# Leveraging learning techniques for agricultural robotics

Harjatin Singh Baweja

CMU-RI-TR-19-65

August, 2019


The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

**Thesis Committee:**
Dr. George Kantor
Dr. David Held
Brian Okorn

*Submitted in partial fulfillment of the requirements for the degree of Masters of Science in Robotics.*

## Abstract

Effective plant breeding requires scientists to find correspondences between genetic markers and desirable physical traits of the genotype. While the cost for gene sequencing has gone down by several orders of magnitude in the last two decades, measuring physical traits of a plant at scale is still a labour intensive task making it borderline intractable. This creates an opportunity for robotics to fill this gap in order to accelerate the breeding pipeline. Aided by current progress in machine learning, in this thesis we elucidate methods to perform high throughput non contact and contact based plant phenotyping. We developed a general purpose computer vision pipeline for plant physical trait detection and semantic segmentation. We tested this pipeline for Sorghum stalk counting and stalk width measurement from images. We discuss the shortcomings of having separate perception and manipulation modules for contact based phenotyping and propose an end to end reinforcement learning framework for learning from RGBD observations. We propose a principled online learning approach to weight different auxiliary losses in order to accelerate reinforcement learning. Our approach gives a 3x speed up for reaching and manipulation tasks in three simulated environments.

## Acknowledgments

This research work would not be possible without the constant guidance and encouragement from my advisor Dr.George Kantor and thesis committee member Dr. David Held. I would like to thank Brian Okorn for his invaluable feedback on my research. I would also like to express my gratitude to Xingyu Lin for his mentorship and support.I would like to take this opportunity to thank my parents for constantly supporting me in all my pursuits.

## Funding

The work detailed in this thesis is facilitated by funding from TERRA program by U.S. Department of Energys Advanced Research Project Agency-Energy(ARPA-E). This initiative focuses on development of robotic systems to autonomously gather phenotypic data for bioenergy sorghum.

# Contents

*When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.*

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Even with advent of modern mechanization, agriculture remains one of the most labour intensive tasks in 21st century. World's population is increasing at a rate higher than ever before and is projected to reach 9.7 billion by year 2050. This will lead to a 50% increase in demand for agricultural produce [1]. Due to shrinking margins and seasonal labour requirements, the number of people taking by agriculture in reducing. Only 1% of current population of United States derive their livelihood solely from agriculture (USDA-NASS, 2014). The need for efficient and sustainable solutions is further exacerbated by growing concerns for land degradation, deforestation and climate change.

Geneticists for decades have used their domain knowledge to cross plants with desirable physical traits to breed better off springs. As shown in figure 1.1 their has been a dramatic reduction in costs of gene sequencing over the past two decades. Geneticists benefit from finding correlations between desirable physical traits of a plant (plant phenotypes) and genetic markers in gene sequences to make more informed decisions for ideal germplasms selection. Currently phenotypic data is collected by manually measuring physical traits of a plants. When preformed at a large scale, this process is labour intensive and prone to inaccuracies. The inability to get large scale phenotypic data slows down the breeding process and is thus known as the Phenotyping Bottleneck in scientific community.

We show that robotics aided by learning based approaches for perception and manipulation can provide a solution for high throughput plant phenotyping and aid

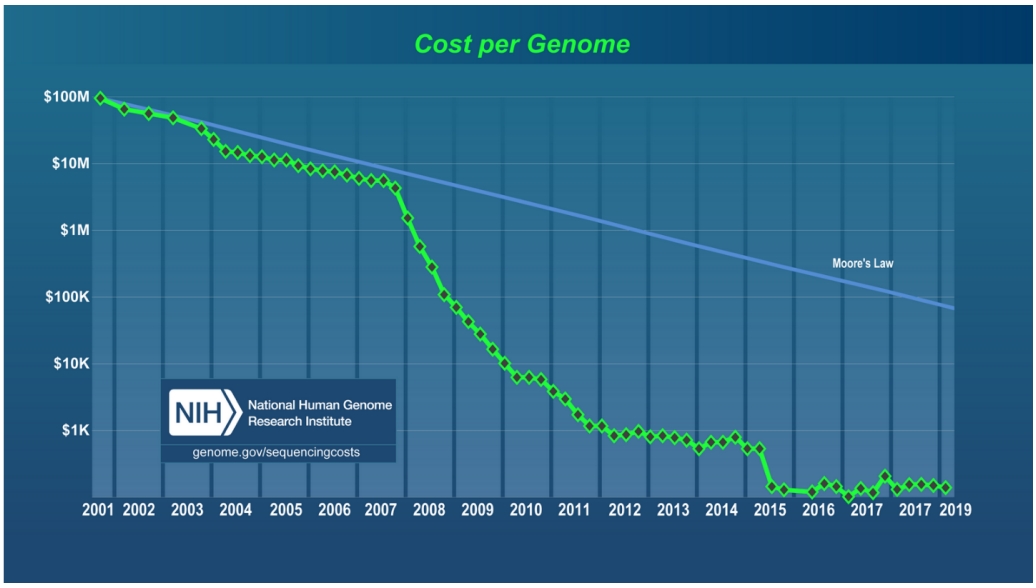


Figure 1.1: Gene sequencing costs from 2001 to 2019

the acceleration of plant breeding.

# Chapter 2

# Background

Background for this work can be broadly broken down into three sections, plant phenotyping, robot manipulation in agriculture and reinforcement learning with visual observation.

## 2.1 Plant Phenotyping

Depending on the physical structure of the object being detected, a wide variety of techniques have been developed for detection in agricultural setting. Xu et al. used a monocular camera and morphological image manipulation techniques to detect buds in grape vines [95]. Other techniques, such as those deployed by Baweja et al. [8] and Amean et al. [57] use Frangi filter [29] to detect tube like structures in the image to detect vine shoots and branches respectively. Pothen et al. [68] and Mirbod et al. use gradient of texture generated by flash light, to detect and size grapes [55] respectively. Jenkins et al. [38] use multiple stereo images taken form a moving platform for point cloud generation and exploit geometric structure of the stalks for detecting them. The authors report a precision of 0.95. The disadvantage of hand crafted feature based techniques is that, they require a sufficient amount of parameter tuning and do not generalize across all environment conditions, variable lighting for instance.

Sodhi et al. [84] combine classical vision techniques with machine learning techniques to segment Sorghum stalks from images. In this work, plants are imaged from various view points and the 3D point clouds are reconstructed. Segmentation

is performed over these point clouds using Support Vector Machines (SVMs) with Conditional Random Fields (CRFs). Accuracy of 80.2% is reported on field data. Paulus et al. use SVMs for classification if histograms of point-clouds for plant organ parametrization [63]. Reported Area Under the Curve (AUC) accuracy is 0.966.

Chen et al. [13] use deep learning based approach to count and detect fruits in images. They use a 2 CNN based architecture, where the first CNN detects blobs where the fruits might be present and the second CNN uses these blobs to estimate the fruit count in those regions. A regressor network then regresses the total fruit count in the image. Baweja et al. [9] use a CNN based architecture for Sorghum stalk detection and width estimation. They use Faster RCNN [69] for stalk detection and use a Fully Convolutional Network (FCN) [51] for segmentation of stalks in the proposals generated by Faster-RCNN. Reported R squared correlation is 0.88 for stalk count and 2.77 mm of absolute error in width estimation. However [9] suffers from poor quality segmentation masks. Parhar et al. [61] address this issue by using a Conditional-Generative Adversarial Network (C-GAN) for generating images, conditioned on the input image, such that the generated images have stalks painted red, which are then used for robotic grasping. They report an F1 score of 0.903 and a grasp accuracy of 74.16%. Owing to the generalizability of the architecture, we build on the work in [9] for detecting and localizing vine buds in 3D, using Faster-RCNN architecture.

## 2.2 Robotic Manipulation in Agriculture

Agricultural manipulation is a widely studied area of research, with a wide variety of techniques focusing towards specialty crops like apples[83]. Scarfe et al.[73] developed an arm and a custom end effector for handling of soft Kiwi fruit, along with a custom lighting system for detection and localization of the fruit with an accuracy of 3.6mm. Benton et al. used an industrial Panasonic VR600L along with a suction cup to pick apples with an accuracy of 80% [6]. [58] and [61] use a 3 DOF manipulator(2 DOF planar movements and 1 DOF in vertical movement along the mast of the robot) to grasp sorghum stalks in field conditions. [61] reports a grasp accuracy of 74.16% and [58] reports 25 out of 25 successful grasp attempts. None of the aforementioned works tackle the task of obstacle avoidance. Tanigaki et al.[88] implemented heuristic

based planning to avoid collision with the plant trunk for cherry harvesting.

Owing to the ability of neural networks to model complex non-linear functions effectively, there has been a huge body of work focusing on using neural-networks as controller for high DOF robotic manipulators. More recently, Reinforcement Learning(RL) techniques are being used to train these neural network controllers. Combining model free techniques with model based techniques like trajectory optimization, Guided Policy Search methods, provide guarantee of robust performance, under supervision from an optimal controller during training [46][47][41]. On the other hand, model free Deep RL methods do not assume the model of the agent. [32] uses model free Deep Q Networks with Normalized Advantage Functions (DQN-NAF) to learn to open doors. And [4] uses Hindsight Experience Replay to learn various tasks from scratch with sparse binary rewards.

Franceschetti et al. [28] preformed a detailed comparison of two model free techniques, TRPO [78] and DQN-NAF for the task of learning jobs like pick and place and goal reaching on a simulated UR5 robot. Owing to theoretic monotonic improvement guarantees of TRPO, Parhar et. al. use an improvement over TRPO, PPO [79] for training simulated UR5 arm to reach goals while avoiding an obstacle in direct line of sight of the end-effector. They further use the trained model to control the real arm to reach buds on a vine.

## 2.3 Reinforcement Learning with Visual Observation

### 2.3.1 Obtaining ground truth state for training

**Adding sensors:** To compute rewards for reinforcement learning, most algorithms assume access to the ground truth state. However, in most real world settings, the ground truth state is not directly observed but must be estimated using a noisy state estimator, which can lead to noisy rewards and poor learning. Attempts have been made to avoid this issue by adding extra sensors during training to accurately record the state. For example, in past work, one robot arm (covered with a cloth at training time) is used to rigidly hold and move an object, while another robot arm learns

to manipulate the object [47]. In such a case, the object position can be inferred directly from the position of the robot gripper that is holding it. In other work on teaching a robot to open a door, an IMU sensor is placed on the door handle to determine the rotation angle of the handle and whether or not the door has been opened [32, 96]. One can also ensure that all relevant objects for a task are placed on scales [74] or affixed with motion capture markers to obtain a precise estimate of their position [44]. However, such instrumentation is challenging for deformable objects, granular material, food, or other settings. Further, such instrumentation is costly and time-consuming to setup; hence most of these previous approaches assume that such instrumentation is only available at training time and these methods do not allow further fine-tuning of the policy after deployment.

**Training in simulation:** Another work-around to the issue of noisy state estimation is to train the policy entirely in simulation, in which the ground truth state can be obtained from the simulator [5, 22, 66, 72, 99]. Many approaches have been explored to try to transfer such a policy from simulation to the real world, such as domain randomization [91] or building or learning a more accurate simulator [12, 86]. However, obtaining an accurate simulator is often very challenging; especially if the simulator differs from the real-world in unknown ways, these methods will not transfer well to the real world. Further, building the simulator itself can be fairly complex. Because these methods require the ground truth state to obtain the reward function, they require training in a simulator and do not allow further fine-tuning after deployment in the real world; our method, in contrast, does not require the ground truth state for the reward function.

### 2.3.2 Robot learning without ground truth state

**Learning a reward function without supervision:** One line of work for learning a reward function is to first learn a latent representation and then use the L2 distance or cosine similarity in the embedding space as a reward. Different approaches for representation learning have been explored, such as maximizing the mutual information between the achieved goal and the intended goal [94], reconstruction of the observation with VAE [59], or learning to match keypoints with spatial autoencoders [26]. Our approach explained later in section 5.1 is much simpler

in that the reward function does not have any parameters that need to be learned.

**Changing the optimization:** Another approach is to forego maximizing a sum of rewards as is typically done in reinforcement learning and instead optimize for another objective. For example, one method is to choose one-step greedy actions based on a learned one-step inverse dynamics model; after training, the policy is then applied directly to a multi-step goal [3]. An alternative method is to learn a predictive forward dynamics model directly in a high-dimensional state space and use visual model-predictive control [19, 20, 21, 24, 25]. Although these methods have shown some promise, predicting future high-dimensional observations (such as images or depth measurements) is challenging. Another approach is to obtain expert demonstrations and define an objective as trying to imitate the expert [27, 64, 80, 81]. Our approach, however, applies even when demonstrations are not available.

**Incentivizing exploration:** An alterantive direction is to reward an agent for visiting unexplored parts of its environment [7, 10, 33, 62, 75, 76, 87]. However, if such an approach is not combined with some kind of goal-oriented learning, then the agent will not actually learn how to later perform intentional actions needed to achieve a task. Thus, the question of incentivizing exploration is somewhat orthogonal to the ideas explored in this paper.

