VIEWPOINT OPTIMIZATION FOR AUTONOMOUS STRAWBERRY
HARVESTING WITH DEEP REINFORCEMENT LEARNING

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Mechanical Engineering

by
Jonathon Sather
March 2019

© 2019
Jonathon Sather
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Viewpoint Optimization for Autonomous
Strawberry Harvesting with Deep Rein-
forcement Learning

AUTHOR: Jonathon Sather

DATE SUBMITTED: March 2019

COMMITTEE CHAIR: John Ridgely, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Jane Zhang, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: Saeed Niku, Ph.D.
Professor of Mechanical Engineering

ABSTRACT

Viewpoint Optimization for Autonomous Strawberry Harvesting with Deep Reinforcement Learning

Jonathon Sather

Autonomous harvesting may provide a viable solution to mounting labor pressures in the United States’s strawberry industry. However, due to bottlenecks in machine perception and economic viability, a profitable and commercially adopted strawberry harvesting system remains elusive. In this research, we explore the feasibility of using deep reinforcement learning to overcome these bottlenecks and develop a practical algorithm to address the sub-objective of viewpoint optimization, or the development of a control policy to direct a camera to favorable vantage points for autonomous harvesting. We evaluate the algorithm’s performance in a custom, open-source simulated environment and observe affirmative results. Our trained agent yields 8.7 times higher returns than random actions and 8.8 percent faster exploration than our best baseline policy, which uses visual servoing. Visual investigation shows the agent is able to fixate on favorable viewpoints, despite having no explicit means to propagate information through time. Overall, we conclude that deep reinforcement learning is a promising area of research to advance the state of the art in autonomous strawberry harvesting.

ACKNOWLEDGMENTS

Thanks to:

- My parents, Ronna and Michael Sather, and sister, Allison Sprague, for continual support and encouragement
- Journey McDowell for facilitating insightful discussions about reinforcement learning
- Dr. John Lin and the California Strawberry Commission production automation team for being my go-to resource for all things strawberry automation

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER	
1 Introduction	1
1.1 Statement of Research Questions	3
1.2 Issues and Assumptions	3
1.3 General Approach	4
2 Background and Theory	5
2.1 Autonomous Strawberry Harvesting	5
2.2 Reinforcement Learning	10
2.2.1 Preliminaries	10
2.2.2 Approaches	12
2.2.3 Q-Learning	14
2.2.4 DDPG	19
2.2.5 Partial Observability	20
2.3 Object Recognition	21
2.3.1 YOLO	26
2.3.2 YOLOv2	29
3 Algorithm and System Design	30
3.1 Environment	30
3.2 Algorithm	33
4 System Implementation	38
4.1 Simulated Environment	38
4.1.1 Plant Model	40
4.2 Detector	42
4.3 Reinforcement Learning	45
5 Testing and Evaluation	51
5.1 Detector	51

5.1.1	Performance on Test Set	51
5.1.2	Visual Analysis	55
5.2	Reinforcement Learning	57
5.2.1	Baseline Comparison	57
5.2.2	Policy Evaluation	58
5.2.3	Hybrid Policy	63
5.2.4	Early Termination Phenomenon	66
5.2.5	Fixation Analysis	69
6	Summary and Conclusions	74
6.1	Contributions	74
6.2	Limitations	75
6.3	Future Works	76
6.3.1	Batch Normalization	77
6.3.2	Noise	77
6.3.3	Parallelization	77
6.3.4	Memory Unit	78
6.3.5	Environment	78
6.3.6	Scope	79
	BIBLIOGRAPHY	81
	APPENDICES	
A	Links	94
B	List of Meshes in Plant Model	95
C	Additional Trajectory Visualizations	100
D	Raw Data	106

LIST OF TABLES

Table		Page
4.1	Manipulator requirements	39
4.2	Plant generation parameters	42

LIST OF FIGURES

Figure	Page
1.1 Strawberry harvesting in Santa Maria, CA. Credit: California Strawberry Commission	1
1.2 Visualization of the reinforcement learning process. Modified image from [6]	2
2.1 Two popular strawberry cultivation environments. Credit: Mercy Olmstead	6
2.2 Directed graph depicting a Markov Decision Process. Source: [33] .	12
2.3 Components of a traditional object recognition system. Source: [58]	23
2.4 Convolutional neural networks simplify the object recognition pipeline. Modified image from [66]	25
2.5 The YOLO detection pipeline. Source: [73]	27
2.6 The YOLO network architecture. Source: [73]	27
2.7 Visual depiction of the IOU similarity metric. Source: [78]	29
3.1 Typical open field cultivation environment (reused from Chapter 2).	31
3.2 Strawberry harvesting robot prototypes.	32
3.3 UR5 articulated robot with Robotiq wrist camera [83].	33
3.4 Husky UGV manipulation package by Clearpath Robotics [84]. . . .	34
3.5 Illustration of the state and action space.	35
4.1 Custom simulated environment displayed in Gazebo.	38
4.2 Harvester’s eye view using RQT.	40
4.3 A generated strawberry bed versus an actual one.	42
4.4 Poorly annotated training image due to camera oscillations.	43
4.5 Properly annotated training image.	44
4.6 The detector comfortably annotates images at over 30fps.	44
4.7 Process diagram showing interaction between system nodes.	46
4.8 Browser-based desktop environment for running experiments. . . .	47
4.9 Returns over 10,000 training episodes. Smoothed using $\gamma = 0.9$. . .	48
4.10 Average max Q values over 10,000 training episodes. Smoothed using $\gamma = 0.9$	48
4.11 Total steps over 10,000 training episodes. Smoothed using $\gamma = 0.9$.	49

4.12	Relative frequency of output activation magnitudes for the actor network over 10,000 training episodes.	49
4.13	Relative frequency of output activation magnitudes for the critic network over 10,000 training episodes.	50
5.1	Sample test image showing ground truth (dark) and predicted (light) bounding boxes.	52
5.2	Histogram of x-coordinate errors on testing set.	52
5.3	Histogram of y-coordinate errors on testing set.	53
5.4	Histogram of width errors on testing set.	53
5.5	Histogram of height errors on testing set.	54
5.6	Precision/recall curve for the strawberry detector. Thresholds listed at each data point.	54
5.7	Screen-shot of the strawberry detector using the real-time camera feed (1).	55
5.8	Screen-shot of the strawberry detector using the real-time camera feed (2).	56
5.9	Screen-shot of the strawberry detector using the real-time camera feed (3).	56
5.10	Average return per episode for each policy.	59
5.11	Average reward per step for each policy.	59
5.12	Average steps until first reward, excluding episodes without a reward.	60
5.13	Average episode length for each policy.	60
5.14	Number of episodes with length < 4	61
5.15	Average return per episode, ignoring episodes with length < 4	62
5.16	Average reward per step, ignoring episodes with length less < 4	62
5.17	Average steps until first reward, excluding episodes without a reward or length < 4	63
5.18	Average episode length, ignoring episodes with length < 4	63
5.19	Average return per episode, including hybrid policy.	64
5.20	Average reward per step, including hybrid policy.	65
5.21	Average episode length, including hybrid policy.	65
5.22	Number of episodes with length < 4 , including hybrid policy.	66
5.23	Range of foliage densities from $\mu_{leaves} = 20$ to $\mu_{leaves} = 50$	67
5.24	Number of early terminations for different levels of foliage.	68

5.25	Comparison of episode lengths for different levels of foliage.	68
5.26	Visualizations where the agent fixates on high-reward viewpoints. .	70
5.27	Visualizations where the agent fixates on low-reward viewpoints. . .	70
5.28	Visualizations where the agent does not appear to fixate.	71
5.29	First-person view from high-return fixation location (1).	72
5.30	First-person view from high-return fixation location (2).	72
5.31	First-person view from low-return fixation location (1).	73
5.32	First-person view from low-return fixation location (2).	73
B.1	Stem models for different leaf heights.	95
B.2	Stem models for different berry orientations.	95
B.3	Leaf models.	96
B.4	Calyx models.	97
B.5	Berry models.	98
B.6	Texture mappings for seven degrees of ripeness.	99

Chapter 1

INTRODUCTION

Driving down Highway 101 through Santa Maria, California in the springtime, it is hard to ignore the abundance of strawberry fields on both sides of the highway. Each field is populated by dozens of seasonal laborers inching their way down the rows and stooping to manually collect ripe berries – a practice that has been largely unchanged for the past 700 years [1]. Recently, the heavy reliance on human labor has become problematic for the California strawberry industry, as an aging and ever-shrinking workforce continues to drive up costs [2].

Figure 1.1: Strawberry harvesting in Santa Maria, CA. Credit: California Strawberry Commission

Autonomous harvesting offers a promising solution to the labor shortage. With the right technology, strawberry harvesting robots could fill the roles of human pickers and improve overall cost-effectiveness for California growers. Although not fully autonomous, mechanical harvesting systems have already demonstrated their utility for several crops, including olives, walnuts, and almonds [3]. A profitable and commercially adopted harvesting system for strawberries remains elusive.

Strawberry plants present several difficulties for autonomous harvesting systems, including high levels of occlusion, lighting variation, and easily-bruised fruit. Tradi-

tional algorithms relying on basic image processing and hard-coded rules have shown to be unsuccessful, either yielding an unacceptably low percentage of ripe strawberries or being too slow for practical usage [4]. More recent strategies making use of specialized hardware or unconventional growing conditions have yet to meet a competitive price point [5]. This suggests that a more sophisticated algorithm for ripe strawberry harvesting not reliant on specialized hardware or growing conditions may be necessary to achieve satisfactory performance at an affordable rate.

Deep reinforcement learning is a powerful framework that may be able to meet these goals. At a high level, reinforcement learning provides the machinery for a robotic harvester to learn how to harvest ripe strawberries through interaction with its environment. “Deep” refers to the use of deep neural networks as function approximators, which has enabled reinforcement learning algorithms to scale to previously-intractable problems. Figure 1.2 shows the reinforcement learning process.

Figure 1.2: Visualization of the reinforcement learning process. Modified image from [6]

In theory, deep reinforcement learning could enable the robotic agent to develop sophisticated control policies that are infeasible to code by hand and capture the complex relationships required to reason about the unstructured harvesting environment. However, reinforcement learning systems are often fragile and have seen limited

success in real-world robotics applications [7].

1.1 Statement of Research Questions

We seek to gain insight on whether reinforcement learning could be used to improve the economic viability of autonomous strawberry harvesting systems. In doing so, we narrow the scope of the autonomous harvesting problem to create a manageable objective that would still facilitate meaningful results. To this end, we focus on the task of viewpoint optimization. By viewpoint optimization, we mean the development of a control policy for a robotic manipulator to direct a camera to favorable vantage points for autonomous harvesting.

Specifically, in this research we seek to answer the following questions:

1. How well can a reinforcement learning solution for viewpoint optimization perform with respect to its own reward scheme in a simulated harvesting environment?
2. How useful would such a policy be in a real autonomous strawberry harvesting application?
3. Can we gain insight on reinforcement learning’s potential for the full autonomous harvesting process?

1.2 Issues and Assumptions

Much of this research hinges on the development of a reinforcement learning algorithm and its application in a simulated harvesting environment. In order to extend conclusions to practical contexts, we make the assumption that if tasks in the simulated environment are on the same magnitude of difficulty as those in the real-world, we can extrapolate meaning from the algorithm’s performance to real-world scenarios. Here, we use intuition to define a simulation’s magnitude of difficulty and impose the requirements that the simulation must contain apparent randomness, frequent occlu-

sions, complex textures, shadows, and lighting variation. Additionally, we require that the agent in the simulated environment does not have access to ground-truth information that would not be available in the real-world.

1.3 General Approach

We begin by formalizing the viewpoint optimization problem within the context of reinforcement learning. Then, we develop a novel reinforcement learning algorithm to train a robotic harvester to navigate to favorable viewpoints for autonomous harvesting. A general-purpose simulated environment is created for our experiments, which spawns a randomly-generated strawberry plant at the beginning of each episode. All parts of the algorithm are then trained in the simulated environment using analogous procedures to the real-world. We compare the learned agent’s performance against several baseline policies and perform visual investigation to gain additional insights on its behavior. Finally, we comment on the results of our analysis with respect to our research questions and conclude by making recommendations for future works.

Chapter 2

BACKGROUND AND THEORY

In this chapter we outline relevant literature and theory for the various components of this research. The domains used in this research can be broken down into three categories: autonomous strawberry harvesting, reinforcement learning, and object recognition. Note that all three of these categories have a considerable amount of overlap, which will be discussed below. We begin our review with autonomous strawberry harvesting.

2.1 Autonomous Strawberry Harvesting

The goal of this section is to provide context for the viewpoint optimization problem within the diverse field of autonomous harvesting. This is accomplished through a top-down approach, starting with an outline of the strawberry production process as a whole and then progressively refining the scope until we reach areas most relevant to this research. When discussing the strawberry production process, it is worth noting that many of its components are largely dependent on the setting in which the strawberries are grown (e.g. greenhouse, tunnel, open field). To simplify matters, we focus solely on the setting of our research, the open field, when surveying background information. For reference, two common cultivation environments are shown in Figure 2.1.

The strawberry production process can be broken down into five operations [8]. The first step is planting, which involves preparing the growing environment and inserting small strawberry plants. For a plasticulture configuration, this includes shaping the beds, setting trickle irrigation lines, laying plastic mulch, and punching holes before manually inserting the plants. The next operation is known as in-season management. As the name implies, this stage involves managing the field for the

Open field

Greenhouse

Figure 2.1: Two popular strawberry cultivation environments. Credit: Mercy Olmstead

duration of the strawberry season, performing tasks such as monitoring and adjusting growth parameters, detecting signs of stress and disease, predicting future yields, and removing runners. The next operation is harvesting, which is then followed by sorting and packaging. Although regarded as different operations, harvesting and packaging are frequently combined in the field to increase efficiency. The final operation is post-harvesting quality inspection. This is necessary to ensure that the strawberries are not diseased or damaged as they come to market and is used as feedback for the efficacy of crop management procedures [9].

In California, over 2 billion dollars are spent on strawberry production per year, roughly 50% spent of which is spent on harvesting alone [10][11]. With global pressures for production and the availability of cheap harvesting labor outside of the United States, it is especially desirable for growers to reduce these costs to maintain a competitive price point [12]. A leading idea for reducing costs are autonomous harvesting systems, which can allow growing operations to persist despite diminishing labor force, provided that commercial requirements can be met. While automated solutions have pervaded the commercial aspects of several other aspects of the strawberry production process, such as planting and quality inspection, there have been technological and financial hurdles that have made interfered with the adoption of

autonomous harvesting systems [8]. To understand these hurdles, we consider the algorithmic requirements for designing such a system.

While for humans, strawberry harvesting can be thought of as one continuous process, it is often useful to break down the process into discrete steps when designing an autonomous harvesting system. In [8], the harvesting workflow is generalized to four steps: ripe strawberry detection, pose determination, path planning and execution, and fruit removal. The detection process involves localizing the ripe strawberries within the space of the sensing modality, e.g. labeling bounding boxes for 2D camera images, and can also be extended to the detection of other relevant harvesting obstacles, such as leaves, stems, and unripe strawberries. From here, the objects of interest are localized in 3D space, and their pose is determined. A common strategy for this is using stereo cameras to infer object depth via triangulation and inferring berry pose from shape or stem information [13]. The poses of the detected objects can then be used as constraints as a trajectory is planned and executed to approach ripe strawberries. A popular way to determine these trajectories is to use sample-based planners like Rapidly-exploring Random Trees (RRT) [14][15]. Once a strawberry is within range, a grasping pose is selected and executed before placing the removed strawberry into a storage receptacle.

With respect to the aforementioned harvesting workflow, viewpoint optimization can be thought of as a preliminary action that improves the remaining steps of the harvesting process. For example, an “optimal” vantage point likely aids in ripe strawberry detection and localization while reducing obstacles to simplify the planning and execution of harvesting trajectories. It is worth noting that a similar effect may be attained using multiple fixed cameras to extensively map out a strawberry plant prior to running the harvesting workflow. However, multiple cameras require additional computational resources for sensor fusion and may still experience challenges due to occlusion under heavy foliage cover.

To the best of our knowledge, this work is the first research exploring viewpoint optimization via reinforcement learning for autonomous harvesting applications (any crop). A related idea is visual servoing [16], which is a class of control algorithms acting on image features that has been applied in the autonomous harvesting domain. Unlike our method, visual servoing typically involves hand-specifying features, whereas our algorithm learns a control policy with a data-driven approach. In [17], Mehta and Burks implement visual servoing by means of two cameras: one in the hand of a citrus harvesting robot and one stationary camera with a wide field of view. The feedback from the cameras is then used to create a perspective image and guide the robotic manipulator towards an artificial citrus fruit. One of the main limitations of this approach is that it requires the target fruit to be visible by the fixed camera, which cannot be guaranteed in unstructured environments.

The work in this research falls under the realm of “deep learning”, which is a subset of machine learning involving hierarchical structures capable of learning complex data representations, such as deep neural networks [18]. A recent survey [19] identifies 40 papers that have used deep learning approaches in agriculture. The topics of these papers include weed identification, land cover classification, plant recognition, fruit counting, and crop type classification. All of the papers use supervised learning, meaning a data set was first collected or generated, and then a deep neural network architecture was trained to perform the respective task using the training data. As will be described in the following sections, the supervised approach is fundamentally different from the reinforcement learning framework used in this paper, which involves dynamically interacting with the environment to learn a control policy.

Like the research in this paper, several of the works covered in [7] use a simulated environment for cost effective data collection and rapid prototyping. In [20], a synthetic dataset of 24,000 images is created for the task of tomato counting. The images are passed to a deep neural network based on the Inception-ResNet architecture [21]

and a scalar value is predicted to estimate the number of tomatoes. In [22], simulated images of plants are created for a weed detection task. The image patches are then fed to a modified version of the VGG-16 convolutional neural network (CNN) architecture [23] that returns pixel labels for class membership as weed, soil, or maize.

Outside of the harvesting realm there are several works which attempt to accomplish tasks similar in spirit to viewpoint optimization using deep reinforcement learning. In [24], a deep reinforcement learning algorithm based on Asynchronous Actor-critic Agents (A3C) [25] is used to solve a navigation task in a simulated environment. The work leverages auxiliary objectives to encourage learning of spatial information that can help with navigation. The network architecture includes long short-term memory (LSTM) layers [26] to propagate contextually relevant information forward through time. During both training and testing, the agent is spawned in random locations in a fixed, partially-observed environment and must use its learned policy to navigate to the goal region.

Perhaps more relevant is [27], where a combined mapping and planning deep architecture is trained using imitation learning to solve both geometric and semantic exploration tasks in unseen simulated environments. A mapping network is trained to output a belief about the environment, which is fed into a planning “value iteration network” [28] that uses a hierarchical approach to efficiently plan a sequence of future actions from the current timestep. The networks are trained using Dataset Aggregation (DAgger) [29], an imitation learning scheme which sequentially updates the training dataset to support the state distribution of the learned policy. The primary limitation with this approach is that it requires an expert policy to be available during all stages of training.

Both [24] and [27] employ memory schemes to propagate information forward through time to combat partial observability in their respective environments. Like these environments, the harvesting environment in our work is also partially observed.

However, we do not incorporate any (direct) methods for information propagation through time in our network architecture. This decision is justified in Section 2.2 and its implications are considered in Chapters 5 and 6.

2.2 Reinforcement Learning

Reinforcement learning is a machine learning framework in which an agent seeks to learn an optimal behavioral policy through interaction with its environment, sequentially improving its actions through trial and error to maximize future rewards. It is a promising technique for accomplishing tasks where direct supervisory data is scarce or challenging to quantify a priori, and has seen landmark success in a wide variety of domains, including simulated environments, video games, and select real-world contexts [30]. In the robotics domain, reinforcement learning is particularly intriguing because it offers the potential to learn behavioral policies for tasks that are difficult or infeasible to code by hand, as is common in unstructured and complex physical environments. This potential has been increasingly evident through the use of deep neural networks, which have enabled algorithms to scale to previously-intractable problems, such as those incorporating high-dimensional sensory data. As such, autonomous strawberry harvesting is a prime candidate for reinforcement learning.

We will begin by defining a common framework for reinforcement learning and then build the fundamentals until we arrive at the base algorithm used in this work. The ensuing discussion assumes some familiarity with machine learning and related topics, including basic theory and applications of probability, linear algebra, and neural networks.

