimization
VWC	Volumetric Water Content

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, and Matthieu Devin. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[AAF+17] Alejandro Agostini, Guillem Alenyà, Andreas Fischbach, Hanno Scharr, Florentin Wörgötter, and Carme Torras. A cognitive architecture for automatic gardening. *Computers and Electronics in Agriculture*, 138:69–79, June 2017. URL: `http://www.sciencedirect.com/science/article/pii/S0168169916304768`, `doi:10.1016/j.compag.2017.04.015`.

[ADW+21] Yahav Avigal, Anna Deza, William Wong, Sebastian Oehme, Mark Presten, Mark Theis, Jackson Chui, Paul Shao, Huang Huang, Atsunobu Kotani, Satvik Sharma, Rishi Parikh, Michael Luo, Sandeep Mukherjee, Stefano Carpin, Joshua H. Viers, Stavros Vougioukas, and Ken Goldberg. Learning Seed Placements and Automation Policies for Polyculture Farming with Companion Plants. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 902–908, May 2021. ISSN: 2577-087X. `doi:10.1109/ICRA48506.2021.9561431`.

[AGW+20] Yahav Avigal, Jensen Gao, William Wong, Kevin Li, Grady Pierroz, Fang Shuo Deng, Mark Theis, Mark Presten, and Ken Goldberg. Simulating Polyculture Farming to Tune Automation Policies for Plant Diversity and Precision Irrigation. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 238–245. IEEE, 2020.

[ATW14] Alejandro Agostini, Carme Torras, and Florentin Wörgötter. Learning weakly correlated cause-effects for gardening with a cognitive system. *Engineering Applications of Artificial Intelligence*, 36:178–194, November 2014. URL: `https://linkinghub.`

`elsevier.com/retrieve/pii/S0952197614001857`, `doi:10.1016/j.engappai.2014.07.017`.

[AWP+22] Yahav Avigal, William Wong, Mark Presten, Mark Theis, Shrey Aeron, Anna Deza, Satvik Sharma, Rishi Parikh, Sebastian Oehme, Stefano Carpin, Joshua H. Viers, Stavros Vougioukas, and Ken Goldberg. Simulating Polyculture Farming to Learn Automation Policiesfor Plant Diversity and Precision Irrigation. *Submitted to: IEEE Transactions on Automation Science and Engineering*, 2022. Conference Name: IEEE Transactions on Automation Science and Engineering.

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

[BK17] 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. IEEE, 2017.

[CB90] T. Czárán and S. Bartha. The effect of spatial pattern on community dynamics; a comparison of simulated and field data. In *Progress in theoretical vegetation science*, pages 229–239. Springer, 1990.

[CCO18] Timothy E. Crews, Wim Carton, and Lennart Olsson. Is the future of agriculture perennial? Imperatives and opportunities to reinvent agriculture by shifting from annual monocultures to perennial polycultures. *Global Sustainability*, 1, 2018. Publisher: Cambridge University Press.

[CCW+21] Mengting Chen, Yuanlai Cui, Xiaonan Wang, Hengwang Xie, Fangping Liu, Tongyuan Luo, Shizong Zheng, and Yufeng Luo. A reinforcement learning approach to irrigation decision-making for rice using weather forecasts. *Agricultural Water Management*, 250:106838, 2021. Publisher: Elsevier.

[DL05] Oliver Deussen and Bernd Lintermann. *Digital design of nature: computer generated plants and organics.* Springer Science & Business Media, 2005.

[FAO20] IFAD FAO. *The State of Food Security and Nutrition in the World 2020: Transforming food systems for affordable healthy diets.* Number 2020 in The State of Food Security and Nutrition in the World (SOFI). FAO, IFAD, UNICEF, WFP and WHO, Rome, Italy, 2020. URL: `https://www.fao.org/documents/card/en/c/ca9692en`, `doi:10.4060/ca9692en`.