### 2.3.3 Multi-task Learning (MTL)

The most similar analogy of auxiliary tasks in the context of supervised learning is multi-task learning [11, 71, 90]. Past work has found that, by sharing a representation among related tasks and jointly learning all the tasks, better generalization can be achieved over independently learning each task [30]. With neural networks, the common way of using MTL is to share the hidden layers between all tasks while keeping task-specific output layers, which is the approach we take. Unlike many works in MTL that try to learn a generic representation that can perform multiple tasks well [16, 43], we hope that, by combining a large set of auxiliary tasks, some of these tasks will be helpful for learning a feature representation suitable for the main control task.

### 2.3.4 Auxiliary Tasks for Reinforcement Learning

While reinforcement learning with low-dimensional state input can also benefit from auxiliary tasks in partially observable environments [48, 50], auxiliary tasks have been more commonly used for reinforcement learning from images or other high-dimensional sensor readings. The auxiliary tasks can either be supervised learning tasks such as depth prediction [56], reward prediction [82], or task specific prediction [53]. Here we focus on self-supervised tasks where labels can be acquired in a self-supervised manner. Although ground-truth state prediction may also be used as an auxiliary task in simulation [97], the ground-truth state might not be available in real world environments when learning from visual observations. Other works use alternative control tasks as auxiliary tasks, such as maximizing pixel changes or network features [36]. The weights for each of the auxiliary tasks are either manually set using prior intuition or through hyperparameter tuning by running the full training prcoedure multiple times. In this work, we propose to determine what auxiliary tasks are useful at each time step of the training procedure and adaptively tune the weights, eliminating the hyper-parameter tuning, which becomes much harder as the number of auxiliary tasks grows.

An alternative use of auxiliary tasks is to learn a hierarchical policy for better exploration [70]. Another example of this is the Horde architecture [85] which learns a general value functions from a set of pseudo-rewards. These types of methods are orthogonal to the use auxiliary tasks in this work, which are intended for feature representation learning.

### 2.3.5 Adaptive Weights for Multiple Losses

Previous works in MTL addressed the problem of specifying weights when multiple task losses are involved, by optimizing the total loss with model parameters and task weights jointly with regularization [52, 98]. In comparison to our work, some works assume that all auxiliary tasks matter equally and adapt the weights based on the gradient norm [14] or task uncertainty [42]. A similar approach for anytime prediction balances the weights based on the average loss over the previous training time [34]. These methods assume that all of the auxiliary tasks are equally important. However, if we scale the number of auxiliary tasks, it is highly probable that some tasks will be

more useful than others. In contrast, our method evaluates the usefulness of each task online and adapts the weights accordingly such that the more useful tasks receive a higher weight.

### 2.3.6 Manipulating deformable objects

Manipulating deformable objects is especially of relevance to agricultural robotics. Most organs of a plant like leaves, stems and fruit itself are often deformable and must be treated accordingly. Treating all organs of a plant as rigid bodies would severely limit a robot's ability to manipulate them. Deformable object manipulation presents many challenges for both perception and control. One approach to the perception problem is to perform non-rigid registration to a deformable model of the object being manipulated [15, 35, 37, 45, 54, 65, 77, 93]. However, such an approach is often slow, leading to slow policy learning, and can produce errors, leading to poor policy performance. Further, such an approach often requires a 3D deformable model of the object being manipulated, which may be difficult to obtain. Our approach applies directly to high-dimensional observations of the deformable object and does not require a prior model of the object being manipulated.

# Chapter 3

# Plant Phenotyping

With the goal of making better informed plant breeding decisions, it is critical for scientists to find correlations between physical features of a plant (plant phenotypes) to the genetic markers in the high dimensional genome sequence. These correlations help accelerate the breeding process by allowing breeders in crossing parents with desirable genetic markers yielding higher quality progeny. Even though the cost of gene sequencing has dropped considerably over the past two decades (figure 1.1), the same cannot be said about plant phenotyping. To achieve the desired result there exists a need for systems that can accurately measure plant phenotypes at high throughput.

The inability to do this with human labour is currently what is referred to as the Phenotyping Bottleneck in the scientific community. It is necessary to collect phenotypic data at scale in order to make decisions that are not prone to sampling bias. Doing this in outdoor farm conditions can be extremely labor intensive and prone to human error. This creates an opportunity to deploy robotic systems that allow high precision repeatable measurements. Using aerial platforms such as UAVs is attractive due to speed to coverage for a large area. However these platforms are severely hampered in terms of payload capacity and flying time. These platforms can also not provide data from underneath the canopy for tall crops such as bio energy sorghum.

In this work we focus on plant phenotyping for bio energy Sorghum crop. Sorghum is a diverse crop with over 40,000 accessions. It is useful for a plethora of purposes

Figure 3.1: Rendering of the ground-based agricultural robot developed at CMU

including but not limited to biofuel, animal fodder and alcohol production. This research is facilitated by a grant from U.S. Department of Energy Advanced Research Project Agency-Energy (ARPA-E)'s TERRA program. Plant stalk width and stalk count are two of the the most important phenotypes required for the sorghum breeding program.

## 3.1 Ground Robot Platform

Since none of the off-the shelf platforms have the runtime and payload capacity to carry oer 50kg of sensors, Tim Mueller Sim, Merritt Jenkins and Justin Abel built a custom ground robot platform [58] to meet the requirements. The platform shown in figure 3.1 is capable of autonomously navigating rows more than 30 inches wide.

The robot carries a suite of sensors including Sick LIDAR, an inertial measurement unit. The robot uses GPS with real time kinematic corrections to get sub centimeter accuracy for pure pursuit autonomous navigation given GPS waypoints. It is equipped with a two degree of freedom arm for contact sensing. The most relevant sensor for this work in the custom 9MP stereo camera attached to the back of the robot. The imager has hardware synced flashes that drown the background and allow us to take similar images invariant of the natural lighting conditions. We collected images at 3Hz. The robot has three Intel NUC i7 computers. All the sensors are synchronized and operated using the Robot Operating System.

## 3.2 Process Pipeline

The motivation behind the work was to develop a high throughput plant phenotyping computer vision based approach that is agnostic to changes in the field conditions and settings such as varying lighting conditions, occlusions, etc. Fig. 3.2 shows the overview of the data-processing pipeline, used by our approach. The faster RCNN takes one of the stereo pair images as its input and produces bounding boxes, each for one stalk. These bounding boxes are extracted from the input image (also called as snips) and fed to the FCN, individually. The FCN outputs a binary mask, classifying each pixel as either belonging to stalk or the background. To this mask, ellipse are fitted to the blobs in the binary mask by minimizing the least-square loss of the pixels in the blob [20]. One snip may have multiple ellipses in cases of multiple blobs. The ellipse with the largest minor axis is used for width calculation. The minor axis of this ellipse gives us the pixel width of the shoot in the current snip. The corresponding pixels in the disparity map are used to convert this pixel width into metric units. The whole pipeline takes on an average 0.19 seconds to process one image on a GTX 1080 GPU. This makes on-borad data-processing viable for systems that collect data at 3Hz.

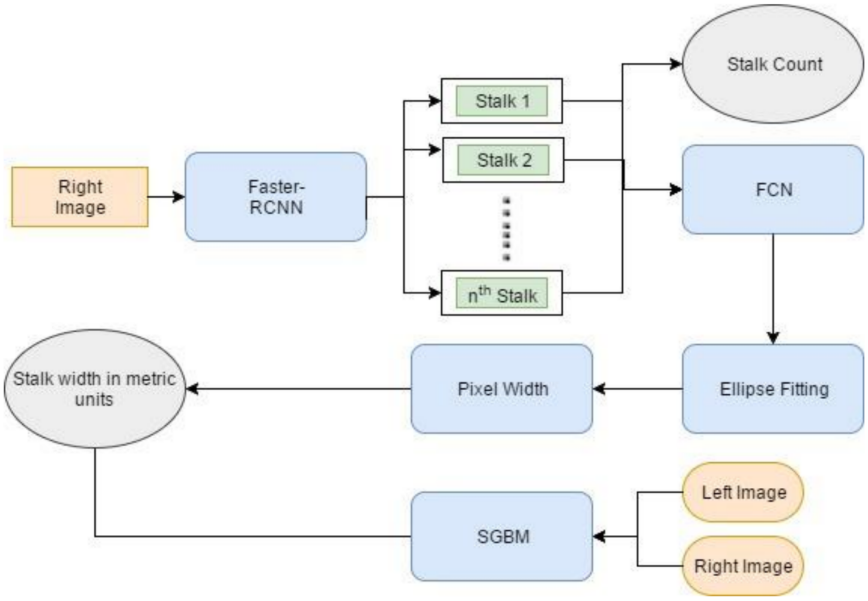


Figure 3.2: Overview of the stalk count and width calculation pipeline

## 3.3 Results

### 3.3.1 Data Collection

Image data was collected in July 2016, in Pendleton, South Carolina using the our ground robot platform. The algorithms were developed on this data. To test the algorithm impartially, another round of data collection with extensive ground truthing was done in February, 2017 in Cruz Farm, Mexico. The images were collected using a 9MP stereo-camera pair with 8 mm focal length, high power flashes triggered at 3Hz by ROS. The sensor was driven at approximately 0.05m/s. A distance of approximately 0.8m was maintained from the plant growth.

Each row of plant growth at Cruz Farm is divided into several 7 feet ranges separated by 5 feet alleys. To ground truth stalk count data, all stalks were counted in 29 ranges by two individuals separately. The mean of these counts was taken as the actual ground truth. Similarly, for width calculations, QR tags were attached to randomly chosen stalks for ground truth registration in images. The width of these stalks at height of 12 inches (30.48 cm) and 24inches (60.96 cm) from the ground

was also measured by two individuals separately using Vernier Calipers of 0.01 mm precision. Humans at an average took 210 seconds to count the stalks in each range and an average of 55 seconds to measure width of each stalk. Each range of plants on an average has 33 stalks, so on an average it would take 33 minutes to measure stalk widths of entire range.

(a) (b)

Figure 3.3: (a) The Robotanist collecting data in Pendleton, South Carolina (b) custom stereo imaging sensor used for imaging stalks

### 3.3.2 Results for stalk count

Due to inability to get accurate homography for data collected at 3Hz, we resorted to calculating stalk-count/meter using stereo data. For ground truth stalk counts which were collected from ranges, each range is of constant length 7 feet (2.134 m). Fig. 3.4 shows the R-squared correlation of 0.88 for robot stalk count vs human stalk count.

### 3.3.3 Results for stalk width

We plot the stalk width values as measured by Human1, Human2 and our pipeline at approximately 12 inches(30.48 cm) from the ground. At the time of data collection, we made sure that a part of ground is visible in every image. This allows us get stalk width at the desired height from the image data. This step is important for consistent width measurement across all the stalks as there is prevalent tapering in

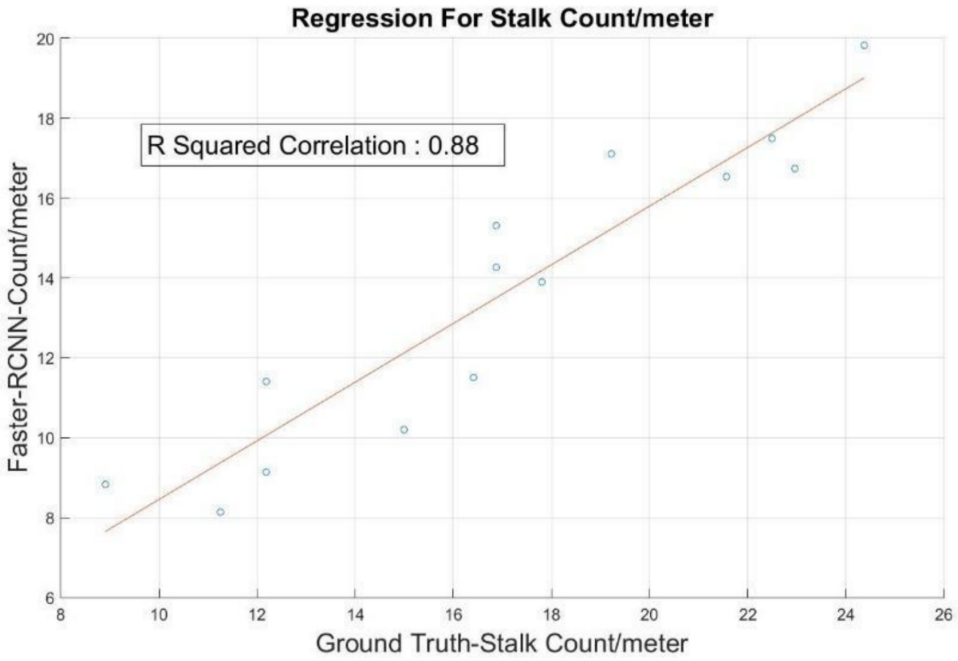


Figure 3.4: R squared correlation for human vs robot stalk count

stalk widths as we go higher up from the ground. Fig. 3.5 shows the widths of each ground truthed stalk as measured by both humans and computed by the algorithm.