2.2.1 Preliminaries

Reinforcement learning problems are often modeled as a Markov Decision Process (MDP), which describes a discrete-time, stochastic environment with a decision-

making agent [31]. MDPs are defined by the 5-tuple $\{S, A, P, R, \gamma\}$, where:

1. $s_t \in S$ is the set of all states, each s_t containing all relevant information about the environment at time t .
2. $a_t \in A$ is the set of all possible actions in the environment at time t .
3. $P(s_{t+1}|s_t, a_t)$ is the state-transition probability for state s_{t+1} given state s_t and action a_t .
4. $R(r_t|s_t, a_t)$ is the reward probability for r_t given state s_t and action a_t .
5. $\gamma \in [0, 1)$ is the discount factor, which is used to geometrically decay the value of future rewards and often aids in algorithm convergence [32].

The resulting process is characterized by a cyclic interplay between an agent and its environment (Figure 2.2). At timestep t , an agent observes state s_t and subsequently takes action a_t according to its policy $\pi(a_t|s_t)$. The agent then arrives at state s_{t+1} through the environment’s dynamics $P(s_{t+1}|s_t, a_t)$ and receives scalar reward r_t according to $R(r_t|s_t, a_t)$. This process repeats, now from s_{t+1} , until a termination criterion is reached.

A roll-out of states and actions from initial state s_0 until termination is called a trajectory, denoted $\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1})$. The probability of a given trajectory (Equation 2.1) can be calculated by exploiting the Markov Property, which asserts that the probability distribution of future states is conditional only upon the present state [34]. The presence of this property is evident from the MDP definition. We denote the probability density function over all trajectories under policy π as $p_\pi(\tau)$.

$$\Pr(\tau) = \Pr(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t) \quad (2.1)$$

It is useful to consider accumulated (discounted) reward, or return, from an arbitrary state until termination. Without loss of generality, we denote a return starting

Figure 2.2: Directed graph depicting a Markov Decision Process. Source: [33]

at $t = 0$:

$$\mathcal{R} = \sum_{t=0}^{T-1} \gamma^t r_t \quad (2.2)$$

The objective of reinforcement learning is to develop an optimal policy π^* to maximize the expected return $\mathcal{R}$ from all states. Mathematically,

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim p_{\pi}(\tau)}[\mathcal{R}] \quad (2.3)$$

2.2.2 Approaches

In practice, there are three main strategies used to find (or approximate) π^* : value functions, policy search, and combined methods. As the name implies, value function methods hinge on the approximation of the value function $V^{\pi}(s_t)$, which represents the expected return under policy π from state s_t :

$$V^{\pi}(s_t) = \mathbb{E}_{\tau \sim p_{\pi}(\tau)}[\mathcal{R}|s_t] \quad (2.4)$$

Under unknown system dynamics $P(s_{t+1}|s_t, a_t)$, it is often more useful to solve for the state-value function $Q^\pi(s_t, a_t)$. The state-value function represents the expected return starting at state s_t , taking action a_t , and following policy π thereafter:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau \sim p_\pi(\tau)}[\mathcal{R}|s_t, a_t] \quad (2.5)$$

Value function methods seek to jointly approximate and maximize V^π or Q^π , from which we can implicitly solve for increasingly competent policies. Let V^* and Q^* denote the value and state-value functions under optimal policy π^* . Given V^* or Q^* without knowing the optimal policy, we can deduce π^* at each state by setting:

$$\pi^*(a_t|s_t) = \arg \max_a \mathbb{E}_{s_{t+1} \sim P}[V^*(s_{t+1})] = \arg \max_a Q^*(s_t, a) \quad (2.6)$$

Policy search methods forgo calculation of a value function and instead seek to directly determine an optimal policy π^* . Here, the policy is often represented by a parameterized function π_θ , where the weights θ are iteratively updated to maximize expected return $\mathcal{R}$ from all states. We can rewrite the reinforcement learning objective of Equation 2.3 in terms of θ :

$$\theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)}[\mathcal{R}] \quad (2.7)$$

The weights θ^* are typically learned via stochastic gradient ascent or evolutionary algorithms to maximize the reinforcement learning objective [30].

Combined methods use both value function and policy search methods to approximate an optimal policy. Compared to pure policy search, these methods can reduce gradient variance for stable learning updates. Additionally, they can be used to extend a popular value function approach, Q-Learning [35], to work with continuous actions. In general, there are pros and cons with each of these reinforcement learning strategies, weighing data efficiency, learning signal, training time, and problem specific details. In this research, we use a combined method known as Deterministic Deep Policy Gradients (DDPG) [36] to solve the strawberry viewpoint optimization

objective. In the following subsections we derive DDPG, starting with its analog in discrete action spaces, Q-learning [35], before extending it to continuous actions.

2.2.3 Q-Learning

Q-learning is a value function method in which we seek to uncover the optimal state-value function Q^* in discrete action environments. We begin by developing a convenient representation of the state-value function for an arbitrary policy Q^π and then show how this function can be sequentially updated to approach Q^* . Here we consider MDPs with deterministic rewards $r_t = \mu(R(s_t, a_t))$, deterministic policies $a_t = \mu(\pi(s_t))$, and finite actions $a_i \in A$.

We begin by defining Q^π recursively to separate reward from the current state and future returns (Equation 2.10). In dynamic programming literature, this is referred to as a Bellman equation, and it is useful for dividing challenging optimization objectives into a series of manageable sub-problems [31].

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau \sim p_\pi(\tau)}[\mathcal{R}|s_t, a_t] \quad (2.8)$$

$$= r_t + \gamma \mathbb{E}_{s_{t+1} \sim P, a_{t+1} \sim \pi}[\mathbb{E}_{\tau \sim p_\pi(\tau)}[\mathcal{R}|s_{t+1}, a_{t+1}]] \quad (2.9)$$

$$= r_t + \gamma \mathbb{E}_{s_{t+1} \sim P, a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})] \quad (2.10)$$

The inner expectation is simplified using a deterministic policy, which assigns probability 1 to a single action a_t given state s_t and probability 0 to all other actions. For deterministic policy π :

$$Q^\pi(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim P}[Q^\pi(s_{t+1}, \pi(s_{t+1}))] \quad (2.11)$$

With the above relationship in mind, define an operator β^π on arbitrary state-value function Q :

$$\beta^\pi Q(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim P}[Q(s_{t+1}, \pi(s_{t+1}))] \quad (2.12)$$

By Equation 2.11, we see that Q^π is a fixed point of Equation 2.12, i.e. $\beta^\pi Q^\pi = Q^\pi$. Further, it can be shown that β^π is a contraction mapping under the L_∞ norm [37]. Using these two facts, it follows that an arbitrary Q will converge to Q^π through continual application of β^π [38]:

$$Q \rightarrow \beta^\pi Q \rightarrow \beta^{\pi^2} Q \rightarrow \dots \rightarrow Q^\pi \quad (2.13)$$

If we could formulate an expression for $\beta^* = \beta^{\pi^*}$, we could repeatedly apply β^* to an arbitrary Q to approach Q^* . Recall that under an optimal policy π^* , an agent takes actions at each timestep to maximize the expected return. This is equivalent to choosing the action that maximizes Q^* at each timestep. Thus,

$$Q^*(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim P} [\max_a Q^*(s_{t+1}, a)] \quad (2.14)$$

and

$$\beta^* Q(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim P} [\max_a Q(s_{t+1}, a)] \quad (2.15)$$

Now we can use β^* to solve for Q^* (Algorithm 1). However, for this formula to be directly useful, we need to have a model of the state transitions $P(s_{t+1}|s_t, a_t)$ to compute the expectation in Equation 2.15 for each Q iteration. This is problematic for most reinforcement learning problems, since $P(s_{t+1}|s_t, a_t)$ is typically not known a priori. We can avoid the expectation by taking an unbiased single sample estimate, denoted $\widehat{\beta^*}$:

$$\beta^* Q(s_t, a_t) \approx \widehat{\beta^*} Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a) \quad (2.16)$$

Algorithm 1 Q-Value Iteration

```

Initialize  $Q^0$ 
for  $n = 0, 1, 2, \dots$  until convergence do
     $Q^{n+1} \leftarrow \beta^* Q^n$ 
return  $Q^{n+1}$ 

```

We then run Monte Carlo Q-value iteration by sampling trajectories in the environment and using $\widehat{\beta^*} Q^n$ to approximate $\beta^* Q^n$ at each iteration. Given the noise of

each sample, it is standard to only partially update Q^{n+1} using averaging parameter $\epsilon_n \in (0, 1)$ (Equation 2.17). It can be shown that this is mathematically equivalent to performing a stochastic gradient ascent-like step to on the squared error between Q^n and $\widehat{\beta^*}Q^n$, holding $\widehat{\beta^*}Q^n$ fixed [37] (Equation 2.18). These updates are shown below for a single sample, (s_t, a_t, r_t, s_{t+1}) .

$$Q^{n+1} \leftarrow (1 - \epsilon)Q^n + \epsilon\widehat{\beta^*}Q^n \quad (2.17)$$

or

$$Q^{n+1} \leftarrow Q^n + \epsilon \nabla_{Q^n} \|y_n - Q^n\|^2 \quad (2.18)$$

where $y_n = \widehat{\beta^*}Q^n$.

This is the premise behind Watkins’s Q-learning algorithm [35] (Algorithm 2). Despite its stochasticity, it can be shown to converge to Q^* using the appropriate learning rates ϵ_n and infinite state-action visitation [35][39].

Algorithm 2 Watkins Q-Learning

```

Initialize  $Q^0$ 
for  $n = 0, 1, 2, \dots$  until convergence do
    Take step in environment under some policy  $\pi$  and observe  $(s_n, a_n, r_n, s_{n+1})$ 
    Set target  $y_n = \widehat{\beta^*}Q^n$ 
     $Q^{n+1} \leftarrow Q^n + \epsilon \nabla_{Q^n} \|y_n - Q^n\|^2$   $\triangleright$  No gradient through  $y_n$ 
return  $Q^{n+1}$ 

```

In practice, this algorithm has limited merit outside of toy problems. It requires a tabular representation for Q , which is only feasible in settings with small and discrete state and action spaces, such as the didactic grid-world environment [40]. This is largely due to the fact that memory requirements for storing Q scale exponentially with respect to the number of state and action dimensions, a phenomenon known as “the curse of dimensionality” [41]. We combat this curse by using a parameterized function Q_θ^π to approximate Q^π , which has linear memory requirements with respect

to the number of parameters $|\theta|$. A popular class of functions for Q_θ^π are neural networks due to their expressive power [42].

Using function approximators, we gain the ability to generalize to unseen areas of the state and action space and can now handle continuous inputs. However, for many classes of functions (including neural networks) this comes at the cost of losing convergence guarantees under standard assumptions, and training is often unstable [43]. As a result, a heavy research focus with approximate Q-learning algorithms is developing strategies to stabilize training and increase the probability of convergence.

Neural-Fitted Q Iteration (NFQ) by Martin Riedmiller [44] is one of the first notable efforts to approximate Q-learning using neural networks. A version of the algorithm is shown in Algorithm 3. NFQ uses the same principle of sampling and partial Q updates as in Watkins Q-learning, but differs in that its updates are performed batch-mode on the weights of a neural network Q_θ . This enables NFQ to leverage stable training methods from supervised learning, e.g. Rprop [45], on the full set of state-transitions. Riedmiller showed that NFQ was capable of learning a proficient state-value representation for simple environments using an order of magnitude less samples than tabular methods. One of the biggest limitations of NFQ is that the computational cost per batch update is proportional to the size of the data set, which is inefficient for training large neural networks [46].

Algorithm 3 Neural Fitted Q Iteration

```

Initialize neural network  $Q_\theta^0$ 
for  $n = 0, 1, 2, \dots$  until convergence do
    Sample trajectories  $(s, a, r, s')$  using policy  $\pi^n$ 
     $y_t = r_t + \gamma \max_a Q_\theta^n(s_{t+1}, a)$ 
     $\theta^{n+1} \leftarrow \arg \min_\theta \sum_t ||y_t - Q_\theta^n(s_t, a_t)||^2$ 
return  $Q_\theta^{n+1}$ 

```

The most relevant breakthrough in Q-learning is Deep Q Networks (DQN) [47], a sample-efficient online Q learning algorithm that is designed to accommodate modern

deep neural network architectures. The algorithm, outlined in Algorithm 4, introduces two new adaptations to make this possible: the experience replay and the target network. The experience replay is a ring buffer that stores (s, a, r, s') from the past N steps in the environment. During training, batches are sampled from the experience replay buffer to decrease intra-batch correlation during the learning process and allows a reuse of state-transitions for multiple update steps. The target network $Q_{\theta'}$ is a “frozen” copy of Q_{θ} used to stabilize training targets Y_t . The target weights θ' are updated to match θ every K timesteps.

Algorithm 4 DQN

```

Initialize experience replay  $\mathbb{D}$  with size  $N$ 
Initialize  $Q_{\theta}$  with random weights
Initialize target network  $Q_{\theta'}$  with weights  $\theta' = \theta$ 
for episode = 1, M do
    Receive initial state  $s_t$ 
    for  $t = 1, T$  do
        Take some action  $a_t$ 
        Observe  $r_t$  and  $s_{t+1}$ 
        Store  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathbb{D}$ 
        Sample mini-batch  $(s_i, a_i, r_i, s_{i+1})$  from  $\mathbb{D}$  uniformly
         $y_i = r_i + \gamma \max_a Q_{\theta}(s_{i+1}, a)$ 
         $\theta \leftarrow \theta - \epsilon \sum_i \nabla_{\theta} ||y_i - Q_{\theta}(s_i, a_i)||^2$   $\triangleright$  Fit  $Q_{\theta}$  to  $y_i$  via SGD
        Every  $K$  timesteps,  $\theta' \leftarrow \theta$ 
    return  $Q_{\theta}$ 

```

In 2015, DQN was used to achieve state-of-the-art performance on several Atari games using raw pixel inputs. Its success inspired an explosion of spin-off algorithms to further improve stability and efficiency [48], and deep Q-learning algorithms are now some of the most popular choices for complex decision making problems. However, traditional DQN is still limited to domains with discrete action spaces, as it uses a brute force approach to calculate $\max_a Q_{\theta}(s_{t+1}, a)$ which is not possible for continuous actions a . This is the primary motivation behind Deep Deterministic Policy Gradients (DDPG).

2.2.4 DDPG

Recall the calculation of the DQN target values:

$$y_i = r_i + \gamma \max_a Q_\theta(s_{i+1}, a) \quad (2.19)$$

In the DQN algorithm, the maximization over actions is performed by computing $Q_\theta(s_{t+1}, a)$ for each action a and choosing the maximum value Q-value. This brute-force method of maximization is not suitable for continuous actions spaces, since we would need to calculate infinite Q values each iteration. Therefore, we need to look into alternative techniques.

One approach is to estimate $\max_a Q_\theta(s_{t+1}, a)$ by Monte Carlo sampling and choosing the best action as an estimate for the argmax. This is successfully performed in [49], where a non-parametric stochastic policy $\pi(s)$ is improved through an iterative resampling scheme. In general, Monte Carlo estimation can provide decent accuracy with a small enough action space and sufficient samples, but does not scale well to high dimensions [50].

Another strategy is to train a function to predict Q_θ in a form that is easily maximizable and determine its maximum analytically. This is done in NAF [51], where a neural network is used to predict parameters for a convex representation of the action-value function. This is sample efficient and requires very few modifications to the DQN algorithm. However, it makes a strong assumption on the distribution of the Q function, which is problematic if the “true” distribution differs significantly from its assumed form.

Finally, we can learn a function to directly predict the action which maximizes Q_θ . This is the premise behind the DDPG algorithm [36]. In DDPG, we learn two neural network function approximators, the familiar Q_θ which approximates the state-value function under an optimal policy, and a new function π_ϕ which approximates the optimal policy. Using π_ϕ we can approximate $\max_a Q_\theta(s_{t+1}, a)$ as $Q_\theta(s_{t+1}, \pi_\phi(s_t))$.

During training this approximation is used to generate target values y_t , which are regressed upon by Q_θ .

The policy parameters ϕ are updated to maximize $Q_\theta(s, \pi_\phi)$ via stochastic gradient ascent. To calculate $\nabla_\phi Q_\theta$, we use the chain rule to decompose the gradient between Q_θ and π_ϕ , followed by standard backpropagation through each network. Symbolically,

$$\nabla_\phi Q_\theta = \nabla_a Q_\theta(s, a) \nabla_\phi \pi(s) \quad (2.20)$$

In [52], it is shown that the expected value of Equation 2.20 under a given state distribution approximates the policy gradient of the reinforcement learning objective. This lends way to the following off-policy update rule:

$$y_i = r_i + \gamma Q_\theta(s_{i+1}, \pi_\phi(i_t)) \quad (2.21)$$

$$\theta \leftarrow \theta - \epsilon_\theta \sum_i \nabla_\theta ||y_i - Q_\theta(s_i, a_i)||^2 \quad (2.22)$$

$$\phi \leftarrow \phi + \epsilon_\phi \sum_i \nabla_a Q_\theta(s_i, \pi(s_i)) \nabla_\phi \pi(s_i) \quad (2.23)$$

DDPG combines the above update rule with several advancements in deep reinforcement learning to make a practically useful algorithm. As in DQN, the authors leverage target networks and experience replay to improve stability and sample efficiency. The authors also add batch normalization [53] to stabilize the learning process, Ornstein-Uhlenbeck noise [54] for exploration in continuous action spaces, and Polyak averaging [55] to smoothly update the target network parameters. Additional details can be found in [36], as well as the listed citations. The full algorithm is shown in Algorithm 5.

2.2.5 Partial Observability

It is worth noting that while the the forgoing discussion assumes the reinforcement learning problem could be framed as a Markov Decision Process, this is not always the case. In many real-world problems, such as our formulation of viewpoint

Algorithm 5 DDPG

```
Initialize experience replay  $\mathbb{D}$  with size  $N$ 
Initialize  $Q_\theta$  with random weights
Initialize  $\pi_\phi$  with random weights
Initialize target network  $Q_{\theta'}$  with weights  $\theta' = \theta$ 
Initialize target network  $\pi_{\phi'}$  with weights  $\phi' = \phi$ 
for episode = 1,  $M$  do
  Initialize random process  $\mathcal{N}$  for action exploration
  Receive initial state  $s_t$ 
  for  $t = 1, T$  do
    Take action  $a_t = \pi_\phi(s_t) + \mathcal{N}$ 
    Observe  $r_t$  and  $s_{t+1}$ 
    Store  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathbb{D}$ 
    Sample mini-batch  $(s_i, a_i, r_i, s_{i+1})$  from  $\mathbb{D}$  uniformly
     $y_i = r_i + \gamma Q_\theta(s_{i+1}, \pi_\phi(s_i))$ 
    Update main networks:
       $\theta \leftarrow \theta - \epsilon_\theta \sum_i \nabla_\theta ||y_i - Q_{\theta'}(s_i, a_i)||^2$ 
       $\phi \leftarrow \phi + \epsilon_\phi \sum_i \nabla_a Q_{\theta'}(s_i, \pi(s_i)) \nabla_\phi \pi_{\phi'}$ 
    Update target networks:
       $\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$ 
       $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$ 
return  $\pi_\phi$ 
```

optimization, each observation taken in the environment may not contain all relevant information to maximize expected return. In this case, the environment is said to be partially observed, which often warrants a generalization of the MDP framework [56]. Despite this, we use algorithms designed for fully observed MDPs in a partially observed setting. Although we lose some theoretical support, this simplifies our implementation while still facilitating meaningful insights for reinforcement learning’s promise in autonomous harvesting.

2.3 Object Recognition

Computer vision is a broad field that relates to how computers can be programmed to extract meaningful information from digital images and videos. For the purposes of this research, we restrict this broad scope to the context of RGB images sensed by

a color camera, unless otherwise specified, since this is the primary sensing modality used here. In [57], the authors segment the computer vision problem into four levels of tasks: detection, localization, recognition, and understanding. They define detection as determining whether a particular item is present in the image, localization as detecting an object and determining its location within the image space, recognition as the localization of all items in the image, and scene understanding as going beyond recognition to uncover the objects’ roles within the scene. In this brief section on background and theory, we develop a specific algorithm for object localization that will later be integrated into our reinforcement learning framework. Since we are solely concerned with localization, we take this opportunity to extend our arsenal of synonyms and use the terms “object detection”, “object localization”, and “object recognition” interchangeably to refer to localization as defined in [57].

In [58], Dickinson describes an object recognition system to include four main steps: feature extraction, feature grouping, object hypothesis, and object verification. A graph showing a traditional object recognition pipeline with these components is outlined in Figure 2.3.

