Symbiosis between pisum sativum and nodule bacteria in varying temperatures

Technical Report · July 2023

DOI: 10.13140/RG.2.2.29146.93129

Suvendu Barai
TH Köln - University of Applied Sciences
11 PUBLICATIONS **2** CITATIONS

TECHNICAL PART

InnoBioDiv 2023

Technische Hochschule Köln
Kharkiv National University of Radio Electronics

FarmBot

PARTICIPANTS:
Daryna Anisimova,
Daryna Nienova,
Illya Para,
Maksym Reshetniak,
Suvendu Barai,
Rinkal Poriya

Table of Contents:

1. INTRODUCTION

FarmBot is a precision farming tool consisting of a Cartesian coordinate robot machine. It is a metal frame with rails, equipped with a robot with interchangeable nozzles for planting seeds, watering, removing weeds, and other farming actions. The plastic components are 3D printed and the robot itself has a built-in microprocessor. With its lighting system, the platform can grow more than 30 crops, working outdoors and indoors around the clock. Thanks to soil sensors, researchers can work with different data about the soil's moisture and humidity, enabling them to make some conclusions. The FarmBot is also equipped with a built-in camera, which will be widely used in this research to explain and improve technical solutions of the biological experiment.

FarmBot uses a web-based interface for operation and configuration that allows control of the platform and supports many popular devices. The web-based application can configure various inputs including water, fertiliser and pesticides, seed interval and environmental factors, including soil and weather conditions, based on sensor readings, location, and time of year. It is also able to create and schedule sequences by combining and modifying basic operations. The software can also manipulate data maps, real-time and logging access to open plant data in the OpenFarm database.

To start the explanation, we'd like to notice some relevant statements of the investigation from the technical side. Thanks to the possibility of working with the robotics system – FarmBot – it's possible to develop new ideas or improve already existing ones to have unique solutions for experiments.

The rapid expansion of Internet technologies, such as the Internet of Things (IoT) and programming, presents a valuable opportunity to incorporate their features into tackling various challenges. This is particularly relevant in addressing global climate change and its impact on different species and the environment.

In this research, we focus on studying the effects of temperature on plants. By utilizing the FarmBot and implementing technical ideas contributed by students from TH Cologne and the Kharkiv National University of Radio Electronics, we can test and validate hypotheses related to this topic. Through this collaboration, we aim to better understand how temperature influences plant growth and overall plant health.

Working with different tools of the FarmBot makes big differences in planning and organizing experiments more properly. It also allows us to optimize the work process regarding the plants’ watering system. During the technical experiment, two main ideas were developed. To make them work FarmBot tools, open FarmBot source, Python programming language, namely OpenCV, PlantCV, Matplotlib, and other packages were used.

2. BIOLOGICAL EXPERIMENT (team 1)

Due to climate change, higher temperatures are expected in the climate and the soil. Therefore, a lot of the biodiversity will change or be harmed. Especially the symbiosis between a plant and microorganisms. A symbiosis is a complex interaction between two different organisms who both have benefited from this interaction.

The main topic of the biological experiment is to investigate how climate change affects plant symbiosis. According to this topic, biological students developed a hypothesis: higher temperatures above 20 degrees improve the symbiotic relationship between nodule bacteria and *Pisum Sativum*, resulting in fast plant growth. So, we decided to develop an idea by measuring the plants' height to make appropriate conclusions which approve or disapprove of the hypothesis, about temperature increases and cause faster plant growth.

So, from the biological point of view, we want to study the symbiosis between a legume plant and rhizobia bacteria. The bacteria help the plant to fixate nitrogen from the air. On the other hand, the plant provides the bacteria with nutrients and water.

3. BIOLOGICAL EXPERIMENT (team 2)

The main idea of the Inno-Bio-Div project is to investigate, approve or disapprove of the practice of some climate-related issues. By that, different biological experiments were developed. In this paragraph, we'd like to consider the second team's biological experiment.

The second biological team decided to investigate other issues, consequently their hypothesis next. Cultivating wheat as the main crop and clover as a cover crop on depleted soil under varying moisture levels will significantly enhance the productivity and yield of the main crop compared to monoculture wheat cultivation.

4. DYNAMIC WATERING

During discussions of the technical solutions for the experiment, the team of technical students were provided with the idea of improving the previous idea that describes possible solutions for creating a specific automated system of watering plants, utilizing the built-in camera of the FarmBot and programming features implemented in a general code that controls watering process.

In the previous investigation in the InnoBioDiv teaching module technical students decided to work with dynamic watering. We were provided with a paper written by students from TH Cologne from the technical team of the experiment. The suggested material discussed the limitations of automated watering systems with fixed watering points, which can result in visible watering holes in the soil and inefficient water distribution. These issues can lead to reduced plant growth and biassed data in experiments.

To address these problems, the technical experiment suggests implementing a dynamic watering solution that considers leaf positions to minimize water loss and eliminate biases in root growth. This approach involves using a camera and a colour segmentation algorithm to identify leaf locations and select randomized watering points. The use of this system is proposed to improve watering consistency, prevent the formation of watering holes, and promote distributed root growth.

4.1. GENERAL METHODOLOGY

The main idea of the dynamic watering system is to create specific conditions for watering the plants to minimize or level the damage that can be made to plants. By the idea of the previous investigation written by technical students from TH Cologne, it's possible to clarify the problem statement. The water flow can damage plants during watering plants, for example, when plants aren't strong and developed enough. For example, when we're talking about the second team's experiment, they use cover crops, which can be damaged by water flow. This is the reason for the importance of using a dynamic watering system.

Consequently, using such a system allows us to organize a more proper watering process, without damaging the leaf. This is how we can create a system, in which water flows directly on needed spots on the soil, without losing some amount of water, caused by spreading the water caused by bouncing the water off the surface of the leaves.

The idea can be implemented utilizing Python programming language, calibrated FarmBot built-in camera, and additional scripts with photo processing to provide the system with more accurate data.

4.2. APPROACHES TO REALIZE THE METHODOLOGY

To make the idea work, we decided to implement different programming tools and ideas, which will be described in this paragraph. We'd like to divide this paragraph into several steps, which will describe the needed data and needed results.

First step: Camera calibration. To add some additional features to the built-in camera of the FarmBot and to change some parameters of its work, we decided to make the calibration of the camera to be able to have more accurate data and graphical representation of needed plants, pots, or sets of pots.

To ensure proper alignment of images with the FarmBot coordinate system, it is essential to calibrate the camera. This calibration process enables scaling, rotation, and positioning adjustments to match the pixels in the images with the FarmBot framework. By achieving this alignment, images can be accurately displayed in the appropriate location on the farm designer map, and FarmBot can effectively identify and locate objects like weeds within the garden.

There are two available methods for camera calibration, we won't consider the second one in the research, because it's not relevant to our project and is much more complicated. The first (and main) method, known as "Calibration via dot grid," is the recommended approach due to its high accuracy and simplicity. This method involves

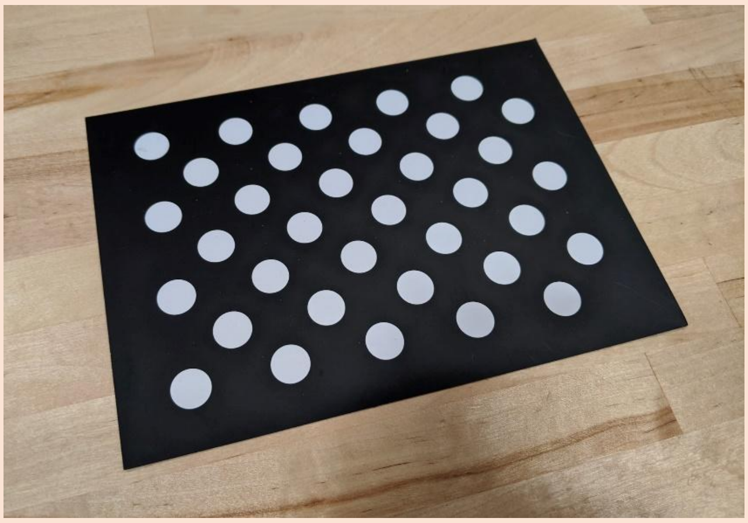


Figure 4.2.1: Printed camera calibration card

utilizing a printed camera calibration card (*Figure 4.2.1*) that features a dot grid.

This calibration card is provided with all Genesis v1.5+ and Express v1.0+ kits, making the calibration process more accessible.

To initiate the camera calibration process, follow these steps:

Step 1: Placement of Calibration Card

Take the camera calibration card and position it on the soil surface of your garden bed, ensuring that the dot grid is facing upwards. The orientation of the card itself is not critical but aligning it square with FarmBot's axes can aid in troubleshooting the calibration process if needed.

Step 2: Calibration Process

Using the controls panel, manoeuvre FarmBot directly above the camera calibration card and raise the Z-axis to its maximum height. Open the photos panel and navigate to the camera calibration section. Expand the section and click on the “Calibrate” button. FarmBot will capture a photo, move 100mm in the +Y direction, capture another photo, move 100mm in the +X direction, capture a third photo, and finally return to its initial position. Usually, it takes approximately 2-3 minutes to complete. Monitor the status ticker for progress updates.

Upon completion of the calibration process, FarmBot will upload the calibrated image and provide calculated values for *ORIGIN LOCATION IN IMAGE*, *PIXEL COORDINATE SCALE*, and *CAMERA ROTATION*.