Since there is a discernible difference in measurements of the two humans, we considered the mean of the two readings as actual ground truth. The mean width of stalks as per this ground truth is 14.354 mm. The mean absolute error between readings of Human1 and Human2 is 1.639mm and the mean absolute error between readings from human ground truth and algorithm is 2.76 mm. The error can be attributed to rare occlusions that force algorithm to calculate height at a location other than 12 inches(30.48 cm) from the ground. Since this as a possibility, we thus measure stalk widths at 2 locations during the ground truthing process: at 12inches(30.48 cm) and 24 inches(60.96 cm) above the ground. Calculations from this data tell us that there was 0.405 mm/inch mean tapering on the measured stalks as we went up from 12 inches(30.48 cm)to 24 inches(60.96 cm)

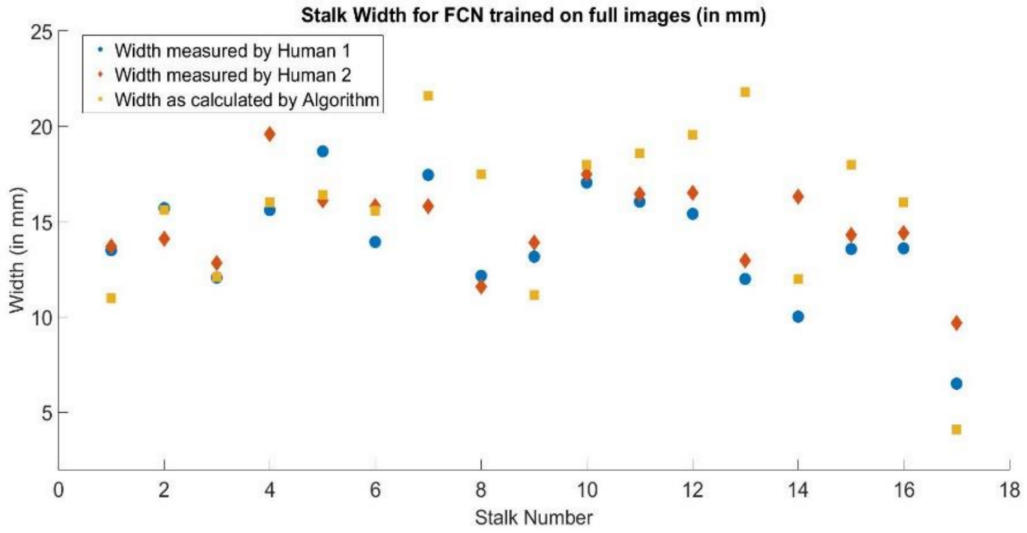


Figure 3.5: Width measured by Human1, Human2 and pipeline

### 3.3.4 Time Analysis

Table 3.1 shows the time comparisons of humans vs algorithm for an average plot. Each plot has approximately 33 stalks and is about 2.133m in length. We observe that automated counting is 30 times faster as compared to humans for stalk counting and 100 times faster than humans for stalk width calculation.

| | Human1 | Human2 | Robot |
|---|---|---|---|
| Stalk Count | 3.33 minutes | 3.66 minutes | |
| Stalk Width | 29 minutes | 30 minutes | 6.5 seconds |
| Total | 32.33 minutes | 33.66 minutes | 6.5 seconds |

Table 3.1: Environment details including observation, goal dimension (in state space, we use Image goals with same dimension as observation for learning), time horizon and distance threshold for successful episode.

# Chapter 4

# Stalk Grasping

Previous chapter highlights how computer vision based techniques can be effectively used for non-contact based plant phenotyping. However there exist several important physical traits of a plant that cannot be measured without physically grasping the plant e.g. hyperstectral imaging based abiotic and biotic stress management, stalk strength, etc. For this purpose we demonstrate a successful deployment of an online, high-throughput deep learning-based sorghum stalk detection and grasping pipeline in an outdoor field.

We utilize the custom three degree of freedom end effector mounted on the vertical linear slide of our ground robot platform. The first DOF is used to move the arm up and down the vertical slide to grasp plants at various heights. The other two DOFs are revolute joints that allow inverse kinematics to move the arm to any given location in plane specified by the height of the linear slide. Figure 4.1 shows a close up rendering of the two DOF planar manipulator.

## 4.1 System Overview

The MultiSense S7 was mounted to the robot at a height of 1m above the ground and 25mm in front of the robots center plane. This positions the camera at a distance from stalks greater than its 200mm minimum range. The camera is synchronized to custom LED flashes visible in Figure 4.2. The LED flashes provide both consistent lighting and foreground brightening, which significantly improves correspondence

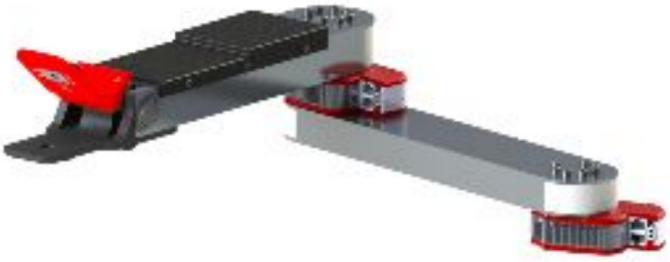

Figure 4.1: Rendering of two DOF arm mounted on vertical slide of ground robot

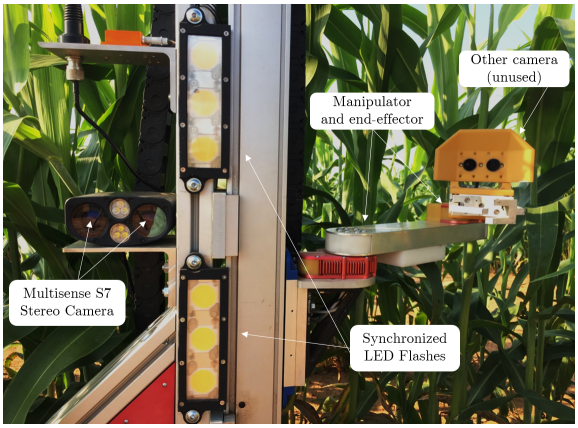


Figure 4.2: The Multisense S7 stereo camera mounted to the robot with synchronized flashes.

matching.

The robot manipulator is designed with three degrees of freedom consisting of a prismatic joint and two rotational joints (PRR). The motivation behind this design is that sorghum stalks are roughly vertical and grasp-based measurements are agnostic to location on a stalk's circumference. A 3-DOF PRR arm can position itself at only two circumferential locations at any height on a plant stalk. The manipulator is actuated by three series-elastic brushless DC motor modules developed by HEBI Inc. which communicate over LAN.

Computation is performed on a ZOTAC Magnus EN1080K mounted to the robot. The computer communicates on the robot local network over LAN. It is equipped with an Intel i7 processor and an Nvidia 1080 discrete GPU. The computer is a ROS slave running a node consisting of an active session of Tensorflow compute graph. The node subscribes to the rectified left image frame of the Multisense S7 and publishes

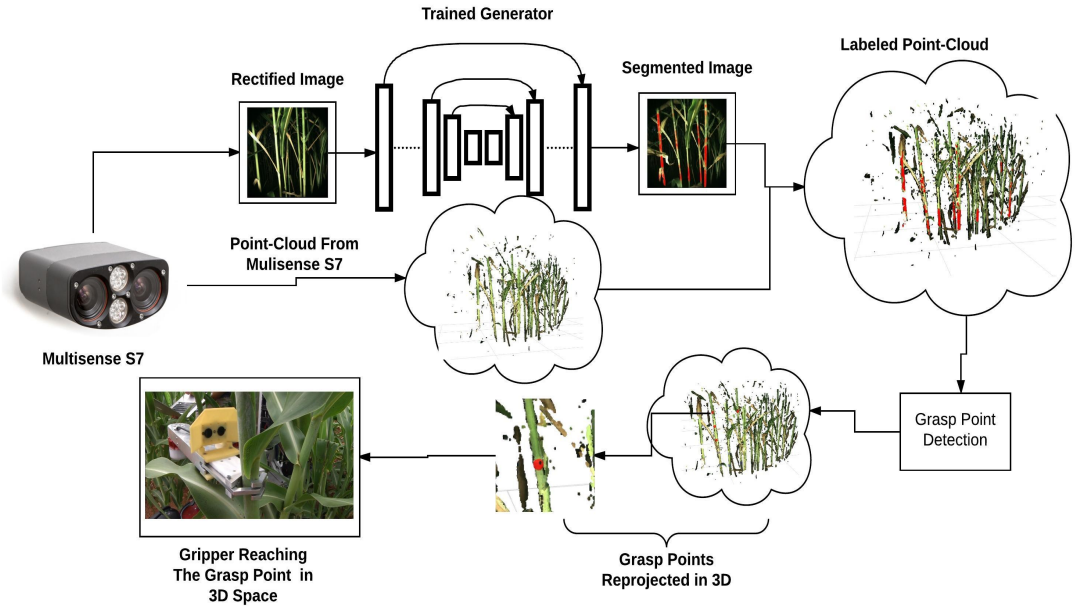


Figure 4.3: The process pipeline: The left image from the Multisense S7 is fed into the trained generator, which masks the stalks in the image with red color. The RGB values of organized point-cloud for the corresponding image is replaced by the output image of generator. Grasp points are detected within this masked point cloud, and the gripper servos to the stalks.

the processed image with segmented stalks. Image processing takes approximately 0.2 seconds per image.

## 4.2 Process Pipeline

### 4.2.1 Perception and grasp point identification

To semantically segment plant stalk pixels in each individual image, we use Conditional GANs. For grasping application using Conditional GANs proved to be better than pixel-wise segmentation networks, as adversarial training regime allows the network to learn the inherent features of target distribution. Hence, it produces more realistic segmentation results with sharper edges delineating stalks from rest of the image. This is understandable, as blob like semantic segmentation output makes the discriminators

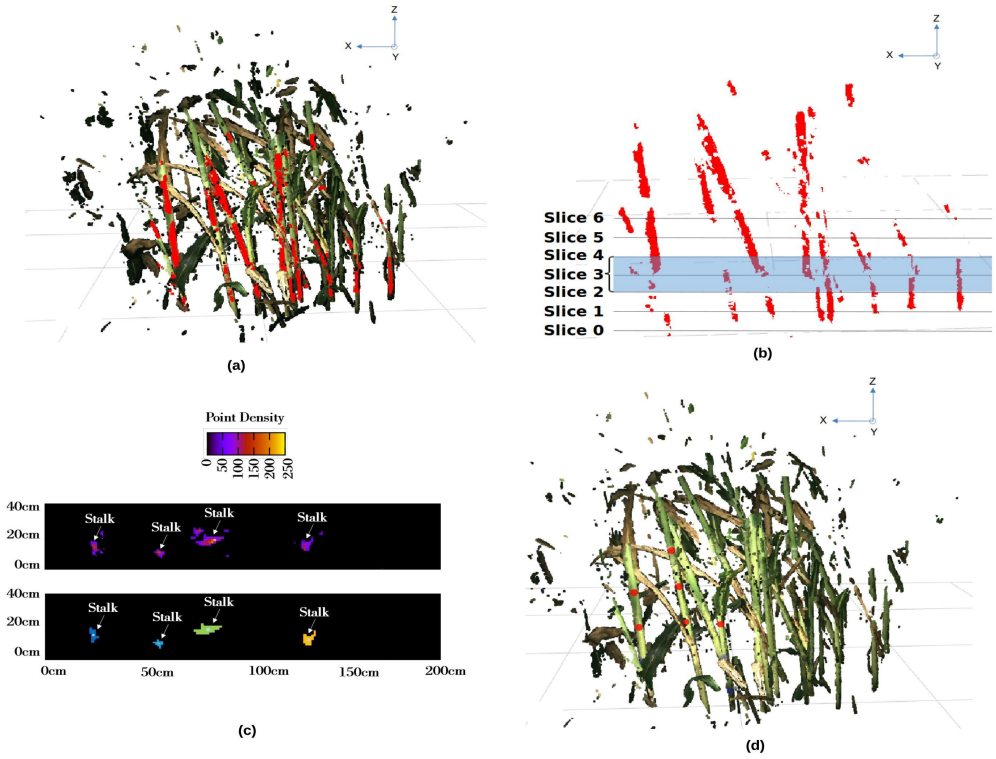


Figure 4.4: (a) Point cloud with masked stalks; (b) Point cloud after thresholding for red color; (c) Result of applying x-y plane projection to a bin in the point cloud. Point density heat-map (top) is filtered and region-growing is applied to segment high density regions (bottom). The centroids of these regions are the prospective grasp points; (d) Shows the detected grasp points (red spheres) within the robot's physical reach, which is only in the left region of the point cloud.

classification task easier.

The Multisense S7 camera uses a hardware trigger to publish an organized point cloud and a rectified image from the left camera and an organized point cloud. Each point in the organized point cloud has a corresponding pixel in the left camera frame. The image from the left camera frame is sent to the conditional GAN, which returns an image where all of the stalks are labelled red. The pixel locations from the labeled image are copied to the point cloud so that the stalks are labelled red in the point cloud.

