Computer Vision to Robot control for root analysis and inoculation for Netherlands Plant Eco-phenotyping Centre

Hubert Waleńczak 220817
2023-2024 Block B
Jason Harty, NPEC
26.01.2024

Table of Contents

Abstract

This documentation outlines a project in collaboration with the Netherlands Plant Eco-phenotyping Centre (NPEC) that focuses on automating the application of fluids to plant root tips using computer vision and robotics. The primary goal is to enhance the throughput of the laboratory's genotype-phenotype system. The project involves the identification of root tips and precise inoculation using the OT2 pipetting robot. The pipeline intended to work with NPEC’s Hades system.

1. Introduction

1.1 Overview of the Project

Plant phenotyping involves the analysis of observable characteristics, development processes, and behavior in plants. To observe how plants react to various substances, different substances can be applied to the roots. This project, in collaboration with the Netherlands Plant Eco-phenotyping Centre (NPEC), concentrates on developing a computer vision and robotics pipeline for automating application of desired fluids on root tips, ultimately enhancing the laboratory's throughput.

The main focus is on identifying root tips and precisely inoculating them using the OT2 pipetting robot. This will help NEPC establish high throughput genotype-phenotype system contributing to development of climate-resilient crops improving crop yeld in times of climate change and unstable wether conditions. This project is designed to work with and supplement one of the systems used by NEPC, Hades, that is responsible for autonomous seeding plants on petri dish and exposing them to controlled conditions to observe difference in plant development. Additional objectives it to measure primary root lengths and total length of lateral roots as well as identifying other parts of the plant other then roots.

1.2 Background

Plant growth can be influenced by both genetic and environmental factors and that influence can be observed as phenotypes. For example, the amount of seeds in fruit is influenced by genes while the fruit size is influenced by both genes and environment, if the environment is not capable of supporting the plant it will grow slower and produce less. How genes and environment influence the growth of the plant is done through phenotyping, that is measuring visible characteristic of the plant during it’s growth.

In nature the plant growth is not the easiest process to speedup, as such to increase amount of experiments and gathered data the integration of computer vision and robotics for use in documenting root lengths and applying desired fluid on root tips in fast automated manner that can work continuously and is easy to scale up.

2. Data

2.1 Data Source

I have received access to the data from NEPC through Breda University of applied science. The provided dataset contains gray scale images of multiple petri dishes containing five Arabidopsis thaliana plants at different growth stages. All images have dimensions of 4202 x 3006 pixels.

In laboratory the growth of sown *Arabidopsis thaliana* in each petri dish is performed daily by photographing them however we are provided with selected images on different growth stages instead of dally.
We can identify 4 stages:
Non-germinated seeds (Figure 1A),
Germinated seed when they begin revealing roots, hypocotyl, and leaves (Figure 1B),
Eventually, smaller lateral roots sprout from the primary (Figure 1C),
And finally when those roots begin to crisscross (Figure 1D).

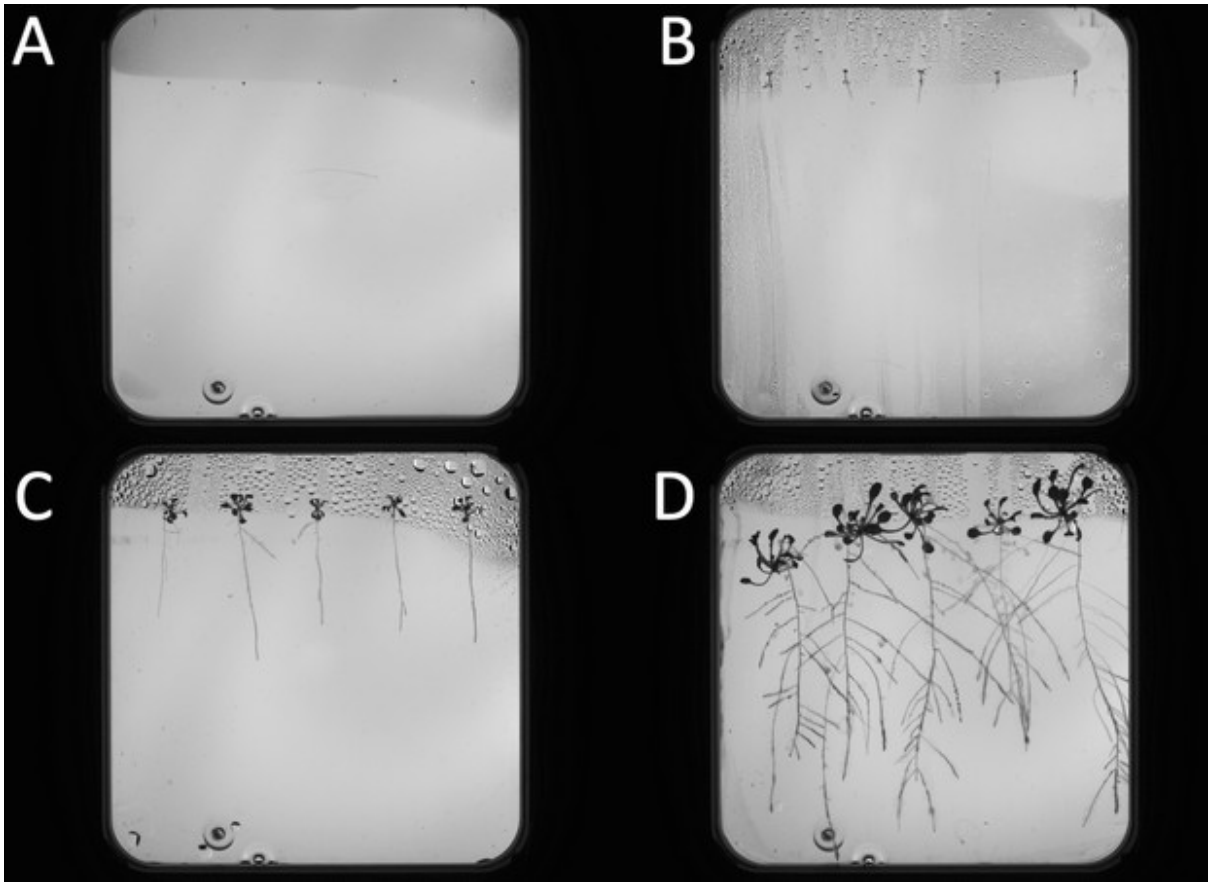


Figure 1 example of growth stages of a plant

2.2 Datasets division

The dataset has been divided into three subsets, Segmentation without any labels containing 126 images in .png format, Measurement dataset containing 11 images with landmark locations, primary and lateral root lengths in .tif format and a Kaggle dataset containing 3 images with primary root length measurements in .tif format.

2.3 Data labeling

As the Segmentation dataset doesn’t have labels and is used to train the models first we needed to annotate the data. Each Image needs 4 masks, root_mask, occluded_root_mask, shoot_mask, and seed_mask saved by appending those masks to the end of the image name that was annotated in .tif format.
Example: original image name - abc.png after annotating would have 4 files named abc_root_maks.tif, abc_occluded_root_mask.tif, abc_shoot_mask.tif and abc_seed_mask.tif.

The four labels for each picture form a comprehensive annotation scheme that enables later trained model/s to indentify distinct components of the plant.

2.4 Potential biases discussion

The dataset contains only images of *Arabidopsis thaliana* in petri dish on black background, so the model can potentially work with only *Arabidopsis thaliana.* As all images are taken in the same conditions there should not be any background bias. Additionally with few exceptions there are always 5 plants evenly spaced so that can also influence model performance with different number of plants.

3. Project process

3.1. Image annotation

By annotating images in computer vision we provide additional information about region or feature on the image, there are many ways to annotate images example methods are: a bounding box or mask with either 0 and 1 where 1 is the region we are interested in. For this project we used 4 separate binary masks of roots, occluded roots, seeds and shoots where 1 is corresponding to that feature on image and saved in .tif format in the same resolution as original image. Inclusion of occluded root mask is experimental where we expect to see increased performance of the system in calculating root lengths by reducing probability of discontinuity in predicted root mask that can lead to incorrect analysis.
The process of annotation of 126 images was spread out among the students where each student had to annotate 2 or 3 images and masks needed to follow naming convention of original_image_name_{TypeOfMask}_mask.tif so image abc.png after annotating would have 4 masks named abc_root_maks.tif, abc_occluded_root_mask.tif, abc_shoot_mask.tif and abc_seed_mask.tif.