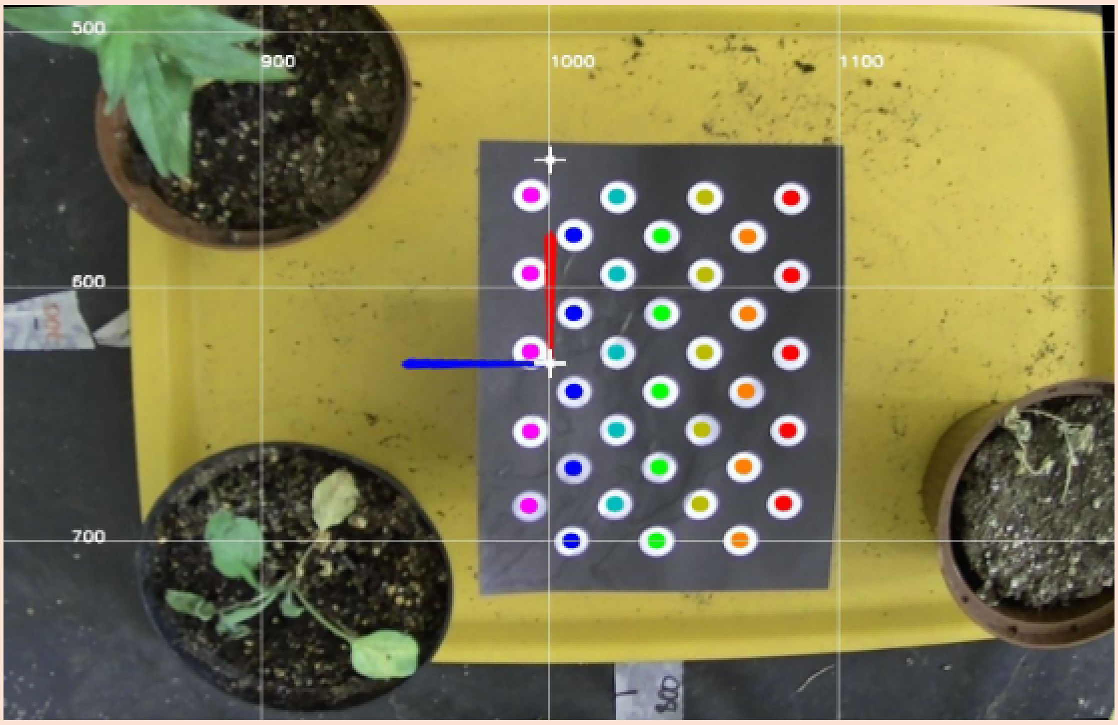


Figure 4.2.2: Calibrated image

Step 3: Result Verification

After camera calibration, the photos captured in your garden should align with the grid when viewed by the farm designer. If any objects, such as plants, appear misaligned in the photos compared to their corresponding map locations, you can adjust the *CAMERA OFFSET X* and *CAMERA OFFSET Y* until they match.

After calibrating the camera, we started to work with the coding part.

Second step: Programming. To build a needed system that performs needed operations and produces accurate results, our team decided to work with Python programming language and its features to create two programming bases. In the appendixes, you can find all the needed scripts which were used in this investigation.

Firstly, it was important to write a script which assists us in detecting exact soil without any other components, including leaves or roots (Appendix A). This code is trained to detect needed areas according to the colour, namely green and brown (with different hues).

Next, the program detects soil

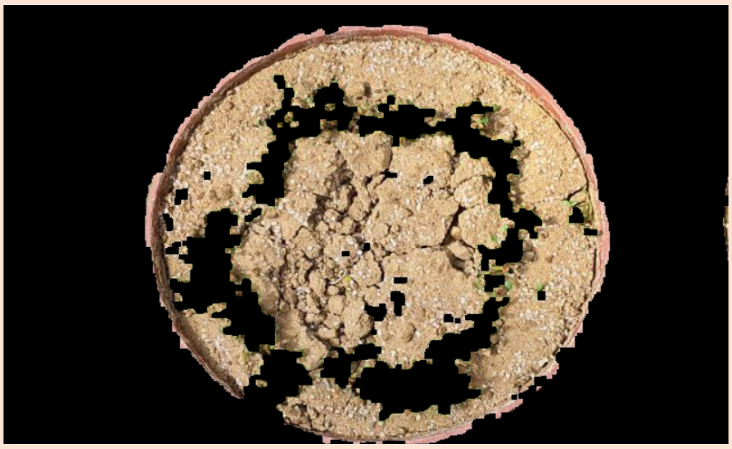


Figure 4.2.3: Identifiable watering spots

(according to colours), saves these results in the form of a NumPy array and leaves these areas on the resulting photo. Next, it detects all other objects, like leaves, roots, parts of pots, weeds, etc. The difference is that the brown areas stay on the resulting pictures, and the green ones are removed from the photo (*Figure 4.2.3*).

As a result, we have a picture of removed unneeded objects, such as leaves, roots, or cover crops. This allows us to work only with soil and detect needed "watering spots" on the soil to water only soil, not leaves or/and plants.

Secondly, the next step was to improve this variant. We decided to implement one more (additional and optional) feature of PlantCV and OpenCV packages, namely, processing the pictures according to colours.

The idea is photo processing. It's also about using PlantCV and OpenCV to work with photos which we can use the FarmBot possibilities. Using this approach, we can detect some problem zones of the plant's leaves and avoid plant diseases the first moment they might occur.

To make this idea read we're going to use photos from the FarmBot's built-in camera, PlantCV, OpenCV, and Matplotlib libraries in Python, which can help us to work with the colours of the photo to make needed results and to present them in a graphical view.

In Figure 4.2.3, you can see the grey-black variant of processed photos. The darkest areas are leaves and cover crops (which next will be removed by code from Appendix A) and more dark and bright parts of the soil. These areas were obtained because of colour processing (darker – wetter, less dark – less wet). This is how we can water less watered areas of the pot.

4.3. USING THE PROPOSED METHODOLOGY

The proposed methodology for the dynamic watering system involves two main steps: camera calibration and programming.

A printed camera calibration card with a dot grid is used in the camera calibration step. The card is placed on the soil surface, and the camera is manoeuvred above it. The calibration process captures multiple photos while moving in different directions, ensuring alignment with the FarmBot coordinate system. This calibration enables the accurate display of images and object identification within the garden.

In the programming step, Python programming language is utilized. The first script focuses on detecting soil areas by removing non-essential objects like leaves and roots based on colour. The second script enhances the process by utilizing PlantCV and OpenCV packages for colour processing. This enables the identification of problem zones on plant leaves, aiding in the prevention of plant diseases. The resulting processed photos provide a graphical representation of wetness levels, allowing targeted watering of less watered areas of the pots.

5. MEASURING THE HEIGHT OF THE PLANTS

From the very beginning, one of the ideas was to measure the height of the plants using FarmBot's built-in camera to take pictures, analyze them, and have particular conclusions about the height of the plants. To implement this idea and to make it real it's possible to use text techniques:

→ First of all, we need to take photos using the FarmBot's built-in camera using the Triangulation method suggested by Ricarda (*Figure 5.1*).

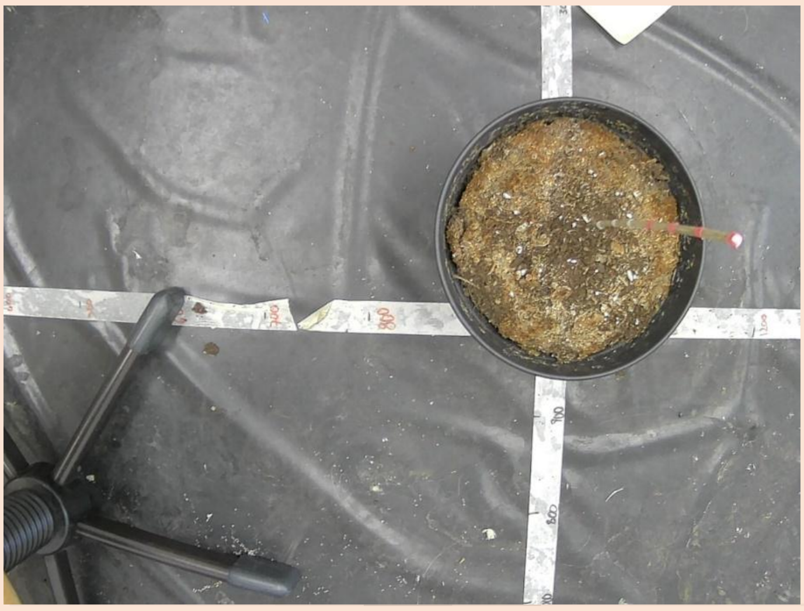


Figure 5.1: Approach for Triangulation method by putting stick with evenly spaced coloured tape

→ next analyze the photo utilizing Python programming language, namely “Matplotlib”, “OpenCV”, and “PlantCV” packages. So, the hypothesis is that such triangulation and programming tools could help us to develop the experiment and measure the height of an object by photo.

During experimental programming, we faced some issues such as that such measurements can provide us with accurate data. Such results still have errors, which are depending on different parameters, such as light and quality of the photo. And secondly, even trying to make training of the system to detect the height there are some errors which occur.

If it’s possible we can implement the next technique. Next, we’re going to use abstract data to justify the idea.