In order to overcome the challenges of visual occlusion, 5 consecutive point clouds

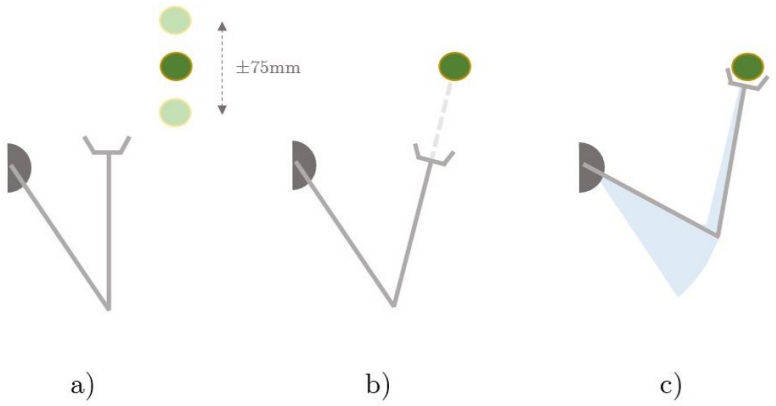


Figure 4.5: Manipulator trajectory outline

are stitched together. The point clouds are stitched using the Iterative Closest Point (ICP) algorithm, which is implemented natively in Point Cloud Library (PCL). The maximum correspondence distance between points was set to 0.15m, and the cloud was downsampled using a 3cm x 3cm x 3cm voxel grid for faster correspondence search. Once the clouds are stitched, they are thresholded for red color. We then deploy the technique used by Jenkins *et. al.* [39] wherein the point cloud is divided into 0.2m segments and each segment is projected onto the x-y plane. Points on the x-y plane associated with stalks tend to be clustered together due to a stalk's vertical profile. After 2D projection and region growing is applied to every slice, the centroids of each stalk region are reprojected to 3D space in the robot frame. Figure 4.3 illustrates the detection pipeline and Figure 4.4 illustrates the method of slicing a point cloud and region-growing the 2D projection.

### 4.2.2 Servoing

The 3D centroids described in the previous section represent points on sorghum stalks to which the manipulator can servo. Trajectory optimization takes approximately 12 microseconds and merely consists of parameterizing a 2D line in the manipulator plane between the manipulator's first joint and the target.

The arm trajectory is controlled using PID on position error and joint velocity is controlled proportional to position error. Error estimation takes place at approximately 250Hz and arm movements take an average of 2s to reach a target. The

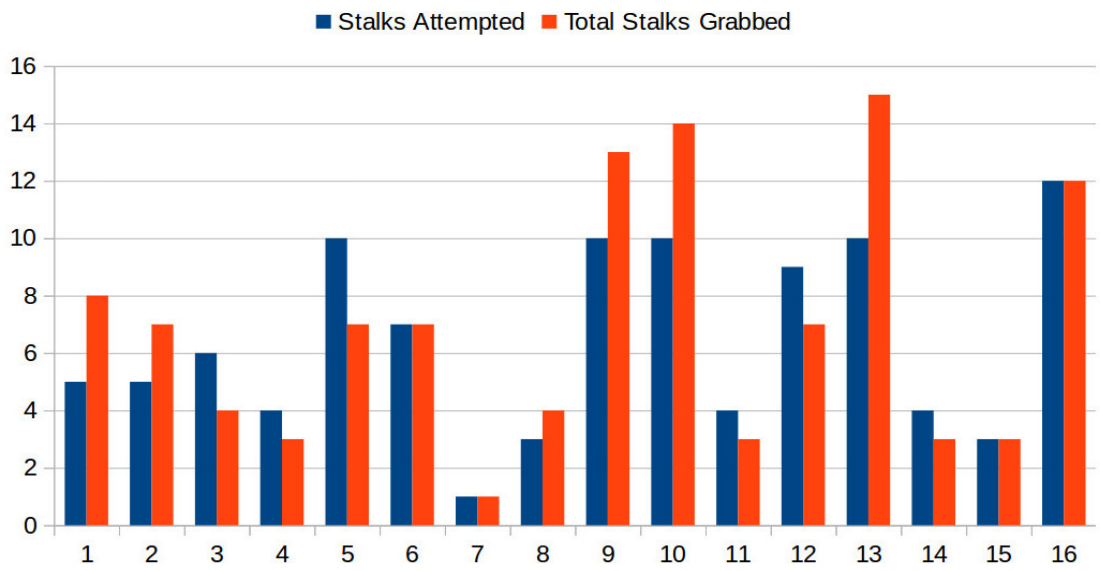


Figure 4.6: Number of stalks successfully grasped vs stalks number of grasps attempted

motor actuating the linear stage is controlled with a PI-controller. This is because the moment of inertia of the carriage on the linear stage is very small and a derivative term is not necessary to prevent overshoot. All three motors were tuned manually using the quarter amplitude decay method. Fig. 4.5 illustrates the planar trajectory of the manipulator.

## 4.3 Results for stalk grasping

The robot was deployed in Florence, South Carolina, USA, on sorghum plants grown for bioenergy. The plants were 90 days old and approximately 2.5m tall. The robot was tested in 16 individual stalk grasping experiments. Each experiment consisted of the robot autonomously driving 0.5m down a sorghum row while running the end-to-end stalk detection and grasping pipeline. While driving down a sorghum row, the system captured five stereo pairs and autonomously servoed the manipulator to every grasp point detected by the Conditional GAN pipeline.

The stereo camera's field of view is wider than the reach space of the manipulator, so not all stalks detected in the camera image are reachable. Therefore we define grasping accuracy as the absolute difference of number of stalks grasped and number

of stalks attempted, divided by the number of stalks attempted: $1 - \frac{abs(\text{error})}{\text{number of attempts}}$. The average grasping accuracy over all 16 experimental runs was 74.13%. Fig. 4.6 shows the results for all 16 experiments. We report accuracy for number of stalks attempted vs. number of stalks reached, rather than the number of detected stalks vs. number of stalks grasped. This is because not all detected stalks are in the reachable proximity of the gripper. As observed in Fig. 4.6, it is often the case that the number of stalks grasped is greater than number of grasps attempted, i.e. the gripper often grips more than one stalk in an attempt. This is a result of high stalk density and lack of collision avoidance in the gripper trajectory planning in our current approach.

# Chapter 5

# Reinforcement Learning with visual observation

The last chapter explored how we can use deep learning techniques for perception and classical methods for manipulation. However using non-learning based techniques for manipulation present several challenges in the field conditions. Decoupling perception and planning for manipulation creates a latency issue in highly dynamic environments resulting in several failure cases that are impossible to recover from due to lack of feedback. This is especially true for agricultural robotics where while trying to manipulate the desired object, the system often ends up altering the environment itself, thus rendering the model of the world created by perception module obsolete. The solution thus requires an end to end reactive approach. Reinforcement learning provides the right framework to tackle this issue. Moreover, using RGB-D observations for learning is beneficial as we can easily get off the shelf camera sensors that can provide high throughput observations for the policy reliably. This approach presents two challenges :-

- How to give rewards to a policy without ground truth state?
- How to make learning more sample efficient when using visual observation?

## 5.1 Reinforcement Learning without Ground-Truth State

Making an accurate state estimator without extensive use of sophisticated sensors in a real world environment is extremely hard. In case of deformable objects, pose is often not descriptive enough and it is hard to determine what features of the state are important.Thus we often resort to using policy learning that maps directly from high dimensional observations to actions. We still need a way to give the learning algorithm rewards in this scenario. We show that using an indicator reward function combined with goal relabelling for goal-conditioned reinforcement learning that gives a positive reward when the robot's observation exactly matches a target goal observation works produces equivalent results as compared to a reward function with access to ground truth state.

### 5.1.1 Formulation

In reinforcement learning, an agent interacts with the environment over discrete time steps. In each time step $t$, the agent observes the current state $s_t$ and takes an action $a_t$. In the next time step, the agent transitions to a new state $s_{t+1}$ based on the transition dynamic $p(s_{t+1}|s_t, a_t)$ and receives a reward $r_{t+1} = r(s_t, s_{t+1})$. The objective for the agent is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected future return $R = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\big]$, where $\gamma$ is a discounted factor in the range of $[0, 1]$.

#### 5.1.1.1 Goal-reaching Reinforcement Learning

In order for the agent to learn diverse and general skills, we define the goal reaching problem as follows. In the beginning of each episode, a goal state $s_g$ is sampled from a goal distribution $\mathcal{G}$. We learn a goal conditioned policy $\pi(a_t|s_t, s_g)$ that tries to reach any goal state from the goal distribution. As such, we use a goal conditioned reward function $r_t = r(s_{t+1}, s_g)$ and optimize for $\mathbb{E}_{s_g \sim \mathcal{G}}\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]$. The transition dynamics $p(s_{t+1}|s_t, a_t)$ of the environment remains independent to the goal.

In many real-world scenarios, it is often difficult to construct a well-shaped reward function. We thus define a sparse reward function that only makes the binary decision

of whether the goal is reached or not. Specifically, let $S_+(s_g)$ be a subset of the state space such that any state in this set is determined to be sufficiently close to $s_g$; in other words, if the environmental state is within $S_+(s_g)$, then the task of reaching $s_g$ can be considered to be achieved. Note that $S_+(s_g)$ can be defined to be an arbitrarily small set, depending on the task specifications.[1] Naturally, we can assume that $s_g \in S_+(s_g)$. A binary reward function can then be defined as

$$r(s_{t+1}, s_g) = \begin{cases} R_+ & s_{t+1} \in S_+(s_g) \\ R_- & s_{t+1} \notin S_+(s_g), \end{cases} \tag{5.1}$$

where $R_+$ and $R_-$ are constants representing the rewards received for achieving the goal and failing to achieve the goal, respectively.

### 5.1.2 Approach

#### 5.1.2.1 Proxy Reward Functions

In many robotic applications, we do not have access to the true state of the environment $s_t$. We might instead have a noisy or high-dimensional sensor (such as a camera) from which we record data. From this sensor, we can try to estimate the state, though this estimate will be noisy, leading to a noisy reward signal.

In such cases, because we do not have access to the true state, we cannot directly use the true reward function defined in Equation 5.1. Instead, we have noisy observations $o_t$, from which we must instead define a proxy reward function $\hat{r}(o_{t+1}, s_g)$. The question now becomes how to choose $\hat{r}$ to be optimal for reinforcement learning, i.e. which choice of $\hat{r}$ will lead to the fastest learning and most accurate policy, when $\hat{r}$ is used in the context of a reinforcement learning algorithm?

The most common approach in robotics is to perform state estimation. Let us define an (unknown) function $f$ that maps an observation $o_t$ to its corresponding ground-truth state $s_t = f(o_t)$. However, since we do not observe the ground-truth state, we must instead infer a noisy estimate of the state $\hat{s}_t = \hat{f}(o_t)$. This noisy state

[1] $S_+$ is a function that maps from the state space to a subset of the space.

can then be used as if it were the true state in equation 5.1:

$$r(\hat{s}_{t+1}, s_g) = \begin{cases} R_+ & \hat{s}_{t+1} \in S_+(s_g) \\ R_- & \hat{s}_{t+1} \notin S_+(s_g), \end{cases} \tag{5.2}$$

However, this approach has two potential issues: first, the state estimator might be noisy, leading to a noisy reward signal. This is especially true in cluttered environments with many occlusions, in which an object detector will often produce many false positives.

Second, in many cases, the state estimator itself might be hard to obtain. This is especially the case for deformable object manipulation, in which the state of the deformable object (e.g. the pose of a rope or cloth) must be estimated. Performing non-rigid registration to a deformabe model of the object being manipulated is often slow and can be noisy[35, 45, 77]. Even in the case of rigid objects, training an object detector and pose estimator typically requires many human-annotated training examples, and using an insufficient number of training examples will lead to noisy state estimation and noisy rewards.

We therefore investigate whether there is another choice of reward function that can be used in cases where a state-estimation based reward function would be either too noisy (due to noisy state estimation) or is not available (due to a lack of a readily-available state estimator, such as for deformable object manipulation or for manipulating a novel object for which a detector and pose estimator have not been trained). Specifically, let us consider a general reward function of the form

$$\hat{r}(o_{t+1}, o_g) = \begin{cases} R_+ & o_{t+1} \in \hat{O}_+(o_g) \\ R_- & o_{t+1} \notin \hat{O}_+(o_g), \end{cases} \tag{5.3}$$

where $o_g$ is a representation of the goal in observation space and $\hat{O}_+(o_g)$ is a subset of the observation space for which we will give positive rewards. For example, if we define $\hat{O}_+(o_g)$ using noisy state estimation, i.e. $\hat{O}_+(o_g) = \{o_t : \hat{f}(o_t) \in S_+(s_g)\}$, then we would recover the reward function of Equation 5.2; however other choices of $\hat{O}_+(o_g)$ are also possible. Our question then becomes how to optimally choose $\hat{O}_+(o_g)$ such that our policy will train the fastest when trained with rewards of $\hat{r}(o_{t+1}, o_g)$.