Prior to the popularization of deep learning and convolutional neural networks, it was common to explicitly perform each step as shown in Figure 2.3 when performing object recognition. The first step, feature extraction, involves using the spatial and intensity components of the pixels to extract relevant information for solving the object recognition task. Features can be low level, such as local changes in pixel intensity corresponding to edges or corners, or they can be more sophisticated, such as a proposed mapping to a constrained class of 3D geometries [59][60][61]. Once extracted, groups of these features are used to query a database of annotated images and return potential object matches. These matches are then evaluated against additional features until the recognized objects are disambiguated.

Feature selection plays a large role in the aforementioned process. Simple features,

Figure 2.3: Components of a traditional object recognition system.
Source: [58]

while easy to compute, are often inefficient at encoding relevant image information, requiring large quantities and computational burden to detect and differentiate between object classes. Simple features also include less bias which may be needed for inference in the presence of partial observability. An extreme example of this would be using a feature vector consisting of all the raw pixel values in an image. This would make the feature extraction step a trivial identity mapping, but would require pixel-wise comparison between the input image and those in the database to generate hypotheses. This is both spatially and computationally expensive, although it can be effective under certain constrained circumstances. One of the earliest object

recognition techniques, template matching [62], is an example of such an approach.

On the other hand, complex and abstract features can provide more concise representations of semantic information, allowing for fast database look-up and computationally cheap object verification. However, extracting complex features can be a computationally expensive and error-prone process and it is not always clear which high-level features to use for a given detection task [58]. One idea to solve this ambiguity, and the subject for the remainder of this section, is to learn a high level feature encoding $f_\theta : \mathbb{R}^{N \times M \times C} \mapsto \mathbb{R}^K$, mapping an $N \times M \times C$ -dimensional digital image to a K -dimensional encoding. Using a technique like gradient-descent, we can optimize parameters θ such that f_θ outputs encodings that are robust and discriminatory for object recognition.

At this point it is useful to ponder what kind of encodings might be the most useful for the object recognition task. In an earlier example we considered trivial feature extraction which led to a template matching procedure. What if, instead of trivial feature extraction, we were able to do trivial database look-up? In other words, can we make the features themselves directly represent the object labels and locations in the image? This would eliminate the need for keeping an object database, and reduce the problem solely to one of feature extraction. It turns out that there is a class of functions that is able to learn such descriptive encodings: convolutional neural networks.

Ever since AlexNet [63] produced landmark results in the 2012 ImageNet Large Scale Visual Recognition Challenge [64], researchers have seen an explosion of success using convolutional neural networks for object classification and recognition. With convolutional neural networks it is possible to learn a function that encodes information from the traditional object recognition pipeline so that explicit database querying and hypothesis generation is no longer needed (Figure 2.4). Currently, there are hundreds of algorithms, or variations thereof, that utilize convolutional neural networks

for object recognition [65].

Figure 2.4: Convolutional neural networks simplify the object recognition pipeline. Modified image from [66]

Liu et al. [65] group convolutional architectures for object recognition into two categories: single stage or two stage detection. As the name implies, a two stage framework breaks the object recognition task into stages: one to determine the location of potential objects and another to determine their respective classification. One of the first notable works in this class is Regional Convolutional Neural Networks (R-CNN) [67]. R-CNN uses selective search [68] to generate class-agnostic region proposals and extracts features from the proposed regions using convolutional neural networks. Then each region is classified using class-specific linear support vector machines. At the time of publication, R-CNN was able to obtain state of the art results on the PASCAL VOC 2010 dataset [69], boasting a mean average precision (mAP) of over 53%.

Since R-CNN there have been countless advancements in two stage object detection, both in terms of speed and accuracy. The general trend has been to move towards simpler, more efficient “end-to-end” convolutional-based architectures for both region proposals and class assignment, eliminating the need for hand crafted features and costly search algorithms [70][71]. There have also been efforts to extend results to pixel-wise object segmentation, instead of traditional bounding boxes [72].

By contrast, single stage frameworks addresses both the region and class assignment tasks in a single forward pass through a convolutional neural network. In

practice, these algorithms tend to be faster than their two stage counterparts while also maintaining a simpler, more-unified structure [65]. Popular algorithms in this category are You Only Look Once (YOLO) (v1-v3) [73][74][75] and Single Shot Multi-Box Detector (SSD) [76]. In the next subsection we take a closer look into YOLO, since it plays a fundamental role in our research.

2.3.1 YOLO

YOLO is a single shot object detector leveraging convolutional neural networks. We will first outline the original formulation described in [73], and then note the further advancements used in YOLO v2, which is used in our viewpoint optimization algorithm. In the original paper, YOLO uses a relatively simple convolutional neural network with a novel training scheme to predict bounding boxes around objects in a given image. According to the authors, versions of the algorithm can process images at 45-155 frames per second on a Titan X GPU while learning a general and robust representation of objects. The details of the YOLO network’s structure and novel training scheme are outlined in the following discussion.

The YOLO algorithm processes an RGB image and outputs bounding box predictions for multiple class labels using only a single pass through the YOLO neural network. This is accomplished by training the network to output an array of values corresponding to the class probabilities, confidence scores, and bounding box locations at various grid locations in the input image and then pruning the output to obtain the final detections. This pipeline is shown in Figure 2.5.

The full network architecture of YOLO uses a modified version of the GoogLeNet architecture [77] (Figure 2.6). The input to the network is a 448x448x3 RGB image. For each grid location, the neural network predicts class-membership probabilities for 20 object classes and predicts two bounding boxes. For a 7x7 grid, this is represented as a 7x7x30 tensor. Each bounding box prediction is represented as a 5-tuple

To train the network, a multi-part loss function is optimized using stochastic gradient descent with momentum:

$$\mathcal{L}_{total} = \mathcal{L}_{coord} + \mathcal{L}_{conf} + \mathcal{L}_{class} \quad (2.24)$$

where

$$\mathcal{L}_{coord} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (2.25)$$

$$\mathcal{L}_{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \quad (2.26)$$

$$\mathcal{L}_{class} = \sum_{i=0}^{S^2} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (2.27)$$

The first term in the loss function, $\mathcal{L}_{coord}$, corresponds to bounding box location error. Since multiple bounding box predictions are made per grid cell location, only the bounding box with deemed “responsible” for the detection is included in the loss function. This allows the two bounding box predictors in each grid cell to be trained independently, with the hope that they will each specialize on learning different representations of the same object and improve localization performance. Additionally, indicator functions are used so that the bounding box loss is zero when there is no ground truth object present. The second loss term, $\mathcal{L}_{conf}$, corresponds to the confidence value error. Here, indicator functions are used to assign different loss values depending on whether a ground truth object was present at each bounding box location. The authors found that scaling the confidence losses for when there is an object present to 2x that of no object led to better model stability. The final loss term, $\mathcal{L}_{class}$ computes the squared error between the predicted classification values and ground truth classes at each grid location.

To determine which bounding box is responsible for detecting a given object, the YOLO algorithm uses the intersection-over-union (IOU) metric to give a normalized score corresponding to how “good” a prediction is. This metric equals the area of the

intersection between the true and proposed bounding box over the area of the union of the true and proposed bounding box. This is depicted in Figure 2.7. The bounding box predictor with the greatest IOU is assumed responsible for bounding box prediction, and the loss is updated accordingly. The IOU metric is again used during evaluation to compare accuracy with ground truth bounding box predictions. Typically, the model output is considered “correct” if the IOU between the ground truth value exceeds 0.5, although different values can be used depending on the application.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Figure 2.7: Visual depiction of the IOU similarity metric. Source: [78]

2.3.2 YOLOv2

In this research, we use the next iteration of YOLO single shot detectors, YOLO v2 [74], which introduces several improvements to the object detection pipeline. The improvements include a simplified convolutional architecture that can accommodate images of different sizes and a new way to specify bounding boxes, known as anchor boxes, using a stable parametric representation. Anchor box priors are selected using k-means on the training set with an IOU distance metric. Additionally, YOLOv2 leverages batch normalization [53] during training to stabilize the learning process. The resulting algorithm reports over 78% mAP on PASCAL VOC 2007 while running at 40 frames per second, a 15% mAP increase over YOLO at comparable speeds [74].

Chapter 3

ALGORITHM AND SYSTEM DESIGN

The goal of this chapter is to develop and propose a reinforcement learning algorithm to solve the strawberry viewpoint optimization problem. As is common in engineering, it is first necessary to define the problem setting in order to make justified design decisions. We begin by briefly constructing general guidelines for our harvesting environment, including plant configuration, types of hardware, and other pertinent details for modeling a Markov Decision Process. Then we develop a reinforcement learning algorithm to operate in the harvesting environment, keeping in mind our research objectives.

3.1 Environment

In this section we specify relevant details about the real-world harvesting environment for which the viewpoint optimization algorithm will be developed. Our primary objective is to make the problem setup as general as possible, so that our experimental results will be applicable to a large number of practical contexts. In this sense, we say that a system component is general if an analogous component, or one that further simplifies the harvesting objective, is prevalent across the strawberry harvesting industry. A secondary objective is to design the system such that it is easy to quantify and interpret the learned algorithm’s performance.

First we consider the cultivation environment in which the plants are grown. As mentioned in Chapter 2, we group harvesting environments into two categories: open field and greenhouse/tunnel. Note that while environment and growing configuration are largely dictated by geographical region, the open field historically constitutes the majority of global strawberry production [79]. The open field is also the most challenging to harvest as it provides the greatest variation in lighting and plant devel-

opment while offering limited visibility and accessibility of fruit [4]. Therefore, results obtained the open field environment will generalize across many practical contexts, and it is used as the setting for this research. A typical open field environment is shown in Figure 3.1.

Figure 3.1: Typical open field cultivation environment (reused from Chapter 2).

Next we consider the autonomous agent that will be interacting with the environment. Since autonomous strawberry harvesting is an active area of research, there are a wide variety of harvester geometries currently being explored. An assortment of these harvesters is shown in Figure 3.2.

Within the context of this research, there are two primary areas of the agent that need to be determined: the type of manipulator and the sensing unit for harvesting perception. The remaining components of the harvesting system, e.g. end-effector, method of locomotion, additional sensors, are outside of the scope of this research, although a potential configuration is suggested at the end of this section.

For the manipulator, we select a high degree-of-freedom articulated robot, such as the one shown in Figure 3.3. This allows the harvester to have a dense workspace,

Agrobot [80]

Harvest CROO [81]

Octinion [82]

Figure 3.2: Strawberry harvesting robot prototypes.

which can later be simplified by more constrained designs. Further, the similarities between the kinetic chain of an articulated robot and a human arm may highlight “human-like behaviors” that suggest proficiency at finding ripe strawberries.

To aid in the detection of these behaviors, we mount a monocular color camera to the robot’s end-effector as our sensing unit. An example of this is shown in Figure 3.3. Compared to other sensing modalities, such as Light Detection and Ranging (LIDAR)

Figure 3.3: UR5 articulated robot with Robotiq wrist camera [83].

or hyperspectral imaging, the signals obtained by a color camera more closely parallel those of the human visual system and lead to a natural interpretation by human observers. Additionally, the color camera gives the opportunity to highlight deep learning’s potential to process high-resolution pixel data and (implicitly) infer spatial relations and semantic membership in the 3D environment.

We envision a complete system consisting of a manipulator mounted to an unmanned ground vehicle, similar to Figure 3.4. This configuration is rugged and cost effective, and, barring any geometric constraints, it is ideal for autonomous or remote-controlled navigation in an open field environment.

3.2 Algorithm

We are now equipped to frame the viewpoint optimization problem as a Markov Decision Process (MDP) and outline our reinforcement learning algorithm. As indicated in Chapter 2, an MDP is defined by a 5-tuple specifying the state space, action space, state-transition probabilities, reward distribution, and discount factor. To avoid plant collisions while still addressing the crux of the viewpoint optimization problem, we constrain the end-effector to move on a hemisphere of a fixed radius above

Figure 3.4: Husky UGV manipulation package by Clearpath Robotics [84].

the strawberry plant as shown in figure 3.5. This allows us to define its position by two rotation angles $j = (\theta, \phi)$. The set of all reachable positions combined with the camera images defines our state space. We define our action space as incremental positions on this hemisphere $a = (\Delta\theta, \Delta\phi)$ and use off-the-shelf trajectory planning software and PID joint controllers to move between states. This allows the agent to focus on viewpoint optimization without having to discover control strategies for already-solved problems like gravity compensation and trajectory planning. State-transitions are defined by the physics of the environment, and we use the standard $\gamma = 0.99$ for our discount factor.

The remaining piece of the puzzle is how to specify the reward distribution. Intuitively, we would like to reward viewpoints that are “good” for autonomous harvesting. There are a myriad of factors that could potentially contribute to the utility of a viewpoint, an obvious one being whether ripe strawberries are visible. We use this observation to develop a novel reward scheme leveraging a pretrained object detector (YOLOv2) to provide positive feedback when ripe strawberries are in view. We also penalize invalid actions and add a light “existence penalty” to instill a notion of urgency. Denoting P_{max} as the maximum confidence value output by the ripe

Figure 3.5: Illustration of the state and action space.

strawberry detector, we define the reward at time t as:

$$R(s_t) = \begin{cases} R_{detect} & P_{max} \geq P_{thresh} \wedge (\theta_{t+1}, \phi_{t+1}) \in \mathcal{W}_{valid} \\ R_{invalid} & (\theta_{t+1}, \phi_{t+1}) \notin \mathcal{W}_{valid} \\ R_{exist} & otherwise \end{cases} \quad (3.1)$$

In our experiments, we use $R_{detect} = 1.0$, $R_{invalid} = -1.0$, and $R_{exist} = -0.1$. The reward scheme eliminates the need for human feedback during the reinforcement learning process but also introduces the burden of collecting and annotating images for pretraining an object detector to detect ripe strawberries. To combat this burden, we develop a labor-efficient data collection procedure (Algorithm 6) that leverages the fact that multiple training images can be annotated per plant given knowledge of its berries 3D poses. When collecting the dataset, we annotate both ripe and unripe

strawberries and then train the detector on both classes with the hopes that it will decrease the frequency of false positive detections.

Algorithm 6 Detector Data Collection

```

Initialize dataset  $\mathcal{D}$ 
for  $i = 0, 1, 2, \dots$  until  $num\_plants$  do
    Move to next plant
    Log location, size, and label of each strawberry. Denote as set  $\mathcal{P}$ 
    for  $k = 0, 1, 2, \dots$  until  $num\_views$  do
        Uniformly sample position  $j_k$  in workspace  $\mathcal{W}$ 
        Actuate camera to  $j_k$  and capture image  $o_k$ 
        Initialize set of bounding boxes  $\mathcal{B}$ 
        for each  $p \in \mathcal{P}$  do
            Map  $p$  to bounding box  $b$  using perspective projection
            if  $b$  is not occluded then ▷ e.g. by color threshold test
                 $\mathcal{B} \leftarrow \mathcal{B} \cup \{b\}$ 
         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{B}, o_k\}$ 
return  $\mathcal{D}$ 

```

The data collection procedure rests on two key observations: a single strawberry plant can be used to generate multiple training images, and, with knowledge of the location and size of each strawberry in 3D-space, bounding boxes can be derived on 2D images using perspective projection. Combining these observations, training data can efficiently be collected via the following steps. For each strawberry plant, an operator logs the location, size, and label of each strawberry by kinesthetically guiding the end-effector to each position. Then, the arm moves to ≈ 20 randomly sampled locations to capture images of the strawberry plant. Finally, training images are autonomously annotated using perspective projection and occlusion heuristics. This significantly reduces the amount of effort and time required to obtain training data compared to the traditional practice of first collecting training images and then manually labeling them. A fraction of this dataset is then used to train YOLOv2 to detect ripe and unripe strawberries, with the remaining images held out for validation and testing.

We use DDPG to train the harvesting agent for viewpoint optimization, as its off-

policy nature allows for sample efficient training updates. Sample efficiency is critical due to the large amount of time required for policy execution. Additionally, DDPG has seen success using convolutional neural networks to process raw pixel inputs [36]. The training procedure is shown in Algorithm 7. It can be executed autonomously provided that the harvesting robot has means navigate from plant to plant during the training process.

Algorithm 7 DDPG for Viewpoint Optimization

```

Initialize experience replay  $\mathbb{D}$  with size  $N$ 
Initialize  $Q_\theta$  with random weights
Initialize  $\pi_\phi$  with random weights
Initialize target network  $Q_{\theta'}$  with weights  $\theta' = \theta$ 
Initialize target network  $\pi_{\phi'}$  with weights  $\phi' = \phi$ 
for episode = 1, M do
    Randomly generate new strawberry plant and reset end-effector
    Initialize random process  $\mathcal{N}$  for action exploration
    Set timestep  $t = 1$ 
    Receive initial state  $s_1 = (o_1, j_1)$ 
    while  $t < T$  do
        In parallel:
            Policy execution:
                Take action  $a_t = (\Delta\theta, \Delta\phi) = \pi_\phi(s_t) + \mathcal{N}$ 
                Receive state  $s_{t+1} = (o_{t+1}, j_{t+1})$ 
                Compute  $r_t = f(s_{t+1})$  according to Equation 3.1
                Store  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathbb{D}$ 
                 $t = t + 1$ 
            Training:
                Sample mini-batch  $(s_i, a_i, r_i, s_{i+1})$  from  $\mathbb{D}$  uniformly
                 $y_i = r_i + \gamma Q_\theta(s_{i+1}, \pi_\phi(s_i))$ 
                Update main networks:
                     $\theta \leftarrow \theta - \epsilon_\theta \sum_i \nabla_\theta ||y_i - Q_{\theta'}(s_i, a_i)||^2$ 
                     $\phi \leftarrow \phi + \epsilon_\phi \sum_i \nabla_a Q_{\theta'}(s_i, \pi(s_i)) \nabla_\phi \pi_{\phi'}$ 
                Update target networks:
                     $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ 
                     $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$ 
    return  $\pi_\phi$ 

```

Chapter 4

SYSTEM IMPLEMENTATION

4.1 Simulated Environment

A simulated environment is created using Robot Operating System (ROS) [85] and Gazebo [86] with the goal of being sufficiently realistic so that results in the simulated environment would be indicative of performance in real-world settings. The virtual world mimics an open strawberry field and consists of a section of a bed with dirt surroundings and a randomly-generated strawberry model. Within the world, we place a “floating arm” harvester centered about the strawberry plant. This arm is controlled through a python interface and serves as the physical agent in our experiments. The setup is shown in Figure 4.1.

Figure 4.1: Custom simulated environment displayed in Gazebo.

The specific robotic arm used in our simulation is the Kinova Jaco 6DOF [87]. It is selected because it is has appropriate size, degrees of freedom, and accuracy for

our problem statement, and it is reasonably priced for a research-quality manipulator ($\approx \$35,000$). Additionally, and perhaps more importantly, Kinova provides a convenient ROS interface [88] upon which we build our simulated environment. A list of primary requirements is shown in Table 4.1.

Table 4.1: Manipulator requirements

Requirement	Kinova Jaco 6DOF
Degrees of freedom ≥ 6	✓
Payload $> 1\text{kg}$	✓
Price $< \$50,000$	✓
Reach $> 50\text{cm}$	✓
Upright mounting	✓
Waterproof	“Weather resistant”

We model the simulated camera based on a consumer-level car backup camera [89] and virtually mount it to the end-effector of our harvesting robot. It is selected because it is low-cost, lightweight, and waterproof. To use it in our simulation, we use Gazebo’s camera plugin and publish the pixel information to a ROS topic. We can then visualize the camera in real-time using RQT [90] and access raw data through ROS’s python interface. The visualization is shown in Figure 4.2.

Low-level control of the arm and camera assembly is performed using the `ros_control` suite with Moveit [91] for actions requiring trajectory planning, e.g. returning to the home position. For the majority of the experiments, the end-effector is constrained to a hemisphere a fixed radius above the strawberry plant. We precompute joint angles for 32,400 evenly spaced positions on the hemisphere, using Moveit to solve inverse kinematics. We store these values in a lookup table, which is later queried for trajectory planning using piecewise constant interpolation. This allows for robust

Figure 4.2: Harvester’s eye view using RQT.

actions on the hemisphere using joint angles that have already been tested, avoiding costly planning errors that can occur when computing inverse kinematics on the fly.

We develop a python interface to communicate between the ROS/Gazebo simulation and the other python-based components of our project. This interface, dubbed **AgentROS**, provides an API modeled after that of OpenAI Gym [92]. The user can then use familiar commands like `env.reset()` and `env.step()` to interact with the environment in a standardized way.

