

**Oasis - A Computer Vision Approach to Self-Watering System for Green  
Air Purifier**

BY

TAN WEI JUN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION  
SYSTEMS**

Faculty of Information and Communication Technology

(Kampar Campus)

JANUARY 2025

# COPYRIGHT STATEMENT

© 2025 Tan Wei Jun. All rights reserved.

This Final Year Project report is submitted in partial fulfilment of the requirements for the degree of **Bachelor of Information Systems (Honours) Business Information Systems** at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

## ACKNOWLEDGEMENTS

I would like to thank and appreciate my supervisor, Dr. Aun Yichiet, for allowing me to work on a computer vision system project. Thank you so much for assisting me with my first step towards a career in computer vision projects, environmental monitoring, and data analysis. Your advice and encouragement have been really helpful during this journey.

Nam Shing, a very special person in my life, for her patience, unconditional support, and love, as well as for remaining by my side through difficult times. I am also very grateful to my parents and family for their love, support, and encouragement throughout the course.

Above all, I give thanks to God, whose grace and guidance have carried me through every step of this project. From the beginning to the end, I have seen doors open, and problems overcome in ways I could never have accomplished on my own. I am humbled and grateful for the strength, wisdom, and perseverance granted to me. *Proverbs 16:3* “Commit to the Lord whatever you do, and He will establish your plans.” This project is a testament to that promise.

# **ABSTRACT**

This project presents an integrated smart system that enhances indoor plant care by combining real-time object detection with automated environmental monitoring and control. At its core, the system leverages the YOLOv10 deep learning model, renowned for its high accuracy in real-time object detection, to identify and classify various indoor plant species from continuous video feeds captured by a webcam. This facilitates real-time visual monitoring and enables a responsive, automated plant care process.

To extend functionality, the project incorporates a self-sustaining air purification system that utilizes natural plants as biofilters. Recognized plants are analysed for their suitability in air purification, with the system adjusting care accordingly. The integration of environmental and moisture sensors allows for dynamic adjustments in watering schedules to maintain optimal plant health.

A key component of the system is the use of Node-RED as a middleware platform to facilitate communication between the plant recognition module and an Arduino-based automated watering system. Detected plants are mapped to corresponding moisture sensors, and each plant receives water based on its specific moisture requirements. This precise matching ensures resource efficiency and tailored care, reducing human intervention while promoting sustainable indoor plant management.

Area of Study: Internet of Things, Computer Vision

Keywords: Data Collection in IoT, Monitoring Application, User-friendly Application, IoT Security, Smart Irrigation, YOLOv10, Real-Time Plant Detection, Node-RED Integration, Arduino-Based Automation, Environmental Sensors.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>COPYRIGHT STATEMENT</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENTS</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF SYMBOLS</b>	<b>x</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xi</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>12</b>
1.1 Problem Statement and Motivation	13
1.2 Research Objectives	14
1.3 Project Scope and Direction	15
1.4 Contributions	16
1.5 Report Organization	17
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>18</b>
2.1 Previous works on Computer Vision	18
2.1.1 Performance of YOLOv5 for Plant Detection in Agricultural Fields	18
2.1.2 Improved YOLOv5-Based Vegetable Disease Detection	19
2.2 Current available computer vision algorithms	19
2.2.1 Computer Vision Algorithm (YOLOv5)	20
2.2.2 Computer Vision Algorithm (YOLOv8)	21
2.3 Comparison Table	23
2.4 Limitation of Reviewed Algorithms	24

<b>CHAPTER 3 SYSTEM METHODOLOGY</b>	<b>26</b>
3.1 System Design Diagram	29
3.1.2 Use Case Diagram and Description	28
<b>CHAPTER 4 SYSTEM DESIGN</b>	<b>30</b>
4.1 System Block Diagram	30
4.2 System Components Specifications	31
4.3 System Components Interaction Operations	31
<b>CHAPTER 5 SYSTEM IMPLEMENTATION</b>	<b>36</b>
5.1 Hardware Setup	36
5.2 Software Setup	40
5.2.1 Python Environment	43
5.2.2 Raspberry Pi OS Configuration	45
5.3 Setting and Configuration	45
5.3.1 Preparation of Datasets	46
5.3.2 Training of the Model	48
5.3.3 Preprocessing and MQTT Integration	49
5.3.4 Sensor Assignment Function	49
5.3.5 Flask Server Setup	50
5.3.6 Setting Up the Pi Camera	50
5.3.7 YOLOv10 Threshold Configuration	50
5.3.8 MQTT publish rate	51
5.3.9 AutoStart Scripts	51
5.3.10 Network Setup	51
5.3.11 Setup of Node-RED on Raspberry Pi	51
5.4 System Operation (with Screenshot)	52
5.4.1 Startup Sequence	52
5.4.2 Detection Demo	53
5.4.3 Watering Action Triggered	54
5.4.4 Visualization Using Node-RED Dashboard 2.0	54