We propose using a proxy reward function that does not have any false positive rewards. To do so, we will use an extreme reward function of $\hat{O}_+(o_g) = \{o_g\}$. In other words, we will use an indicator reward function:

$$\hat{r}_{ind}(o_{t+1}, o_g) = \begin{cases} R_+ & o_{t+1} = o_g \\ R_- & o_{t+1} \neq o_g, \end{cases} \tag{5.4}$$

It should be clear that this reward function will have no false positives, since the reward is positive only if $o_{t+1} = o_g$, which implies that $f(o_{t+1}) = f(o_g)$, or equivalently, $s_{t+1} = s_g$. Since $s_g \in S_+(s_g)$ by definition, then all positive rewards are true positives, and there will be no false positive rewards under this indicator reward function. However, this reward function will be extremely sparse and will have many false negatives. In fact, without goal relabeling, we would expect all rewards to be negative under this indicator reward function, since, no two observations in continuous spaces will ever be identical. However, we will next describe how to learn with this reward function, even in continuous state spaces, with goal relabeling.

### 5.1.3 Goal Relabeling for Off-policy learning

As mentioned above, the proposed indicator reward function cannot be directly applied in continuous observation spaces due to the lack of positive rewards, since the current observation will never be identically equal to the goal observation. Fortunately, in the off-policy learning case, we can adopt the goal-relabeling technique introduced in [4, 40] to learn the goal-conditioned Q function.

Suppose that some transitions $(o_t, a, o_{t+1})$ were observed when the agent took an action $a \sim \pi(o_t, o_g)$ with a goal of $o_g$. When we train our critic with this transition, we use an off-policy reinforcement learning algorithm such as Q-learning (or DDPG in continuous state and action spaces), which directly minimizes the Bellman error:

$$E_{o_t, a_t \sim \pi, o_g \sim \mathcal{G}} \Big[ Q(o_t, a_t, o_g) - \big( \hat{r}_{ind}(o_{t+1}, o_g) + \gamma Q(o_{t+1}, \pi(o_{t+1}, o_g), o_g) \big) \Big]. \tag{5.5}$$

Because Q-learning is an off-policy reinforcement learning algorithm, we can replace the goal observation $o_g$ with any other observation $o_{g'}$ in our Bellman update of the Q-function. Specifically, with some probability we will choose to replace $o_g$ with

the observation $o_{t+1}$. By re-labeling $o_g$ with $o_{t+1}$, then using our indicator reward function, $\hat{r}_{ind}(o_{t+1}, o_g) = \hat{r}_{ind}(o_{t+1}, o_{t+1}) = R_+$. Thus, using goal relabeling, we can achieve positive rewards, even using an indicator reward function in continuous state spaces.

### 5.1.4 Reward Balancing and Filtering

After sampling a batch of data, we train the Q-function with goal relabeling. Let $p_1$ and $p_2$ determine the relative frequency between providing positive rewards and propagating rewards to other timesteps in the episode , we use three different strategies for goal relabeling: with probability $p_1 p_2$, we relabel $o_g \leftarrow o_{t+1}$, which will receive a positive reward under our indicator reward function. With probability $p_1(1 - p_2)$, we relabel the goal $o_g \leftarrow o_{t'}$ with an observation from some future time $t'$ step within the episode. The indicator reward function will most likely give a negative reward in this case, which is possibly a false negative. Finally, with probability $1 - p_1$, we use the original goal (with no relabeling), which will again most likely give a negative reward under the indicator reward function; as before, this might be a false negative.

We refer to "reward balancing" as setting $p_1 = 0.9$ and $p_2 = 0.5$, leading us to receive positive rewards approximately 0.45 of the time and negative rewards approximately 0.55 of the time. Thus the ratio of positive and negative rewards that we use to train the Q-function are approximately balanced, even with indicator rewards.

We experimented with removing the second type of goal relabeling, i.e. with probability 0.45 of relabeling $o_g \leftarrow o_{t+1}$ and otherwise using the original goal without relabeling. More specifically, this is equivalent to setting $p_1 = 0.45$ and $p_2 = 1$. We found that this leads to much worse performance. One explanation is that training with the goal relabeled to a future state in the episode helps to propagate the positive reward to other states in the episode, using the Bellman update. This is similar to the relaxation operation used in solving shortest path problems. With this perspective, $p_1$ can be seen as a weighting factor between providing positive rewards and propagating rewards. In future work, it might be interesting to explore the relationship between $p_1$ and other factors relating to value iteration, such as length of the episode, usage of N-step returns, and prioritized replay.

We also experimented with setting $p_1 = 1$, e.g. only using relabeled goals taken from observations from the same episode. We found that this also leads to much worse performance; the optimization becomes stuck in a trivial solution where the agent takes no action and stays at the same observation.

While false negative rewards do not adversely affect learning significantly, we still wish to avoid them if possible to improve the convergence time of the learned policy. We achieve this using "reward filtering," in which we filter out transitions that we suspect of having a high chance of being false negatives.For a given transition $(o_t, a_t, o_{t+1})$, if $o_{t+1} = o_g$, we know that $\hat{r}_{ind}(o_{t+1}, o_g) = R_+$. In this case, $Q^*(o_t, a_t, o_g) = R_+/(1-\gamma)$, where $Q^*$ is the optimal Q-function. Similarly, if $o_{t+1} \neq o_g$, then $\hat{r}_{ind}(o_{t+1}, o_g) = R_-$. Since we know that the policy starting from $o_t$ will thus receive at least one negative reward before receiving positive rewards, then $Q^*(o_t, a_t, o_g) \leq R_- + \gamma R_+/(1-\gamma)$. Thus, we can set a threshold $q_0$, where $R_- + \gamma R_+/(1-\gamma) < q_0 < R_+/(1-\gamma)$; if we find that $Q(o_t, o_g, a_t) > q_0$, then the corresponding reward $\hat{r}_{ind}(o_{t+1}, o_g)$ is likely to be a false negative (assuming that the Q-function has been trained well); we thus filter out such rewards, to reduce the number of false negatives that we use for training.

### 5.1.5 Experiments

We denote our method, which uses indicator rewards with reward balancing and filtering, as **Indicator+Balance+Filter**. We compare our method with the following methods:

- **Oracle** This method assumes access to the ground truth reward $r(s_t, s_g)$.
- **Auto Encoder** For vision-based tasks, we train an autoencoder with an $L2$ reconstruction loss of the image observation, jointly with the RL agent. We then use cosine similarity in the learned embedding space to provide dense rewards, as similarly compared in [94]. Specifically, assuming the learned encoding of an observation $o$ is $\phi(o)$ after $L2$ normalization, the reward will be $r(o, o_g) = max(0, \phi(o)^T \phi(o_g))$, similar to the baseline compared in [18].
- **Indicator** This is the same as our method, without reward balancing and filtering.

We use the standard off-policy learning algorithm DDPG [49] with goal relabeling [4]. For the baselines, we use the goal-relabeling and sampling strategy that result in

the best performance. Specifically, for all these methods, with a probability of 0.9, we relabel the current goal with an achieved goal sampled uniformly from one of the future time steps within the same episode, otherwise, the original goals are used. For all the environments, the ground truth rewards are based on the $L_2$ distance in the state space:

$$r(s_{t+1}, s_g) = \begin{cases} R_+ & ||s_{t+1} - s_g|| \leq \epsilon \\ R_- & o.w. \end{cases} \tag{5.6}$$

#### 5.1.5.1 Learning with Indicator Rewards

We first evaluate all the methods in simulated environments in MuJoCo [92], where all the methods receive the state representation as input:

- Reacher: Teach a two-link arm to reach a randomly located position in 2D space.
- FetchReach: Move the end effector of the Fetch robot to a random position in 3D.

Both environments above are relatively easy, standard environments from Gym [60].

We test all the algorithms in the Reacher and FetchReach environments where both current and goal observation are given in RGBD images. The results are shown in Figure 5.1. While Oracle achieves the best performance, we can see that other methods with indicator rewards are also able to solve the tasks, though converging slower. Compared to the Auto Encoder and Indicator baselines, Indicator+Balance+Filter achieves better performances in FetchReach, both in terms of the success and the final distance to goal. Interestingly, in Reacher environment, although Indicator+Balance+Filter learns slower when evaluated in terms of success rate, it has a lower distance to goal (distance in the state space), suggesting a different learning priority when learning with indicator rewards.

#### 5.1.5.2 Learning with Indicator Rewards from Real Images

Using RGB-D observations and goals, we train a Sawyer robot for 3 dimensional reaching task with Oracle, Indicator rewards and Indicator rewards with balance and filter. The goal observations are sampled by moving the robot arm to a uniformly

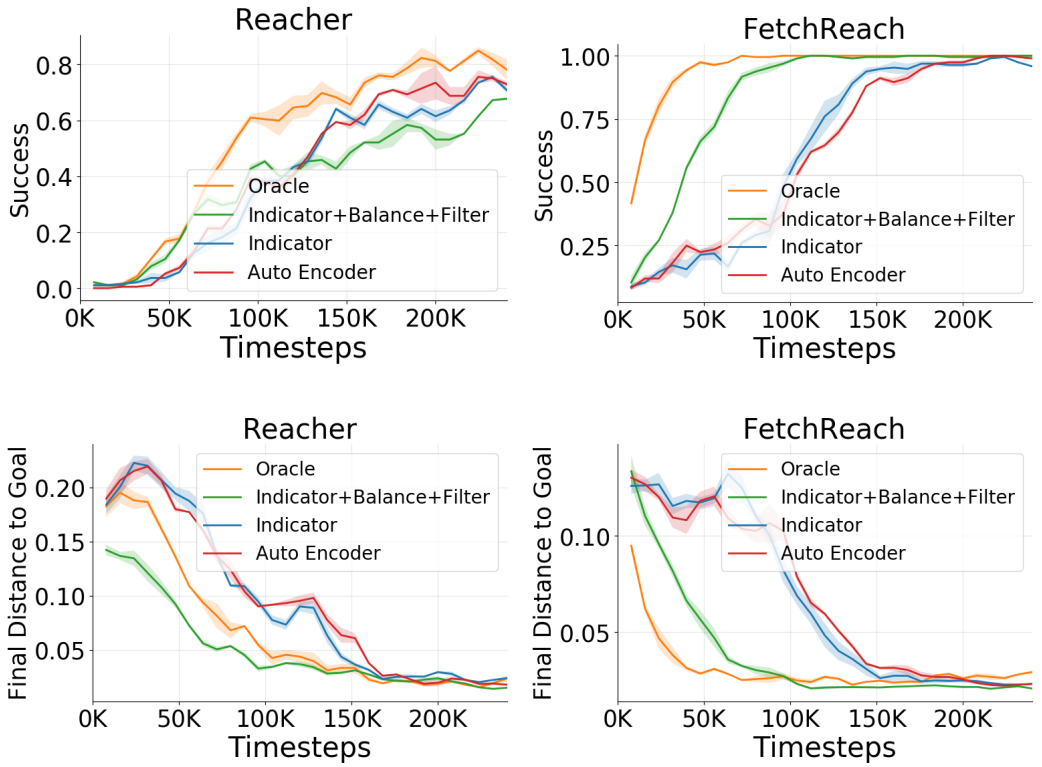


Figure 5.1: The success(first row) and the final distance to goal(second row) of different methods in different environments throughout the training. The observation are based on the RGBD images rendered in simulation.

sampled location in a cuboid of diagonal length 1.3m and the episode was considered successful if the end effector was within 0.1m from goal end effector location at the end of the episode with fixed time horizon of 25 steps. The trained policy performs position control and outputs end effector displacement within a range of -0.05m to 0.05m in each direction. Figure 5.2 shows the training curves for using the three rewards. Indicator+Balance+Filter reward is comparable to Oracle both in terms of success rate and final goal distance while it outperforms just the Indicator reward in both these metrics.

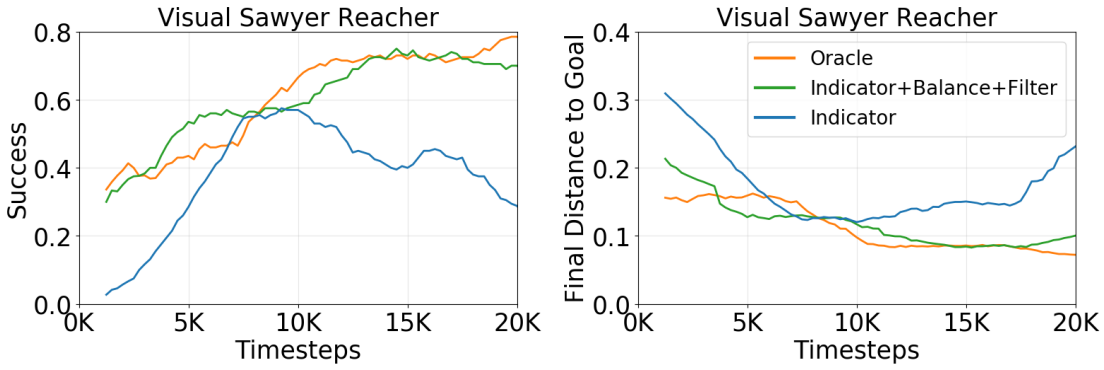


Figure 5.2: Success and final distance to goal on learning visual reacher on a physical Sawyer robot.