4.1.1 Plant Model

Plant models are generated using using Gazebo’s native SDF file format with embedded Ruby [93]. The algorithm creates plants based on a simple structural model in which various parameters, such as berry pose, mesh, and ripeness, are sampled to create unique plant configurations on the fly. It is outlined in Algorithm 8, and key parameters are listed in Table 4.2.

Algorithm 8 Plant Generation

```
Sample # ripe, unripe, and green berries
Sample # leaf stems
for each berry do
    Sample berry mesh
    Sample texture corresponding to ripeness level within ripeness class
    Sample berry scale according to ripeness level
    Sample calyx mesh
    while valid pose not selected do
        Sample stem length
        Sample stem pitch and yaw
        Set stem roll = 0
        Set stem model based on pitch
        Set berry yaw = stem yaw
        Set berry roll = 0
        Calculate berry position  $(x, y, z)$ 
        if  $z < \text{berry radius}$  then
            Set stem pitch such that  $z = \text{radius}$ 
            Set berry pitch horizontal
        else if  $z < 2(\text{berry radius})$  then
            Set berry pitch such that it makes contact with ground
        else
            Set berry pitch vertical
        if no berry collisions then
            Break
for each leaf stem do
    Sample stem length
    Sample stem pitch and yaw
    Set stem roll = 0
    Set stem model based on stem pitch
    Calculate leaf position  $(x, y, z)$ 
    for each of 3 leaves do
        Sample leaf yaw
        Sample leaf mesh
        Set leaf roll and pitch = 0
    Sample leaves' scale
```

Note that the plant creation algorithm is based solely on observation and does not contain biological considerations that may be found in more sophisticated models [94]. Despite this, our generated plant models appear complex and sufficiently realistic to conduct our experiments. Textured meshes for each plant component are shown in

Table 4.2: Plant generation parameters

Name	Distribution
ripe berries	$\mathcal{N}(\mu = 4, \sigma = 1)$
unripe berries	$\mathcal{N}(\mu = 3, \sigma = 1)$
green berries	$\mathcal{N}(\mu = 3, \sigma = 1)$
leaf stems	$\mathcal{N}(\mu = 35, \sigma = 5)$

Appendix B. All of the meshes are custom made, with the exception of the strawberry fruit, which uses down-sampled meshes from the UC Davis Strawberry Database [95]. A side-by-side comparison of a generated plant bed with a real image is shown in Figure 4.3. Note that training occurs on a single strawberry plant, not on a populated bed as depicted.

Figure 4.3: A generated strawberry bed versus an actual one.

4.2 Detector

An annotated dataset of over 7000 images is collected in the simulated environment using the procedure outlined in Algorithm 6. At each iteration, we generate

a new plant model and move to 20 uniformly sampled positions to collect camera images. For each image, we project the ground truth berry locations (modeled as spheres) to the image plane to create labeled bounding boxes. Bounding boxes are considered valid if they contained over 50 pixels within a class-specific RGB range, indicating that a portion of the berry is visible. During the data collection procedure it is necessary to ensure the manipulator has sufficient time to settle before projecting the berry locations to the image plane. In our early attempts, this results in inaccurate bounding boxes that are not suitable for training. An example of this is shown in Figure 4.4. A properly labeled image is shown in Figure 4.5.

Figure 4.4: Poorly annotated training image due to camera oscillations.

We split the dataset 80/20 and trained YOLOv2 to detect ripe and unripe strawberries. For this, we use the Darknet framework [96] and train for 50,000 epochs using default hyperparameters as specified in [74]. The trained detector is able to perform real-time detections ($> 30\text{fps}$) on the simulated camera feed using a Tesla K80 GPU. A screenshot is shown in Figure 4.6.

Since we perform no hyperparameter tuning, there is no need to hold out images for validation. We use the remaining 20% of images as our testing set to evaluate

Figure 4.5: Properly annotated training image.

Figure 4.6: The detector comfortably annotates images at over 30fps.

localization errors and construct a precision-recall curve. The results are outlined in Chapter 5. Once tested, the detector is integrated into **AgentROS** to provide feedback during reinforcement learning.

4.3 Reinforcement Learning

We implement DDPG in python using TensorFlow [97] as our deep learning framework. For both the actor and critic networks we use five $3 \times 3 \times 32$ convolutional layers with stride 2 to process the $800 \times 800 \times 3$ raw pixel input. In the actor network this is followed by two 200-neuron fully connected layers with hyperbolic tangent activation functions. The critic network has a similar structure, except the first fully connected layer is concatenated with the action input and the final (scalar) output is a linear activation. The networks use the same uniform initialization scheme and weight regularization as in [36]. We include batch normalization on all layers, although in hindsight this may not have been the best choice (see Chapter 6). Training is performed in batch sizes of 16 and updates are performed with Adam optimizer [98] using learning rates of 1×10^{-4} and 1×10^{-3} for the actor and critic, respectively, and default values $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\hat{\epsilon} = 1 \times 10^{-8}$ from the paper. The target networks are updated using Polyak Averaging [55] with mixing parameter $\tau = 1 \times 10^{-3}$.

The DDPG algorithm, detector network, and simulated environment must interact harmoniously during the training process (Figure 4.7). When running the policy, DDPG interfaces with **AgentROS** to take actions in the simulated environment. After each action is performed, **AgentROS** grabs the present joint encoder readings and converts the end-effector position to location (θ, ϕ) on the hemisphere. The agent also grabs the camera image and runs it through the detector network to obtain bounding box predictions. The predictions and position are then processed to determine the scalar reward at the present state. The new state and reward information is sent back from **AgentROS** to DDPG, and the episode continues until a termination signal is received.

The training process is computationally demanding, as it must run the simulation and perform forward and backward passes through several deep neural networks.

Figure 4.7: Process diagram showing interaction between system nodes.

To handle these computational demands, we implement the system using Google Compute Engine (GCE) with a Tesla K80 GPU. GCE does not provide a graphical user interface, so we use VirtualGL with TurboVNC and NoVNC [99][100] to create an interactive desktop in the browser (Figure 4.8). We use Docker [101] to package all needed modules into a container that can be deployed from an Ubuntu server.

Using this setup, the DDPG agent was trained for approximately one week, during which time it completed roughly 10,000 episodes over 500,000 total timesteps. Each episode we log (undiscounted) returns, average max Q values, and total steps. We plot these values using TensorFlow’s TensorBoard visualization tool to get feedback on the learning process (Figures 4.9-4.11). Training is terminated when the average returns appear to stagnate or regress. For testing, we use the policy network weights that yield maximum returns (indicated by a star in Figure 4.9).

Figure 4.8: Browser-based desktop environment for running experiments.

From the figures, we see that the critic network learns Q values much faster than the actor learns a proficient policy. This is by design (recall the different learning rates) to increase stability for the policy gradient updates. Another interesting observation is that the amount of steps per episode initially declines until about 2000 episodes before beginning to increase. We believe this phenomenon is in part due to the differing convergence rates between the actor and the critic. It is further investigated in Chapter 5.

We also used TensorBoard to plot the distribution of network activations over the course of each episode. Perhaps the most illuminating of these plots were the output activations for the actor and critic networks (Figures 4.12-4.13). We see that the actor quickly learns a maximal control scheme, which is likely the result of not including an effort penalty while also encouraging fast exploration. The critic’s activations are less dramatic, but they show a trend of decreased spread to within reasonable

bounds. This behavior agrees with the actor and critic approaching convergence over the course of training.

Figure 4.9: Returns over 10,000 training episodes. Smoothed using $\gamma = 0.9$.

Figure 4.10: Average max Q values over 10,000 training episodes. Smoothed using $\gamma = 0.9$.

Figure 4.11: Total steps over 10,000 training episodes. Smoothed using $\gamma = 0.9$.

Figure 4.12: Relative frequency of output activation magnitudes for the actor network over 10,000 training episodes.

Figure 4.13: Relative frequency of output activation magnitudes for the critic network over 10,000 training episodes.

Chapter 5

TESTING AND EVALUATION

5.1 Detector

In Chapter 1, we laid out the research objective to assess reinforcement learning’s promise for viewpoint optimization and set the stage for the development of a practical reinforcement learning algorithm. As described in Chapter 3, our algorithm leverages a pretrained strawberry detector to generate reward signals at each state, making the detector an integral part of the autonomous training process. In the following analyses, we take an in-depth look at the pretrained strawberry detector and evaluate its performance on key metrics for reinforcement learning.

We begin by looking at basic performance data of the trained detector on a held-out testing set, such as bounding box errors and average IOU, and note any biases or irregularities that could potentially be exploited by the reinforcement learning agent during training. Next, we investigate the relationship between the precision and recall of the trained detector by evaluating its performance with various confidence thresholds. This gives insight on how well the confidence values correspond to actual detection performance, and gives a chance to revisit our choice of reward threshold. We conclude with a visual analysis to look at the “intangibles” that are hard to observe from the raw data alone, such as whether the detector’s performance is consistent, robust, and shares any similarities with human intuition.

5.1.1 Performance on Test Set

We tested the learned strawberry detector on 2000 held-out test images and compared detections to ground truth values. We logged the distribution of intersection over union (IOU), error in x, y, w, and h for each bounding box (here, predicted and ground truth bounding boxes are considered to be for the same berry if they have an

IOU greater than 0.5). A sample annotated test image is shown in Figure 5.1. The plots for these statistics on the testing set are shown in Figures 5.2-5.5. Additionally, we assessed the detector’s precision and recall across all thresholds in increments of 0.05. The resulting precision-recall curve is shown in Figure 5.6.

Figure 5.1: Sample test image showing ground truth (dark) and predicted (light) bounding boxes.

Figure 5.2: Histogram of x-coordinate errors on testing set.

Figure 5.3: Histogram of y-coordinate errors on testing set.

Figure 5.4: Histogram of width errors on testing set.

The distribution of the IOU plots and errors in the four bounding box statistics appear to be symmetrically distributed with nearly zero mean and similar standard deviations. Intuitively, this means the detector is performing “nicely” with consistent performance across the many viewpoints of the testing set. The width and height errors had a slightly larger spread than the localization errors. We suspect this is due

Figure 5.5: Histogram of height errors on testing set.

Figure 5.6: Precision/recall curve for the strawberry detector. Thresholds listed at each data point.

to increased difficulty uncovering bounding boxes dimensions derived from a spherical approximation of the berry shape.

The Precision-Recall curve shows that there is a gradual linear decrease in precision with a large increase in recall as the threshold is lowered from 0.75 to 0.5. Lowering the threshold further results in a sharp decrease in precision and asymp-

otic recall to 0.5. In reinforcement learning, it is important to have a strong reward signal so that the agent can understand the consequences of its actions. As such, we err on the side of high precision/low recall and select $R_{thresh} = 0.6$ as our reward threshold, corresponding to a precision of 0.9 and a recall of less than 0.2. While such a low recall may initially raise some red flags, it is important to note that its implications largely depend on the nature of the strawberries that are ignored. For this, we turn to a visual analysis.

5.1.2 Visual Analysis

Many of the desired attributes of the static detector are hard to quantify analytically. In this section, we look at real-time detections from the trained detector on the camera’s feed and see if the the observed behavior aligns with our intuition. Although less “concrete”, this proved to be one of the most useful evaluations because it allowed use to get inside the mind of the detector and assess the whether we believe the detector to be suitable for use in reinforcement learning. Several screen-shots from the visual evaluation are shown in Figures 5.7-5.9.

Figure 5.7: Screen-shot of the strawberry detector using the real-time camera feed (1).

Figure 5.8: Screen-shot of the strawberry detector using the real-time camera feed (2).

Figure 5.9: Screen-shot of the strawberry detector using the real-time camera feed (3).

Looking at the behavior of the detector frame-by-frame, we note its performance is relatively consistent across the camera images. It appears to prefer strawberries that are closer to the camera and larger, with more red flesh of the strawberry showing corresponding to higher confidence values. The strawberries that are missed by the

detector are typically smaller or heavily occluded. Therefore, as a feedback mechanism for viewpoint optimization, the biases exhibited by the detector are likely beneficial.

5.2 Reinforcement Learning

In this section, we quantify the performance of the trained reinforcement learning algorithm in the simulated environment, seeking to gain insight on its utility for viewpoint optimization. We begin by comparing the learned policy’s performance versus several handcrafted baseline policies. Then we perform a visual analysis to gain insight on specific characteristics of the learned behavior.

5.2.1 Baseline Comparison

To assess the performance of the learned policy, we compared metrics such as episode return and length versus five baseline policies. These policies are listed below in order of increasing complexity.

1. Random policy, π_1 : At each time step, the agent takes a uniformly sampled action in $\Delta\theta$ and $\Delta\phi$.
2. Random policy with boundary awareness, π_2 : At each time step, the agent takes a uniformly sampled action in $\Delta\theta$ and $\Delta\phi$, taking opposite actions to stay in bounds as needed.
3. Downward heuristic with boundary awareness, π_3 : Given threshold $\phi^* \in (0, \frac{\pi}{2})$. At each time step, the policy selects a downward action along the hemisphere until it reaches threshold ϕ^* . Once it crosses the threshold, the policy takes random actions in accordance with π_2 .
4. Frozen detector with downward heuristic and boundary awareness, π_4 : At each time step, the policy first runs the pretrained strawberry detector on observation o_t , and obtains coordinates (x_t, y_t) of the most confident ripe detection in the image frame. If a ripe strawberry is detected, the policy outputs zero action.

Otherwise, the policy moves in accordance with π_3 .

5. Proportional detector with downward heuristic and boundary awareness, π_5 :

At each time step, the policy first runs the pretrained strawberry detector on observation o_t , and obtains coordinates (x_t, y_t) of the most confident ripe detection in the image frame. If a ripe strawberry is detected, the policy outputs action proportional to the direction of its bounding box in the image plane.

Otherwise, the policy moves in accordance with π_3 .

Note that for the detector-based policies we used a bounding box threshold of 0.5 to represent a positive detection, instead of 0.6 used to derive the reward values. This prevented the baseline detectors from having “insider knowledge” of the reward scheme and makes the resulting comparison less biased.

5.2.2 Policy Evaluation

We ran each of the policies for 100 episodes, recording the reward at each timestep and number of steps per episode. Using this information, we calculated the expected return for each episode, average reward per timestep, average number of timesteps until first reward (excluding trials without a reward), and average episode length. The resulting histograms are shown in Figures 5.10-5.13.

We observe that the trained detector performed significantly better with respect to the reinforcement learning objective than the “naive” baseline policies, and performed on-par with the best of the “intelligent” baseline policies. Given that the intelligent baseline policies had access to a pretrained strawberry detector during testing (unlike the learned policy), these results are encouraging for reinforcement learning’s promise for ripe strawberry detection. The results are mixed with respect to the other metrics.

One interesting result was that, despite having a high average episode return, the average reward per step of the learned detector was the lowest among all the baseline policies. Recall the reinforcement learning objective is to maximize expected return

Figure 5.10: Average return per episode for each policy.

Figure 5.11: Average reward per step for each policy.

per episode, irrespective of the number of time steps. Looking at the episode lengths, we see that the learned detector had many episodes of extremely short length, which generated a very low return value due to the early termination penalty for going out of bounds. As a result, the average reward per timestep in these episodes was very low compared to the average reward per timestep of a longer, higher return episode,

Figure 5.12: Average steps until first reward, excluding episodes without a reward.

Figure 5.13: Average episode length for each policy.

since there were less steps to average over.

Another metric considered was the average timesteps until a positive reward. Although not explicitly solved for in the reinforcement learning objective, the learned detector performed favorably in this metric with respect to all the baseline policies. This suggests that minimizing time to first detection is a useful strategy for maxi-

mizing our reward objective, which aligns with intuition. Under the current reward function, an agent can obtain maximum return by rapidly moving to a position where a ripe strawberry is in view, and then staying fixated on that location for the remainder of the episode.

The comparison between reward metrics highlights the importance of selecting a reward function that represents the true design objective. Fortunately for this experiment, we had some play with the design objective since the primary concern was to assess reinforcement learning’s promise in the automated harvesting domain, instead of accomplishing a specific task. In Chapter 6, we explore modifications to the reinforcement learning objective that may provide greater utility.

As mentioned prior, the learned detector had a large quantity of episodes with very short length. Given the steep penalty for going out of bounds, these episodes all had negative return ranging from -1.1 to -1.5 and contribute to the low average return per timestep. Out of curiosity, we sought to assess the performance of the learned detector if we excluded the extremely short episodes. The filtered results are plotted in Figures 5.14-5.18.

Figure 5.14: Number of episodes with length < 4 .

Figure 5.15: Average return per episode, ignoring episodes with length < 4 .

Figure 5.16: Average reward per step, ignoring episodes with length less < 4 .

By ignoring extremely short episodes, we see that our trained agent performed significantly better in every metric. Notably, the average reward per timestep is now positive, and the average episode return is now on par with the most proficient “intelligent” policies.

Figure 5.17: Average steps until first reward, excluding episodes without a reward or length < 4 .

Figure 5.18: Average episode length, ignoring episodes with length < 4 .

5.2.3 Hybrid Policy

In the preceding section, we noted that an improvement could be seen in the learned policy by ignoring episodes where the agent performs early termination behaviors. This begs the question that if such behaviors were somehow hard-coded into the learned policy, if there would be a notable increase in performance. To test

this hypothesis in the simulated environment, we created a “hybrid” policy that augmented the learned behaviors with boundary awareness and a downward heuristic. This policy is outlined below.

1. Hybrid policy: Given threshold $\phi^* \in (0, \frac{\pi}{2})$. At each time step, the policy selects a downward action along the hemisphere until it reaches threshold ϕ^* . Once it crosses the threshold, the agent follows the learned policy (DDPG), taking opposite actions to stay in-bounds as needed.

A comparison the hybrid policy with the previous policies over 100 episodes is shown in Figures 5.19- 5.22.

Figure 5.19: Average return per episode, including hybrid policy.

Adding the hard-coded heuristics, the hybrid policy outperformed the top “intelligent” baselines. While this brings further promise to reinforcement learning’s applicability to the detection process, it is important to note that some of these heuristics, such as the initial downward actions, exploit simplifications in structure of the virtual strawberry plant that may differ from the real-world environment. For example, strawberries and leaves are generated uniformly within a fixed range of angles which generally favor low altitude vantage points. Although this results in

Figure 5.20: Average reward per step, including hybrid policy.

Figure 5.21: Average episode length, including hybrid policy.

reasonable-looking virtual plants, it does not account for all real-world plant configurations. Thus the effects of the hard-coded heuristics may be less pronounced in a practical setting.

Figure 5.22: Number of episodes with length < 4 , including hybrid policy.

5.2.4 Early Termination Phenomenon

In this section we revisit the learned policy’s early termination phenomenon and seek to explain the underlying reason(s) why it occurs. In general, understanding the reason behind the policy’s behaviors can allow us to improve the system design and quantify how well it is performing in the environment. We begin our analysis by recalling the agent’s learning curves from the training process.

In the early stages of training, we noted in Figures 4.9 and 4.10 that the critic’s Q values converged much faster than the actual returns obtained by the actor’s policy. From these plots, we suspect that the critic was “better” at critiquing than the actor was at acting during these initial stages. In this case, there would be many states where the critic would predict a low expected reward because it perceived the environment to be too challenging for the actor. A potential instance for this to occur is at the beginning of the episode, where the camera typically sees heavy foliage, and detection of ripe strawberries often requires a long sequence of coordinated actions.

Given that our current problem setup spawns the agent a mere three steps from an out-of-bounds configuration, it often will incur a higher reward by immediately

terminating the episode than aimlessly wandering without a detection (recall the reward scheme in Equation 3.1). In the extreme case, the agent gets a total reward of -1.2 from immediate termination, and a total reward of -10.0 for no detections for the duration of the episode. Therefore, we hypothesize that the reason for the early termination phenomenon is that it is mathematically favorable to end an episode early than continue exploring a heavily leaved plant under a poor policy.

To (informally) test this hypothesis, we looked at the learned agent’s behavior under five levels of plant foliage density and recorded the number of early terminations episodes. A positive correlation between the two variables would support our suspicions. Five sample plant models from these experiments are shown in Figure 5.23. We ran 100 episodes of the trained detector on each level of foliage variation and compared the number of early terminations and average episode lengths. The results are shown in Figures 5.24 and 5.25.

Figure 5.23: Range of foliage densities from $\mu_{leaves} = 20$ to $\mu_{leaves} = 50$.

With the original early termination criteria of episode length < 4 , there is no

Figure 5.24: Number of early terminations for different levels of foliage.