For example, from the table to the camera there is 50 cm, and this dimension is constant, so we won’t change this parameter during the experiment. Also, we have the height of the pot, in which the plant grows, for example, 7 cm. So, we can already make an equation that we have the dimension between the beginning of the plant (from the soil surface) to the camera distance equal to 43 cm (fig. 3). So, using OpenCV and PlantCV in Python it’s probably possible to find a “hot spot” which is the closest to the FarmBot’s camera and to detect how far it is from the camera. So, we’ll have for example 10 cm, so our general height is 50 minus pot height 7 cm, minus hot-spot-to-camera-distance equal to a plant height: 50 – 7 – 10 = 33.0 cm. And this number with an error of approximately +1.0 cm or –1.0 cm can be considered as a plant’s height.

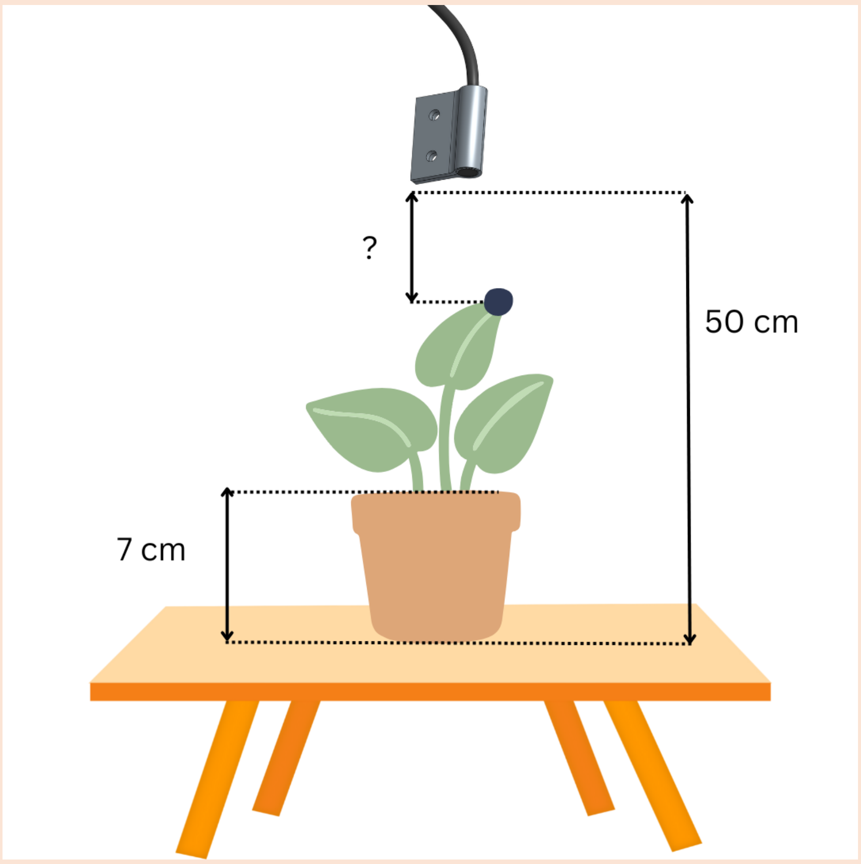


Figure 5.2: Measuring plant's height by Triangulation method

1 cm is a great error, so to minimize this error we can try to provide the program with almost errorless input data. This is how we can reduce an error in the general result of the calculation.

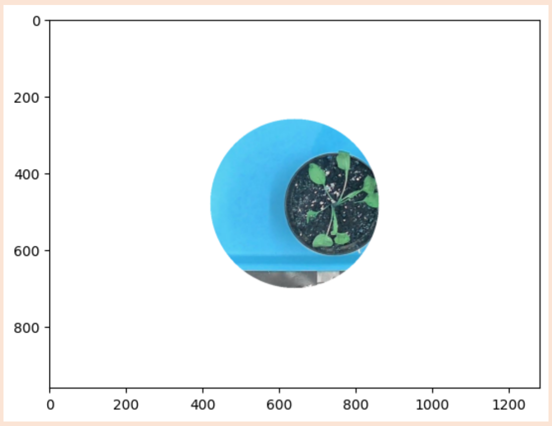
This is the tech hypothesis on how we can detect plant height using the camera, so the next step is to find a solution on how to transform it into a programmable view in Python.

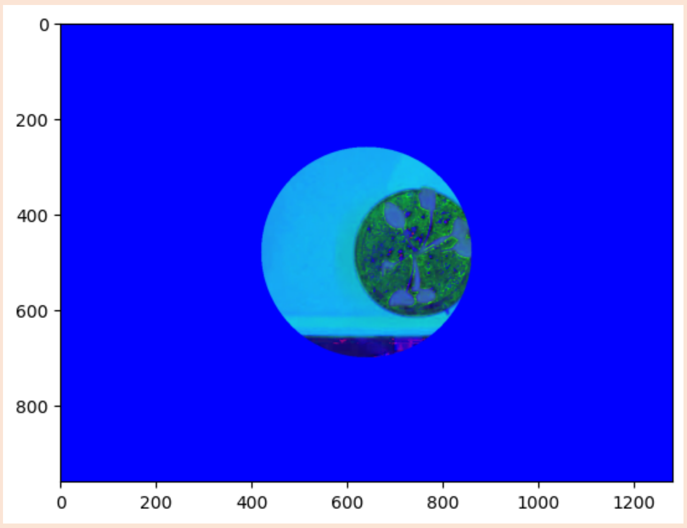
5.1. GENERAL METHODOLOGY

The methodology for measuring the height of plants using FarmBot's built-in camera involves the following steps:

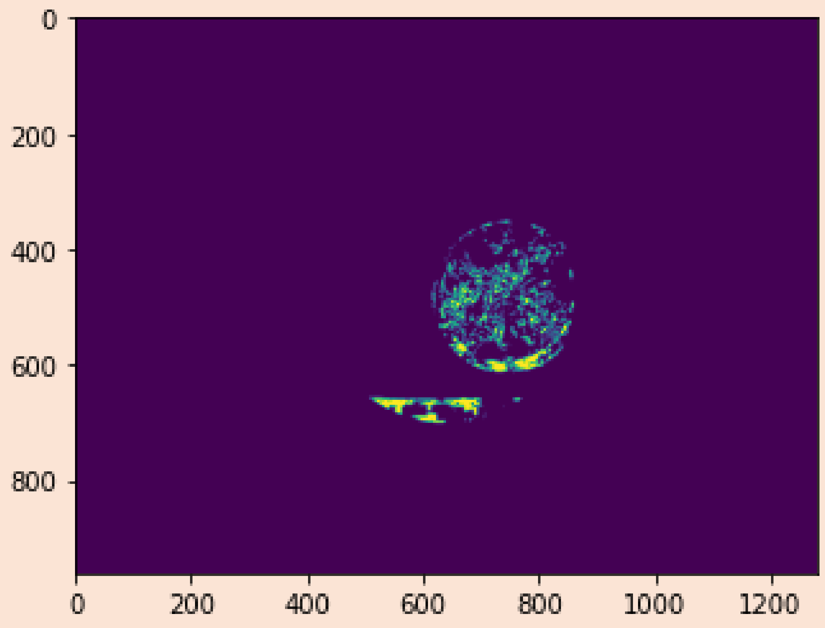
1. Photo Capture: Utilize the FarmBot's built-in camera to capture photos of the plants. The Triangulation method suggested by Ricarda (*Figure 5.2*) can be used to capture accurate images.

2. Photo Analysis: Analyse the captured photos using Python programming language and relevant packages such as "Matplotlib," "OpenCV," and "PlantCV." The hypothesis is that these triangulation and programming tools can aid in developing the experiment and measuring the height of plants through photographs.





(a) cropped image to only get the pot portion *(b) cropped image converted from BGR to HSV*



(c) final masked image

3. Error Handling: During the experimental programming, it is important to address potential issues and sources of error. Measurements may still contain errors influenced by various parameters, including lighting conditions and photo quality. Additionally, training the system to detect height may also result in errors.

4. Utilization of Abstract Data: Incorporate abstract data to support the methodology. For example, if the distance between the camera and the table is constant at 50 cm, and the height of the pot in which the plant grows is 7 cm, an equation can be established to calculate the distance between the plant's base (from the soil surface) and the camera, which would be 43 cm (fig. 3). By using OpenCV and PlantCV in Python, it may be possible to identify a "hot spot" closest to the FarmBot's camera and determine its distance. Subtracting the pot height and the hot-spot-to-camera distance from the total height, the plant's height can be calculated, with an error margin of approximately +1.0 cm or -1.0 cm.

5. Error Reduction: To minimize errors in the overall calculation, efforts can be made to provide the program with highly accurate input data. Minimizing errors in the input data can help reduce the error in the final height measurement.

6. Implementation and Programming: The next step is to transform the proposed methodology into a programmable approach using Python. This involves translating the steps and techniques discussed above into code that can be executed by the software.

By following this methodology, it is expected to detect and measure the height of plants accurately using the FarmBot's camera, while also considering potential sources of error and implementing measures to minimize them.

5.2. USING THE PROPOSED METHODOLOGY

Such an idea allows us to have a hypothesis of the possibility of measuring the height of the plants by photo processing and calculations done in Python programming language. To make the calculations real it is possible to use some additional sensors and/or lasers for improving photos which will be used in further operations and processing.

Using the proposed methodology, we can explore the hypothesis of measuring plant height through photo processing and calculations conducted in the Python programming language. To implement this approach effectively, additional sensors and/or lasers can be employed to enhance the quality of the photos, which will be utilized in subsequent operations and processing.