Example of correctly annotated masks can be seen on Figure 2.

Due to the academic nature of the project, the image labeling process was distributed among multiple participants, with each individual responsible for labeling 2-3 images. This collaborative effort spanned a period of 9 days, ensuring a thorough and meticulous labeling process. Additionally, every mask underwent a quality check to meet the desired standards and conditions, contributing to the overall reliability of the annotated dataset.

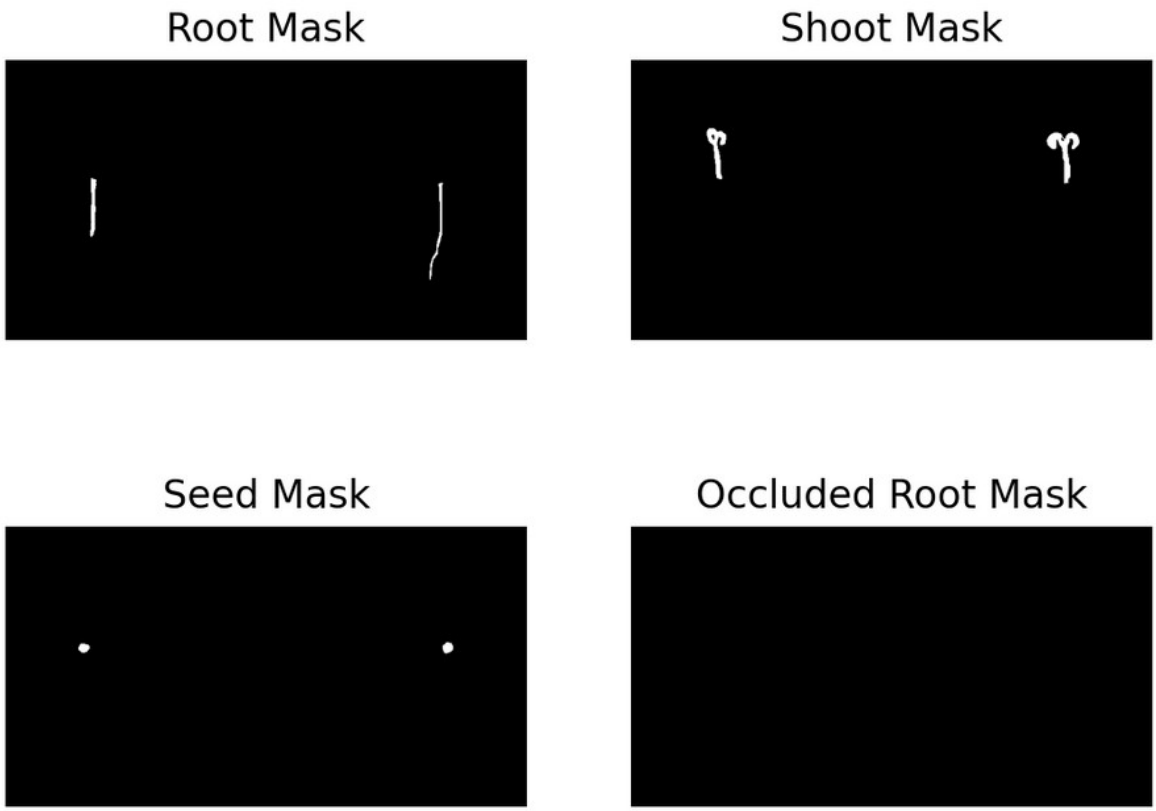


Figure 2 example of correctly annotated masks

3.2. Petri dish extraction

First step in in the pipeline is pre-processing the images. The pre-processing consists of cropping image to the petri dish as the background is outside our Region of Interest(RoI), resulting in square image and cropping the masks to the same dimensions to avoid misalignment between actual plant parts and masks. This step can be performed by traditional computer vision techniques which are faster and less resource intensive then deep learning solution.

To crop the image to petri dish following steps are taken:

- First image is thresholded using thresholding function from cv2 library,
- Then from the thresholded image different components and their statistics such as position and size is extracted using “connected_components_with_stats” function from cv2,
- As “connected_components_with_stats” also returns background as component we remove the component that starts at coordinates x:0 and y:0,
- Petri dish is the biggest component on the image so I took the statistics with highest value in height column,
- Now knowing the x and y position of top-left corner of petri dish and it’s width and hight I crop the image and masks to those coordinates and size, to ensure that the result is a square I overwrite smaller value between width and hight with the higher value.

The result of those steps is a square image cropped to RoI that is petri dish resulting in less pixels that are irrelevant to identifying plant parts. The image before cropping can be seen on left side of figure 3 and resulting cropped image on right side of figure 3

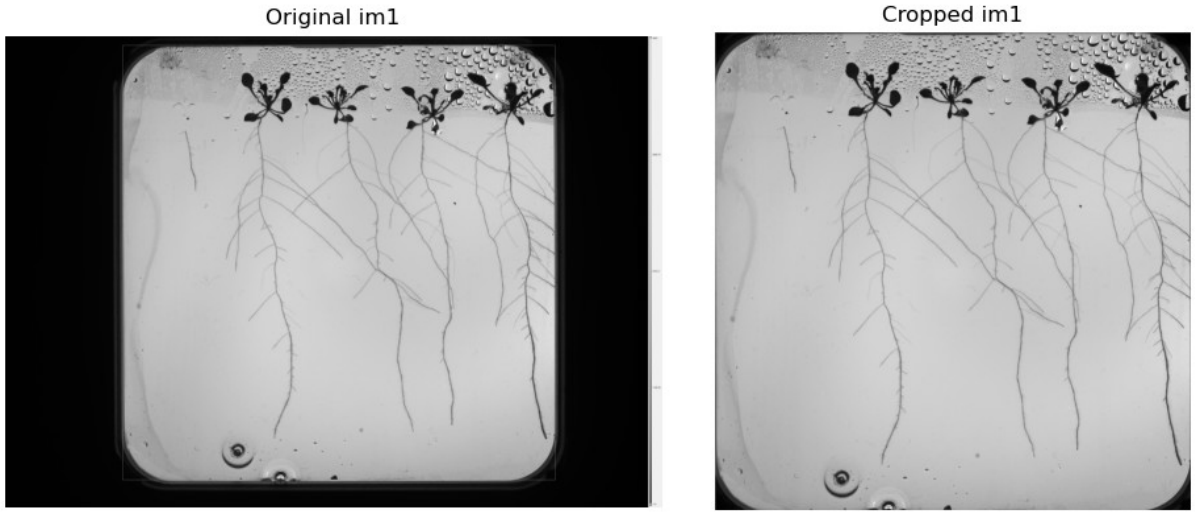


Figure 3 example of cropping image to petri dish

3.3. Instance segmentation

Instance segmentation is a process of finding and delineating each individual object (i.e. instance) in an image at the pixel level. Unlike semantic segmentation, which clusters all objects of the same class together, instance segmentation distinguishes each unique object, even if they belong to the same class.

Instance segmentation at this part of the project is performed on whole plants using traditional computer vision techniques. The process is similar to steps taken for cropping the image to petri dish. The steps are to threshold the image, extract components with corresponding stats, filtering noise by applying positional, size and width/hight ratio filters. In the end resulting with each instance having different label value starting from 1 up, 0 is background. While this approach is fast it is very inflexible, potent to errors and hard to make it work with wide range of different raw images however it is very effective for analysis later in the pipeline. The result of the instance segmentation process can be seen on Figure 4.

Figure 4 example of segmented individual plants

3.4. Semantic segmentation

Semantic segmentation is a process of assigning the same class to all pixels that belong to that class so for example image with multiple clouds would assign the same class to all of them. In the case of this pipeline, we want to identify specific parts of the plants, mainly root, but also occluded root, seed, and shoot. The best approach in our case is to use deep learning model for that trained on images and masks prepared in section 3.1 from segmentation dataset.