Figure 5.25: Comparison of episode lengths for different levels of foliage.

obvious pattern between canopy density and early termination frequency. However, when we increase the minimum episode length to 10, a positive correlation emerges between the two variables. Looking at the full range of episode lengths, it appears that the dense canopies result in many more episodes of shorter length, but do not

necessarily translate into immediate termination behavior. This observation is supported by comparing the average episode lengths over 100 trials for each canopy density (Figure 5.25). As a whole, the results suggest that there is a relationship between canopy density and the agent’s tendency to go out of bounds, but there are likely other phenomena at play to explain the full trajectory distribution.

5.2.5 Fixation Analysis

After determining a high-reward vantage point for detecting ripe strawberries, it is optimal for an agent to remain fixated at that location for the remainder of the episode. Fixation behaviors can also be detrimental if they occur on a vantage point with low return. In this section, we seek to characterize the learned agent’s fixation tendencies in an attempt to better understand the inner workings of its policy.

To detect instances of fixation, we ran the DDPG policy on over 400 plants and tracked positions and rewards for each episode. We then plotted the corresponding trajectories on a hemisphere superimposed above the plant models, denoting detection states with a blue star. Several of these visualizations are shown in Figures 5.26-5.28, with additional visualizations in Appendix C. We then manually inspected the plots and noted trajectories where the agent exhibited fixation behaviors.

Of the 213 trajectories with more than 50 steps, 82 fixated on high-return regions, 64 fixated on low-return regions, and 67 did not appear to fixate. From this, it appears the agent learned the desired fixation behavior, but the policy lacked robustness to extend to all states. We suspect some of the low-return clusters are a pitfall of partial observability: Conflicting interpretations of plant geometry at adjacent viewpoints could result in opposite actions for exploration, even if no ripe strawberries are currently visible.

The visualizations also bring to light the agent’s location biases. For the majority of the episodes, the agent spends the entirety of its life constrained to a limited

Figure 5.26: Visualizations where the agent fixates on high-reward view-points.

Figure 5.27: Visualizations where the agent fixates on low-reward view-points.

proximity of its spawn location. On instances where the agent did venture to other regions, we noted that the agent was less likely to fixate, even if ripe strawberries were in frame. We believe this is the result of insufficient experience on these regions of the hemisphere, which results in a weak learning signal during training. In future works, this could be remedied by spawning at different starting locations or by using an improved exploration heuristic.

Figure 5.28: Visualizations where the agent does not appear to fixate.

Next, we investigated the nature of the fixated viewpoints. Using saved plant models from experimental trials, we moved the agent to known fixation locations and manually inspected the camera images. On regions with high return, we observed the agent tended to favor viewpoints with ripe strawberries in close proximity with minimal occlusions (Figure 5.29 and 5.30). As mentioned in Chapter 2, such viewpoints are not only advantageous for detection but likely benefit the remaining steps of the harvesting process, providing better angles for pose prediction and reducing obstacles to simplify planning and execution of harvesting trajectories.

On the other regions, the image contents were more varied, but we noticed that several of the viewpoints displayed ripe, or nearly-ripe, strawberries in frame that were not being picked up by the pretrained detector (Figure 5.31). In these instances, it is possible that the DDPG policy over-estimated the returns from these viewpoints and thus gravitated towards them. Other times, the agent did not appear to fixate on anything notable at all (Figure 5.32). However, as a whole, it appears that the agent learned to exploit the strengths of the pretrained strawberry detector and generally navigated towards regions where it had a high probability of detection. This is particularly impressive considering the known limitations of partial observability.

camera view

Figure 5.29: First-person view from high-return fixation location (1).

camera view

Figure 5.30: First-person view from high-return fixation location (2).

camera view

Figure 5.31: First-person view from low-return fixation location (1).

camera view

Figure 5.32: First-person view from low-return fixation location (2).

SUMMARY AND CONCLUSIONS

6.1 Contributions

In this research, we formulated a novel application of reinforcement learning to solve the viewpoint optimization problem for autonomous strawberry harvesting. We showed that feedback from a pretrained strawberry detector could be used as an autonomous reward scheme, eliminating the need for a human in the loop during the training process. In doing so, we developed a labor efficient data collection procedure for capturing and annotating strawberry images to train the strawberry detector. To train and test our algorithms, we created a realistic simulated environment incorporating many harvesting challenges found in real-world contexts. The simulated environment, along with source code for the python interface, will be made public at the URLs shown in Appendix A.

In Chapter 1, we posed three research questions of increasing generality. First, we wanted to determine how well a reinforcement learning solution for viewpoint optimization could perform in the context of a simulated harvesting environment with respect to the reinforcement learning objective. Our reinforcement learning algorithm was proficient in this regard: In our experiments, we saw that the trained agent was able to achieve episodic returns on par with sophisticated baseline policies, and its performance could be significantly improved by hard-coding a few simple heuristics.

Next, we wanted to assess the reinforcement learning algorithm’s utility in real-world harvesting contexts. In this assessment, we not only need to consider how well results in the simulated environment would transfer to real-world settings, but also what additional metrics may be important besides the reinforcement learning objective. We note that the simulated environment, while not perfect, contains many

real-world challenges, such as apparent randomness, frequent occlusions, complex textures, shadows, and lighting variation. In Section 5, we considered pertinent metrics like time-to-detection and expected return, and observed that the learned agent is generally competent in these areas. Further, we noted that the nature of the agent’s fixated viewpoints not only aids ripe strawberry detection, but also provides favorable angles for pose determination, trajectory planning, and trajectory execution. For these reasons, we are optimistic that the viewpoint optimization algorithm would have practical merit in real-world settings.

Finally, we wanted to know if we could gain insight on reinforcement learning’s potential for the full autonomous harvesting process. In our experiments, we witnessed a successful application of reinforcement learning to process high-dimensional sensory input and optimize with respect to an abstract objective specified by a complex reward function. The generality and success of this approach suggests that the reinforcement learning framework may provide a viable solution to other components of the harvesting process. In Section 6.3, we propose several means to extend the scope of our algorithm to increase its utility for harvesting applications.

6.2 Limitations

Many of the limitations faced in this research were due to financial obstacles. For a proper evaluation of the reinforcement learning algorithm we would have liked to run several separate training iterations with different random seeds, as research shows that there is often a high variability between training runs due to initial configuration [102]. However, using Google Compute Engine to run the trials using Tesla K80 GPUs, we did not have sufficient funding to perform these tests. Instead, we ran a single trial with the final configuration and evaluated its performance. With additional resources we could run many of these ideas in parallel, taking advantage of Google’s scalable infrastructure to obtain rapid results. Fortunately, we were able to obtain

positive results with this approach, which could either mean that we were lucky with our performance, or that the algorithm is robust to different random initialization. Without further testing, it is hard to know the answer to this question.

Also due to limited resources, we did not do much experimentation with tuning of hyperparameters, such as the number of layers in our neural networks, learning rates, and exploration parameters. Because of this, it is likely that we could have found improved performance with another configuration, as we picked values solely on an intuitive basis and from previous results in literature without verification. Certain choices, such as the use of batch normalization, were based on ambiguous implementation details that may have been misunderstood in hindsight, leading to inferior performance. Beyond standard hyperparameters, we also would have liked to experiment with small tweaks to the environment or reward scheme, such as increasing weight of the out of bounds penalty and experimenting with different complexity of strawberry plants. Several of these ideas are developed in Section 6.3.

Another limitation of this work was that it was solely implemented in a simulated environment. Although efforts were taken to make the simulated environment sufficiently complex and representative of a real-world strawberry harvesting setup, it is inherently different than real-world conditions and thus it is still unknown the algorithm’s performance on a real system. So, while a simulated environment is convenient and enables a rapid development cycle, further testing is needed on a physical system to solidify our research conclusions.

6.3 Future Works

The research presented in this paper indicates that reinforcement learning is a promising method to improve the autonomous strawberry harvesting pipeline and serves as a branching-off point for many future developments. In this section, we briefly outline several of these directions for future work.

6.3.1 Batch Normalization

We used batch normalization [53] in an attempt to improve the distribution of layer weights during training. Research suggests that batch normalization works best with batches representative of the training domain and can become unstable with small minibatch sizes [103]. In future works, we hypothesize that our training would improve without batch normalization using the current batch size of 16. Another option would be to increase the batch size to ≈ 64 to stabilize the running mean and variance parameters, although this may pose challenges due to memory constraints.

6.3.2 Noise

We used an Ornstein-Uhlenbach process [54] to add exploration noise to the agent’s policy during training. One consideration when doing this is whether to reset the noise values between episodes, since the noise has a “momentum” term that takes into account previous noise states. In this research, we reset the noise before each training episodes, however we think that it would be good to explore not resetting between trials. By resetting the noise, the agent on average will have lower noise values at the beginning of the episode, until the noise magnitudes increase and the noise gains momentum. This inconsistent behavior could limit exploration in the early states of an episode, and may be detrimental to performance. In future works, we recommend omitting the reset step to see if this improves performance.

6.3.3 Parallelization

Our training procedure consisted of two independent processes: a worker collecting training data and an asynchronous update procedure using samples from the experience replay. In future work, this could be extended to include multiple workers for the data collection process with minimal modifications to the underlying algorithm. In [104], parallelization was used to significantly speed up deep reinforcement learn-

ing for a real-world manipulation task. Such a modification is a natural extension to autonomous harvesting since most existing systems already use multiple workers during the harvesting process.

6.3.4 Memory Unit

In Chapter 3, we defined the agent’s “state” to be its raw camera image and its end-effector position. This presents problems in a heavily occluded environments due to partial observability at each timestep, which may lead to sub-optimal actions. If the agent had some mechanism to propagate relevant state information through time, then it would be able to overcome this limitation. We propose several improvements for future work. First, the state itself could be modified to include additional information from previous state(s). In the simplest case, this could consist of appending the previous position to the current state vector, while a more extreme case is frame stacking. These strategies are limited by their finite horizon and linear memory complexity. A more sophisticated approach would be to provide a framework for the agent to learn which information to propagate through time. Possible frameworks for this include recurrent neural networks [24][105][106] or an external memory unit [107][108].

6.3.5 Environment

With the goal of viewpoint optimization in open field harvesting contexts, we have freedom to select many design variables (Chapter 3). After developing and testing our reinforcement learning algorithm, we suggest several modifications of to the underlying Markov Decision Process that may improve performance. First, we recommend experimenting with different action spaces for the manipulator. A small change would be allowing the agent to go “over the top” of the hemisphere instead of penalizing out of bounds. This should discourage the early termination behavior described in Chapter 5. A more drastic modifications would be relaxing the hemispherical con-

straint and enabling the agent to move freely about its workspace. This would allow the end-effector to bring the camera to better viewpoints, but introduces the new concern of incurring plant damage. Another viable modification would be changing the episodic termination criteria. In our research, we saw that the current framework encouraged the agent to move quickly to an optimal viewpoint and then fixate for the remainder of the episode. In practice, this fixation behavior is not necessary as long as the agent can communicate that it has found a good viewpoint. In future works, we recommend experimenting with termination criteria that eliminate the need for fixation and more directly address practical concerns.

A key component of our viewpoint optimization algorithm was the use of a pre-trained strawberry detector to determine the utility of the camera observations. For the most part, this reward scheme aligned with our goal of viewpoint optimization, i.e. maximizing rewards generally corresponds to finding a good viewpoint for viewing ripe strawberries. However, there are a few edge cases where the reinforcement learning algorithm could potentially “outsmart” the reward scheme. For instance, there are certain viewpoints where, if given a full view of the strawberry, the detector would predict that it is unripe due to light coloration towards the top. However, if the top of the strawberry is not visible, the detector may indicate that the strawberry is ripe. To exploit this, the agent could learn to predominantly focus on the tips of strawberries to avoid lower returns from additional information. Although we did not directly observe this behavior, it is important to keep in mind ways that the reward scheme could be exploited to derive an unintended policy.

6.3.6 Scope

In Chapter 1, we defined the overarching goal of this research to gain insight on reinforcement learning’s potential for autonomous harvesting. In addressing this goal, we narrowed the scope of the research to the task of viewpoint optimization, which

we identified as a key component in the autonomous harvesting process. Given the success of our approach, a natural progression for future works is to extend the application of reinforcement learning to other aspects of the harvesting process, including pose determination, path planning and execution, and fruit removal. For future works, we recommend exploring techniques like hierarchical or curriculum learning [109][110] to develop complex policies for the full harvesting pipeline.

We reiterate that the most pertinent change of scope is to go beyond the simulated environment and evaluate our algorithms on a physical system. While the simulated environment is convenient and enables a rapid development cycle, without physical testing we can only speculate on reinforcement learning’s utility in real-world harvesting applications. By comparing results between the two domains, we also will be able to improve the simulated environment and better understand its limitations for future research.

BIBLIOGRAPHY

- [1] Vern Grubinger. History of the strawberry, Jun 2012. Online; Accessed October 2018.
- [2] Geoffrey Mohan. As california’s labor shortage grows, farmers race to replace workers with robots, Jul 2017. Online; Accessed October 2018.
- [3] Sicma agriculture machinery. <https://www.sicmasrl.com/en>. Online; Accessed October 2018.
- [4] Cornelis Wouter Bac, E.J. Van Henten, Jochen Hemming, and Yael Edan. Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. *Journal of Field Robotics*, 31, 07 2014.
- [5] Dan Charles. Robots are trying to pick strawberries. so far, they’re not very good at it. <https://www.npr.org>, Mar 2018. Online; Accessed October 2018.
- [6] Megajuce. Reinforcement learning diagram, 2017. Online; Accessed 3 February 2019.
- [7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017.
- [8] S.G. Defterli, Yeyin Shi, Yunjun Xu, and Reza Ehsani. Review of robotic technology for strawberry production. 32:301–318, 01 2016.
- [9] J.E. Holt, D. Schoorl, and I.F. Muirhead. Post-harvest quality control strategies for fruit and vegetables. *Agricultural Systems*, 10(1):21 – 37, 1983.

- [10] California Strawberry Commission. California strawberry farming.
<http://www.calstrawberry.com>, Jan 2018. Online; Accessed October 2018.
- [11] Mark P. Bolda, Laura Tourte, Jeremy Murdock, and Daniel A. Sumner. 2016 sample costs to produce and harvest strawberries.
<https://coststudyfiles.ucdavis.edu>, Jan 2017. Online; Accessed October 2018.
- [12] Megan Crivelli. Agricultural labor in the united states: Current situation threats. <https://www.theproducenerd.com>, Oct 2017. Online; Accessed October 2018.
- [13] Naoshi Kondo, Kazunori Ninomiya, Shigehiko Hayashi, Tomohiko Ota, and Kotaro Kubota. A new challenge of robot for harvesting strawberry grown on table top culture. 01 2005.
- [14] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.
- [15] Tien Thanh Nguyen, Erdal Kayacan, Josse De Baedemaeker, and Wouter Saeys. Task and motion planning for apple harvesting robot*. *IFAC Proceedings Volumes*, 46(18):247 – 252, 2013. 4th IFAC Conference on Modelling and Control in Agriculture, Horticulture and Post Harvest Industry.
- [16] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.
- [17] S.S. Mehta and T.F. Burks. Vision-based control of robotic manipulator for citrus harvesting. *Computers and Electronics in Agriculture*, 102:146 – 158, 2014.

- [18] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.
- [19] Andreas Kamilaris and Francesc X. Prenafeta-Bold. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70 – 90, 2018.
- [20] Maryam Rahnemoonfar and Clay Sheppard. Deep count: Fruit counting based on deep simulated learning. *Sensors (Basel, Switzerland)*, 17, 04 2017.
- [21] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.
- [22] Mads Dyrmann, Anders Krogh Mortensen, Henrik Skov Midtiby, and Rasmus Nyholm Jørgensen. Pixel-wise classification of weeds and crops in images by using a fully convolutional neural network. 2016. Paper No. 608, CIGR 2016; null ; Conference date: 26-06-2016 Through 29-06-2016.
- [23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [24] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.
- [25] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

- [26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [27] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *CoRR*, abs/1702.03920, 2017.
- [28] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. *CoRR*, abs/1602.02867, 2016.
- [29] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.
- [30] Kailash Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil A. Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.
- [31] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.
- [32] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.
- [33] waldoalvarez. Markov decision process, 2017. Online; Accessed 3 February 2019.
- [34] R. Serfozo. *Basics of Applied Stochastic Processes*. Probability and Its Applications. Springer Berlin Heidelberg, 2009.
- [35] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

- [36] 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.
- [37] Josh Schulman. Q-learning function methods, cs294.
<http://rll.berkeley.edu/deeprlcoursesp17/docs/lec3.pdf>, 2017.
- [38] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181, 1922.
- [39] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.
- [40] Alex Karpathy. Gridworld: Dynamic programming demo. https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html. Online; Accessed December 2018.
- [41] Eamonn Keogh and Abdullah Mueen. *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA, 2017.
- [42] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.
- [43] Leemon C. Baird, III. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning, ICML’95*, pages 30–37, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

- [44] Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [45] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.
- [46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.
- [47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [48] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.
- [49] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *NIPS*, 2007.

- [50] Sergey Levine. Advanced q-learning, slide: 28, cs294.
<http://rail.eecs.berkeley.edu/deeprlcourse-fa17>, 2018.
- [51] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine.
Continuous deep q-learning with model-based acceleration. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML’16, pages 2829–2838. JMLR.org, 2016.
- [52] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.
- [53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [54] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Phys. Rev.*, 36:823–841, Sep 1930.
- [55] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July 1992.
- [56] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra.
Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134, 1998.
- [57] Alexander Andreopoulos and John Tsotsos. 50 years of object recognition: Directions forward. *Computer Vision and Image Understanding*, 117:827891, 08 2013.

- [58] Sven J Dickinson. Object representation and recognition. *What is cognitive science*, 7:172–207, 1999.
- [59] Gerald J Agin. Representation and description of curved objects. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1972.
- [60] Alex P. Pentland. Perceptual organization and the representation of natural form. *Artif. Intell.*, 28(3):293–331, May 1986.
- [61] Irving Biederman. Human image understanding: Recent research and a theory. *Computer Vision, Graphics, and Image Processing*, 32(1):29 – 73, 1985.
- [62] Azriel Rosenfeld. Picture processing by computer. *ACM Comput. Surv.*, 1(3):147–176, September 1969.
- [63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.
- [64] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.
- [65] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *CoRR*, abs/1809.02165, 2018.
- [66] Aphex34. Typical cnn architecture, 2015. Online; Accessed 3 February 2019.
- [67] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

- [68] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.
- [69] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge, 2010.
- [70] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [71] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.
- [72] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.
- [73] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [74] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 1612, 2016.
- [75] Joseph Redmon and Ali Farhadi. Yolo3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.
- [76] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

- [77] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
- [78] Adrian Rosebrock. A visual equation for intersection over union (jaccard index)., 2016. Online; Accessed 1 February 2019.
- [79] Fumiomi Takeda. Strawberry production in soilless culture systems. *Acta Horticulturae*, 481:289–296, 01 1999.
- [80] Agrobot. <http://www.agrobot.com/>. Online; Accessed 23 January 2019.
- [81] Harvest croo. <https://harvestcroo.com>. Online; Accessed 23 January 2019.
- [82] Octinion. octinion.com.
- [83] Universal robot ur5.
<https://www.universal-robots.com/products/ur5-robot/>.
- [84] Husky ugv. <https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/>.
- [85] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [86] Nathan P Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. Citeseer.
- [87] Kinova website.
<https://www.kinovarobotics.com/en/products/assistive-technologies/kinova-jaco-assistive-robotic-arm>.

- [88] Kinova github repository.
<https://github.com/Kinovarobotics/kinova-ros>, Dec 2018.
- [89] Yanees website. www.yanees.net.
- [90] Rqt ros wiki. <http://wiki.ros.org/rqt>.
- [91] Ioan A. Sucas and Sachin Chitta. Moveit! <http://moveit.ros.org>. Online.
- [92] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.
- [93] Erb - ruby templating.
<http://ruby-doc.org/stdlib-2.6.1/libdoc/erb/rdoc/ERB.html>.
- [94] Oliver Deussen and Bernd Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [95] Adrien Durand-Petiteville, Dennis Sadowski, and Stavros Vougioukas. A strawberry database: Geometric properties, images and 3d scans, uc davis dash, dataset. 2018.
- [96] Joseph Redmon. Darknet: Open source neural networks in c.
<http://pjreddie.com/darknet/>, 2013–2016.
- [97] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit

- Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [98] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [99] A brief introduction to virtualgl. virtualgl.org. Online; Accessed 26 January 2019.
- [100] novnc: Open source vnc client using html5. <https://novnc.com>. Online; Accessed 26 January 2019.
- [101] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.
- [102] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *CoRR*, abs/1806.08295, 2018.
- [103] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *CoRR*, abs/1702.03275, 2017.
- [104] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.
- [105] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver.

- Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.
- [106] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, *abs/1507.06527*, 2015.
- [107] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.
- [108] Marvin Zhang, Zoe McCarthy, Chelsea Finn, Sergey Levine, and Pieter Abbeel. Learning deep neural network policies with continuous memory states. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 520–527. IEEE, 2016.
- [109] Mostafa Al-Emran. Hierarchical reinforcement learning: A survey. *International Journal of Computing and Digital Systems*, 4:2210–142, 04 2015.
- [110] Sanmit Narvekar. Curriculum learning in reinforcement learning: (doctoral consortium). In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS ’16, pages 1528–1529, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems.
- [111] K.J strm. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174 – 205, 1965.

APPENDICES

Appendix A

LINKS

Project repositories:

<https://github.com/jsather/harvester-docker> (Docker container)

<https://github.com/jsather/harvester-python> (python interface)

<https://github.com/jsather/harvester-sim> (simulated environment)

Video summary:

<https://youtu.be/C6hrCVv2B-o>

Appendix B

LIST OF MESHES IN PLANT MODEL

Figure B.1: Stem models for different leaf heights.

Figure B.2: Stem models for different berry orientations.

Figure B.3: Leaf models.

Figure B.4: Calyx models.

Figure B.5: Berry models.

Figure B.6: Texture mappings for seven degrees of ripeness.

Appendix C

ADDITIONAL TRAJECTORY VISUALIZATIONS

High Return Fixations

Low Return Fixations

No Fixations

Appendix D

RAW DATA

Detector Performance - Thresh = 0.5 (100/2000 images)

iou	dx	dy	dh	dw
0.642486	3	5	-3	-1
0.61316	6	-5	-7	-6
0.616048	0	5	5	5
0.753902	-1	4	2	1
0.715582	-4	4	-6	-8
0.592937	0	7	6	5
0.54993	6	3	-1	3.55E-15
0.735811	-1	1	7	4
0.584958	4	-2	-5	-4
0.669291	4	-4	2	4
0.660523	4	8	2	-6
0.695125	4	4	3	-2
0.706789	0	-5	-1	4
0.824718	2	1	-1	-5
0.686928	0	-4	-5	-4
0.509885	2	-5	-15	-17
0.532766	-3	-10	11	8
0.670988	3	6	-5	-6
0.811224	-3	1	-3	-8
0.546473	-4	-11	-15	-16
0.788396	-3	2	2	3
0.646516	-4	-6	-3	-4
0.524954	-13	-1	6	1
0.763649	-1	5	0	-1
0.65545	-5	6	2	-3
0.502033	-7	5	-3	-4
0.706723	2	0	6	5
0.613268	3	11	-2	-2
0.587244	-6	4	4	5
0.557851	-2	-8	4	7
0.544041	5	6	2	4
0.747863	-3	-2	4	7
0.596448	-5	-6	7	12
0.614074	11	-4	7	10
0.573478	4	6	-3	-3
0.594561	6	6	0	4
0.742262	-5	3	-2	-4
0.75132	2	2	7	9
0.730864	-2	1	-8	-5
0.648525	3	6	6	10
0.640152	0	3	7	6
0.646518	-5	-1	5	7
0.599305	3	-4	3	2
0.513766	6	1	6	9
0.603113	4	-1	-1	1

0.543307	5	2	7	8
0.625475	1	13	6	7.11E-15
0.687831	-3	-8	7.11E-15	-6
0.591801	-3	-4	3	5
0.611067	-5	4	5	5
0.646865	1	-4	-4	-6
0.597444	-10	4	-3	-5
0.744931	-4	7	-4	-8
0.544276	6	-6	-4	-5
0.713476	-5	-3	1	-1
0.5629	3	5	-4	-4
0.530612	4	2	-10	-9
0.765924	-4	-1	3	4
0.654378	0	-4	7	9
0.702111	-4	1	-4	-3
0.641469	1	7	0	-4
0.710496	0	1	-9	-7
0.72208	3	4	5	3
0.781632	0	-4	9	12
0.716454	-4	2	3	2
0.693222	3	-7	-6	-12
0.814707	-3	-3	-7.11E-15	5
0.714697	2	-4	0	-2
0.680798	-6	0	-4	-4
0.660044	-9	1	0	2
0.713086	-3	1	-7	-7
0.81011	0	-4	1	-1
0.676334	-1	6	-16	-12
0.581227	-6	-1	-10	-9
0.736765	-5	-1	-3	-4
0.555647	2	11	4	2
0.676247	7	5	-13	-14
0.56044	-5	7	4	2
0.532135	6	5	-4	-3
0.795839	-1	-4	-6	-4
0.581389	0	-13	-11	-3
0.811258	4	3	2	-9
0.655329	-1	-1	4	4
0.526998	6	2	12	9
0.558282	-2	4	5	6
0.737397	-2	-1	1	0
0.527925	-3	4	5	5
0.708081	7	1	2	-3
0.5264	6	6	-5	-7
0.596939	-3	2	10	13
0.824799	-5.68E-14	3	2	8
0.679817	-4	2	-1	-2

0.765496	1	1	-5	-6
0.606695	-5	8	12	18
0.516685	-4	18	6	2
0.88317	-2	1	2	2
0.835941	-3	-1	0	0
0.68	-2	5	3	4
0.720568	-11	4	-4	-5
0.847079	3	1	5	-1

Detector Performance - Thresh = 0.5 (100/2000 images)

iou	dx	dy	dh	dw
0.642486	3	5	-3	-1
0.61316	6	-5	-7	-6
0.616048	0	5	5	5
0.753902	-1	4	2	1
0.715582	-4	4	-6	-8
0.592937	0	7	6	5
0.54993	6	3	-1	3.55E-15
0.735811	-1	1	7	4
0.584958	4	-2	-5	-4
0.669291	4	-4	2	4
0.660523	4	8	2	-6
0.695125	4	4	3	-2
0.706789	0	-5	-1	4
0.824718	2	1	-1	-5
0.686928	0	-4	-5	-4
0.509885	2	-5	-15	-17
0.532766	-3	-10	11	8
0.670988	3	6	-5	-6
0.811224	-3	1	-3	-8
0.546473	-4	-11	-15	-16
0.788396	-3	2	2	3
0.646516	-4	-6	-3	-4
0.524954	-13	-1	6	1
0.763649	-1	5	0	-1
0.65545	-5	6	2	-3
0.502033	-7	5	-3	-4
0.706723	2	0	6	5
0.613268	3	11	-2	-2
0.587244	-6	4	4	5
0.557851	-2	-8	4	7
0.544041	5	6	2	4
0.747863	-3	-2	4	7
0.596448	-5	-6	7	12
0.614074	11	-4	7	10
0.573478	4	6	-3	-3
0.594561	6	6	0	4
0.742262	-5	3	-2	-4
0.75132	2	2	7	9
0.730864	-2	1	-8	-5
0.648525	3	6	6	10
0.640152	0	3	7	6
0.646518	-5	-1	5	7
0.599305	3	-4	3	2
0.513766	6	1	6	9
0.603113	4	-1	-1	1

0.543307	5	2	7	8
0.625475	1	13	6	7.11E-15
0.687831	-3	-8	7.11E-15	-6
0.591801	-3	-4	3	5
0.611067	-5	4	5	5
0.646865	1	-4	-4	-6
0.597444	-10	4	-3	-5
0.744931	-4	7	-4	-8
0.544276	6	-6	-4	-5
0.713476	-5	-3	1	-1
0.5629	3	5	-4	-4
0.530612	4	2	-10	-9
0.765924	-4	-1	3	4
0.654378	0	-4	7	9
0.702111	-4	1	-4	-3
0.641469	1	7	0	-4
0.710496	0	1	-9	-7
0.72208	3	4	5	3
0.781632	0	-4	9	12
0.716454	-4	2	3	2
0.693222	3	-7	-6	-12
0.814707	-3	-3	-7.11E-15	5
0.714697	2	-4	0	-2
0.680798	-6	0	-4	-4
0.660044	-9	1	0	2
0.713086	-3	1	-7	-7
0.81011	0	-4	1	-1
0.676334	-1	6	-16	-12
0.581227	-6	-1	-10	-9
0.736765	-5	-1	-3	-4
0.555647	2	11	4	2
0.676247	7	5	-13	-14
0.56044	-5	7	4	2
0.532135	6	5	-4	-3
0.795839	-1	-4	-6	-4
0.581389	0	-13	-11	-3
0.811258	4	3	2	-9
0.655329	-1	-1	4	4
0.526998	6	2	12	9
0.558282	-2	4	5	6
0.737397	-2	-1	1	0
0.527925	-3	4	5	5
0.708081	7	1	2	-3
0.5264	6	6	-5	-7
0.596939	-3	2	10	13
0.824799	-5.68E-14	3	2	8
0.679817	-4	2	-1	-2

0.765496	1	1	-5	-6
0.606695	-5	8	12	18
0.516685	-4	18	6	2
0.88317	-2	1	2	2
0.835941	-3	-1	0	0
0.68	-2	5	3	4
0.720568	-11	4	-4	-5
0.847079	3	1	5	-1

Baseline Test - π_1

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	46	-0.12	-1	-5.5	-0.1
-1	36	0.00	25	-0.1	1
-1	53	-0.12	-1	-6.2	-0.1
-1	55	-0.12	-1	-6.4	-0.1
-1	48	-0.10	44	-4.6	1
-1	36	-0.13	-1	-4.5	-0.1
-1	37	0.02	25	0.9	1
-1	65	-0.06	55	-4.1	1
-1	57	-0.08	33	-4.4	1
-1	58	0.32	1	18.6	1
-1	48	0.04	1	2	1
-1	30	0.24	4	7.1	1
-1	33	0.11	1	3.5	1
-1	53	-0.01	32	-0.7	1
-1	41	0.09	33	3.8	1
-1	37	-0.01	33	-0.2	1
-1	28	0.10	2	2.9	1
-1	29	0.02	22	0.6	1
-1	58	0.17	16	9.8	1
-1	34	-0.13	-1	-4.3	-0.1
-1	45	0.27	7	12.2	1
-1	15	-0.16	-1	-2.4	-0.1
-1	35	0.13	1	4.4	1
-1	45	0.15	34	6.7	1
-1	9	-0.20	-1	-1.8	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	40	0.48	1	19.3	1
-1	30	-0.13	-1	-3.9	-0.1
-1	66	-0.08	58	-5.3	1
-1	36	-0.13	-1	-4.5	-0.1
-1	41	-0.04	38	-1.7	1
-1	31	0.01	3	0.4	1
-1	8	-0.21	-1	-1.7	-0.1
-1	33	0.11	1	3.5	1
-1	12	-0.18	-1	-2.1	-0.1
-1	12	-0.18	-1	-2.1	-0.1
-1	47	0.14	21	6.5	1
-1	53	0.32	1	16.9	1
-1	39	-0.09	2	-3.7	1
-1	42	-0.12	-1	-5.1	-0.1
-1	41	-0.12	-1	-5	-0.1
-1	54	-0.12	-1	-6.3	-0.1
-1	38	-0.12	-1	-4.7	-0.1
-1	55	-0.04	40	-2	1

-1	47	-0.12	-1	-5.6	-0.1
-1	7	0.40	1	2.8	1
-1	9	-0.20	-1	-1.8	-0.1
-1	27	-0.13	-1	-3.6	-0.1
-1	37	-0.06	35	-2.4	1
-1	48	-0.12	-1	-5.7	-0.1
-1	28	-0.13	-1	-3.7	-0.1
-1	50	0.32	1	16.1	1
-1	51	-0.12	-1	-6	-0.1
-1	31	0.30	1	9.2	1
-1	41	0.17	25	7.1	1
-1	48	0.00	42	-0.2	1
-1	54	0.01	42	0.3	1
-1	31	0.69	1	21.3	1
-1	46	-0.05	6	-2.2	1
-1	50	0.43	24	21.6	1
-1	32	0.46	1	14.6	1
-1	32	0.46	9	14.6	1
-1	35	-0.13	-1	-4.4	-0.1
-1	33	0.01	19	0.2	1
-1	49	-0.12	-1	-5.8	-0.1
-1	33	-0.13	-1	-4.2	-0.1
-1	8	-0.21	-1	-1.7	-0.1
-1	5	-0.28	-1	-1.4	-0.1
-1	8	-0.21	-1	-1.7	-0.1
-1	44	-0.12	-1	-5.3	-0.1
-1	47	0.11	1	5.4	1
-1	14	-0.16	-1	-2.3	-0.1
-1	40	-0.10	39	-3.8	1
-1	36	-0.13	-1	-4.5	-0.1
-1	33	-0.09	8	-3.1	1
-1	48	-0.12	-1	-5.7	-0.1
-1	66	0.49	1	32.1	1
-1	44	-0.12	-1	-5.3	-0.1
-1	40	0.15	29	6.1	1
-1	48	-0.12	-1	-5.7	-0.1
-1	35	0.94	1	33	1
-1	38	-0.01	9	-0.3	1
-1	34	-0.13	-1	-4.3	-0.1
-1	52	-0.10	18	-5	1
-1	49	-0.03	6	-1.4	1
-1	71	-0.11	-1	-8	-0.1
-1	52	0.05	43	2.7	1
-1	54	-0.12	-1	-6.3	-0.1
-1	43	-0.12	-1	-5.2	-0.1
-1	37	-0.12	-1	-4.6	-0.1
-1	34	-0.13	-1	-4.3	-0.1

-1	43	0.47	1	20.1	1
-1	39	-0.12	-1	-4.8	-0.1
-1	44	-0.07	29	-3.1	1
-1	35	0.00	28	-9.99201E-16	1
-1	54	0.15	17	8	1
-1	46	-0.12	-1	-5.5	-0.1
-1	50	-0.12	-1	-5.9	-0.1
-1	48	-0.10	47	-4.6	1
-1	64	0.45	18	29	1

Baseline Test - π_2

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	31	0.23	2	7	1
-1	50	-0.12	-1	-5.9	-0.1
-1	36	0.00	18	-0.1	1
-1	53	-0.12	-1	-6.2	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	49	-0.10	44	-4.7	1
-1	51	-0.12	-1	-6	-0.1
-1	74	-0.11	-1	-8.3	-0.1
-1	68	-0.11	-1	-7.7	-0.1
-1	31	0.23	5	7	1
-1	56	-0.08	6	-4.3	1
-1	55	0.22	22	12.3	1
-1	64	0.11	32	7	1
-1	34	-0.06	1	-2.1	1
-1	46	-0.12	-1	-5.5	-0.1
-1	50	0.19	25	9.5	1
-1	41	0.12	32	4.9	1
-1	48	-0.03	3	-1.3	1
-1	43	-0.12	-1	-5.2	-0.1
-1	35	-0.13	-1	-4.4	-0.1
-1	56	-0.12	-1	-6.5	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	48	-0.12	-1	-5.7	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	52	0.24	23	12.6	1
-1	44	0.28	5	12.3	1
-1	68	0.00	3	-7.77156E-16	1
-1	39	-0.01	23	-0.4	1
-1	55	0.24	35	13.4	1
-1	37	0.74	1	27.3	1
-1	42	0.59	12	24.6	1
-1	43	-0.12	-1	-5.2	-0.1
-1	28	0.61	1	17.2	1
-1	69	-0.11	-1	-7.8	-0.1
-1	51	-0.12	-1	-6	-0.1
-1	37	-0.12	-1	-4.6	-0.1
-1	30	0.38	9	11.5	1
-1	41	0.09	26	3.8	1
-0.1	100	-0.10	-1	-10	-0.1
-1	33	0.61	1	20	1
-1	72	-0.07	69	-4.8	1
-1	79	-0.10	46	-7.7	1
-1	48	-0.12	-1	-5.7	-0.1

-1	56	-0.12	-1	-6.5	-0.1
-1	54	-0.12	-1	-6.3	-0.1
-1	42	0.66	7	27.9	1
-1	73	-0.05	1	-3.8	1
-1	46	0.19	25	8.8	1
-1	35	0.41	16	14.3	1
-1	62	0.35	16	21.5	1
-1	52	0.18	12	9.3	1
-1	51	-0.12	-1	-6	-0.1
-1	37	0.29	21	10.8	1
-1	36	0.03	2	1	1
-1	78	0.00	57	0.1	1
-1	58	0.13	2	7.6	1
-1	66	-0.11	-1	-7.5	-0.1
-1	55	-0.10	54	-5.3	1
-1	51	-0.05	2	-2.7	1
-1	50	0.65	1	32.6	1
-1	58	0.00	51	-0.1	1
-1	54	0.09	21	4.7	1
-1	80	0.01	8	1	1
-1	68	0.03	58	2.2	1
-1	51	-0.10	11	-4.9	1
-1	47	0.44	17	20.8	1
-1	40	-0.07	36	-2.7	1
-1	31	0.62	4	19.1	1
-1	44	-0.05	35	-2	1
-1	48	0.78	3	37.2	1
-1	31	-0.13	-1	-4	-0.1
-1	61	-0.11	-1	-7	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	30	-0.13	-1	-3.9	-0.1
-1	50	-0.05	40	-2.6	1
-1	73	-0.11	-1	-8.2	-0.1
-1	57	-0.12	-1	-6.6	-0.1
-1	42	0.66	8	27.9	1
-1	40	-0.12	-1	-4.9	-0.1
-1	58	-0.12	-1	-6.7	-0.1
-1	49	-0.12	-1	-5.8	-0.1
-1	57	-0.12	-1	-6.6	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	45	0.83	1	37.5	1
-1	44	-0.12	-1	-5.3	-0.1
-1	36	0.36	1	13.1	1
-1	45	-0.12	-1	-5.4	-0.1
-1	55	-0.12	-1	-6.4	-0.1
-1	39	-0.12	-1	-4.8	-0.1
-1	57	0.04	48	2.2	1

-1	43	0.88	1	37.7	1
-1	40	0.79	1	31.4	1
-1	29	-0.13	-1	-3.8	-0.1
-1	48	0.02	41	0.9	1
-1	36	0.09	3	3.2	1
-1	38	0.48	2	18.4	1
-1	43	-0.12	-1	-5.2	-0.1
-1	45	0.05	33	2.3	1
-1	49	0.89	1	43.7	1

Baseline Test - π_3

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	62	0.75	1	46.8	1
-1	39	0.95	1	37	1
-1	42	-0.12	-1	-5.1	-0.1
-1	51	0.81	1	41.3	1
-1	53	-0.05	28	-2.9	1
-1	56	0.81	1	45.2	1
-1	39	0.02	14	0.7	1
-1	35	0.88	1	30.8	1
-1	76	-0.11	-1	-8.5	-0.1
-1	55	0.10	1	5.7	1
-1	63	-0.11	-1	-7.2	-0.1
-1	78	-0.04	4	-3.2	1
-1	91	-0.01	80	-1.2	1
-1	61	0.77	3	46.9	1
-1	33	0.07	20	2.4	1
-1	45	-0.12	-1	-5.4	-0.1
-1	48	0.13	36	6.4	1
-1	56	0.30	33	16.6	1
-1	73	-0.08	70	-6	1
-1	55	-0.04	40	-2	1
-1	76	0.28	45	21.2	1
-1	40	0.02	4	0.6	1
-1	37	0.29	1	10.8	1
-1	61	0.35	4	21.6	1
-1	45	0.30	28	13.3	1
-1	62	0.28	24	17.1	1
-1	71	-0.07	62	-4.7	1
-1	40	-0.07	36	-2.7	1
-1	64	0.04	3	2.6	1
-1	36	-0.06	29	-2.3	1
-1	35	-0.13	-1	-4.4	-0.1
-1	48	-0.12	-1	-5.7	-0.1
-1	25	-0.09	5	-2.3	1
-1	38	0.17	28	6.3	1
-1	46	-0.12	-1	-5.5	-0.1
-1	38	0.14	21	5.2	1
-1	42	-0.02	15	-0.7	1
-1	59	0.09	15	5.3	1
-1	46	-0.12	-1	-5.5	-0.1
-1	63	-0.10	8	-6.1	1
-1	71	0.49	17	34.9	1
-0.1	100	-0.10	-1	-10	-0.1
-1	37	0.05	12	2	1
-1	39	0.24	23	9.5	1