## 5.2 Accelerating Reinforcement Learning using Auxiliary Losses

Reinforcement learning is known to be sample inefficient, preventing its application to many real-world problems, especially with high dimensional observations like images. Learning auxiliary tasks along with the reinforcement learning objective could be a powerful tool to improve the learning efficiency. However, the usage of auxiliary tasks has been limited so far due to the difficulty in selecting and combining different auxiliary tasks. In this work, we propose a principled online learning algorithm that dynamically combines different auxiliary tasks to speed up training for reinforcement learning. We argue that good auxiliary tasks should provide gradient directions that, in the long term, help to decrease the loss of the main task. Our approach is illustrated in Figure 5.3. We show that our algorithm can effectively combine a variety of different auxiliary tasks and achieves about a 3x speedup compared to using no auxiliary tasks in a set of robotic manipulation environments.

### 5.2.1 Problem Definition

Assume that we have a main task $\mathcal{T}_{main}$ that we want to complete and a set of auxiliary tasks $\mathcal{T}_{aux,i}$, where $i \in \{1, 2, ..., K\}$, that will be used for representation learning. Each task has a corresponding loss $\mathcal{L}_{main}$ and $\mathcal{L}_{aux,i}$. In the context of

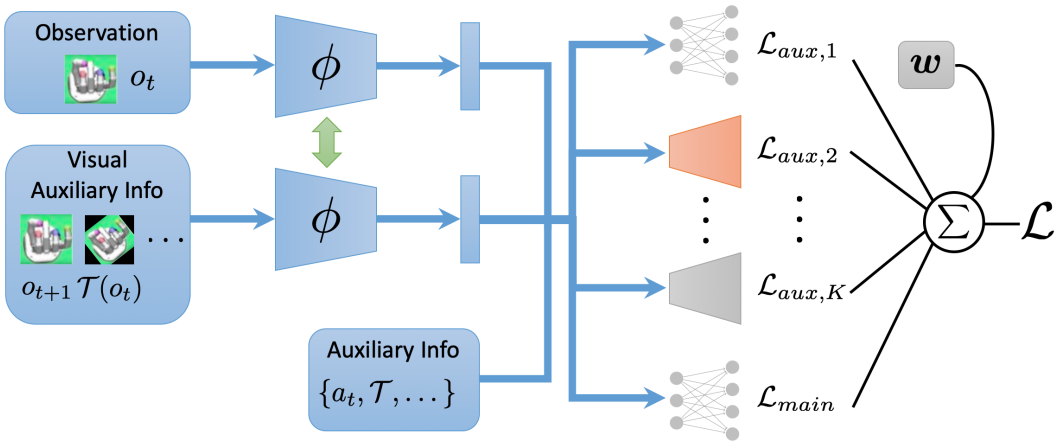


Figure 5.3: An illustration of learning with auxiliary tasks. All of the visual observations and any auxiliary visual information are passed through a shared weight CNN to get a low dimension representation. The representation, along with other auxiliary information are then used to perform different auxiliary tasks, as well as the main task. A final loss is computed by weighting the main loss and all the auxiliary losses.

reinforcement learning, $\mathcal{L}_{main}$ can either be the the policy loss $\mathcal{L}_\pi$ or the Bellman error $\mathcal{L}_Q$, such as for actor-critic methods. The losses are functions of the model parameters $\theta_t$ at each training time step $t$.

Our goal is to optimize the main loss $\mathcal{L}_{main}$. However, using gradient-based optimization with only the main task gradient $\nabla_\theta \mathcal{L}_{main}$ is often slow and unstable, due to the high variance of reinforcement learning. Thus, auxiliary tasks are commonly used, especially for image based tasks, to help to learn a good feature representation. We can combine the main loss with the loss from the auxiliary tasks as

$$\mathcal{L}(\theta_t) = \mathcal{L}_{main}(\theta_t) + \sum_{i=1}^{K} w_i \mathcal{L}_{aux,i}(\theta_t), \tag{5.7}$$

where $w_i$ is the weight for auxiliary task $i$ and $\theta_t$ is the set of all model parameters at training step $t$. We assume that we update the parameters $\theta_t$ using gradient descent on this combined objective:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t). \tag{5.8}$$

If a large number of auxiliary tasks are used, some auxiliary tasks may be more beneficial than others for learning a feature representation for the main task; thus the weights $w_i$ of each auxiliary task (Eqn. 5.7) need to be tuned. Previous work manually tunes the auxiliary task weights $w_i$ [36]. However, a number of issues arise when we try to scale the number of auxiliary tasks. First, tuning the parameters $w_i$ becomes more computationally intensive as the number of auxiliary tasks $K$ increases. Second, if the values of $w_i$ are learned via hyperparameter optimization, then the reinforcement learning optimization must be run to near-convergence multiple times to determine the optimal values of $w_i$; ideally the weights would be learned online so that the reinforcement learning optimization only needs to be performed once. Last, the importance of each auxiliary task, and hence the optimal weight $w_i$, might change throughout the learning process; using a fixed value for $w_i$ might limit the performance.

### 5.2.2 Approach

We propose to dynamically tune the weights for each auxiliary task. We will first describe an approach that uses a one-step gradient in Section 5.2.2.1; we will then extend this framework in Section 5.2.2.2.

#### 5.2.2.1 Local Update from One-step Gradient

In our initial approach, we aim to find the weights for the auxiliary tasks such that $\mathcal{L}_{main}$ decreases the fastest. Specifically, define $\mathcal{V}_t(\boldsymbol{w})$ as the speed at which the main task loss decreases at the time step $t$, where $\boldsymbol{w} = [w_1, ..., w_k]^T$. We then have

$$\begin{aligned}
\mathcal{V}_t(\boldsymbol{w}) = \frac{d\mathcal{L}_{main}(\theta_t)}{dt} &\approx \mathcal{L}_{main}(\theta_{t+1}) - \mathcal{L}_{main}(\theta_t) \\
&= \mathcal{L}_{main}(\theta_t + \alpha\nabla_{\theta_t}\mathcal{L}(\theta_t)) - \mathcal{L}_{main}(\theta_t) \\
&\approx \mathcal{L}_{main}(\theta_t) + \alpha\nabla_{\theta_t}\mathcal{L}_{main}(\theta_t)^T\nabla_{\theta_t}\mathcal{L}(\theta_t) - \mathcal{L}_{main}(\theta_t) \\
&= \alpha\nabla_{\theta_t}\mathcal{L}_{main}(\theta_t)^T\nabla_{\theta_t}\mathcal{L}(\theta_t),
\end{aligned} \tag{5.9}$$

where $\alpha$ is the gradient step size. The first line is obtained from a finite difference approximation of the time derivative, with $\Delta t = 1$ (where $t$ is the iteration number of

the learning process). We then write out $\theta_{t+1}$ using Eqn. 5.8; followed by a first-order Taylor approximation.

To update $\boldsymbol{w}$, we can simply calculate its gradient:

$$\frac{\partial \mathcal{V}_t(w_i)}{\partial w_i} = \alpha \nabla_{\theta_t} \mathcal{L}_{main}(\theta_t)^T \nabla_{\theta_t} \mathcal{L}_{aux,i}(\theta_t), \forall i = 1, ..., K. \tag{5.10}$$

This leads to an update rule that is based on the dot product between the gradient of the auxiliary task and the gradient of the main task. An intuitive explanation would be, up weight an auxiliary task if its gradient aligns with the main task and down weight it otherwise. The form of this equation resembles recent work which uses a thresholded cosine similarity to determine whether to use each auxiliary task [17]; however, our update rule is a product of our derivation of maximizing the speed at which the main task loss decreases.

#### 5.2.2.2 N-step Update

The gradient in Eqn. 5.10 is optimized for the instantaneous update rate of the main task, $d\mathcal{L}_{main}(\theta_t)/dt$, as shown in Equation 5.9. However, we are not actually concerned with the one-step update of the main task loss $\mathcal{L}_{main}(\theta_t)$; rather, we are concerned with the long-term value of $\mathcal{L}_{main}(\theta_t)$ after multiple gradient updates. In this section, we extend the method of the previous section to obtain an optimization objective for $\boldsymbol{w}$ that accounts for the performance on $\mathcal{L}_{main}(\theta_t)$ in the longer term.

Since the loss changes in one step does not necessarily reflect the long-term performance, we instead seek to optimize the N-step decrease of the main task loss:

$$\mathcal{V}_t^N(\boldsymbol{w}) = \mathcal{L}_{main}(\theta_{t+N}) - \mathcal{L}_{main}(\theta_t).$$

However, exact computation of the gradient with respect to $\boldsymbol{w}$ requires calculating higher order Jacobians, which can be computationally expensive. We thus adopt a

first-order approximation:

$$\begin{aligned}\mathcal{V}_t^N(\boldsymbol{w}) &\doteq \mathcal{L}_{main}(\theta_{t+N}) - \mathcal{L}_{main}(\theta_t) \\ &= \mathcal{L}_{main}\Big(\theta_{t+N-1} + \alpha\nabla_{\theta_{t+N-1}}\mathcal{L}(\theta_{t+N-1})\Big) - \mathcal{L}_{main}(\theta_t) \\ &\approx \mathcal{L}_{main}(\theta_{t+N-1}) - \mathcal{L}_{main}(\theta_t) + \alpha\nabla_{\theta_{t+N-1}}\mathcal{L}_{main}(\theta_{t+N-1})^T\nabla_{\theta_{t+N-1}}\mathcal{L}(\theta_{t+N-1}) \\ &\vdots \\ &\approx \alpha\sum_{j=0}^{N-1}\nabla_{\theta_{t+j}}\mathcal{L}_{main}(\theta_{t+j})^T\nabla_{\theta_{t+j}}\mathcal{L}(\theta_{t+j})\end{aligned} \tag{5.11}$$

resulting in,

$$\mathcal{V}_t^N(\boldsymbol{w}) \approx \alpha\sum_{j=0}^{N-1}\nabla_{\theta_{t+j}}\mathcal{L}_{main}(\theta_{t+j})^T\nabla_{\theta_{t+j}}\mathcal{L}(\theta_{t+j}) \tag{5.12}$$

Next, we want to update $\boldsymbol{w}$ by calculating $\nabla_{\boldsymbol{w}}\mathcal{V}_t^N(\boldsymbol{w})$, which requires differentiating through the optimization process. To avoid this cumbersome computation in a online process, we ignore all the higher order terms, essentially assuming that a small perturbation of $\boldsymbol{w}$ only affects the immediate next step. With this approximation, we get that $\forall i = 1, ..., K$:

$$\nabla_{w_i}\mathcal{V}_t^N(w_i) \approx \alpha\sum_{j=0}^{N-1}\nabla_{\theta_{t+j}}\mathcal{L}_{main}(\theta_{t+j})^T\nabla_{\theta_{t+j}}\mathcal{L}_{aux,i}(\theta_{t+j}). \tag{5.13}$$

We call this approach Online Learning for Auxiliary losses (OL-AUX). The full algorithm is described in Algorithm 1. As an implementation detail, to balance the norm of the gradient between different losses, we adopt the Adaptive Loss Balancing technique [34] and wrap all the auxiliary task losses inside a log operator. Figure 5.3 provides an illustration of the pipeline for computing all the individual losses that constitute $\mathcal{L}(\theta_t)$.

The benefit of the N-step update, compared to the one-step update, comes from two sources. First, as shown in Eqn. 5.11, the N-step objective considers the long-term effect on the main task loss of updating the weights $\boldsymbol{w}$, which aligns with our longer term goal. Second, as shown in Eqn. 5.13, the N-step method also averages the

**Algorithm 1** Learning with OL-AUX

**Input:**
Main task loss: $\mathcal{L}_{main}$
K auxiliary task losses: $\mathcal{L}_{aux,1}, \ldots, \mathcal{L}_{aux,K}$
Horizon $N$
Step size $\alpha, \beta$
**Initialize** $\theta_0, \boldsymbol{w} = \mathbf{1}, t = 0,$
**for** $i = 0$ **to** $TrainingEpoch - 1$ **do**
  Collect new data using $\theta_t$
  **for** $j = 0$ **to** $UpdateIteration - 1$ **do**
    $t \leftarrow i \cdot UpdateIteration + j$
    Sample a mini-batch from dataset
    $\mathcal{L}(\theta_t) \leftarrow \log \mathcal{L}_{main}(\theta_t) + \sum_{i=1}^{K} w_i \log \mathcal{L}_{aux,i}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t)$
    **if** $t + 1 \bmod N == 0$ **then**
      $\nabla_{w_i} \mathcal{V}_{t-N+1}^{N}(w_i) \leftarrow \alpha \sum_{j=0}^{N-1} \nabla_{\theta_{t-j}} \log \mathcal{L}_{main}(\theta_{t-j})^T \nabla_{\theta_{t-j}} \log \mathcal{L}_{aux,i}(\theta_{t-j})$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} - \beta \nabla_{\boldsymbol{w}} \mathcal{V}_{t-N+1}^{N}(\boldsymbol{w})$
    **end if**
  **end for**
**end for**

auxiliary weight gradient over more $\theta$ update iterations, which will compute a less noisy gradient. Ablation experiments are shown in Section 5 to differentiate between these effects, and we show that both of these effects contribute to our performance.