The dataset is further divided into train and test dataset containing 96 and 30 image, respectively. The U-net architecture(Figure 5) is used for training four different models each for predicting different part of the plant. I tested few approaches to train one multi-class model but none of them had satisfactory results. To train the model input size needs to be constant and the cropped image is too large to pass straight to the model, as such I have decided to patch the images to 256x256px and use that size as input for the model.

To achieve that I needed to perform following additional pre-processing steps:

- crop the images and masks to petri dish using code form section 3.2,
- pad images that they will evenly divide by 256,
- patch the images and masks into 256x256px patches and save them into train and test folders

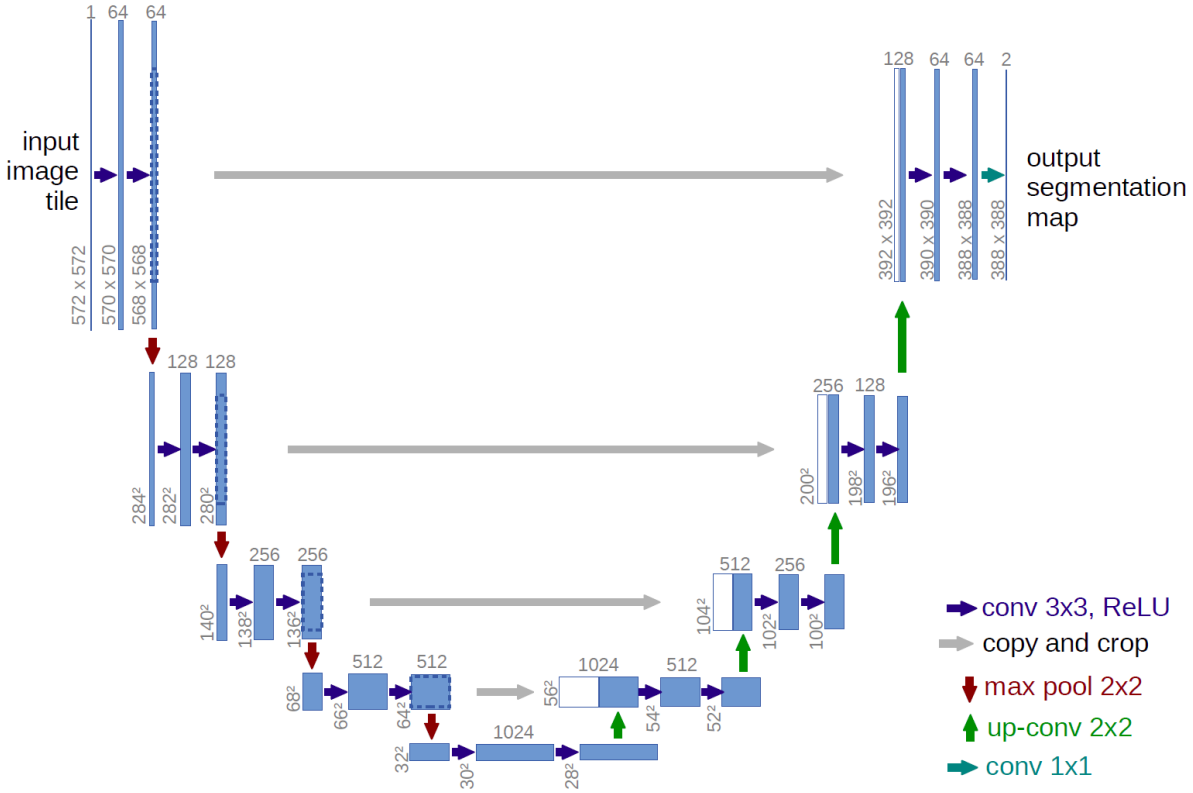
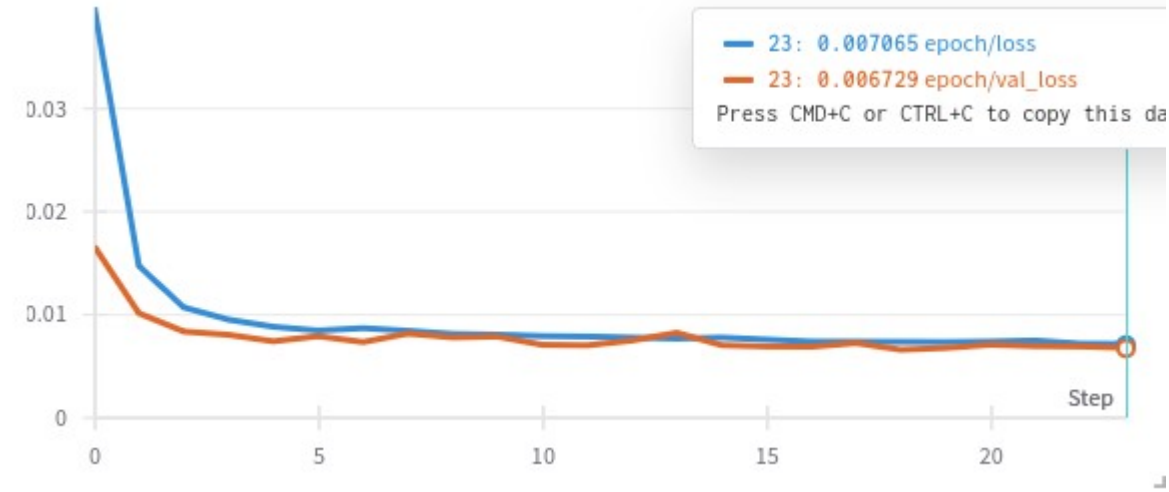


Figure 5 Diagram of U-net architecture

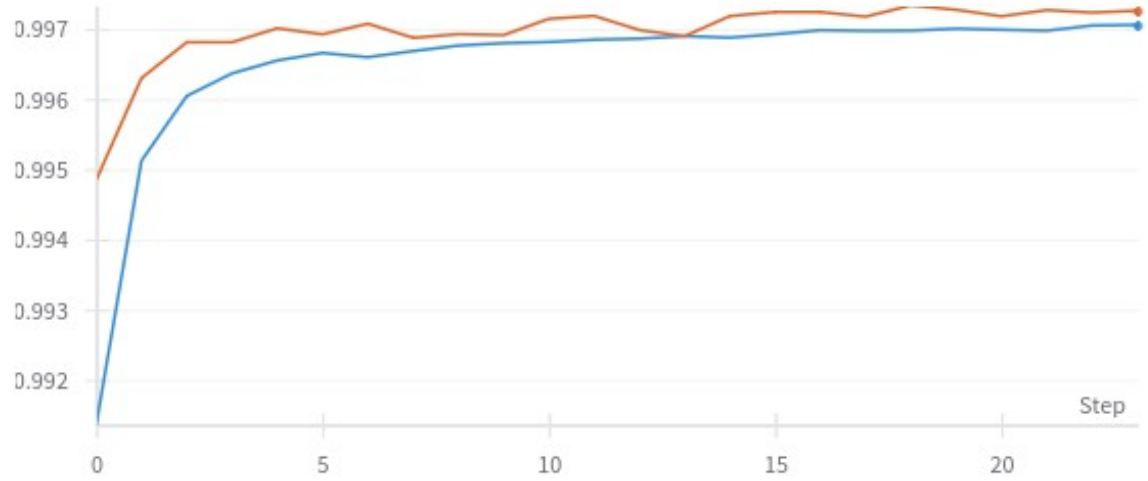
For training of the models, the images were loaded as gray scale and model performance was monitored during training process using “Weights & Biases” platform, as well as the trained model was additionally tested for Intersection over Union (IoU) where 1 means complete overlap and 0 means that nothing overlaps.
The performance of the models can be observed on following subsections, blue line represents performance on train dataset and orange line on test dataset

3.4.1 Root model

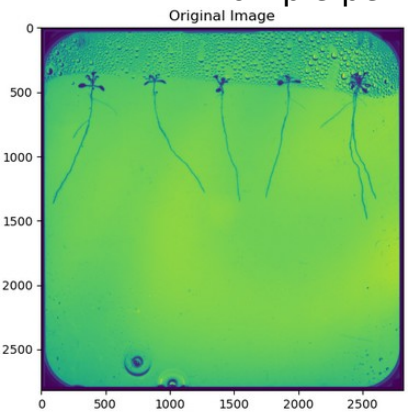
Loss:

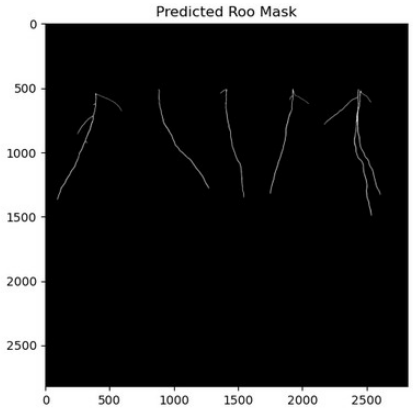


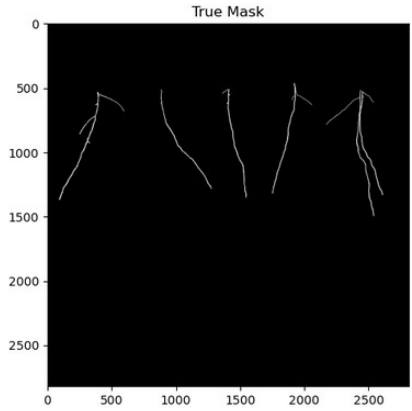
Accuracy:



Example performance:



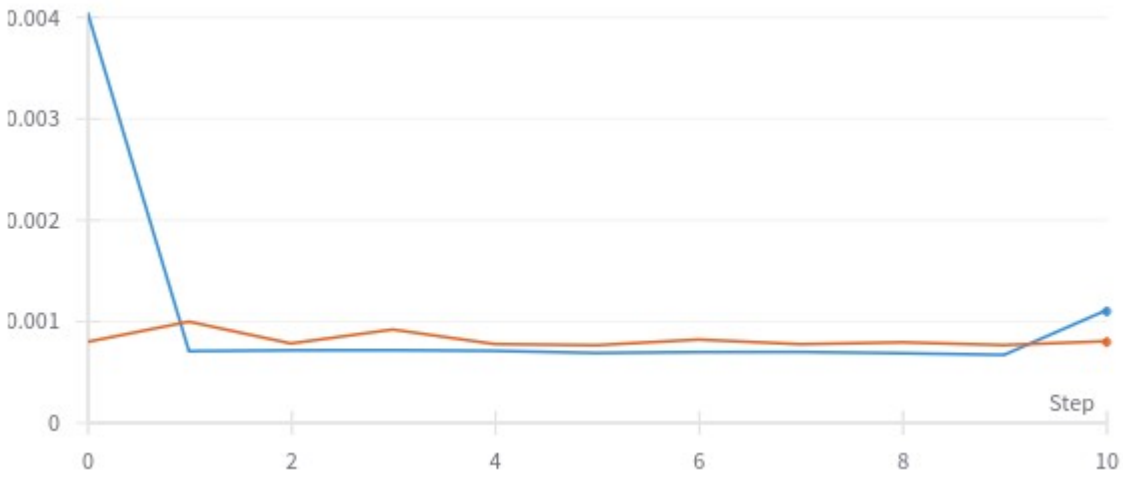




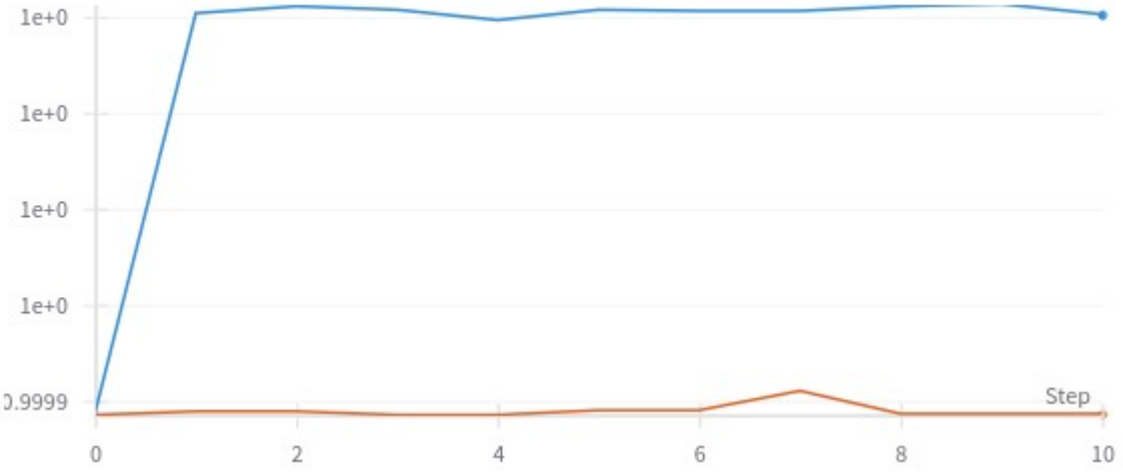
IoU score for the root model is: 0.897834837436676
which suggests hight overlap between predicted masks and actual masks and above client's requirement of >0.5IoU.

3.4.2 Occluded root model

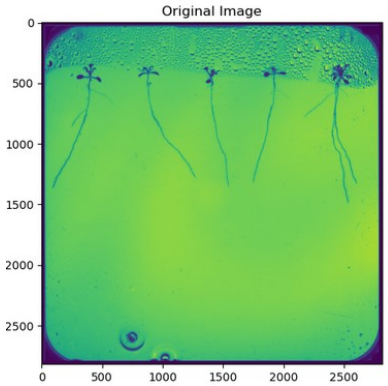
Loss:

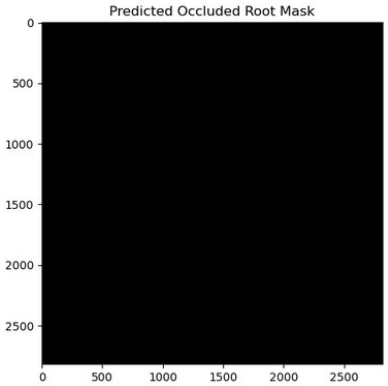


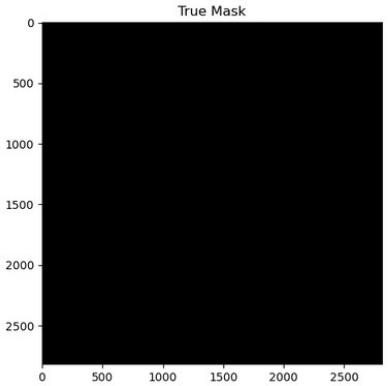
Accuracy:



Example performance:



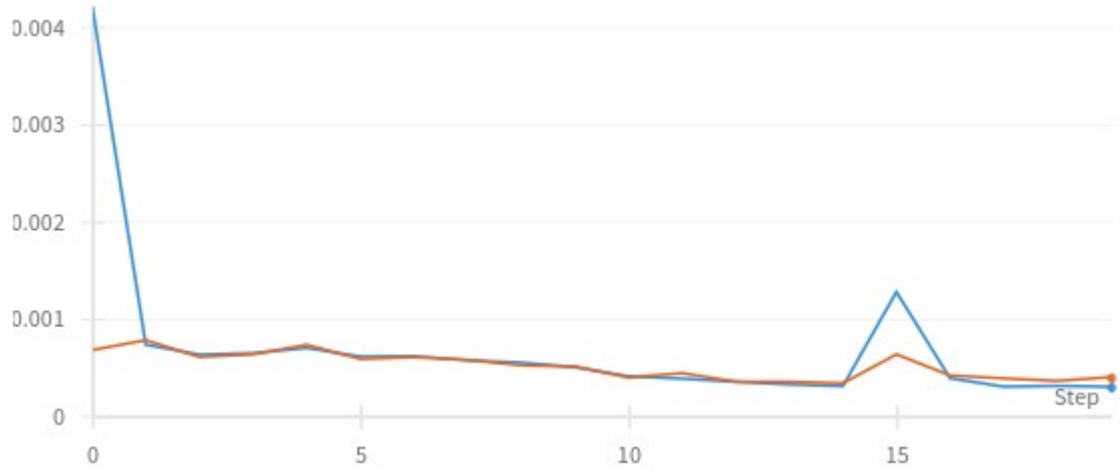




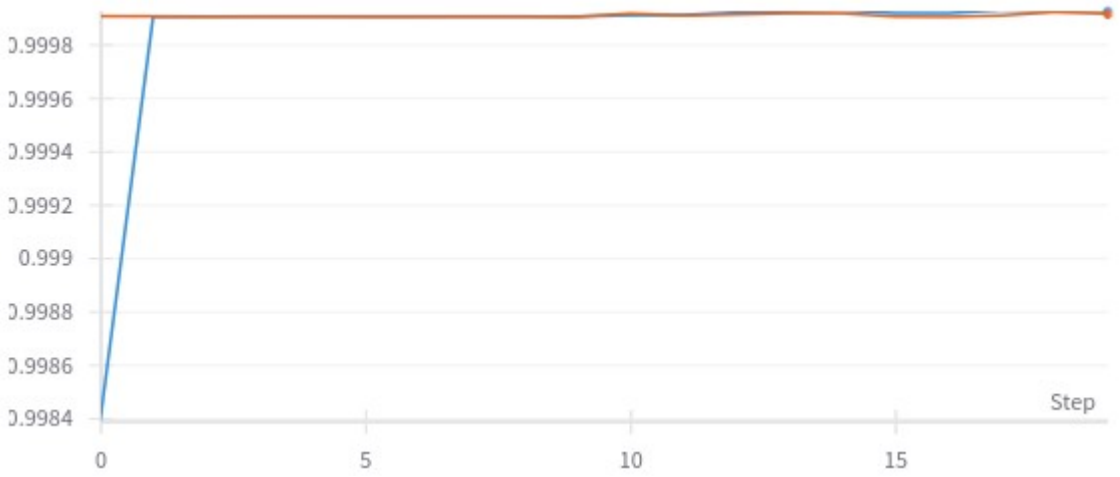
IoU score for the occluded root model is: 0.9830800890922546
which suggest very high overlap however as there is a very few examples of occluded roots in dataset it may be that model doesn’t predict any occluded roots and still gets high score, further proved by model accuracy which is almost 100% form the start. It’s above client's requirement of >0.5IoU.

3.4.3 Seed model

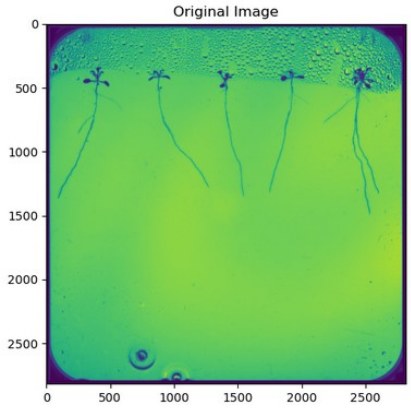
Loss:

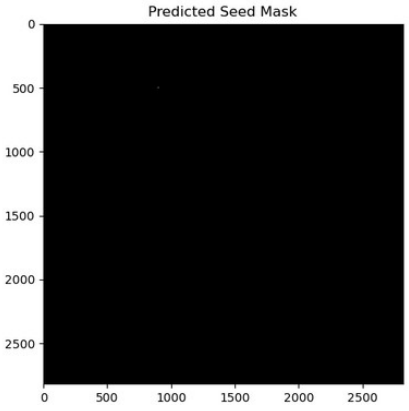


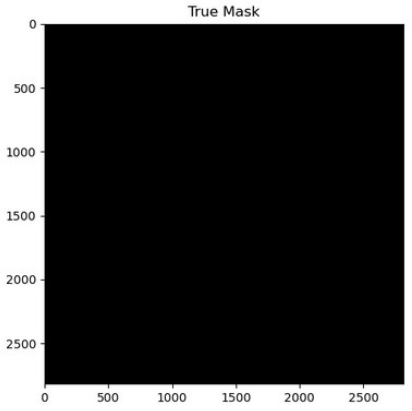
Accuracy:



Example performance:



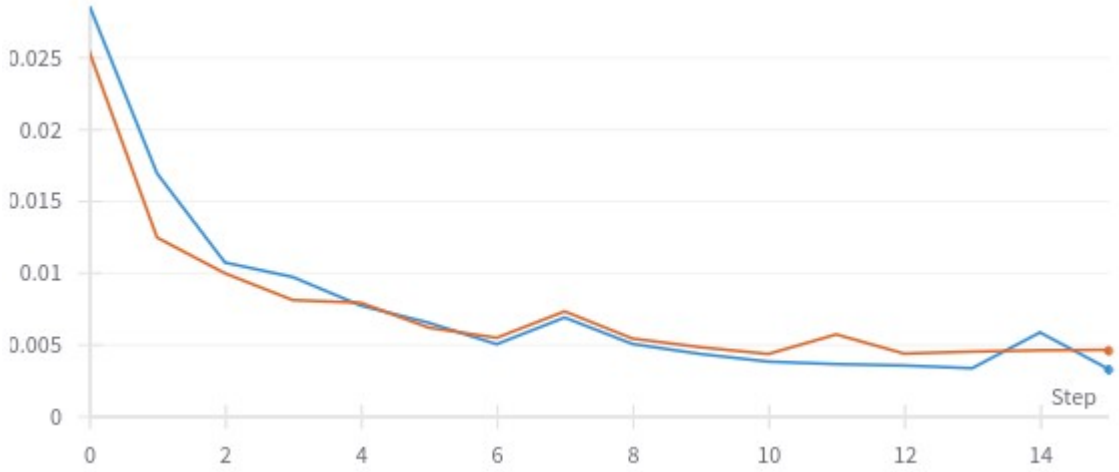




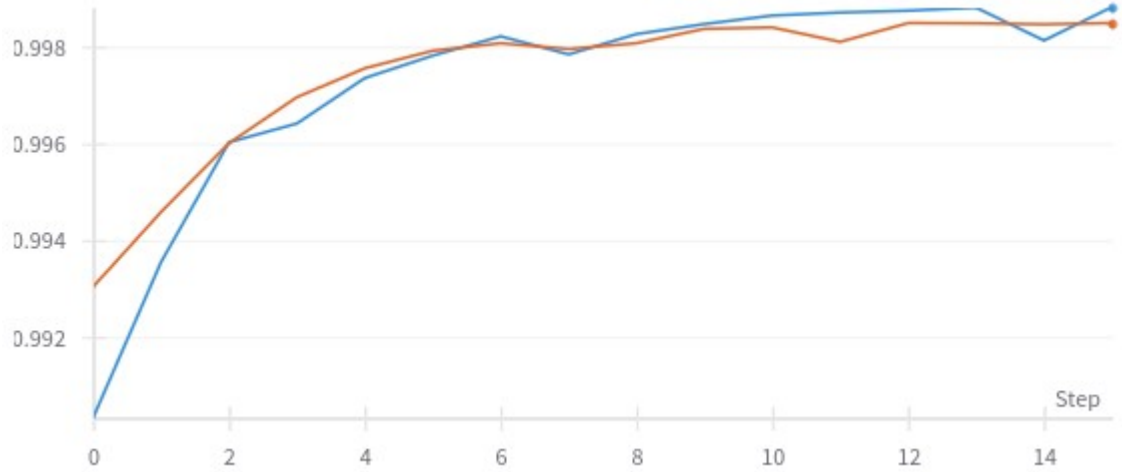
IoU of the seed model is: 0.9389204978942871
Which also suggests high performance of the model but also doesn't have much dataset and may achieve that performance by not predicting any seeds. It's above client's requirement of >0.5IoU.