-1	45	0.00	11	0.1	1
-1	77	-0.10	57	-7.5	1
-1	52	0.05	3	2.7	1
-1	33	0.67	6	22.2	1
-1	52	-0.12	-1	-6.1	-0.1
-1	49	-0.05	21	-2.5	1
-1	41	0.68	1	28	1
-0.1	100	-0.10	-1	-10	-0.1
-1	57	-0.10	9	-5.5	1
-1	70	0.26	1	18.5	1
-1	40	0.04	10	1.7	1
-1	67	0.25	19	16.6	1
-1	51	-0.12	-1	-6	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	41	-0.07	39	-2.8	1
-1	75	-0.11	-1	-8.4	-0.1
-1	50	-0.12	-1	-5.9	-0.1
-1	40	0.04	34	1.7	1
-1	61	-0.11	-1	-7	-0.1
-1	41	-0.12	-1	-5	-0.1
-1	41	-0.12	-1	-5	-0.1
-1	62	0.06	48	3.9	1
-1	33	0.24	2	7.9	1
-1	55	0.90	1	49.7	1
-1	33	0.27	1	9	1
-1	48	-0.12	-1	-5.7	-0.1
-1	57	-0.12	-1	-6.6	-0.1
-1	42	-0.12	-1	-5.1	-0.1
-1	53	-0.12	-1	-6.2	-0.1
-1	53	-0.12	-1	-6.2	-0.1
-1	38	-0.12	-1	-4.7	-0.1
-1	43	0.90	1	38.8	1
-1	53	0.46	20	24.6	1
-1	83	-0.11	-1	-9.2	-0.1
-1	73	-0.08	56	-6	1
-1	42	-0.10	38	-4	1
-1	57	-0.10	3	-5.5	1
-1	77	0.10	8	7.9	1
-1	42	-0.10	41	-4	1
-1	32	-0.13	-1	-4.1	-0.1
-1	47	0.61	1	28.5	1
-1	49	-0.03	22	-1.4	1
-1	56	-0.12	-1	-6.5	-0.1
-1	51	-0.12	-1	-6	-0.1
-1	69	-0.11	-1	-7.8	-0.1
-1	57	-0.10	56	-5.5	1
-1	32	-0.13	-1	-4.1	-0.1

-1	62	0.13	2	8.3	1
-1	43	0.24	29	10.2	1
-1	29	0.44	3	12.7	1
-1	50	0.30	31	15	1
-1	34	-0.13	-1	-4.3	-0.1
-1	76	-0.11	-1	-8.5	-0.1
-1	49	0.13	1	6.3	1
-1	70	0.12	46	8.6	1
-1	59	0.24	34	14.1	1

Baseline Test - π_4

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	78	0.71	11	55.1	1
-1	33	0.81	4	26.6	1
1	100	0.82	11	82.4	1
-1	49	0.42	6	20.6	1
-1	34	0.04	22	1.2	1
1	100	0.68	30	68.1	1
-1	36	-0.13	-1	-4.5	-0.1
-1	36	-0.13	-1	-4.5	-0.1
-1	63	-0.08	51	-5	1
-1	34	0.23	1	7.8	1
-1	39	-0.12	-1	-4.8	-0.1
-1	39	0.16	19	6.2	1
-1	78	-0.11	-1	-8.7	-0.1
1	100	0.96	5	95.6	1
-1	42	-0.12	-1	-5.1	-0.1
-1	53	-0.12	-1	-6.2	-0.1
-1	37	-0.12	-1	-4.6	-0.1
-1	51	-0.12	-1	-6	-0.1
-1	57	-0.12	-1	-6.6	-0.1
-1	75	-0.11	-1	-8.4	-0.1
-1	42	-0.12	-1	-5.1	-0.1
-1	58	-0.12	-1	-6.7	-0.1
-1	43	-0.12	-1	-5.2	-0.1
-1	53	0.01	7	0.4	1
-0.1	100	-0.06	24	-5.6	1
-1	45	0.49	4	22.1	1
-1	49	0.31	5	15.1	1
-1	35	-0.13	-1	-4.4	-0.1
-1	46	0.43	3	19.8	1
-1	34	0.20	1	6.7	1
-1	31	-0.09	27	-2.9	1
-1	48	-0.12	-1	-5.7	-0.1
-1	32	-0.06	28	-1.9	1
-1	58	-0.12	-1	-6.7	-0.1
-1	43	0.01	27	0.3	1
-1	49	0.62	3	30.5	1
-1	43	-0.12	-1	-5.2	-0.1
-1	37	0.02	9	0.9	1
-1	39	-0.01	35	-0.4	1
-1	36	-0.09	34	-3.4	1
-1	48	-0.12	-1	-5.7	-0.1
-1	47	-0.10	46	-4.5	1
-1	57	-0.12	-1	-6.6	-0.1
-1	54	-0.12	-1	-6.3	-0.1

-1	37	-0.12	-1	-4.6	-0.1
-1	36	-0.13	-1	-4.5	-0.1
-1	43	0.47	1	20.1	1
1	100	0.56	38	56	1
-1	94	-0.10	43	-9.2	1
-1	31	-0.02	10	-0.7	1
-1	41	-0.01	35	-0.6	1
-1	46	-0.12	-1	-5.5	-0.1
-1	64	-0.11	-1	-7.3	-0.1
-1	78	0.16	2	12.2	1
-1	39	-0.09	33	-3.7	1
-1	40	-0.07	38	-2.7	1
-1	47	-0.03	3	-1.2	1
-1	36	0.33	21	12	1
-1	40	-0.12	-1	-4.9	-0.1
-1	60	-0.11	-1	-6.9	-0.1
-1	36	-0.13	-1	-4.5	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	35	-0.13	-1	-4.4	-0.1
-1	53	-0.12	-1	-6.2	-0.1
-1	39	-0.12	-1	-4.8	-0.1
-1	50	0.01	22	0.7	1
-1	44	-0.10	41	-4.2	1
-1	43	0.29	26	12.4	1
-1	66	-0.11	-1	-7.5	-0.1
-1	44	0.48	5	21.1	1
-1	46	-0.10	45	-4.4	1
-1	39	-0.12	-1	-4.8	-0.1
-1	45	-0.12	-1	-5.4	-0.1
-1	41	-0.12	-1	-5	-0.1
-1	50	-0.05	1	-2.6	1
-1	42	-0.12	-1	-5.1	-0.1
-1	39	-0.04	36	-1.5	1
-1	46	-0.12	-1	-5.5	-0.1
-1	60	0.53	2	31.6	1
-1	40	0.40	4	16	1
-1	49	0.02	42	0.8	1
-1	51	-0.12	-1	-6	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	34	0.00	27	0.1	1
-1	38	-0.09	1	-3.6	1
-1	68	0.03	1	2.2	1
-1	33	-0.13	-1	-4.2	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	73	-0.11	-1	-8.2	-0.1
-1	47	0.19	3	8.7	1

-1	34	0.07	21	2.3	1
-1	37	0.29	1	10.8	1
-1	47	-0.12	-1	-5.6	-0.1
-1	61	0.01	15	0.7	1
-1	51	-0.12	-1	-6	-0.1
-1	27	-0.13	-1	-3.6	-0.1
-1	51	-0.10	17	-4.9	1
-1	53	-0.12	-1	-6.2	-0.1
-1	59	-0.12	-1	-6.8	-0.1

Baseline Test - π_5

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	89	-0.11	-1	-9.8	-0.1
-1	41	-0.12	-1	-5	-0.1
-1	28	-0.01	1	-0.4	1
-1	63	0.43	1	26.9	1
-1	61	0.12	1	7.3	1
-1	59	-0.12	-1	-6.8	-0.1
-0.1	100	-0.09	1	-8.9	1
-1	38	-0.12	-1	-4.7	-0.1
-1	70	-0.11	-1	-7.9	-0.1
-1	47	-0.12	-1	-5.6	-0.1
-1	41	-0.12	-1	-5	-0.1
-0.1	100	0.35	7	35.1	1
-1	34	-0.13	-1	-4.3	-0.1
-1	41	0.01	6	0.5	1
-1	100	-0.10	27	-9.8	1
-1	36	-0.09	11	-3.4	1
-1	46	-0.12	-1	-5.5	-0.1
-1	44	0.65	3	28.8	1
-1	92	-0.11	-1	-10.1	-0.1
-1	48	-0.12	-1	-5.7	-0.1
-1	19	0.26	5	4.9	1
-1	92	0.24	24	21.8	1
-1	39	0.33	22	12.8	1
-0.1	100	0.00	35	-0.1	1
-1	63	-0.11	-1	-7.2	-0.1
-1	44	-0.12	-1	-5.3	-0.1
-1	33	0.91	1	29.9	1
-1	39	-0.12	-1	-4.8	-0.1
-1	51	-0.12	-1	-6	-0.1
-1	46	-0.12	-1	-5.5	-0.1
-1	39	-0.12	-1	-4.8	-0.1
1	100	0.53	36	52.7	1
-1	2	0.00	1	0	1
-1	42	0.04	36	1.5	1
-0.1	100	-0.10	-1	-10	-0.1
-1	30	-0.09	29	-2.8	1
-0.1	100	0.13	9	13.1	1
-1	89	0.58	1	51.8	1
-1	94	0.86	2	81	1
-0.1	100	0.54	3	53.8	1
-1	49	-0.05	1	-2.5	1
-1	85	-0.11	-1	-9.4	-0.1
-1	47	0.28	1	13.1	1
-1	64	-0.06	14	-4	1

-1	44	-0.12	-1	-5.3	-0.1
-0.1	100	0.03	2	3.2	1
1	100	0.75	22	74.7	1
-1	49	-0.10	4	-4.7	1
-1	27	0.89	1	23.9	1
-1	36	0.82	1	29.6	1
-1	34	0.13	1	4.5	1
-1	62	-0.11	-1	-7.1	-0.1
1	100	0.64	10	63.7	1
-1	33	-0.13	-1	-4.2	-0.1
1	100	0.67	2	67	1
1	100	0.91	7	91.2	1
-1	51	-0.12	-1	-6	-0.1
1	100	0.52	37	51.6	1
-1	28	0.22	14	6.2	1
1	100	0.76	2	75.8	1
-1	31	-0.06	1	-1.8	1
-1	40	-0.04	36	-1.6	1
1	100	0.88	1	87.9	1
-1	56	-0.12	-1	-6.5	-0.1
-1	92	0.27	7	25.1	1
-0.1	100	0.04	11	4.3	1
-1	39	-0.12	-1	-4.8	-0.1
-1	38	0.02	26	0.8	1
-1	36	-0.06	34	-2.3	1
-1	32	-0.13	-1	-4.1	-0.1
-1	39	-0.09	1	-3.7	1
-1	46	0.10	1	4.4	1
-1	66	-0.11	-1	-7.5	-0.1
-1	26	0.54	1	14.1	1
-1	78	-0.11	-1	-8.7	-0.1
-1	27	-0.05	1	-1.4	1
-1	80	0.19	3	15.3	1
-1	97	0.38	1	36.7	1
-1	55	-0.12	-1	-6.4	-0.1
-1	41	-0.10	4	-3.9	1
-1	95	-0.05	67	-4.9	1
-1	45	0.08	32	3.4	1
-1	28	-0.05	26	-1.5	1
-1	59	0.59	5	35	1
-1	93	0.14	1	12.9	1
-1	45	-0.05	34	-2.1	1
-1	98	0.69	1	67.4	1
-1	68	-0.11	-1	-7.7	-0.1
1	100	0.49	40	49.4	1
-1	37	-0.09	1	-3.5	1
-1	84	-0.08	57	-7.1	1

-1	34	0.04	1	1.2	1
-1	48	-0.12	-1	-5.7	-0.1
-0.1	100	-0.07	8	-6.7	1
-1	56	-0.12	-1	-6.5	-0.1
-1	44	-0.12	-1	-5.3	-0.1
-1	47	0.14	4	6.5	1
-0.1	100	-0.04	66	-4.5	1
-1	36	0.27	5	9.8	1
-1	51	-0.12	-1	-6	-0.1

Baseline Test - Hybrid

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
1	100	0.81	18	81.3	1
-0.1	100	-0.08	16	-7.8	1
-1	78	0.79	12	61.7	1
-0.1	100	-0.09	49	-8.9	1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.81	11	81.3	1
-0.1	100	0.24	27	24.1	1
-0.1	100	-0.08	22	-7.8	1
-0.1	100	-0.09	54	-8.9	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.40	9	39.5	1
-0.1	100	-0.07	40	-6.7	1
-0.1	100	-0.08	30	-7.8	1
-1	79	-0.10	61	-7.7	1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.30	11	29.6	1
-0.1	100	0.65	1	64.8	1
-1	63	0.62	1	39	1
-0.1	100	-0.10	-1	-10	-0.1
-1	72	0.13	1	9.5	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	0.08	38	7.6	1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.12	17	12	1
1	100	0.86	6	85.7	1
1	100	0.27	29	27.4	1
1	100	0.70	21	70.3	1
1	100	0.11	1	10.9	1
-0.1	100	0.05	32	5.4	1
-0.1	100	-0.08	3	-7.8	1
-0.1	100	-0.03	55	-3.4	1
-1	29	-0.13	-1	-3.8	-0.1
1	100	0.79	16	79.1	1
-0.1	100	-0.02	50	-2.3	1
-0.1	100	0.10	10	9.8	1
-1	73	-0.10	7	-7.1	1
1	100	0.43	10	42.8	1
1	100	0.54	27	53.8	1
-0.1	100	-0.09	28	-8.9	1
1	100	0.14	44	14.2	1
-1	2	-0.55	-1	-1.1	-0.1
1	100	0.60	31	60.4	1
1	100	0.92	3	92.3	1
1	100	0.96	5	95.6	1

-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	0.19	4	18.6	1
1	100	0.75	3	74.7	1
-0.1	100	0.52	8	51.6	1
1	100	0.63	4	62.6	1
1	100	0.77	15	76.9	1
-0.1	100	0.12	13	12	1
-0.1	100	-0.07	63	-6.7	1
1	100	0.69	1	69.2	1
1	100	0.70	2	70.3	1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.90	9	90.1	1
-1	81	0.43	36	35	1
1	100	0.99	1	98.9	1
-0.1	100	0.79	1	79.1	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	0.02	17	2.1	1
1	100	0.81	13	81.3	1
1	100	0.53	6	52.7	1
1	100	0.55	21	54.9	1
1	100	0.79	11	79.1	1
1	100	0.80	19	80.2	1
1	100	0.76	5	75.8	1
-0.1	100	-0.08	3	-7.8	1
1	100	0.56	20	56	1
1	100	-0.04	52	-4.5	1
-0.1	100	0.54	7	53.8	1
-0.1	100	-0.01	1	-1.2	1
-0.1	100	-0.08	6	-7.8	1
-0.1	100	0.14	2	14.2	1
-0.1	100	0.48	12	48.3	1
-0.1	100	0.27	4	27.4	1
-1	96	-0.02	1	-1.7	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	0.07	18	6.5	1
-0.1	100	0.32	18	31.8	1
-0.1	100	-0.06	8	-5.6	1
1	100	0.58	29	58.2	1
-0.1	100	-0.10	-1	-10	-0.1
1	100	0.98	1	97.8	1
-1	80	0.62	14	49.4	1
-1	39	-0.12	-1	-4.8	-0.1
-0.1	100	0.41	4	40.6	1
-1	40	0.26	5	10.5	1
-0.1	100	0.08	8	7.6	1
-0.1	100	-0.02	23	-2.3	1
-1	45	-0.12	-1	-5.4	-0.1

1	100	0.44	14	43.9	1
-1	1	-1.00	-1	-1	-1
-0.1	100	-0.01	27	-1.2	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	0.32	19	31.8	1
-0.1	100	-0.09	76	-8.9	1
-0.1	100	-0.10	-1	-10	-0.1

Baseline Test - DDPG

terminal rewards	episode lengths	average rewards	steps to reward	total rewards	max rewards
-1	47	0.35	22	16.4	1
-1	70	0.48	15	33.9	1
-0.1	100	-0.10	-1	-10	-0.1
-1	36	-0.13	-1	-4.5	-0.1
-1	45	0.52	12	23.2	1
-1	15	-0.16	-1	-2.4	-0.1
-1	2	-0.55	-1	-1.1	-0.1
-1	8	-0.21	-1	-1.7	-0.1
-1	11	-0.18	-1	-2	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	30	0.05	8	1.6	1
-1	2	-0.55	-1	-1.1	-0.1
-1	27	-0.13	-1	-3.6	-0.1
-1	45	0.66	11	29.8	1
-1	2	-0.55	-1	-1.1	-0.1
-1	32	-0.13	-1	-4.1	-0.1
-1	54	0.47	15	25.6	1
1	100	0.69	13	69.2	1
1	100	0.66	3	65.9	1
-0.1	100	-0.10	-1	-10	-0.1
-0.1	100	-0.06	4	-5.6	1
-1	71	0.15	36	10.7	1
-1	10	0.03	3	0.3	1
-1	49	-0.12	-1	-5.8	-0.1
1	100	0.45	3	45	1
-0.1	100	-0.04	16	-4.5	1
-1	8	-0.08	1	-0.6	1
-1	39	0.50	14	19.4	1
-1	2	-0.55	-1	-1.1	-0.1
-1	15	-0.09	1	-1.3	1
-1	16	-0.16	-1	-2.5	-0.1
-1	16	-0.16	-1	-2.5	-0.1
-1	43	-0.12	-1	-5.2	-0.1
-1	13	0.08	8	1.1	1
-1	10	0.36	1	3.6	1
1	100	0.32	47	31.8	1
-1	60	0.12	2	7.4	1
-0.1	100	0.13	2	13.1	1
-1	51	0.29	2	14.9	1
-0.1	100	0.70	2	70.3	1
-1	29	-0.13	-1	-3.8	-0.1
1	100	0.58	1	58.2	1
1	100	0.43	8	42.8	1
-1	51	-0.12	-1	-6	-0.1
-0.1	100	-0.07	4	-6.7	1
-0.1	100	0.15	1	15.3	1
-1	42	-0.10	41	-4	1
-1	7	-0.23	-1	-1.6	-0.1
-0.1	100	-0.09	80	-8.9	1
-1	99	-0.09	2	-8.6	1

-0.1	100	0.32	20	31.8	1
-1	17	-0.15	-1	-2.6	-0.1
-1	15	0.06	10	0.9	1
-1	3	-0.40	-1	-1.2	-0.1
-1	2	-0.55	-1	-1.1	-0.1
-1	21	-0.14	-1	-3	-0.1
-1	13	-0.17	-1	-2.2	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	50	-0.12	-1	-5.9	-0.1
1	100	0.23	2	23	1
1	100	0.45	11	45	1
-1	12	0.28	1	3.4	1
-1	2	-0.55	-1	-1.1	-0.1
-1	8	-0.21	-1	-1.7	-0.1
-0.1	100	0.33	17	32.9	1
-1	2	-0.55	-1	-1.1	-0.1
-1	10	0.69	2	6.9	1
-1	21	-0.04	2	-0.8	1
-1	2	-0.55	-1	-1.1	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	8	-0.21	-1	-1.7	-0.1
-1	25	0.35	3	8.7	1
-1	2	-0.55	-1	-1.1	-0.1
-1	36	0.76	7	27.4	1
-0.1	100	0.19	1	18.6	1
-1	81	0.12	3	9.7	1
-1	9	0.53	2	4.8	1
-0.1	100	-0.09	88	-8.9	1
-1	60	0.11	26	6.3	1
1	100	0.47	3	47.2	1
-1	31	0.23	15	7	1
1	100	0.75	12	74.7	1
-1	15	-0.16	-1	-2.4	-0.1
-1	19	-0.15	-1	-2.8	-0.1
-1	59	-0.12	-1	-6.8	-0.1
-0.1	100	-0.10	-1	-10	-0.1
-1	33	-0.13	-1	-4.2	-0.1
-1	55	-0.12	-1	-6.4	-0.1
-0.1	100	0.44	3	43.9	1
-0.1	100	0.70	2	70.3	1
-0.1	100	-0.02	2	-2.3	1
-1	30	0.27	19	8.2	1
1	100	0.41	1	40.6	1
-0.1	100	0.10	23	9.8	1
-1	38	0.25	2	9.6	1
1	100	0.68	4	68.1	1
-1	16	-0.16	-1	-2.5	-0.1
-1	8	-0.21	-1	-1.7	-0.1
1	100	0.07	19	6.5	1
-1	41	0.20	9	8.2	1