### 5.2.3 Experiments

In the following experiments, we aim to answer the following questions:

- How much can we improve sample efficiency by leveraging a set of diverse auxiliary tasks?
- Is dynamically tuning the weights of the auxiliary tasks important to the achieved sample efficiency, compared to using a fixed set of weights?
- Is it beneficial to adapt the auxiliary task weight based on its longer term effect, i.e. N-step update (Section 5.2.2.2) compared to the 1-step update (Section 5.2.2.1)?

To answer these questions, we empirically evaluate different approaches on a few

goal-oriented reinforcement learning environments with visual observations, where the issue of sample complexity is exacerbated due to the high dimensional input. We use DDPG [49] with hindsight experience replay [4] as our base learning algorithm.

#### 5.2.3.1 Auxiliary Tasks

In the context of manipulation from visual observations, we consider the set of auxiliary tasks briefly described in Table 5.1.

1. **Forward Dynamics** [3]: This task enforces forward consistency in the learned latent representation. Given the current visual observation and the action, the network is asked to predict the latent representation of the next state. The loss is defined as the following:

$$\mathcal{L}_{fk} = ||f_{fk}(e(o_t;\phi), a_t; \phi_{fk}) - e(o_{t+1};\phi)||_2^2,$$

   where $f_{fk}$ and $e$ are the latent space forward model and the CNN encoder respectively, $o_t$ is an observation, and $a_t$ is an action.

2. **Inverse Dynamics** [3]: Given two consecutive image observations, this task predicts the action taken. The loss is specified by

$$\mathcal{L}_{ik} = ||f_{ik}(e(o_t;\phi), e(o_{t+1};\phi); \phi_{ik}) - a_t||_2^2.$$

   where $f_{ik}$ is the latent space inverse dynamics model.

3. **Egomotion** [2]: Given an image observation and a random transformation of this image, the network needs to predict the performed transformation. This task forces the network to learn visual correspondences between the transformed image and the original image. In our experiments, the transformation is constrained to be a planar rotation with a degree of $\theta \in [-30°, 30°]$. The transformed image is then clipped and scaled to have the same size as the original image. The input to the egomotion prediction network also shares the convolutional features with other tasks, including the main task. The loss is defined as:

$$\mathcal{L}_{eg} = ||f_{eg}(e(o_t;\phi), e(\mathcal{T}o_t;\phi); \phi_{eg}) - \theta||_2^2,$$

where $\mathcal{T}$ is the transformation.

4. **Autoencoder**: This task aims to reconstruct the image observation given the latent representation $e(o;\phi)$. It enforces a representation that preserves the information in the original observation as much as possible. The loss is defined as:

$$\mathcal{L}_{ae} = ||f_{ae}(e(o_t;\phi));\phi_{ae}) - o_t||_2^2$$

5. **Optical Flow** [31]: Between every two consecutive visual observations, we first compute the visual representation of optical flow using Farneback's [23] algorithm. Then, the network needs to predict the optical flow result from the latent representation of the two images. This task encourages a latent representation that focuses more on the moving pixels and could be helpful for the object manipulation tasks. The optical flow loss is defined as:

$$\mathcal{L}_{op} = ||f_{op}(e(o_t;\phi), e(o_{t+1};\phi));\phi_{op}) - \text{FK}(o_t, o_{t+1})||_2^2,$$

where $f_{op}$ denotes the optical flow prediction network to be learned and FK denotes the Farneback's algorithm. Target optical flow representation for two NxNxD frames is an NxNx1 mask where the value of each pixel in the mask represents the strength of the flow vector at that location.

| Auxiliary Task | Description |
|---|---|
| Forward Dynamics [3] | Given current visual observation and action, predict latent space representation of next observation. |
| Inverse Dynamics [3] | Given consecutive image observations, predict the action taken. |
| Egomotion [2] | Given raw and transformed visual observation, predict the transformation applied. |
| Autoencoder | Reconstruct visual observation from latent space representation. |
| Optical Flow [31] | Given two consecutive visual observations, predict the optical flow. |

Table 5.1: Brief description of the auxiliary losses used.

#### 5.2.3.2 Environments

We evaluate all methods on robotic manipulation tasks simulated in MuJoCo [92]:

- **Visual Fetch Reach** (OpenAI Gym [67]). The goal is to move the end effector of a Fetch Robot to a randomly sampled 3D location.
- **Visual Hand Reach** (OpenAI Gym [67]). A target hand pose with the positions of the five fingers of a 24 DOF Shadow hand is randomly sampled from 3D space; the policy is required to control the hand to reach the corresponding positions of all five fingers. The policy outputs motors commands for the 24 DOFs of the hand.
- **Visual Finger Turn** (DeepMind Control Suite [89]). A policy needs to control a 3 DOF robot finger to rotate a body on an unactuated hinge. The agent receives a positive reward if the body tip coincides with a randomly sampled target location.

For all the environments, the goal is an RGBD image with objects in the desired configuration. For Visual Fetch Reach and Visual Hand Reach environments, the observation is the current RGBD image, while for Visual Finger Turn, the observation is 3 stacked consecutive RGBD frames.

For all the environments, we use sparse rewards specified by the $L_2$ distance of the underlying ground truth state from the goal state. The rewards are defined as:

$$r(s_{t+1}, s_g) = \begin{cases} R_+ & ||s_{t+1} - s_g|| \leq \epsilon \\ R_- & o.w. \end{cases} \tag{5.14}$$

where both $s_{t+1}$ and $s_g$ are in state space, $R_+ = 1$ and $R_- = -1$. For hindsight experience replay, with a probability of 0.9 we relabel the original goal with another observation from a future time step of the same episode. More details on the environments and the algorithm used can be found in Appendix A and B.

#### 5.2.3.3 Baselines

We compare the following approaches:

1. **No Auxiliary Losses** This is our base learning algorithm without using any

auxiliary tasks.

2. **Gradient Balancing** This baseline combines all the auxiliary tasks with the same weight of 1 but uses adaptive loss balancing [34] to balance the norm of the gradient for different auxiliary tasks.
3. **Cosine Similarity** This baseline combines the gradients from the auxiliary tasks and the main task based on their cosine similarities [17]. Specifically, $\nabla_\theta \mathcal{L}_{aux,i}$ is added to $\nabla_\theta \mathcal{L}_{main}$ to update the shared parameter $\theta$ only if $\cos(\nabla_\theta \mathcal{L}_{main}, \nabla_\theta \mathcal{L}_{aux,i}) \geq 0$.
4. **OL-AUX** This is our method as described in Algorithm 1, with N=5 (OL-AUX-5).

#### 5.2.3.4 Online learning of auxiliary task weighting gives significant improvement

The learning curves of all methods are shown in Figure 5.4. We can see that, in all environments, using auxiliary tasks with proper gradient balancing gives consistent improvement over not using auxiliary tasks. By online updating the weighting of the auxiliary tasks, our method(OL-AUX-5) shows further improvement, achieving around 3x reduction in sample complexity for Visual Hand Reach and Visual Finger Turn tasks and 2x reduction for Visual Fetch Reach.

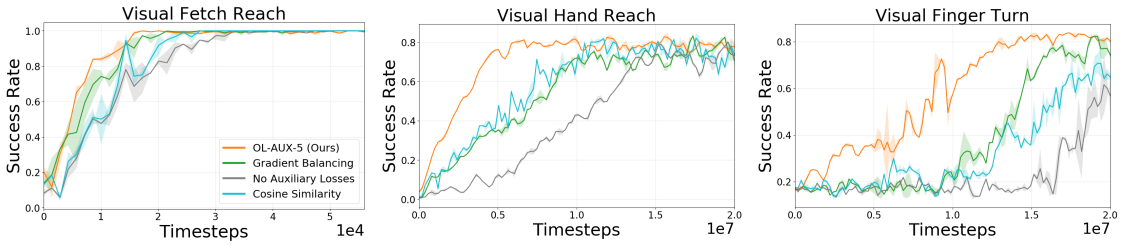


Figure 5.4: The training curves of our method compared to other baselines. The y-axis shows the percentage of the time that the goal is reached.

We further compare with using only a single auxiliary task along with gradient balancing in Figure 5.5. We can see that using a single auxiliary task often only gives marginal improvement over the baseline. On the other hand, our method, by leveraging the combination of all auxiliary tasks, always performs better or as well

as the best single auxiliary task. Additionally, we can see that the importance of the auxiliary task depends on the RL task. For example, inverse dynamics is the best single auxiliary task for the Visual Hand Reach environment but does not help much in the Visual Finger Turn environment. Our method is able to exploit the best combination of the auxiliary tasks without prior knowledge about the relationships among the auxiliary tasks as well the relationship between the auxiliary tasks and the main RL task.

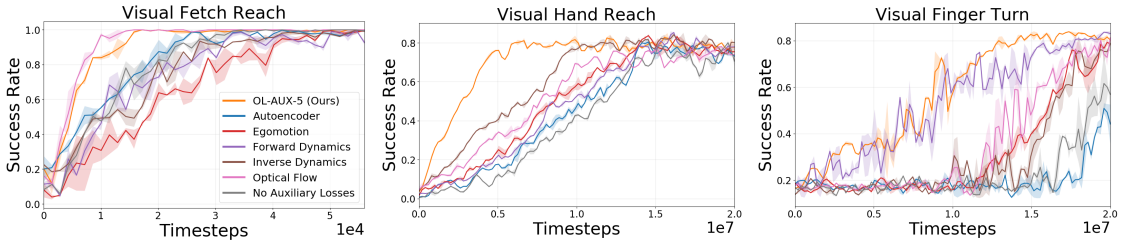


Figure 5.5: The training curve for our method (which combines multiple auxiliary losses) compared to using each individual auxiliary loss one at a time. The y-axis shows the percentage of the time that the goal is reached.

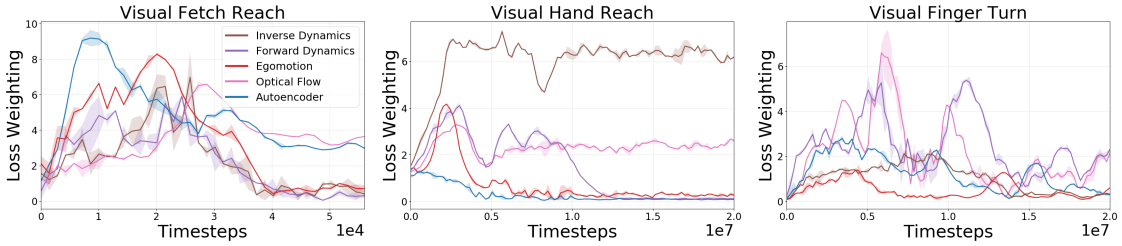


Figure 5.6: Change of the weights of the auxiliary tasks during training.

For our OL-AUX-5 method, we show how the weights of all the auxiliary tasks change during training in Figure 5.6. Looking at the weight of an auxiliary task alongside the single auxiliary task ablation in Figure 5.5, we can see that they often align. For example, inverse dynamics is the best single auxiliary task in Hand Reach and it also retains a large weight when combined with other auxiliary tasks. There are also exceptions: In Finger Turn, optical flow does not work well as a single auxiliary task. But when combined with other auxiliary tasks, it still has the largest weight for a small amount of time; In Hand Reach, optical flow performs well as a single task but when combined with others, the weight is kept at a relatively low level. This

shows that our method is able to take advantage of the auxiliary tasks that best suits the RL task at hand, while taking into consideration the interplay of different auxiliary tasks, without any prior knowledge.

#### 5.2.3.5 Auxiliary task gradients should provide long-term guidance

Our N-step update method incorporates the idea that the auxiliary tasks should be used to decrease the loss of the main task in the long term. To verify that this long-term reasoning is important, we compare OL-AUX-5 with OL-AUX-1 where the weights are updated every time step (as in Sec. 5.2.2.1). For OL-AUX-1, we scale the learning rate $\beta$ down by a factor of 5 to make a fair comparison, as it updates $\boldsymbol{w}$ more frequently. The results are shown in Figure 5.7. As shown, OL-AUX-1 performs much worse than OL-AUX-5, providing evidence of the importance of using auxiliary tasks to provide long-term guidance for reinforcement learning.