[FZS+08] Thierry Fourcaud, Xiaopeng Zhang, Alexia Stokes, Hans Lambers, and Christian Körner. Plant growth modelling and applications: the increasing importance of plant architecture in growth models. *Annals of Botany*, 101(8):1053–1063, 2008. Publisher: Oxford University Press.

[GA82] S. Gliessman and M. Altieri. Polyculture cropping has advantages. *California Agriculture*, 36(7):14–16, 1982. Publisher: University of California, Agriculture and Natural Resources.

[HBDV+21] Yasmeen Hitti, Ionelia Buzatu, Manuel Del Verme, Mark Lefsrud, Florian Golemo, and Audrey Durand. GrowSpace: Learning How to Shape Plants. 2021.

[Heu99] E. Heuvelink. Evaluation of a dynamic simulation model for tomato crop growth and development. *Annals of Botany*, 83(4):413–422, 1999. Publisher: Elsevier.

[HHA21] Victoria J. Hodge, Richard Hawkins, and Rob Alexander. Deep reinforcement learning for drone navigation using sensor data. *Neural Computing and Applications*, 33(6):2015–2033, 2021. Publisher: Springer.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, and Nathaniel J. Smith. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. Publisher: Nature Publishing Group.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997. URL: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`, `doi:10.1162/neco.1997.9.8.1735`.

[HS15] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *2015 AAAI Fall Symposium Series*, Arlington, November 2015.

[JHP+03] James W. Jones, Gerrit Hoogenboom, Cheryl H. Porter, Ken J. Boote, William D. Batchelor, L. A. Hunt, Paul W. Wilkens, Upendra Singh, Arjan J. Gijsman, and Joe T. Ritchie. The DSSAT cropping system model. *European journal of agronomy*, 18(3-4):235–265, 2003. Publisher: Elsevier.

[Jon07] Hamlyn G. Jones. Monitoring plant and soil water status: established and novel methods revisited and their relevance to studies of drought tolerance. *Journal of Experimental Botany*, 58(2):119–130, January 2007. Publisher: Oxford Academic. URL: `https://academic.oup.com/jxb/article/58/2/119/531950`, `doi:10.1093/jxb/erl118`.

[KBP13] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. page 38, 2013.

[Kel05] Andrew Keller. Evapotranspiration and crop water productivity: making sense of the yield-ET relationship. In *Impacts of Global Climate Change*, pages 1–11. 2005.

[LEL$^+$18] Brenda B. Lin, Monika H. Egerer, Heidi Liere, Shalene Jha, and Stacy M. Philpott. Soil management is key to maintaining soil moisture in urban gardens facing changing climatic conditions. *Scientific Reports*, 8(1):17565, December 2018. Number: 1 Publisher: Nature Publishing Group. URL: `https://www.nature.com/articles/s41598-018-35731-7`, `doi:10.1038/s41598-018-35731-7`.

[LHP$^+$15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[Lie87] Matt Liebman. Polyculture Cropping Systems. In *Agroecology*. CRC Press, 2 edition, 1987. Num Pages: 14.

[LKJ01] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001. Publisher: SAGE Publications.

[LLN$^+$17] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, page 85, 2017.

[MBM$^+$16] 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 *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MKS+15] 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, February 2015. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7540 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Computer science Subject_term_id: computer-science. URL: https://www.nature.com/articles/nature14236, doi:10.1038/nature14236.

[MP16] Jane Mt. Pleasant. Food yields and nutrient analyses of the Three Sisters: A Haudenosaunee cropping system. *Ethnobiology Letters*, 7(1):87–98, 2016. Publisher: JSTOR.

[OBA21] Hiske Overweg, Herman NC Berghuijs, and Ioannis N. Athanasiadis. CropGym: a Reinforcement Learning Environment for Crop Management. *arXiv preprint arXiv:2104.04326*, 2021.

[PAT+21] Mark Presten, Yahav Avigal, Mark Theis, Satvik Sharma, Rishi Parikh, Shrey Aeron, Sandeep Mukherjee, Sebastian Oehme, Simeon Adebola, and Walter Teitelbaum. AlphaGarden: Learning to Autonomously Tend a Polyculture Garden. *arXiv preprint arXiv:2111.06014*, 2021.