The methodology involves the following steps:

1. Data Acquisition: Utilize FarmBot's built-in camera and additional sensors/lasers to capture high-quality photos of the plants. The use of sensors/lasers can help enhance the accuracy and precision of the acquired data, providing clearer images for analysis.

2. Image Preprocessing: Apply preprocessing techniques using Python packages such as "OpenCV" to enhance the quality of the acquired photos. This may involve operations such as noise reduction, image denoising, and contrast enhancement to ensure optimal data quality for subsequent analysis.

3. Feature Extraction: Utilize image processing algorithms and techniques to extract relevant features from the preprocessed photos. This may involve identifying key points, edges, or specific regions of interest within the images that correspond to plant height.

4. Triangulation: Implement the triangulation method, as suggested by Ricarda (figure 2), to estimate the height of the plants based on the identified features in the photos. By measuring the relative positions of the features from different angles, triangulation can help calculate the plant height accurately.

5. Calculation and Analysis: Employ Python programming language, along with packages like "Matplotlib" and "PlantCV," to perform the necessary calculations and analysis. The captured photos and extracted features will be processed to determine the precise height of the plants. This step may involve complex mathematical algorithms and equations to derive the final measurements.

6. Error Mitigation: Address potential sources of error during the calculation and analysis process. Factors such as lighting conditions, image distortion, or variations in plant growth patterns can introduce inaccuracies. By implementing error mitigation strategies, such as considering error margins or conducting multiple measurements, the reliability and accuracy of the calculated plant height can be improved.

By following this detailed methodology, which includes the integration of additional sensors/lasers and utilizing advanced image processing techniques in Python, it is expected to achieve more accurate and reliable measurements of plant height. The combination of triangulation, calculations, and error mitigation strategies will contribute to a comprehensive approach for assessing the height of plants based on photo data.

6 EXPECTED RESULTS

To sum up, it is worth admitting that using both technical ideas will help us to develop engineering-related things to make biological experiments more reliable and accurate.

The first idea will help us and further researchers to improve the watering process and turn ordinary watering systems into more accurate dynamic watering systems. It will help biological researchers as well because this methodology and its direct implementation will help to level, reduce, and even completely avoid the spreading water on the surface, losing some amount of water, and control the watering system.

The second idea is theoretical in our case, but practically it can be done utilizing some additional equipment. The idea of measuring the height of the plants through photo processing will reduce the implementation of human work, such as manual measuring of the height of plants.

7 WORKING WITH WINRHIZO

After harvesting, we listed all info about 72 plants (except number 35 died) into an Excel sheet, segregating in different columns like Genotype, Microbe, Temperature, Fresh weight shoot, Fresh weight root, Shoot root ratio, Shoot height, Leaves number, Nodules number. Again, it was needed to extract data of Root length and Root volume to understand how much or to what extent these were influenced by different temperatures (RT, 26, 30). Root length and root volume are crucial parameters for assessing plant growth and developments. Others include, assessing nutrient and water uptake efficiency (larger root system with increased length and volume indicates a greater potential for resource acquisition from the soil), Studying root-microbe interactions, Assessing stress tolerance and adaptation, Evaluating treatment efficacy. With WinRhizo, we got precise data and visualisation of these parameters.

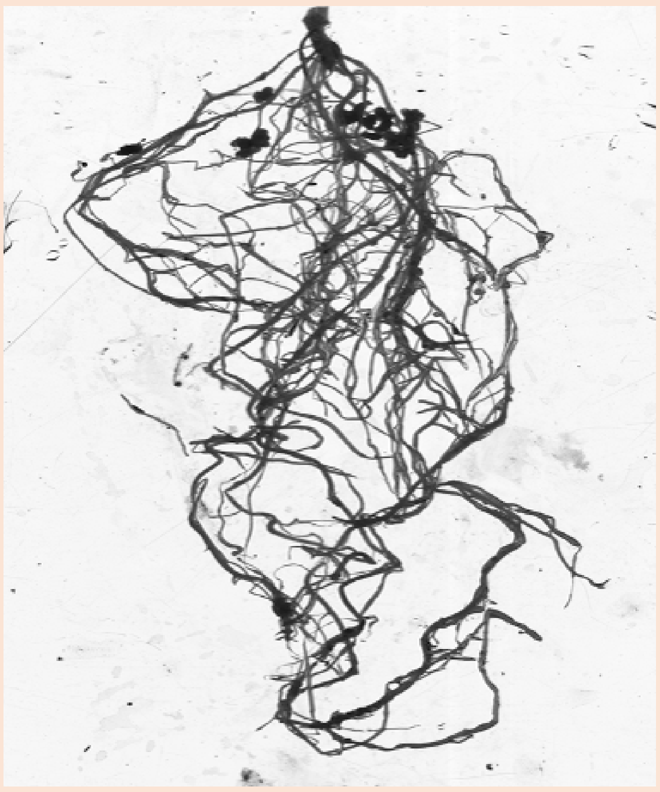


Figure 7.1: root of plant 18 (WT, +Rhizobia, RT - having highest recorded Nodules_number: 46 and Root_length and volume: 5.444.214 cm and 3.023 cm^3)

The next task is to analyse and compare the collected data in the R studio. For that, an R script was prepared.

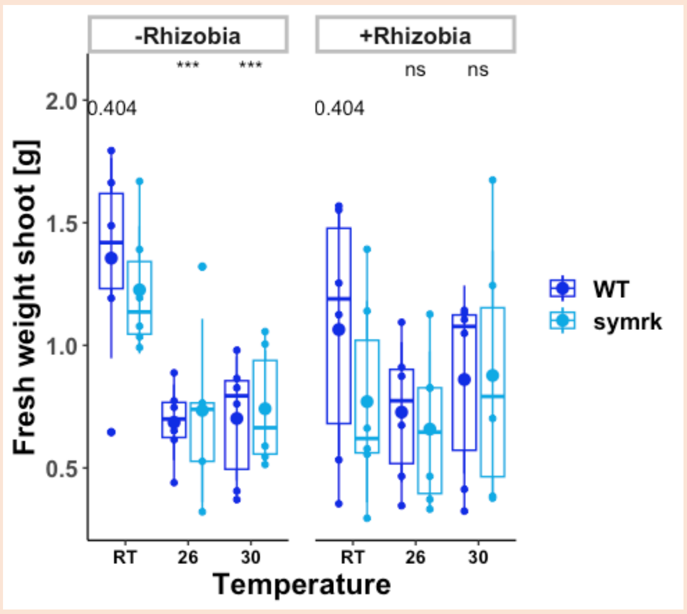


Figure 7.2: Shoot weight vs Temperature for WT and symrk (with or without the presence of Rhizobia)

This plot scrutinizes temperature influence on shoot weight which is also crucial because changes in shoot weight can indicate how well the plant is benefiting from the nitrogen-fixing ability of rhizobia.

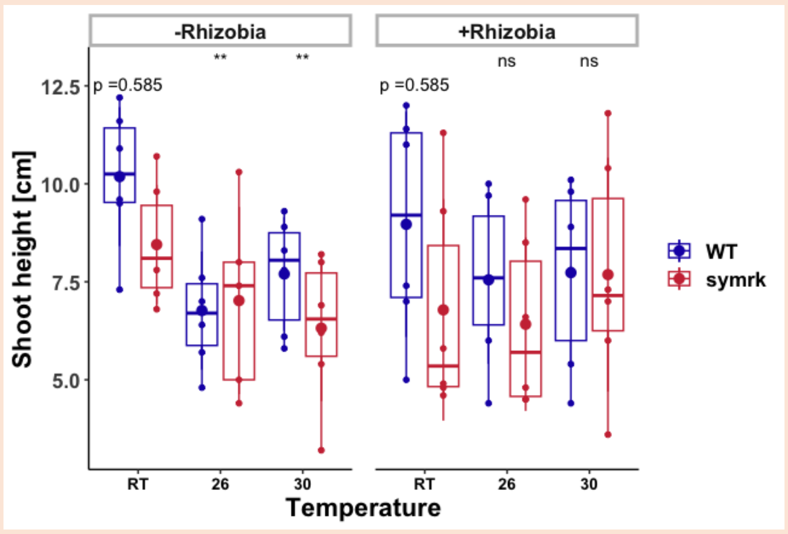


Figure 7.3: Shoot height vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying shoot height, we can gain insights into how different temperatures influence the growth patterns and overall development of the plant, particularly in the presence or absence of rhizobia.

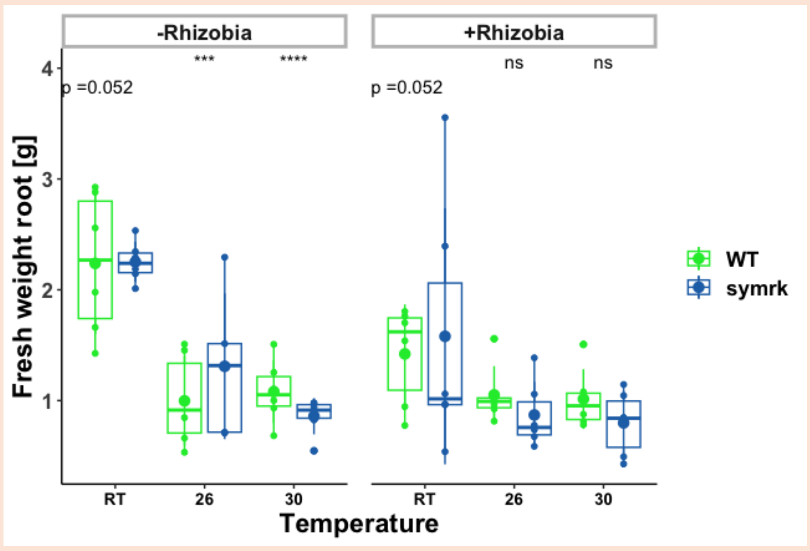


Figure 7.4: Root Weight vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying root weight, we can understand how different temperatures affect the allocation of resources, particularly in the presence or absence of rhizobia. Rhizobia symbiosis can influence root biomass by enhancing nutrient availability, such as nitrogen, which can affect root growth and weight.