3.4.4 Shoot model

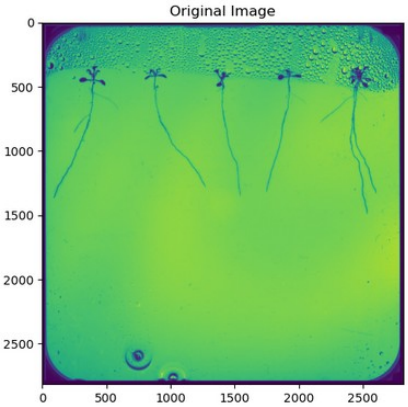
Loss:

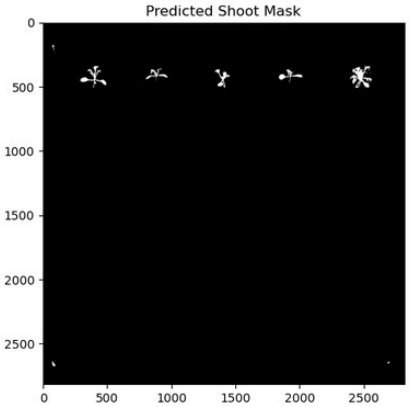


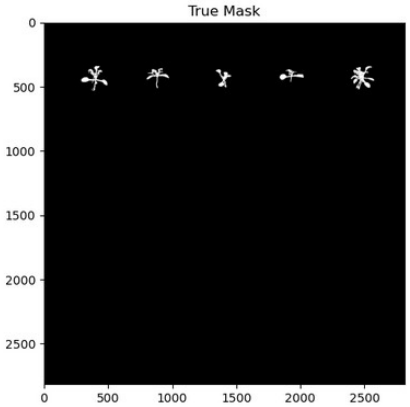
Accuracy:



Example performance:







IoU of the shoot model is: 0.9680633544921875
this is better performance than the root model while I am sure that it outputs predictions, that may be because shoots are more concentrated and easier to predict than thin roots and above client's requirement of >0.5IoU.

3.5. Instance segmentation

Now that I have a clear mask of root thanks to model form section 3.4 I can use Instance segmentation again on that mask resulting in different instances of the same class(root) in the image that are not connected. Model notifies if the number of detected roots is different than 5 and if it's more then 5 it only takes 5 biggest roots. As the roots are mostly constant in terms of placement in X axis if the number of detected roots less then 5 I can identify which root is missing by filtering them by X position. Example output is on Figure 6

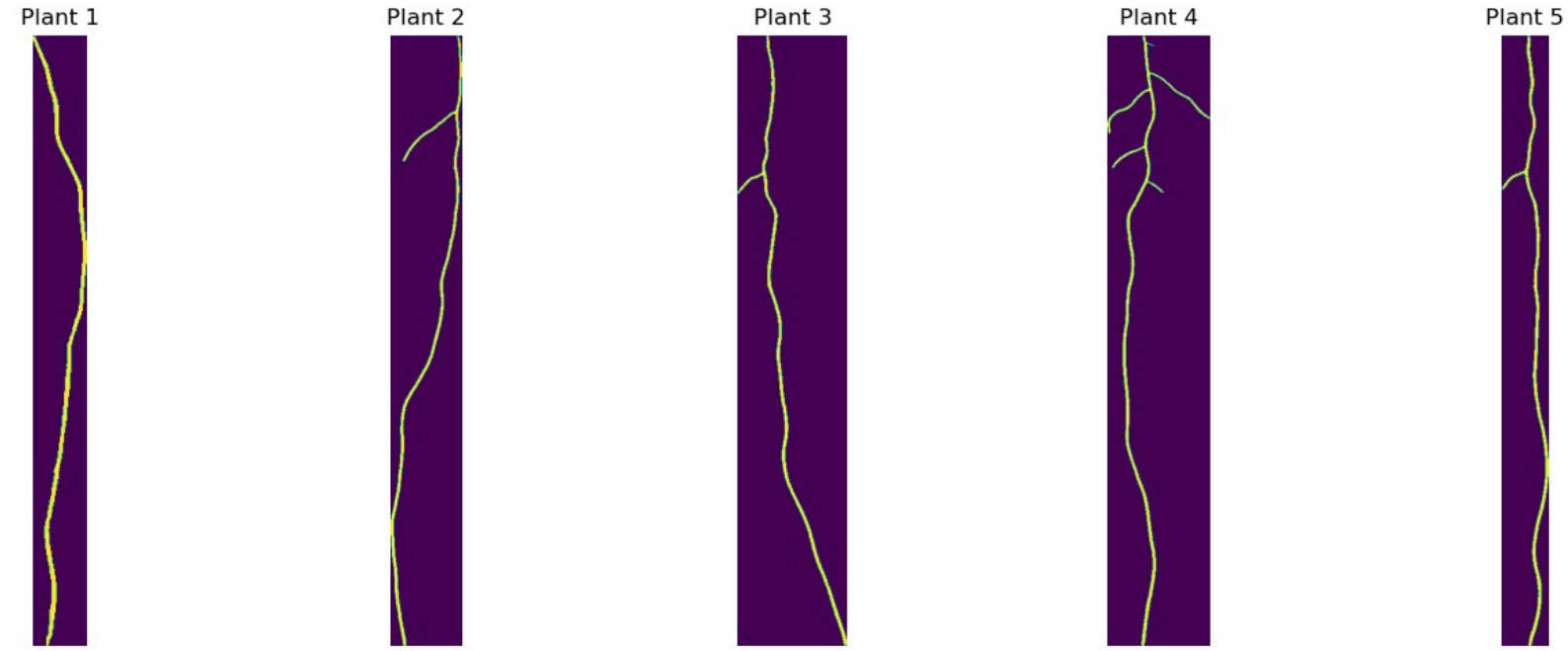


Figure 6 example of instance segmentation on predictedroot mask

3.6. Root landmark detection

Landmark detection is a process of identifying characteristic points in the data in case of this project characteristic points in root, them being main root tip, lateral root tips, junctions and junction between hypocotyl and root.
I perform the analysis of root landmarks by first skeletonizing the mask using skeletonize and close_small_holes functions form skimage library, to generate skeleton and reduce risk of discontinued skeleton by closing potential small gaps in the skeleton. After that I use Skan library to generate summary form skeleton with image and plant ID. Example of skan summary is on Figure 7, additionally found landmarks example are on Figure 8 and 9

	skeleton-id	node-id-src	node-id-dst	branch-distance	branch-type	mean-pixel-value	stdev-pixel-value	image-coord-src-0	image-coord-src-1	image-coord-dst-0	image-coord-dst-1	coord-src-0	coord-src-1	coord-dst-0	coord-dst-1	euclidean-distance	Image_ID	Plant_ID
0	0	0	54	48.828427	1	1.0	0.0	0	61	48	59	0	61	48	59	48.041649	2	4
1	0	54	59	10.828427	1	1.0	0.0	48	59	50	49	48	59	50	49	10.198039	2	4
2	0	54	127	70.656854	2	1.0	0.0	48	59	117	55	48	59	117	55	69.115845	2	4
3	0	127	247	80.497475	1	1.0	0.0	117	55	171	102	117	55	171	102	71.589105	2	4
4	0	127	976	821.521861	1	1.0	0.0	117	55	900	2	117	55	900	2	784.791692	2	4

Figure 7 example root skeleton summary

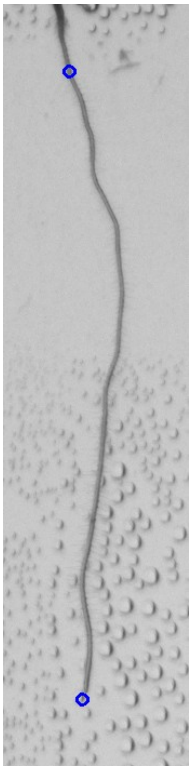


Figure 8 example identified landmarks on simple root

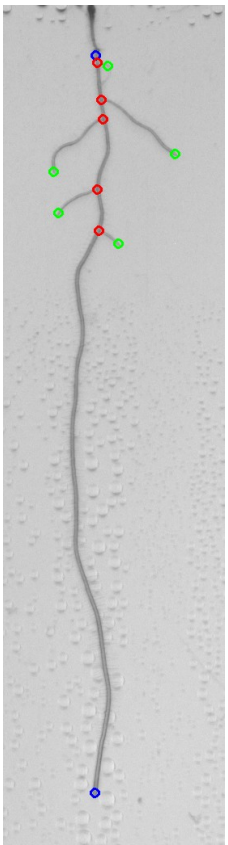


Figure 9 example identified landmarks on complex root

3.7. Morphometric analysis

Within computer vision, morphometric analysis involves the quantitative measurement of shapes or structures within an image. This includes the extraction and examination of dimensions such as lengths, areas, and volumes, providing insights into and facilitating the comparison of object geometries.