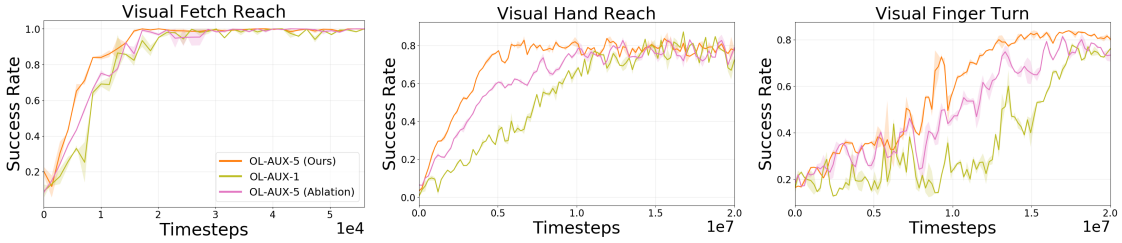


Figure 5.7: Learning curves comparing different ablation methods.

However, as discussed Sec. 5.2.2.2, there are two reasons why our method might outperform the OL-AUX-1 baseline: our method takes into account the long-term effect of the auxiliary weight update on the main objective; also, our method averages the auxiliary weight gradient over more $\theta$ update iterations, which will result in a a less noisy gradient update. To isolate the influence of taking into account the long-term effects, we perform another ablation experiment. In this ablation, we perform a one-step gradient update using Eqn. 5.10, but only update $\boldsymbol{w}$ every N (N = 5) steps. When updating $\boldsymbol{w}$, we use N times as much data to compute the gradient, and we use the same learning rate $\beta$ as OL-AUX-5. This makes the algorithm the same as OL-AUX-5 in terms of update frequency, learning rate, and the amount of data used for the update. The plots from this method are labeled **OL-AUX-5 (Ablation)** in Figure 5.7. The gap between OL-AUX-5 and OL-AUX-5 (Ablation) shows the

effect of "long-term reasoning" while the gap between OL-AUX-5 (Ablation) and OL-AUX-1 shows the effect of "gradient smoothing". We can see that both factors contribute to the gained improvement in training time.

# Chapter 6

# Conclusions

We show that robotics aided by learning based approaches for perception and manipulation can provide a solution for high throughput plant phenotyping and aid the acceleration of plant breeding process. We find the automated measurements are accurate to within 10% of human validation measurements for stalk count and measure stalk width with 2.76 mm on average. Ultimately though, we identify that the human measurements are 30 times slower than the robotic measurements for count and 270 times slower for measuring stalk width over an experimental plot. Using our perception approach with simple inverse kinematics based planning for plant stalk grasping achieves grasping accuracy of 74.13% with a stalk detection F1 score of 0.90.

We argue that decoupling the perception and manipulation pipeline results in latency induced failure cases due to lack of feedback based end to end control. Moreover it is hard to model an environment that requires manipulation of deformable objects. Reinforcement Learning provides a framework ideal for this scenario. Our experiments demonstrate that plugging a simple indicator reward function coupled with goal relabelling can perform just as well as a ground truth based reward function with a Reinforcement Learning based approaches. We also propose a principled approach to combining several self supervised auxiliary losses to speed up learning when using visual observation. Our method achieves about a 3x speedup compared to using no auxiliary tasks in a set of robotic manipulation environments.

# Bibliography


[1] *The future of food and agriculture Trends and challenges*, 2019 (accessed February 21, 2019). URL http://www.fao.org/3/a-i6583e.pdf. 1

[2] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 37–45, 2015. 3, ??

[3] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016. 2.3.2, 1, 2, ??, ??

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017. 2.2, 5.1.3, 5.1.5, 5.2.3

[5] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018. 2.3.1

[6] Johan Baeten, Kevin Donné, Sven Boedrij, Wim Beckers, and Eric Claesen. Autonomous fruit picking machine: A robotic apple harvester. In *Field and service robotics*, pages 531–539. Springer, 2008. 2.2

[7] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013. 2.3.2

[8] HARJATIN SINGH BAWEJA, Tanvir Parhar, and Stephen Nuske. Early-season vineyard shoot and leaf estimation using computer vision techniques. In *2017 ASABE Annual International Meeting*, page 1. American Society of Agricultural and Biological Engineers, 2017. 2.1

[9] Harjatin Singh Baweja, Tanvir Parhar, Omeed Mirbod, and Stephen Nuske. Stalknet: A deep learning pipeline for high-throughput measurement of plant

stalk count and stalk width. In *Field and Service Robotics*, pages 271–284. Springer, 2018. 2.1

[10] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016. 2.3.2

[11] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. 2.3.3

[12] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018. 2.3.1

[13] Steven W Chen, Shreyas S Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J Taylor, and Vijay Kumar. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017. 2.1

[14] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017. 2.3.5

[15] Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O'Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3893–3900. IEEE, 2011. 2.3.6

[16] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017. 2.3.3

[17] Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018. 5.2.2.1, 3

[18] Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierre Sermanet. Learning Actionable Representations from Visual Observations. 2018. ISSN 0032-8332, 1610-7365. doi: 10.1007/s10329-008-0079-0. URL http://arxiv.org/abs/1808.00928. 5.1.5

[19] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017. 2.3.2

[20] Frederik Ebert, Sudeep Dasari, Alex X Lee, Sergey Levine, and Chelsea Finn. Robustness via retrying: Closed-loop robotic manipulation with self-supervised

learning. *arXiv preprint arXiv:1810.03043*, 2018. 2.3.2

[21] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. 2.3.2

[22] Kuan Fang, Yuke Zhu, Animesh Garg, Andrey Kurenkov, Viraj Mehta, Li Fei-Fei, and Silvio Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *arXiv preprint arXiv:1806.09266*, 2018. 2.3.1

[23] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003. 5

[24] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017. 2.3.2

[25] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016. 2.3.2

[26] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016. 2.3.2

[27] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017. 2.3.2

[28] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni. Robotic arm control and task training through deep reinforcement learning. 2.2

[29] Alejandro F Frangi, Wiro J Niessen, Koen L Vincken, and Max A Viergever. Multiscale vessel enhancement filtering. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 130–137. Springer, 1998. 2.1

[30] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2.3.3

[31] Vikash Goel, Jameson Weng, and Pascal Poupart. Unsupervised video object segmentation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5683–5694, 2018. 5, **??**

[32] 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. 2.2, 2.3.1

[33] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016. 2.3.2

[34] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. *arXiv preprint arXiv:1708.06832*, 2017. 2.3.5, 5.2.2.2, 2

[35] Sandy H Huang, Jia Pan, George Mulcaire, and Pieter Abbeel. Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 878–885. IEEE, 2015. 2.3.6, 5.1.2.1

[36] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016. 2.3.4, 5.2.1

[37] Shervin Javdani, Sameep Tandon, Jie Tang, James F O'Brien, and Pieter Abbeel. Modeling and perception of deformable one-dimensional objects. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1607–1614. IEEE, 2011. 2.3.6

[38] Merritt Jenkins and George Kantor. Online detection of occluded plant stalks for manipulation. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 5162–5167. IEEE, 2017. 2.1

[39] Merritt Jenkins and George Kantor. Online detection of occluded plant stalks for manipulation. In *International Conference on Intelligent Robots and Systems*, pages to–appear. IEEE, (accepted) 2017. 4.2.1

[40] Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993. Morgan Kaufmann. URL `http://people.csail.mit.edu/lpk/papers/ijcai93.ps`. 5.1.3

[41] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3342–3349. IEEE, 2017. 2.2

[42] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018. 2.3.5

[43] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In

*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6129–6138, 2017. 2.3.3

[44] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE, 2010. 2.3.1

[45] Alex X Lee, Max A Goldstein, Shane T Barratt, and Pieter Abbeel. A non-rigid point and normal registration algorithm with applications to learning from demonstrations. In *ICRA*, pages 935–942, 2015. 2.3.6, 5.1.2.1

[46] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013. 2.2

[47] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 2.2, 2.3.1

[48] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: a hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015. 2.3.4

[49] 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. 5.1.5, 5.2.3

[50] Long-Ji Lin and Tom M Mitchell. *Memory approaches to reinforcement learning in non-Markovian domains*. Citeseer, 1992. 2.3.4

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 2.1

[52] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and S Yu Philip. Learning multiple tasks with multilinear relationship networks. In *Advances in Neural Information Processing Systems*, pages 1594–1603, 2017. 2.3.5

[53] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018. 2.3.4

[54] Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4861–4868. IEEE, 2011. 2.3.6

[55] Omeed Mirbod, Luke Yoder, and Stephen Nuske. Automated measurement of berry size in images. *IFAC-PapersOnLine*, 49(16):79–84, 2016. 2.1

[56] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Bal-

lard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 2.3.4

[57] Z Mohammed Amean, Tobias Low, Cheryl McCarthy, and Nigel Hancock. Automatic plant branch segmentation and classification using vesselness measure. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2013)*, pages 1–9. Australasian Robotics and Automation Association, 2013. 2.1

[58] Tim Mueller-Sim, Merritt Jenkins, Justin Abel, and George Kantor. The robotanist: a ground-based agricultural robot for high-throughput crop phenotyping. In *IEEE International Conference on Robotics and Automation (ICRA), Singapore, Singapore*, 2017. 2.2, 3.1

[59] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *NeurIPS*, pages 9209–9220, 2018. 2.3.2

[60] OpenAI. Learning Dexterous In-Hand Manipulation. pages 1–27, 2018. URL http://arxiv.org/abs/1808.00177. 5.1.5.1

[61] Tanvir Parhar, Harjatin Baweja, Merritt Jenkins, and G. Kantor. A deep learning-based stalk grasping pipeline. In *Proceedings of the IEEE International Conference on Robotics and Automation, accepted*, May 2018. 2.1, 2.2

[62] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017. 2.3.2

[63] Stefan Paulus, Jan Dupuis, Anne-Katrin Mahlein, and Heiner Kuhlmann. Surface feature based classification of plant organs from 3d laserscanned point clouds for plant phenotyping. *BMC bioinformatics*, 14(1):238, 2013. 2.1

[64] Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. *arXiv preprint arXiv:1810.00821*, 2018. 2.3.2

[65] Calder Phillips-Grafflin and Dmitry Berenson. A representation of deformable objects for motion planning with no physical simulation. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 98–105. IEEE, 2014. 2.3.6

[66] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. 06 2018. doi: 10.15607/RSS.2018.XIV.008. 2.3.1

[67] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker,

Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018. 5.2.3.2

[68] Zania S Pothen and Stephen Nuske. Texture-based fruit detection via images using the smooth patterns on the fruit. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5171–5176. IEEE, 2016. 2.1

[69] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1137–1149, 2017. 2.1

[70] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018. 2.3.4

[71] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 2.3.3

[72] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 2.3.1

[73] Alistair John Scarfe. *Development of an autonomous kiwifruit harvester: a thesis presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Industrial Automation at Massey University, Manawatu, New Zealand.* PhD thesis, Massey University, 2012. 2.2

[74] Connor Schenck and Dieter Fox. Guided policy search with delayed sensor measurements. *arXiv preprint arXiv:1609.03076*, 2016. 2.3.1

[75] Jürgen Schmidhuber. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pages 1458–1463. IEEE, 1991. 2.3.2

[76] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3): 230–247, 2010. 2.3.2

[77] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1130–1137. IEEE, 2013. 2.3.6, 5.1.2.1

[78] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015. 2.2

[79] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

2.2

[80] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016. 2.3.2

[81] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018. 2.3.2

[82] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016. 2.3.4

[83] Abhisesh Silwal, Joseph R Davidson, Manoj Karkee, Changki Mo, Qin Zhang, and Karen Lewis. Design, integration, and field evaluation of a robotic apple harvester. *Journal of Field Robotics*, 34(6):1140–1159, 2017. 2.2

[84] Paloma Sodhi, Srinivasan Vijayarangan, and David Wettergreen. In-field segmentation and identification of plant structures using 3d imaging. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 5180–5187. IEEE, 2017. 2.1

[85] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011. 2.3.4

[86] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018. 2.3.1

[87] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017. 2.3.2

[88] Kanae Tanigaki, Tateshi Fujiura, Akira Akase, and Junichi Imagawa. Cherry-harvesting robot. *Computers and electronics in agriculture*, 63(1):65–72, 2008. 2.2

[89] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 5.2.3.2

[90] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science &

Business Media, 2012. 2.3.3

[91] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017. 2.3.1

[92] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. 5.1.5.1, 5.2.3.2

[93] Ping Chuan Wang, Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Perception for the manipulation of socks. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4877–4884. IEEE, 2011. 2.3.6

[94] David Warde-farley, Tom Van De Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised Control through Non-parameteric Discriminative Rewards. pages 1–17, 2018. 2.3.2, 5.1.5

[95] Sheng Xu, Yi Xun, Tingmeng Jia, and Qinghua Yang. Detection method for the buds on winter vines based on computer vision. In *Computational Intelligence and Design (ISCID), 2014 Seventh International Symposium on*, volume 2, pages 44–48. IEEE, 2014. 2.1

[96] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 79–86. IEEE, 2017. 2.3.1

[97] Fangyi Zhang, Jürgen Leitner, Michael Milford, and Peter Corke. Modular deep q networks for sim-to-real transfer of visuo-motor policies. *arXiv preprint arXiv:1610.06781*, 2016. 2.3.4

[98] Yu Zhang and Dit-Yan Yeung. A regularization approach to learning task relationships in multitask learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3):12, 2014. 2.3.5

[99] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018. 2.3.1