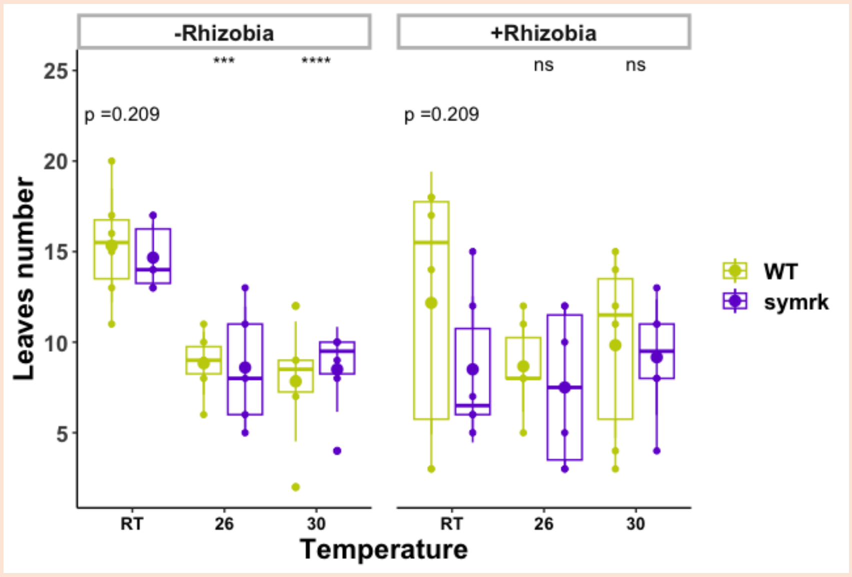


Figure 7.5: Leaves number vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying leaf number, we can gain insights into how different temperatures influence leaf initiation, expansion, and overall plant development, particularly in the presence or absence of rhizobia.

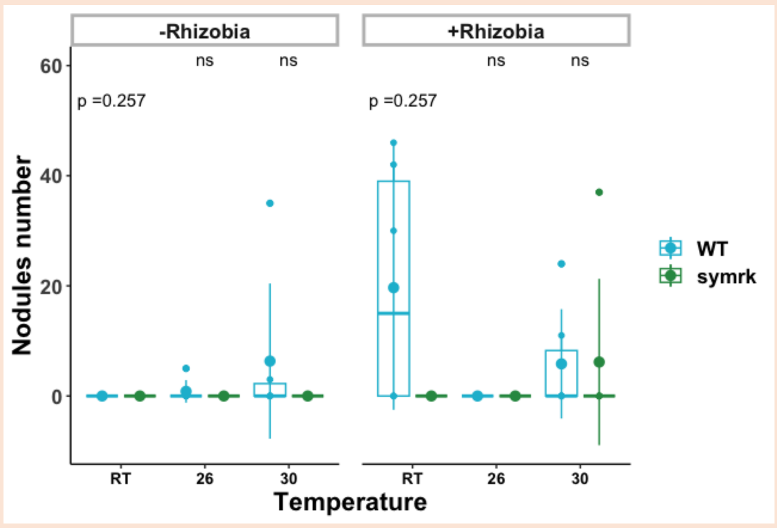


Figure 7.6: Nodules number vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying nodule number, we can assess the effectiveness of the symbiotic relationship between the plant and rhizobia. Understanding how nodule number varies with different temperatures provides insights into the efficiency of rhizobia colonization and nodulation under different environmental conditions.

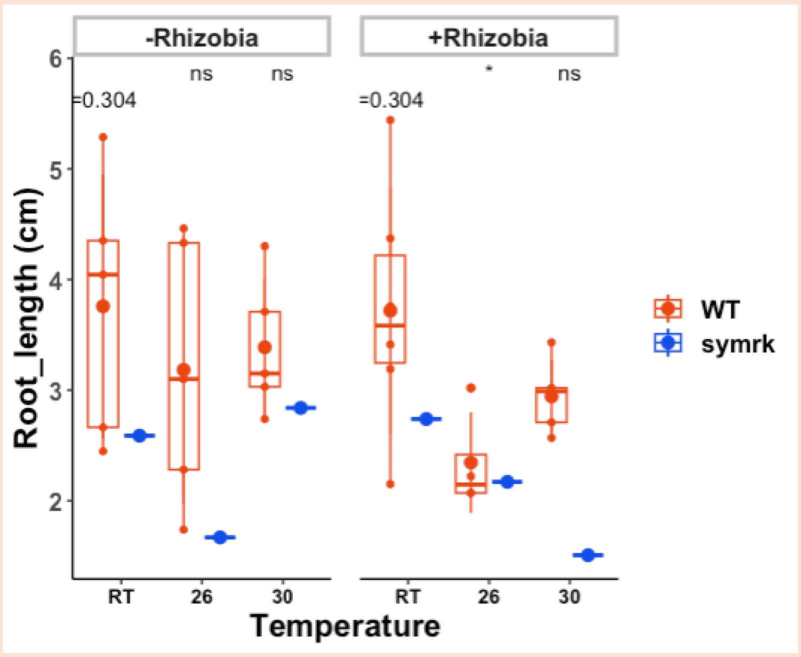


Figure 7.7: Root length vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying root length, we can understand how different temperatures affect root growth and architecture in the presence or absence of rhizobia.

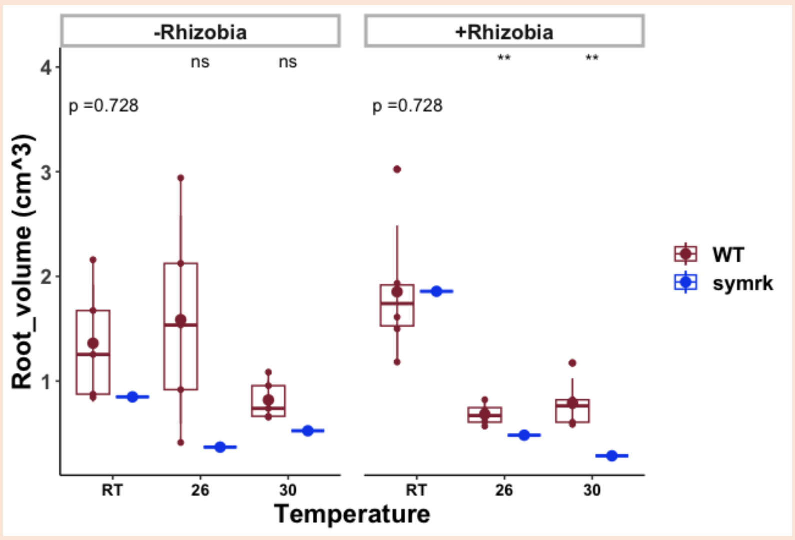


Figure 7.8: Root volume vs Temperature for WT and symrk (with or without the presence of Rhizobia)

By studying root volume, we can gain insights into how different temperatures affect the architectural development of the root system in the presence or absence of rhizobia.

CONCLUSION

In conclusion, the proposed methodology of implementing a dynamic watering system and measuring the height of plants through photo processing is expected to yield several positive outcomes.

1.1. Dynamic Watering System

The implementation of a dynamic watering system based on the proposed methodology is anticipated to offer the following benefits:

Improved watering process: By organizing a more targeted watering process, the system will minimize damage to plants, especially when they are not fully developed or have cover crops that are susceptible to water flow damage.

Reduced water wastage: The system will ensure that water flows directly to the desired spots on the soil, minimizing water loss caused by water bouncing off the surface of leaves.

Enhanced accuracy: The calibration of the FarmBot's built-in camera, coupled with photo processing and analysis using Python programming language, will provide more precise data for effective watering decisions.

Optimal plant health: By utilizing photo processing techniques, such as PlantCV and OpenCV, potential problem zones on plant leaves can be detected early, helping to prevent plant diseases and promote overall plant health.

1.2. Plant Height Measurement

The proposed methodology for measuring plant height through photo processing is expected to offer the following outcomes:

Automation and efficiency: By leveraging the FarmBot's built-in camera and Python programming language, the manual process of measuring plant height can be automated, saving time and effort.

Improved accuracy: While some errors may still exist due to factors like lighting conditions and photo quality, incorporating additional sensors and lasers can help enhance the accuracy of the measurements.

Error mitigation: By implementing error mitigation strategies, such as considering error margins and conducting multiple measurements, the reliability of the calculated plant height can be increased.

Reduction of manual work: The methodology eliminates the need for manual measurement of plant height, making the process more convenient and reducing the potential for human errors.

In summary, the implementation of the proposed methodology for the dynamic watering system and plant height measurement is expected to enhance the accuracy, efficiency, and reliability of these processes. These advancements have the potential to improve agricultural practices, facilitate biological research, and contribute to the development of more sophisticated and automated systems in the field of plant cultivation.