In this project the morphic analysis consists of just measuring main root length and total lateral root length, I use networkx library to perform that analysis on data form section 3.6 by selecting highest and lowest value in y axis for each plant and calculating distance between them following the skeleton for the main root, and for total lateral root length I take out 2 previous points form data and sum the length of each junction to endpoint nodes. After calculating the lengths I calculate Symmetric Mean Absolute Percentage Error(sMAPE) for main and lateral root lengths on Measurement dataset. The sMAPE for main root is: 3.4013135048603074% which is a good performance and well within client's requirements of <20%, and sMAPE for total lateral root length is: 31.720253942566533% which is not the best performance however it is within client's requirements of <50%

3.8 The Kaggle Competition

The pipeline consists of 4 jupyter notebooks, each saves data in designated folder and next notebook reads data from where the previous notebook saved the data. In this approach, any pipeline issues can be identified and examined at each step of the process. The position on kaggle leader board on private dataset(Figure 10) is 19th place with 26.722% sMAPE, the performance is below the baseline and ideal preference of <5% however it is within the requirement of <50%

Figure 10 Screenshot of private leader board position

3.9. Simulation Environment

To test the simulation I created a simple script that initializes the simulation and than goes over the list of actions to go to each corner in simulation space, the next action is chosen based on condition that the robot didn't move for 10 simulation steps.

3.10. Creating a Gym Environment

I created a wrapper for the provided pyBullet simulation environment to make it compatible with the gymnasium standard. This was necessary for training a reinforcement learning algorithm using Stable Baselines 3. The objective of the Reinforcement Learning (RL) algorithm is to enable the movement of the pipette's tip to any specified position within the working envelope. I had to define the following properties and methods.

- Properties:
  - Action Space
  - Observation Space
  - Possible actions
  - Possible Observations
  - Reward function
  - Done condition/s
- Methods:
  - Reset
  - Step
  - Render
  - Close

To test the wrapper I wrote a simple script that takes random actions form action space.

The wrapper is capable of displaying the simulation and returning rgb image of the simulation, although number of agents can be specified the wrapper is made with only one agent in mind. Possible actions are from -1 to 1 for x, y and z axis. The observation space is from -infinity to infinity. The reward function is a simple negative of distance to goal, if the simulation truncates and distance is bigger than 0.1 than the reward is -50 otherwise it is normal reward and if the distance is lower than 0.001 than the reward is 2000.

3.11. Reinforcement Learning

I used PPO model form stabe_baselines3 library and trained it using the wrapper mentioned in section 3.10 to be able to go to any position within envelope. To measure and compare the performance of different hyperparameters, in collaboration with other students, were tracked using “Weights & Biases” (Figure 11) platform. My best performing hyperparameters were n_steps = 1024, batch_size = 32 and learning_ rate = 0.0001.
Within my hyperparameter testing group peak highest performance were with hyperparameters of n_steps = 4096,batch_size = 64 and learning_rate = 0.0001. The groups best performing hyperparameters are avaiable in Table 1

	Timesteps	Number of steps	Learning rate	Factor gamma	Clip range	Entropy coefficient	Batch size
Max	5.000.000	4096	0.0001	0.99	0.2	0.01	64
Hubert	100.000.000	1024	0.0001	/	/	/	32
Shan	1.000.000	4096	0.0001	/	/	/	64

Table 1 Group's best hyperparameters

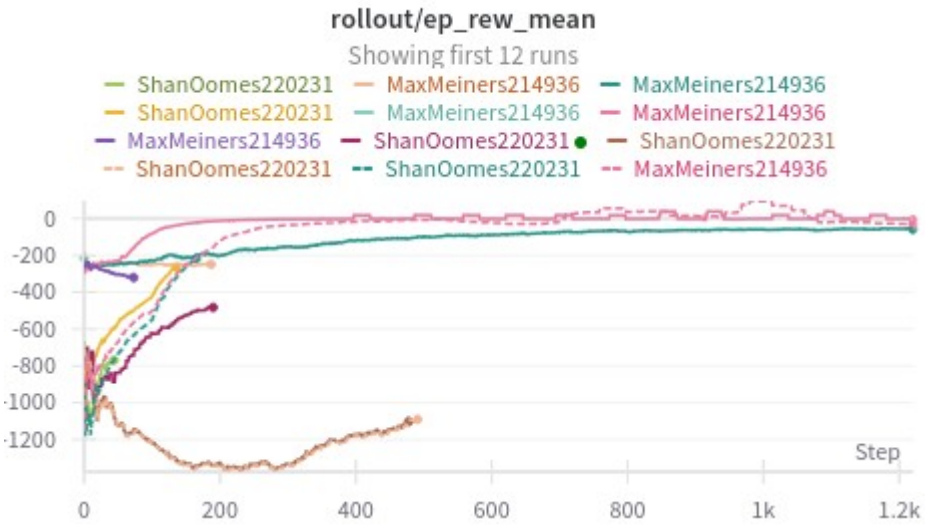


Figure 11 mean reward of each epoch for 12 runs

3.12. Creating a Controller

A simple PID (Proportional, Integral, Derivative) controler for x,y and z axis is more accurate and more resource efficient than the RL model. Initial PID values of P=10, I=0 and D=3 were already better in performance than the RL models in accuracy. However speed at which PID completes the movement to the goal can be increased by changing D value to 0. The distance from goal over steps for initial PID values is on Figure 12 and for PID with D = 0 on Figure 13. Further improvements can be possible by increasing P value, however that may result in oscillation and possibly put

a lot of stress on the machine by trying to quickly reverse it’s movement.

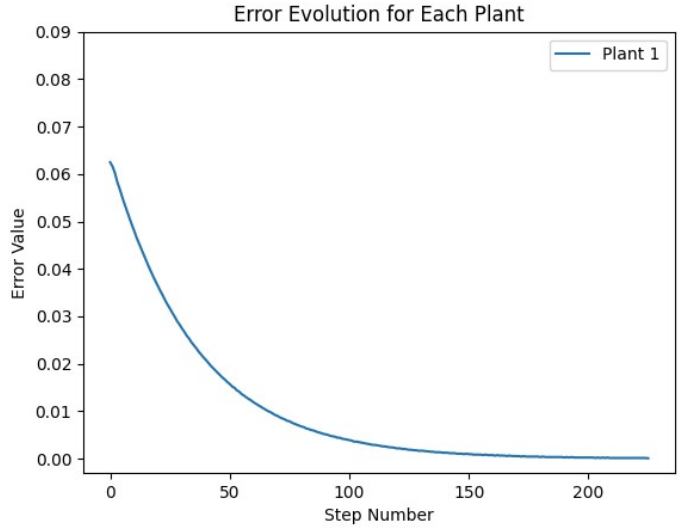


Figure 12 distance over steps for P=10, I=0, D=3

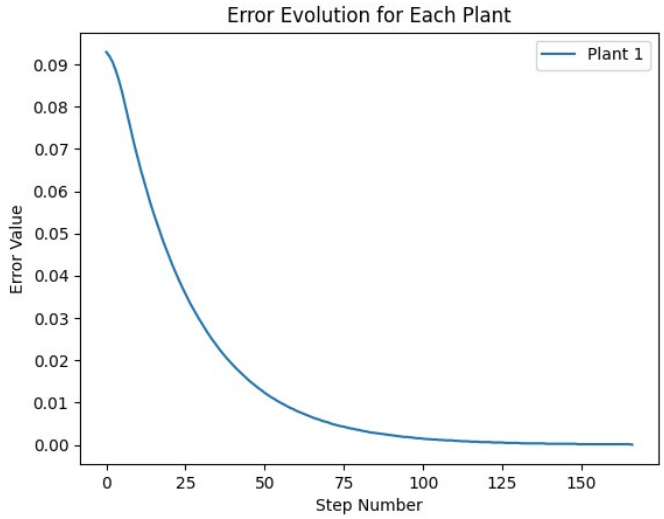


Figure 13 distance over steps for P=10, I=0, D=0

3.13. Pipeline

Pipeline connects all the previous components to apply fluid on detected coordinates within the robot envelope. All steps taken can be seen on Figure 14. The pipeline is limited to reading images from file, it also requires PredictedPlants and SegmentedPlants folder to work properly. Demonstration of the working pipeline implemented to work with simulation wrapper in file Pipeline/main.py

3.14. Benchmark

To benchmark the performance of PID and RL solution for robot control they are compared by number of steps taken to move to goal position. RL model is quick at going to distance of 10mm however it doesn’t achieve higher accuracy with distance drastically increasing and rapidly changing directions (Figure 15). The PID control method on the other hand is achieving distance of 1/10mm in less than 200 steps (Figure 13) resulting in a lot higher accuracy than RL model while being predictable and fast. Furthermore, PID solution is less computationally intensive resulting in lower energy usage of the system than using RL model.