[PAZA18] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[PMV+12] CE Timothy Paine, Toby R. Marthews, Deborah R. Vogt, Drew Purves, Mark Rees, Andy Hector, and Lindsay A. Turnbull. How to fit nonlinear plant growth models and calculate growth rates: an update for ecologists. *Methods in Ecology and Evolution*, 3(2):245–256, 2012. Publisher: Wiley Online Library.

[Ris83] Stephen J. Risch. Intercropping as cultural pest control: Prospects and limitations. *Environmental Management*, 7(1):9–14, January 1983. doi:10.1007/BF01867035.

[RLD+14] Todd S. Rosenstock, Daniel Liptzin, Kristin Dzurella, Anna Fryjoff-Hung, Allan Hollander, Vivian Jensen, Aaron King, George Kourakos, Alison McNally, G. Stuart Pettygrove, Jim Quinn, Joshua H. Viers, Thomas P. Tomich, and Thomas Harter. Agriculture's Contribution to Nitrate Contamination of Californian Groundwater (1945-2005). *Journal of Environmental Quality*, 43(3):895–907, May 2014. `doi:10.2134/jeq2013.10.0411`.

[RSHF09] Dirk Raes, Pasquale Steduto, Theodore C. Hsiao, and Elias Fereres. AquaCrop—the FAO crop model to simulate yield response to water: II. Main algorithms and software description. *Agronomy Journal*, 101(3):438–447, 2009. Publisher: Wiley Online Library.

[RVR$^+$16] Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SDB$^+$20] TjeerdJan Stomph, Christos Dordas, Alain Baranger, Joshua de Rijk, Bei Dong, Jochem Evers, Chunfeng Gu, Long Li, Johan Simon, and Erik Steen Jensen. Designing intercrops for high yield, yield stability and efficient use of resources: are there principles? *Advances in Agronomy*, 160(1):1–50, 2020. Publisher: Elsevier.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[SLH$^+$14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[SML$^+$15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, June 2015. arXiv: 1506.02438 version: 1. URL: `http://arxiv.org/abs/1506.02438`.

[SWD$^+$17] 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`.

[TKBL05] Marc Tchamitchian, Constantin Kittas, Thomas Bartzanas, and Christos Lykas. Daily temperature optimisation in greenhouse by reinforcement learning. *IFAC Proceedings Volumes*, 38(1):131–136, 2005. Publisher: Elsevier.

[TS10] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[TWC+18] David Tseng, David Wang, Carolyn Chen, Lauren Miller, William Song, Joshua Viers, Stavros Vougioukas, Stefano Carpin, Juan Aparicio Ojea, and Ken Goldberg. Towards automating precision irrigation: Deep learning to infer local soil moisture conditions from synthetic aerial agricultural images. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 284–291. IEEE, 2018.

[VME07] J. Vos, L. F. M. Marcelis, and J. B. Evers. Functional-Structural plant modelling in crop production: adding a dimension. *Frontis*, pages 1–12, February 2007. URL: `https://library.wur.nl/ojs/index.php/frontis/article/view/1367`.

[WAB+19] Marius Wiggert, Leela Amladi, Ron Berenstein, Stefano Carpin, Joshua Viers, Stavros Vougioukas, and Ken Goldberg. RAPID-MOLT: A Meso-scale, Open-source, Low-cost Testbed for Robot Assisted Precision Irrigation and Delivery. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1489–1496, August 2019. ISSN: 2161-8089. `doi:10.1109/COASE.2019.8842877`.

[WFPS07] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *International conference on artificial neural networks*, pages 697–706. Springer, 2007.

[WHL20] Lu Wang, Xiaofeng He, and Dijun Luo. Deep reinforcement learning for greenhouse climate control. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 474–480. IEEE, 2020.

[Wil92] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. Publisher: Springer.

License