REFERENCES

1. Alexander Schumski Chhavi Dadhich Helen Schmidt Omar Raad. Dynamic Watering Algorithm. A Computer Vision Approach. 2022.

2. “Camera Calibration.” FarmBot Software Documentation, software.farm.bot/v12/The-FarmBot-Web-App/photos/camera-calibration.html. Accessed 22 June 2023.

3. “Intro to FarmBot Express.” FarmBot Express Documentation, express.farm.bot/v1.1/assembly/intro. Accessed 21 June 2023.

4. “Intro to FarmBot Genesis.” FarmBot Genesis Documentation, genesis.farm.bot/v1.6/assembly/intro. Accessed 21 June 2023.

5. Farmbot | Drone.ua. drone.ua/farmbot/. Accessed 21 June 2023.

6. “FarmBot.” Wikipedia, 19 Mar. 2020, en.wikipedia.org/wiki/FarmBot.

7. “OpenCV: OpenCV-Python Tutorials.” Docs.opencv.org, docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

8. “Roi Tutorial - PlantCV.” Plantcv.readthedocs.io, plantcv.readthedocs.io/en/stable/tutorials/roi_tutorial/. Accessed 25 May 2023.

9. “The FarmBot Web App.” My.farm.bot, my.farm.bot/app/designer/controls. Accessed 21 June 2023.

APPENDIX A

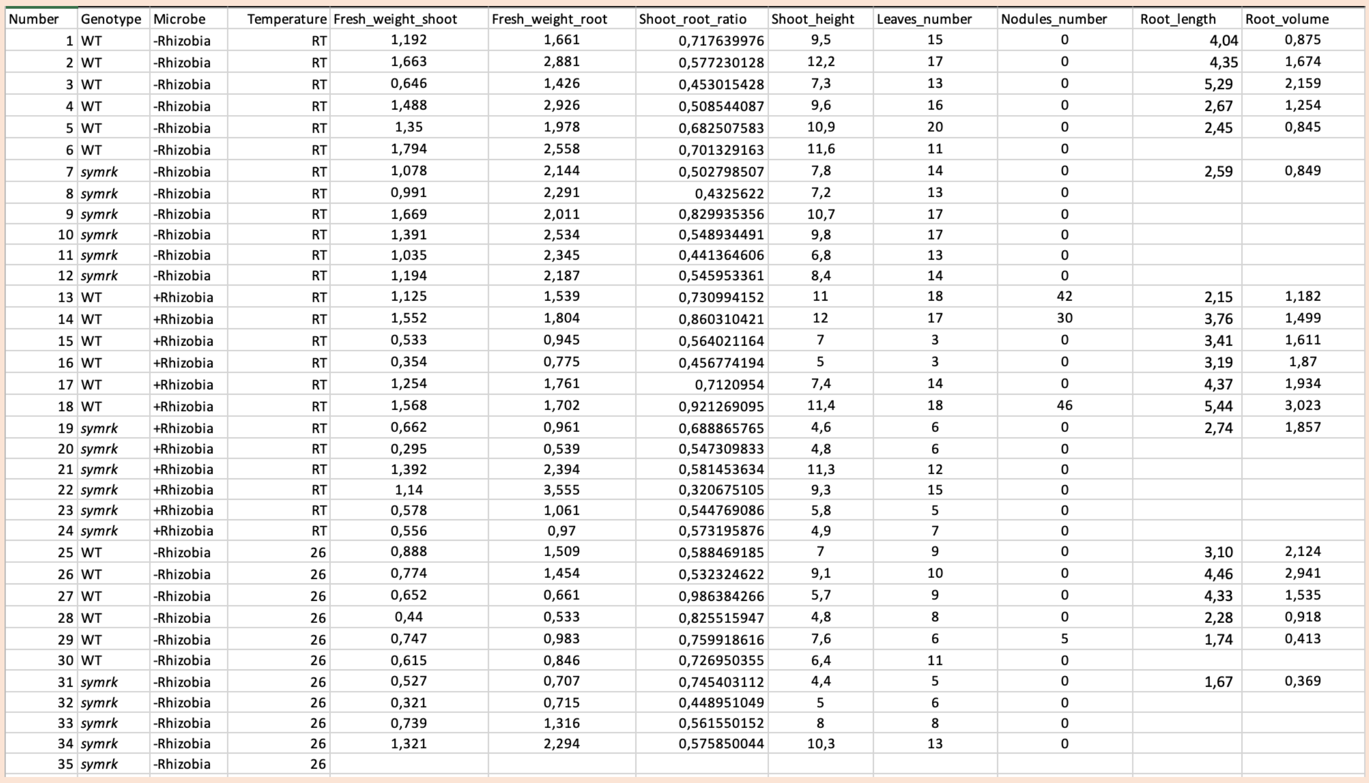
```
import cv2
import numpy as np

# Load the image
image = cv2.imread('soil_plant.jpg')

# Resize the image if needed (optional)
# image = cv2.resize(image, (new_width, new_height))

new_width = 1800
new_height = 1600

# Convert the image from BGR to HSV colour space
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Define the lower and upper thresholds for the brown colour representing soil
lower_brown = np.array([0, 50, 20])  # Adjust these values according to the specific shade of brown
upper_brown = np.array([30, 255, 255])  # Adjust these values according to the specific shade of brown

# Define the lower and upper thresholds for the green colour representing leaves
lower_green = np.array([36, 25, 25])  # Adjust these values according to the specific shade of green
upper_green = np.array([86, 255, 255])  # Adjust these values according to the specific shade of green

# Create masks based on the defined thresholds
mask_brown = cv2.inRange(hsv, lower_brown, upper_brown)
mask_green = cv2.inRange(hsv, lower_green, upper_green)

# Apply morphological operations to enhance the masks
kernel = np.ones((5, 5), np.uint8)
mask_brown = cv2.morphologyEx(mask_brown, cv2.MORPH_OPEN, kernel)
mask_brown = cv2.morphologyEx(mask_brown, cv2.MORPH_CLOSE, kernel)
mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_OPEN, kernel)
mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_CLOSE, kernel)

# Subtract the green mask from the brown mask to remove green elements
mask_soil = cv2.subtract(mask_brown, mask_green)

# Apply the soil mask to the original image
result = cv2.bitwise_and(image, image, mask=mask_soil)

# Display the resulting image
cv2.imshow('Detected Soil', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

APPENDIX B

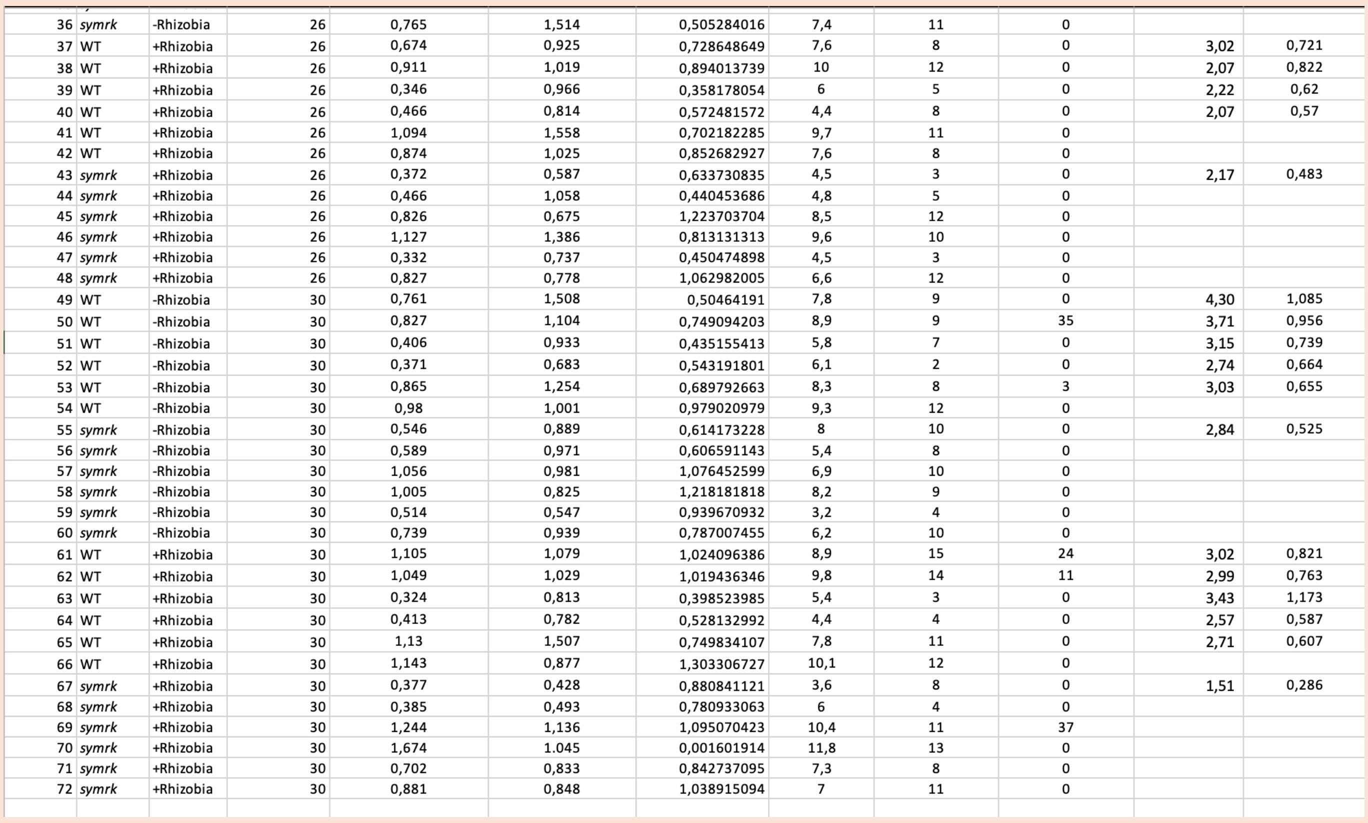
```
from plantcv import plantcv as pcv
import matplotlib

class options:
    def __init__(self):
        self.image = "soil_plant.jpg"
        self.debug = "plot"
        self.writeimg= False
        self.result = "vis_tutorial_results.json"
        self.outdir = "." # Store the output to the current directory