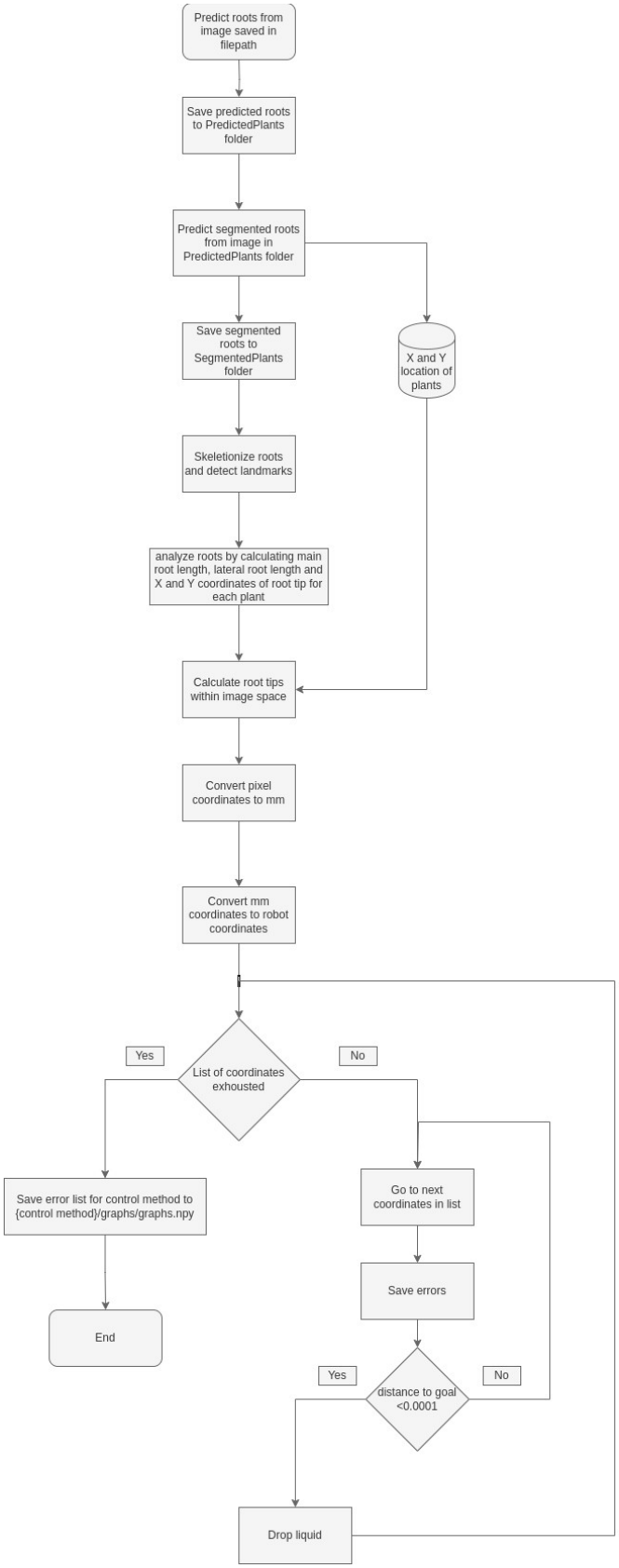


Figure 14 Flowchart of the pipeline

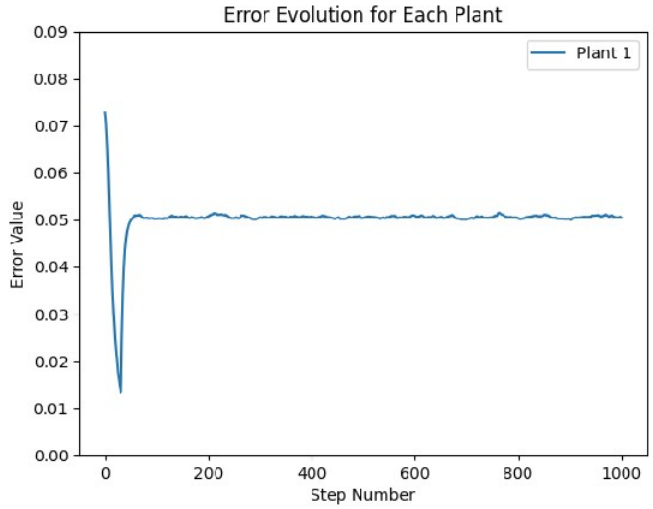


Figure 15 distance over steps for RL model

4. Overall results

The result of the pipeline is predicted roots saved as .png image in PredictedPlants folder, segmented roots saved as .png images in SegmentedPlants folder and graphs.npy with detected landmarks on each of the found roots. The pipeline reads images from the specified location, analyses it and returns the location of roots within robot space that is passed to the main loop using PID control to move robot's arm to that location and drop the fluid on root tip. Instead of PID the RL model can be used instead but from the benchmark the RL model is not able to achieve higher precision, however it's very fast if the requirement is 10mm (Figure 15). The accuracy of PID controller is much higher at 1/10mm while also being very fast (Figure 13). The space and scale of real-life robots may differ and adjustments in code may be needed. Limitations of the pipeline is that it needs the required folder structure to work properly.

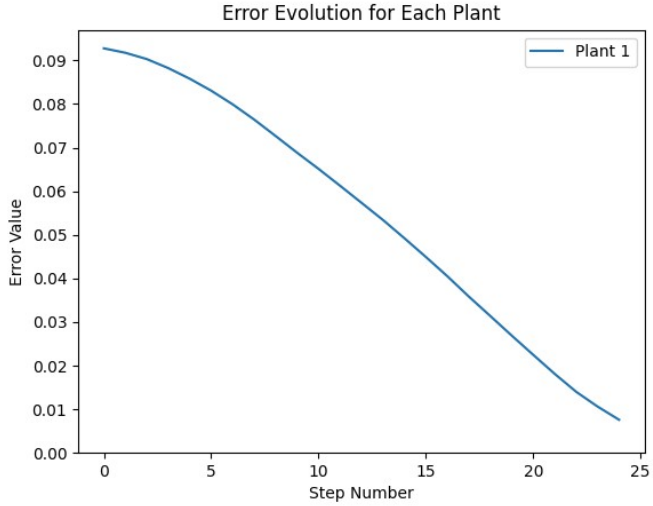


Figure 15 Best RL model if marigin is 10mm

5. Conclusion

In conclusion, the project successfully addressed the objectives of plant part segmentation, landmark detection, morphometric analysis, and robot control. The combination of traditional computer vision techniques and deep learning models proved effective in identifying and analyzing different plant parts.

The performance of the models and the PID controller met or exceeded client requirements. The pipeline will need to be adjusted manually to fit the rest of the system; demonstration of pipeline is in a simulation environment. The use of "Weights & Biases" for performance monitoring added transparency and facilitated collaboration on Reinforcement learning.

The successful implementation of reinforcement learning and PID control provided insights into their strengths and weaknesses. The PID controller, with its superior accuracy and efficiency, emerged as a more practical solution for robot control in this context.

In summary, the project displayed a comprehensive strategy for plant analysis, integrating a variety of methods spanning from computer vision to reinforcement learning. The culmination of these diverse techniques resulted in the development of a pipeline easy to adapt for real-world application.

The space and scale of real-life robots may differ and adjustments in code may be needed.

6. References

Noyan, M. A. (2022). *Uncovering bias in the PlantVillage dataset*.