Canopy Test - $\mu = 20$

terminal rewards	total rewards	average rewards	steps to reward	episode lengths	max rewards
-1	-2	-0.181818182	-1	11	-0.1
-1	21.7	0.571052632	1	38	1
-0.1	-8.9	-0.089	99	100	1
-0.1	9.8	0.098	1	100	1
-0.1	61.5	0.615	1	100	1
-0.1	31.8	0.318	3	100	1
1	100	1	1	100	1
1	95.6	0.956	1	100	1
-1	5.6	0.124444444	2	45	1
-0.1	51.6	0.516	20	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-2.1	-0.091304348	3	23	1
-1	34.3	0.51969697	2	66	1
1	91.2	0.912	1	100	1
-1	-2.9	-0.038666667	12	75	1
-1	6.3	0.39375	2	16	1
-1	-1.2	-0.4	-1	3	-0.1
-1	0.6	0.085714286	2	7	1
1	83.5	0.835	4	100	1
-1	6.9	0.079310345	37	87	1
-0.1	-8.9	-0.089	66	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-4.5	-0.125	-1	36	-0.1
1	47.2	0.472	1	100	1
1	89	0.89	1	100	1
1	100	1	1	100	1
1	98.9	0.989	1	100	1
1	54.9	0.549	4	100	1
-1	-2.7	-0.15	-1	18	-0.1
-1	-1.1	-0.55	-1	2	-0.1
1	79.1	0.791	9	100	1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	8.6	0.573333333	1	15	1
-1	11.6	0.644444444	2	18	1
-1	51.2	0.701369863	3	73	1
-0.1	-0.1	-0.001	33	100	1
1	89	0.89	1	100	1
-1	1.9	0.11875	11	16	1
-1	11.6	0.644444444	1	18	1
1	51.6	0.516	4	100	1
-1	-0.9	-0.081818182	1	11	1
1	42.8	0.428	1	100	1
-1	0	0	1	2	1
1	65.9	0.659	4	100	1
-0.1	-7.8	-0.078	3	100	1
-1	-1.6	-0.055172414	1	29	1
-1	0.4	0.044444444	2	9	1
-1	7.1	0.236666667	1	30	1
-1	43.3	0.446391753	35	97	1

-1	0.1	0.008333333	1	12	1
-1	14.4	0.626086957	2	23	1
1	65.9	0.659	2	100	1
-0.1	62.6	0.626	1	100	1
-1	3.5	0.318181818	1	11	1
-0.1	-6.7	-0.067	1	100	1
1	100	1	1	100	1
-1	9.9	0.282857143	20	35	1
-1	5	0.277777778	1	18	1
-1	0	0	1	2	1
-1	-0.9	-0.081818182	10	11	1
-1	-2	-0.045454545	20	44	1
-1	-1	-1	-1	1	-1
-0.1	93.4	0.934	3	100	1
-1	15.3	0.425	1	36	1
-1	-2.5	-0.15625	-1	16	-0.1
-1	1.9	0.38	1	5	1
-0.1	9.8	0.098	1	100	1
-1	13.2	0.55	2	24	1
-1	-1.1	-0.55	-1	2	-0.1
1	80.2	0.802	1	100	1
1	56	0.56	1	100	1
1	73.6	0.736	1	100	1
-1	83.3	0.895698925	2	93	1
-1	-0.2	-0.013333333	1	15	1
-1	0	0	1	2	1
-0.1	29.6	0.296	1	100	1
-1	0.7	0.041176471	4	17	1
-0.1	16.4	0.164	1	100	1
-0.1	-5.6	-0.056	38	100	1
1	76.9	0.769	1	100	1
-1	-1.1	-0.084615385	12	13	1
1	78	0.78	17	100	1
1	67	0.67	1	100	1
-1	18.7	0.779166667	2	24	1
-0.1	64.8	0.648	1	100	1
1	35.1	0.351	1	100	1
1	52.7	0.527	23	100	1
-1	4.9	0.257894737	4	19	1
-1	-9.9	-0.11	-1	90	-0.1
-1	0	0	1	2	1
-0.1	32.9	0.329	5	100	1
-0.1	51.6	0.516	1	100	1
1	98.9	0.989	1	100	1
1	75.8	0.758	1	100	1
1	94.5	0.945	1	100	1
-1	6.1	0.338888889	8	18	1
-0.1	90.1	0.901	1	100	1
1	85.7	0.857	1	100	1
-1	0.3	0.03	1	10	1
-1	-1.2	-0.4	-1	3	-0.1

Canopy Test - $\mu = 27.5$

terminal rewards	total rewards	average rewards	steps to reward	episode lengths	max rewards
-1	-2.8	-0.147368421	-1	19	-0.1
-0.1	7.6	0.076	1	100	1
-1	-3.5	-0.134615385	-1	26	-0.1
-1	-5.9	-0.118	-1	50	-0.1
-1	0	0	1	2	1
-1	5.4	0.0675	8	80	1
1	57.1	0.571	17	100	1
-1	5.5	0.157142857	1	35	1
-1	-1	-1	-1	1	-1
1	91.2	0.912	1	100	1
-1	1.9	0.38	1	5	1
-0.1	-7.8	-0.078	3	100	1
-1	-3.5	-0.134615385	-1	26	-0.1
-0.1	-5.6	-0.056	21	100	1
1	75.8	0.758	1	100	1
-0.1	26.3	0.263	15	100	1
1	96.7	0.967	1	100	1
-1	-1.9	-0.059375	2	32	1
-0.1	10.9	0.109	2	100	1
-1	2.9	0.058	9	50	1
-1	0	0	1	2	1
-1	13.8	0.270588235	32	51	1
1	78	0.78	1	100	1
-1	47.8	0.562352941	7	85	1
-1	0	0	1	2	1
-1	35.4	0.357575758	10	99	1
-1	0.7	0.041176471	2	17	1
1	47.2	0.472	13	100	1
-0.1	28.5	0.285	1	100	1
-1	-5.55112E-16	-2.56205E-17	1	13	1
-1	-3.6	-0.133333333	-1	27	-0.1
1	76.9	0.769	1	100	1
-0.1	6.5	0.065	1	100	1
-1	14.4	0.257142857	8	56	1
-1	-0.1	-0.033333333	1	3	1
1	43.9	0.439	16	100	1
-1	-4.5	-0.125	-1	36	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-7.8	-0.078	85	100	1
-1	4.6	0.139393939	8	33	1
-1	0	0	1	2	1
-1	4	0.102564103	6	39	1
-1	10.9	0.11978022	41	91	1
-1	0.8	0.05	2	16	1
-1	-5	-0.096153846	51	52	1
-1	0	0	1	2	1
-1	4.3	0.074137931	1	58	1
-0.1	58.2	0.582	1	100	1
-0.1	30.7	0.307	1	100	1
1	62.6	0.626	4	100	1

-1	0	-8.54018E-18	1	13	1
-1	-4.9	-0.096078431	1	51	1
-1	32.3	0.430666667	1	75	1
-0.1	53.8	0.538	8	100	1
1	15.3	0.153	6	100	1
-1	-5	-0.096153846	47	52	1
1	76.9	0.769	1	100	1
-1	47.1	0.672857143	2	70	1
-1	-0.5	-0.071428571	1	7	1
1	51.6	0.516	14	100	1
-1	28.1	0.7025	2	40	1
-1	7.2	0.116129032	13	62	1
-1	-1	-1	-1	1	-1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	2.1	0.021	16	100	1
1	94.5	0.945	1	100	1
-0.1	-4.5	-0.045	1	100	1
-0.1	-5.6	-0.056	1	100	1
-1	1.2	0.035294118	1	34	1
-0.1	34	0.34	11	100	1
-0.1	1	0.01	1	100	1
-1	47.1	0.798305085	1	59	1
-1	-1.1	-0.55	-1	2	-0.1
-1	-4.7	-0.123684211	-1	38	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	46.1	0.461	3	100	1
1	83.5	0.835	1	100	1
-1	6.8	0.618181818	1	11	1
-1	15.6	0.709090909	1	22	1
-1	-6.9	-0.115	-1	60	-0.1
-1	-1.7	-0.2125	-1	8	-0.1
-0.1	16.4	0.164	3	100	1
-1	5.8	0.276190476	1	21	1
-1	0	0	1	2	1
-1	17.5	0.192307692	2	91	1
-1	0	0	1	2	1
-1	-3.5	-0.134615385	-1	26	-0.1
-0.1	30.7	0.307	1	100	1
-0.1	6.5	0.065	1	100	1
-1	8.7	0.621428571	2	14	1
-1	5.7	0.518181818	1	11	1
-1	-1.1	-0.55	-1	2	-0.1
-1	30.1	0.35	14	86	1
-1	-0.1	-0.033333333	1	3	1
-0.1	75.8	0.758	4	100	1
-1	2.1	0.084	4	25	1
-1	-1.1	-0.55	-1	2	-0.1
1	81.3	0.813	1	100	1
-1	8.1	0.261290323	15	31	1
-1	0	0	1	2	1

Canopy Test - $\mu = 35$

terminal rewards	total rewards	average rewards	steps to reward	episode lengths	max rewards
-1	-1.4	-0.28	-1	5	-0.1
-1	-2	-0.181818182	-1	11	-0.1
-1	-7.2	-0.114285714	-1	63	-0.1
1	73.6	0.736	1	100	1
-1	2	0.133333333	10	15	1
-1	0	0	1	2	1
1	49.4	0.494	9	100	1
-1	-1.1	-0.084615385	1	13	1
-1	2.1	0.15	10	14	1
-1	-1.6	-0.228571429	-1	7	-0.1
-1	3.5	0.106060606	2	33	1
-1	11.8	0.310526316	3	38	1
-1	2.3	0.191666667	5	12	1
-1	31	0.402597403	12	77	1
-1	-1	-1	-1	1	-1
-0.1	50.5	0.505	1	100	1
-1	-1.6	-0.055172414	18	29	1
-0.1	-1.2	-0.012	35	100	1
-1	11.3	0.262790698	1	43	1
-1	11.1	0.326470588	2	34	1
1	70.3	0.703	1	100	1
-1	-5.5	-0.119565217	-1	46	-0.1
-1	0	0	1	2	1
-1	-1.1	-0.55	-1	2	-0.1
-1	12.4	0.288372093	1	43	1
-1	0	0	1	2	1
-0.1	32.9	0.329	8	100	1
-1	-1.2	-0.4	-1	3	-0.1
1	7.6	0.076	26	100	1
-1	-1.5	-0.088235294	1	17	1
-0.1	-8.9	-0.089	1	100	1
1	52.7	0.527	1	100	1
-1	31.5	0.516393443	20	61	1
-1	0	0	1	2	1
-0.1	-7.8	-0.078	46	100	1
1	39.5	0.395	1	100	1
-1	-4.9	-0.096078431	4	51	1
1	70.3	0.703	1	100	1
-1	4.6	0.418181818	2	11	1
-1	8.9	0.197777778	3	45	1
1	65.9	0.659	1	100	1
-1	1.6	0.016666667	43	96	1
-0.1	-10	-0.1	-1	100	-0.1
-1	0.9	0.034615385	7	26	1
1	49.4	0.494	3	100	1
1	86.8	0.868	1	100	1
-1	-1.7	-0.2125	-1	8	-0.1
-1	-3.1	-0.140909091	-1	22	-0.1
1	57.1	0.571	6	100	1
-0.1	-10	-0.1	-1	100	-0.1

-1	-5.1	-0.096226415	51	53	1
-1	-1.1	-0.55	-1	2	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-4.4	-0.077192982	46	57	1
-0.1	-4.5	-0.045	1	100	1
-1	24.8	0.4	23	62	1
-1	0	0	1	2	1
-0.1	8.7	0.087	12	100	1
-1	2.8	0.155555556	2	18	1
-1	-1.1	-0.55	-1	2	-0.1
-1	2.9	0.170588235	3	17	1
1	53.8	0.538	14	100	1
-1	-1.7	-0.2125	-1	8	-0.1
-1	-1.2	-0.4	-1	3	-0.1
1	47.2	0.472	11	100	1
1	98.9	0.989	1	100	1
-1	5.7	0.172727273	4	33	1
-1	0	0	1	2	1
-0.1	-10	-0.1	-1	100	-0.1
-1	2.1	0.15	9	14	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-5.6	-0.08115942	59	69	1
-1	0	0	1	2	1
1	72.5	0.725	1	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-6	-0.096774194	40	62	1
-1	-1.1	-0.55	-1	2	-0.1
-1	-1.2	-0.4	-1	3	-0.1
1	47.2	0.472	23	100	1
-0.1	40.6	0.406	1	100	1
-1	8.3	0.098809524	2	84	1
-0.1	-8.9	-0.089	75	100	1
-1	-0.2	-0.007692308	1	26	1
-1	0.2	0.004545455	25	44	1
-1	-1.1	-0.55	-1	2	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-1.7	-0.2125	-1	8	-0.1
-1	12.3	0.279545455	1	44	1
-1	0.4	0.004651163	1	86	1
1	48.3	0.483	45	100	1
-1	1.3	0.039393939	7	33	1
-1	0	0	1	2	1
-1	-1.2	-0.4	-1	3	-0.1
-1	-2.2	-0.169230769	-1	13	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	18	0.18556701	20	97	1
-1	-1.1	-0.55	-1	2	-0.1

Canopy Test - $\mu = 42.5$

terminal rewards	total rewards	average rewards	steps to reward	episode lengths	max rewards
-0.1	46.1	0.461	21	100	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-2.6	-0.152941176	-1	17	-0.1
1	84.6	0.846	8	100	1
1	-0.1	-0.001	20	100	1
-1	-2.6	-0.152941176	-1	17	-0.1
-1	25.3	0.443859649	11	57	1
-1	1.3	0.059090909	1	22	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-9	-0.111111111	-1	81	-0.1
-1	-4.1	-0.047126437	76	87	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-3.1	-0.140909091	-1	22	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-5.4	-0.096428571	55	56	1
-1	-5.1	-0.121428571	-1	42	-0.1
-1	-2.3	-0.164285714	-1	14	-0.1
1	69.2	0.692	29	100	1
-1	-0.9	-0.040909091	20	22	1
-1	47.4	0.707462687	5	67	1
-1	-3.3	-0.1375	-1	24	-0.1
-1	-1.3	-0.325	-1	4	-0.1
1	16.4	0.164	9	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	29.9	0.679545455	8	44	1
-1	-1.4	-0.28	-1	5	-0.1
-0.1	86.8	0.868	6	100	1
-1	8.3	0.2075	17	40	1
-0.1	-10	-0.1	-1	100	-0.1
-1	15.6	0.283636364	1	55	1
-1	-3.3	-0.094285714	34	35	1
-0.1	-2.3	-0.023	90	100	1
1	27.4	0.274	11	100	1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-1.2	-0.012	50	100	1
1	84.6	0.846	6	100	1
-1	-1.8	-0.09	16	20	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.5	-0.088235294	12	17	1
-1	4.9	0.077777778	26	63	1
-1	-0.3	-0.06	2	5	1
-1	1	0.333333333	1	3	1
-1	-1.2	-0.4	-1	3	-0.1
-1	-2.9	-0.145	-1	20	-0.1
-1	3.3	0.048529412	41	68	1
-1	-5.1	-0.121428571	-1	42	-0.1
-0.1	-10	-0.1	-1	100	-0.1

1	80.2	0.802	11	100	1
-1	-1.2	-0.4	-1	3	-0.1
-1	12.1	0.263043478	25	46	1
-1	18.7	0.207777778	55	90	1
-1	-2.6	-0.152941176	-1	17	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	8.7	0.087	1	100	1
1	59.3	0.593	26	100	1
-1	-1.2	-0.4	-1	3	-0.1
1	70.3	0.703	6	100	1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	64.8	0.648	1	100	1
-1	-1.2	-0.4	-1	3	-0.1
-1	16.5	0.183333333	49	90	1
-1	-1.2	-0.4	-1	3	-0.1
-1	13.4	0.243636364	30	55	1
-1	-3.8	-0.131034483	-1	29	-0.1
-1	-1.2	-0.4	-1	3	-0.1
1	96.7	0.967	4	100	1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	5.7	0.074025974	1	77	1
1	28.5	0.285	13	100	1
-1	58.6	0.901538462	3	65	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-4.2	-0.127272727	-1	33	-0.1
-1	-2.8	-0.147368421	-1	19	-0.1
-1	-1.5	-0.25	-1	6	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	12.9	0.263265306	28	49	1
-1	-8	-0.112676056	-1	71	-0.1
1	32.9	0.329	31	100	1
-1	-1.4	-0.28	-1	5	-0.1
-0.1	-3.4	-0.034	2	100	1
-1	-1.2	-0.4	-1	3	-0.1
-1	40.9	0.61969697	13	66	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-3.1	-0.140909091	-1	22	-0.1
-1	-5.1	-0.0796875	17	64	1
-1	0.8	0.02962963	1	27	1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	-6.7	-0.067	44	100	1
-1	-3.6	-0.133333333	-1	27	-0.1
-1	-2.5	-0.15625	-1	16	-0.1
-1	0	0	11	13	1
1	79.1	0.791	8	100	1
-0.1	-10	-0.1	-1	100	-0.1

Canopy Test - $\mu = 50$

terminal rewards	total rewards	average rewards	steps to reward	episode lengths	max rewards
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-2.7	-0.15	-1	18	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-0.1	-3.4	-0.034	13	100	1
-1	0.8	0.16	1	5	1
-1	-2.8	-0.147368421	-1	19	-0.1
-1	-3	-0.142857143	-1	21	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.5	-0.25	-1	6	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
1	60.4	0.604	14	100	1
-0.1	13.1	0.131	4	100	1
-1	4.2	0.113513514	25	37	1
-1	-1.4	-0.28	-1	5	-0.1
1	35.1	0.351	46	100	1
-1	-4.2	-0.127272727	-1	33	-0.1
-1	-2.3	-0.164285714	-1	14	-0.1
-1	-8.7	-0.111538462	-1	78	-0.1
-1	-5.2	-0.096296296	45	54	1
-1	24.7	0.392063492	11	63	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-4.7	-0.123684211	-1	38	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	19	0.351851852	17	54	1
1	68.1	0.681	11	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-3	-0.142857143	-1	21	-0.1
1	65.9	0.659	2	100	1
-1	10	0.833333333	1	12	1
-1	-2.8	-0.093333333	14	30	1
-1	10.3	0.332258065	6	31	1
-1	-1.5	-0.25	-1	6	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	1	0.333333333	1	3	1
-1	-1.2	-0.4	-1	3	-0.1
-1	-2.1	-0.046666667	42	45	1
-1	-1.2	-0.4	-1	3	-0.1
-1	-8.9	-0.11125	-1	80	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-5.9	-0.118	-1	50	-0.1
-1	-3.4	-0.136	-1	25	-0.1
-1	-1.6	-0.228571429	-1	7	-0.1

-1	-1.2	-0.4	-1	3	-0.1
-1	-3.1	-0.140909091	-1	22	-0.1
-1	-1.5	-0.25	-1	6	-0.1
-1	-2.2	-0.169230769	-1	13	-0.1
-1	13.8	0.270588235	30	51	1
-1	-2.7	-0.15	-1	18	-0.1
-1	-4.5	-0.077586207	55	58	1
-1	-1.9	-0.19	-1	10	-0.1
-1	2.4	0.072727273	1	33	1
-1	-2.5	-0.15625	-1	16	-0.1
-1	0.8	0.021052632	28	38	1
-1	-2	-0.181818182	-1	11	-0.1
1	50.5	0.505	9	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	22.6	0.309589041	22	73	1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-4.4	-0.077192982	49	57	1
1	8.7	0.087	52	100	1
-1	-6.3	-0.116666667	-1	54	-0.1
-1	16.1	0.193975904	45	83	1
-1	72.9	0.837931034	10	87	1
-1	-0.8	-0.012307692	1	65	1
-1	-1.5	-0.088235294	15	17	1
-1	-1.9	-0.19	-1	10	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.5	-0.25	-1	6	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.4	-0.28	-1	5	-0.1
1	81.3	0.813	2	100	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-1	-1.3	-0.325	-1	4	-0.1
-1	-2	-0.181818182	-1	11	-0.1
-0.1	34	0.34	16	100	1
-0.1	-10	-0.1	-1	100	-0.1
-1	-0.3	-0.06	2	5	1
-1	-1.5	-0.25	-1	6	-0.1
-1	16.4	0.282758621	23	58	1
-1	-1.4	-0.28	-1	5	-0.1
-1	-1.2	-0.4	-1	3	-0.1
-0.1	-10	-0.1	-1	100	-0.1
-1	-4	-0.046511628	34	86	1
-1	-6.9	-0.115	-1	60	-0.1
-1	3	0.6	1	5	1