# Get options
args = options()

# Set debug to the global parameter
pcv.params.debug = args.debug

img, path, filename = pcv.read image(filename=args.image)
s = pcv.rgb2gray_hsv(rgb_img=img, channel='s')
s_thresh = pcv.threshold.binary(gray_img=s, threshold=85, max_value=255, object_type='light')
s_mblur = pcv.median_blur(gray_img=s_thresh, ksize=5)
gaussian_img = pcv.gaussian_blur(img=s_thresh, ksize=(5, 5), sigma_x=0, sigma_y=None)
b = pcv.rgb2gray_lab(rgb_img=img, channel='b')
b_thresh = pcv.threshold.binary(gray_img=b, threshold=160, max_value=255,
                                object_type='light')
bs = pcv.logical_or(bin_img1=s_mblur, bin_img2=b_thresh)
masked = pcv.apply_mask(img=img, mask=bs, mask_color='white')
masked_a = pcv.rgb2gray_lab(rgb_img=masked, channel='a')
masked_b = pcv.rgb2gray_lab(rgb_img=masked, channel='b')
maskeda_thresh = pcv.threshold.binary(gray_img=masked_a, threshold=115,
                                      max_value=255, object_type='dark')
maskeda_thresh1 = pcv.threshold.binary(gray_img=masked_a, threshold=135,
                                       max_value=255, object_type='light')
maskedb_thresh = pcv.threshold.binary(gray_img=masked_b, threshold=128,
                                      max_value=255, object_type='light')
ab1 = pcv.logical_or(bin_img1=maskeda_thresh, bin_img2=maskedb_thresh)
ab = pcv.logical_or(bin_img1=maskeda_thresh1, bin_img2=ab1)
opened_ab = pcv.opening(gray_img=ab)
xor_img = pcv.logical_xor(bin_img1=maskeda_thresh, bin_img2=maskedb_thresh)
ab_fill = pcv.fill(bin_img=ab, size=200)
closed_ab = pcv.closing(gray_img=ab_fill)
masked2 = pcv.apply_mask(img=masked, mask=ab_fill, mask_color='white')
id_objects, obj_hierarchy = pcv.find_objects(img=masked2, mask=ab_fill)
id_objects, obj_hierarchy = pcv.find_objects(img=masked2, mask=ab_fill)
roi1, roi_hierarchy= pcv.roi.rectangle(img=masked2, x=100, y=100, h=200, w=200)
roi_objects, hierarchy3, kept_mask, obj_area = pcv.roi_objects(img=img, roi_contour=roi1,
                                                               roi_hierarchy=roi_hierarchy,
                                                object_contour=id_objects,
                                                obj_hierarchy=obj_hierarchy,
                                                roi_type='partial')
obj, mask = pcv.object_composition(img=img, contours=roi_objects, hierarchy=hierarchy3)
analysis_image = pcv.analyze_object(img=img, obj=obj, mask=mask, label="default")
boundary_image2 = pcv.analyze_bound_horizontal(img=img, obj=obj, mask=mask, line_position=370,
        label="default")
color_histogram = pcv.analyze_color(rgb_img=img, mask=kept_mask, colorspaces='all', label="default")
pcv.print_image(img=color_histogram, filename="vis_tutorial_color_hist.jpg")
top_x, bottom_x, center_v_x = pcv.x_axis_pseudolandmarks(img=img, obj=obj, mask=mask,
        label="default")
top_y, bottom_y, center_v_y = pcv.y_axis_pseudolandmarks(img=img, obj=obj, mask=mask)
pcv.outputs.save_results(filename=args.result)
```

APPENDIX C

Excel sheet (Group 1)



Number	Genotype	Microbe	Temperature	Fresh_weight_shoot	Fresh_weight_root	Shoot_root_ratio	Shoot_height	Leaves_number	Nodules_number	Root_length	Root_volume
1	WT	-Rhizobia	RT	1,192	1,661	0,717639976	9,5	15	0	4,04	0,875
2	WT	-Rhizobia	RT	1,663	2,881	0,577230128	12,2	17	0	4,35	1,674
3	WT	-Rhizobia	RT	0,646	1,426	0,453015428	7,3	13	0	5,29	2,159
4	WT	-Rhizobia	RT	1,488	2,926	0,508544087	9,6	16	0	2,67	1,254
5	WT	-Rhizobia	RT	1,35	1,978	0,682507583	10,9	20	0	2,45	0,845
6	WT	-Rhizobia	RT	1,794	2,558	0,701329163	11,6	11	0		
7	*symrk*	-Rhizobia	RT	1,078	2,144	0,502798507	7,8	14	0	2,59	0,849
8	*symrk*	-Rhizobia	RT	0,991	2,291	0,4325622	7,2	13	0		
9	*symrk*	-Rhizobia	RT	1,669	2,011	0,829935356	10,7	17	0		
10	*symrk*	-Rhizobia	RT	1,391	2,534	0,548934491	9,8	17	0		
11	*symrk*	-Rhizobia	RT	1,035	2,345	0,441364606	6,8	13	0		
12	*symrk*	-Rhizobia	RT	1,194	2,187	0,545953361	8,4	14	0		
13	WT	+Rhizobia	RT	1,125	1,539	0,730994152	11	18	42	2,15	1,182
14	WT	+Rhizobia	RT	1,552	1,804	0,860310421	12	17	30	3,76	1,499
15	WT	+Rhizobia	RT	0,533	0,945	0,564021164	7	3	0	3,41	1,611
16	WT	+Rhizobia	RT	0,354	0,775	0,456774194	5	3	0	3,19	1,87
17	WT	+Rhizobia	RT	1,254	1,761	0,7120954	7,4	14	0	4,37	1,934
18	WT	+Rhizobia	RT	1,568	1,702	0,921269095	11,4	18	46	5,44	3,023
19	*symrk*	+Rhizobia	RT	0,662	0,961	0,688865765	4,6	6	0	2,74	1,857
20	*symrk*	+Rhizobia	RT	0,295	0,539	0,547309833	4,8	6	0		
21	*symrk*	+Rhizobia	RT	1,392	2,394	0,581453634	11,3	12	0		
22	*symrk*	+Rhizobia	RT	1,14	3,555	0,320675105	9,3	15	0		
23	*symrk*	+Rhizobia	RT	0,578	1,061	0,544769086	5,8	5	0		
24	*symrk*	+Rhizobia	RT	0,556	0,97	0,573195876	4,9	7	0		
25	WT	-Rhizobia	26	0,888	1,509	0,588469185	7	9	0	3,10	2,124
26	WT	-Rhizobia	26	0,774	1,454	0,532324622	9,1	10	0	4,46	2,941
27	WT	-Rhizobia	26	0,652	0,661	0,986384266	5,7	9	0	4,33	1,535
28	WT	-Rhizobia	26	0,44	0,533	0,825515947	4,8	8	0	2,28	0,918
29	WT	-Rhizobia	26	0,747	0,983	0,759918616	7,6	6	5	1,74	0,413
30	WT	-Rhizobia	26	0,615	0,846	0,726950355	6,4	11	0		
31	*symrk*	-Rhizobia	26	0,527	0,707	0,745403112	4,4	5	0	1,67	0,369
32	*symrk*	-Rhizobia	26	0,321	0,715	0,448951049	5	6	0		
33	*symrk*	-Rhizobia	26	0,739	1,316	0,561550152	8	8	0		
34	*symrk*	-Rhizobia	26	1,321	2,294	0,575850044	10,3	13	0		
35	*symrk*	-Rhizobia	26								



36	*symrk*	-Rhizobia	26	0,765	1,514	0,505284016	7,4	11	0		
37	WT	+Rhizobia	26	0,674	0,925	0,728648649	7,6	8	0	3,02	0,721
38	WT	+Rhizobia	26	0,911	1,019	0,894013739	10	12	0	2,07	0,822
39	WT	+Rhizobia	26	0,346	0,966	0,358178054	6	5	0	2,22	0,62
40	WT	+Rhizobia	26	0,466	0,814	0,572481572	4,4	8	0	2,07	0,57
41	WT	+Rhizobia	26	1,094	1,558	0,702182285	9,7	11	0		
42	WT	+Rhizobia	26	0,874	1,025	0,852682927	7,6	8	0		
43	*symrk*	+Rhizobia	26	0,372	0,587	0,633730835	4,5	3	0	2,17	0,483
44	*symrk*	+Rhizobia	26	0,466	1,058	0,440453686	4,8	5	0		
45	*symrk*	+Rhizobia	26	0,826	0,675	1,223703704	8,5	12	0		
46	*symrk*	+Rhizobia	26	1,127	1,386	0,813131313	9,6	10	0		
47	*symrk*	+Rhizobia	26	0,332	0,737	0,450474898	4,5	3	0		
48	*symrk*	+Rhizobia	26	0,827	0,778	1,062982005	6,6	12	0		
49	WT	-Rhizobia	30	0,761	1,508	0,50464191	7,8	9	0	4,30	1,085
50	WT	-Rhizobia	30	0,827	1,104	0,749094203	8,9	9	35	3,71	0,956
51	WT	-Rhizobia	30	0,406	0,933	0,435155413	5,8	7	0	3,15	0,739
52	WT	-Rhizobia	30	0,371	0,683	0,543191801	6,1	2	0	2,74	0,664
53	WT	-Rhizobia	30	0,865	1,254	0,689792663	8,3	8	3	3,03	0,655
54	WT	-Rhizobia	30	0,98	1,001	0,979020979	9,3	12	0		
55	*symrk*	-Rhizobia	30	0,546	0,889	0,614173228	8	10	0	2,84	0,525
56	*symrk*	-Rhizobia	30	0,589	0,971	0,606591143	5,4	8	0		
57	*symrk*	-Rhizobia	30	1,056	0,981	1,076452599	6,9	10	0		
58	*symrk*	-Rhizobia	30	1,005	0,825	1,218181818	8,2	9	0		
59	*symrk*	-Rhizobia	30	0,514	0,547	0,939670932	3,2	4	0		
60	*symrk*	-Rhizobia	30	0,739	0,939	0,787007455	6,2	10	0		
61	WT	+Rhizobia	30	1,105	1,079	1,024096386	8,9	15	24	3,02	0,821
62	WT	+Rhizobia	30	1,049	1,029	1,019436346	9,8	14	11	2,99	0,763
63	WT	+Rhizobia	30	0,324	0,813	0,398523985	5,4	3	0	3,43	1,173
64	WT	+Rhizobia	30	0,413	0,782	0,528132992	4,4	4	0	2,57	0,587
65	WT	+Rhizobia	30	1,13	1,507	0,749834107	7,8	11	0	2,71	0,607
66	WT	+Rhizobia	30	1,143	0,877	1,303306727	10,1	12	0		
67	*symrk*	+Rhizobia	30	0,377	0,428	0,880841121	3,6	8	0	1,51	0,286
68	*symrk*	+Rhizobia	30	0,385	0,493	0,780933063	6	4	0		
69	*symrk*	+Rhizobia	30	1,244	1,136	1,095070423	10,4	11	37		
70	*symrk*	+Rhizobia	30	1,674	1.045	0,001601914	11,8	13	0		
71	*symrk*	+Rhizobia	30	0,702	0,833	0,842737095	7,3	8	0		
72	*symrk*	+Rhizobia	30	0,881	0,848	1,038915094	7	11	0		

APPENDIX D

R script

Load packages----

install.packages("ggpubr")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("RColorBrewer")
install.packages("tidyverse")
install.packages("broom")
install.packages("agricolae")
install.packages("dplyr")
install.packages("purrr")
install.packages("tidyr")
install.packages("Hmisc")
install.packages("ade4")
install.packages("ggrepel")

```
install.packages("ggsignif")
install.packages("readxl")
install.packages("FSA")
install.packages("rcompanion")
install.packages("rstatix")
install.packages("ggsignif")
install.packages("stats")
```

#Load packages into current R session

```
library(ggpubr)  #kruskal test
library(ggplot2) #visualize data
library(RColorBrewer)#color packages
library(tidyverse)#data manipulation
library(broom)#data manipulation
library(agricolae)#statistics
library(dplyr)#data management
library(purrr)#data management
library(tidyr)#data management
library(Hmisc)#data management
library(ade4)#statistics
library(ggrepel)#add text to a plot
library(ggsignif)#add p values + statistics on a plot
library(readxl)#read excel files
library(FSA)
library(rcompanion)
library(rstatix)
library(ggsignif)
```

#Loading data----

```
dataset <- read_excel("Experiment_results Root_length vs Temp.xlsx",sheet = "Group1")

View(dataset)
head(dataset)
nrow(dataset)
ncol(dataset)
dim(dataset)
```

#Tidying up data----

```
dataset %>%
  drop_na(Root_length)->dataset2

dataset2$`Fresh_weight_shoot`<-as.numeric(dataset2$`Fresh_weight_shoot`) #it was set as a
character, this interfered with our data cleaning, so we are setting as numeric
dataset2$`Fresh_weight_root`<-as.numeric(dataset2$`Fresh_weight_root`)
dataset2$`Shoot_height`<-as.numeric(dataset2$`Shoot_height`)
dataset2$`Shoot_root_ratio`<-as.numeric(dataset2$`Shoot_root_ratio`)
dataset2$`Leaves_number`<-as.numeric(dataset2$`Leaves_number`)
```

```
dataset2$`Nodules_number`<-as.numeric(dataset2$`Nodules_number`)
dataset2$`Root_length`<-as.numeric(dataset2$`Root_length`)
```

#Segmenting measured variable into one column and values into another column

```
dataset2 %>%
  pivot_longer(c(Fresh_weight_shoot,Fresh_weight_root,Shoot_height, Shoot_root_ratio,
Leaves_number, Nodules_number, Root_length),names_to="Measurement", values_to="values" )->
dataset3
```

#Types
#Converting to factors

```
dataset3$Measurement<-as.factor(dataset3$Measurement)


#converting to factors and ordering our levels
dataset2$Microbe= factor(dataset2$Microbe)
dataset2$Genotype = factor(dataset2$Genotype, levels=c("WT", "symrk"))
dataset2$Temperature = factor(dataset2$Temperature, levels=c("RT", "26", "30"))

#for statistical analysis we are going to generate a new variable, which is a combination of our three
variables
dataset2$Condition<-paste(dataset2$Genotype,dataset2$Microbe,dataset2$Temperature, sep="_")
dataset2$Condition<-as.factor(dataset2$Condition)
```

#Statistics for significance----

```
  arrange(Microbe, Genotype, Temperature, Measurement) %>% #arrange your data according to your
variables of desire
  na.omit() %>% #get rid of remaining NAs
  group_by(Microbe, Genotype, Temperature, Measurement) %>% #grouping your data according to
variables for the next functions
  summarise(Average = mean(values), #calculates the mean of your groups
            standard_deviation= sd(values),#calculates the sd of your groups
            number_of_replicates = n()) -> desc_stats_data #calculates the number of replicates and
generates a new data frame

write.table(desc_stats_data, "desc_stats_group1.txt", sep = "\t", row.names = F)
```

#Checking data distribution

```
hist(dataset2$Root_length)#to visualize data distribution
shapiro.test(dataset2$Root_length)#Compute Shapiro-Wilk test of normality
#The shapiro test is not significant (p>0.05). Therefore, we can assume normal distribution of the data.

#Since we have normal distributed data, we can do an anova for our three variables
#Since we don't have normal distributed data, we can do a Kruskal test for our three variables
kruskal.test(Root_length~Microbe, data = dataset2) -> kruskal.result
p_value <- round(kruskal.result$p.value, 3)
```

#Visualization----
#Plot setting----
#Position

```
posn_d <- position_dodge(0.9)  #values could be between 0 (no separation between columns) and 1
(wide separation between the columns)
posn_j<-position_jitter(0.3)  #adds jittter to your data points in geom_point
```

#Colours

```
myColours<-c('WT'= "#ea4610",'symrk' ="#104fea")
```

#Visualization of data----

```
#Plot----
#getting the groups for our significance to add on the plot
#Get highest quantile for Tukey's 5 number summary and add a bit of space to buffer between upper
quantile and label placement
abs_max<-max(dataset2$Root_length)
maxs<-dataset2 %>%
  group_by(Genotype, Microbe, Temperature, Condition) %>%
  summarise(Fresh_weight_shoot=max(Root_length)+0.1*abs_max)

#using ggplot to plot data
ggplot(dataset2, aes(Temperature, Root_length, facet.by = "Microbe", short.panel.labs = FALSE))+
  geom_boxplot(position=posn_d,aes(col=Genotype))+
  stat_summary(fun.data = mean_sdl,
               fun.args = list(mult=1),
               position = posn_d, aes(col=Genotype))+#define your geom
  geom_jitter(position = posn_d, aes(col = Genotype), size = 1.3) +
  #stat_compare_means(method = "kruskal.test", label.y = 2.3)+
  scale_colour_manual(values=myColours)+#setting the colours
  stat_compare_means(ref.group = "RT", method = "wilcox.test",label="p.signif", label.y = 5.8, hide.ns =
FALSE)+
  facet_wrap(.~Microbe)+#dividing the plot in factes according to variable
  labs(y= "Root_length (cm)", x = "Temperature")+#rename your y and x axis labels
  theme_classic()+
  theme(rect = element_blank(), text = element_text(size=16, face = "bold", color = "black"),
        legend.title = element_blank(), axis.text.x =element_text(color = "black", size = 10), axis.text.y =
element_text(hjust=1),
        panel.spacing.x =(unit(1,"lines")), strip.background = element_rect(color="grey") )+#defining your
theme
  annotate("text", x = 1.2, y = 5.7, hjust = 1, vjust = 1, label = paste0("p =", p_value), color = "black") +
  NULL
```