5.5 Implementation Issues and Challenges	55
5.5.1 Hardware Limitations	55
5.5.2 Integration with Arduino Leonardo	56
5.5.3 Network Communication Delay	56
5.5.4 Camera Positioning	56
5.6 Concluding Remark	57
 <b>CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION</b>	 <b>58</b>
6.1 System Testing and Performance Metrics	58
6.2 Testing Setup and Result	62
6.3 Project Challenges	65
6.4 Objectives Evaluation	66
6.5 Concluding Remark	67
 <b>CHAPTER 7 CONCLUSION AND RECOMMENDATION</b>	 <b>69</b>
7.1 Conclusion	69
7.2 Recommendation	70
 <b>REFERENCES</b>	 <b>73</b>
<b>APPENDIX</b>	<b>74</b>
<b>POSTER</b>	<b>150</b>

# LIST OF FIGURES

Figure Number	Title	Page
2.1	YOLOv5 for Plant Detection in Agricultural Fields	1
2.2	Tomato Leaf Classification	10
2.3	General architecture of the YOLOv5	13
2.4	Comparison of YOLO models	14
2.5	Architecture of the YOLOv8 Model	19
3.1	System Diagram	20
3.2	Custom YOLOv8 architecture	28
3.3	Use Case Diagram	29
4.1	System Design	31
4.2	Flask Video Stream Code	34
4.3	Published MQTT message	34
4.4	Detection Results	35
4.5	Dashboard Visualization	35
4.6	System Node Flow	36
5.1	Raspberry Pi 4 Model B	39
5.2	Raspberry Pi Camera Module V2	39
5.3	Raspberry Pi Power Supply	39
5.4	Arduino Leonardo	40
5.5	Soil Moisture Sensors	40
5.6	Water Valve	40
5.7	USB Serial Cable	40
5.8	Node-RED Dashboard	41
5.11	Annotation of datasets	48
5.12	Model Training for YOLOv10	49
5.13	MQTT Integration	50
5.14	Sensor Assignment Function	50
5.15	Threshold Configuration	51
5.16	Raspberry Pi Flask Stream	53



5.17	CV Model Execution	54
5.18	Detection Demo	54
5.19	Terminal Output	55
5.20	Moisture Assignments	55
5.21	Dashboard Visualization	56
5.22	Gauge Visualization	56
5.23	Control Panel	56
5.24	Function Node Configuration	57
5.25	Test Image of Camera Module	58
6.1	Metrics Curve	60
6.2	Training Loss Metrics	61
6.3	Validation Loss Metrics	63
6.4	Performance Metrics	64
6.5	Inference testing of Model	65

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
2.1	YOLOv5 variants	22
2.2	YOLOv8 variants	24
2.3	Comparison Table of Reviewed Computer Vision Models	25
5.1	Specifications of laptop	37
5.2	Components of Computer Vision and Automation System	39
5.3	Libraries Requirements	44
5.4	Library Summary Contributions by Category	45
5.5	Dataset Summary	47
6.1	Performance Metrics table	64
6.2	Testing Procedures and Results	65

## LIST OF SYMBOLS

$(mAP)$	Mean Average Precision
$train/box\_loss$	Train Box Loss
$train/cls\_loss$	Train Class Loss
$train/dfl\_loss$	Train Distribution Focal Loss
$val/box\_loss$	Validation Box Loss
$val/cls\_loss$	Validation Class Loss
$val/dfl\_loss$	Validation Distribution Focal Loss

## LIST OF ABBREVIATIONS

<i>CV</i>	Computer Vision
<i>YOLO</i>	(You Only Looked Once)

# CHAPTER 1

## Introduction

As society continues to evolve and cost of living increases, there is a growing concern on the importance of maintaining indoor air quality. With the increased use of air conditioning and recycled air, the air we breathe ends up being recycled repeatedly. This has contributed to what is commonly referred to as modern "sick" buildings, where the very systems designed to provide comfort but inevitably contributed to the decrease in air quality [1].

These buildings, which contribute to roughly 30% of global energy due to the extensive use of indoor furniture and facilities, can trap harmful pollutants and negatively impact our health[1]. One promising approach is the integration of computer vision technology with plant-based systems for indoor air purification [4]. The advancements in computer vision for plant monitoring offer significant potential for improving the efficiency and effectiveness of plant-based air filtration systems. By leveraging computer vision, it is possible to monitor plant health in real time, ensuring that plants are optimally maintained to perform their natural air-purifying ability.

Ensuring the health of these biofilter plants is crucial for the continuous air purification in indoor spaces. A computer vision system plays a role in maintaining these plants by providing real-time monitoring and analysis of their health. By capturing real-time images of the plants, the system can detect early signs of stress, disease, or lack of growth, reducing the need for manual plant care and preventing the risk of human errors. The system can analyse various visual signs, such as leaf colour, shape, and texture, to assess plant health and make data-driven decisions [5]. This adaptability ensures that the plants receive the necessary care, allowing for timely interventions and adjustments to their environment. By integrating with real-time data, the computer vision system enhances the overall effectiveness of the green air purification system, ensuring that the plants remain in peak condition and continue to contribute to a healthier indoor environment.

## 1.1 Problem Statement and Motivation

Indoor plants are known to act as natural air purifiers, as they clear the pollutants in the air by absorbing the impurities in the environment and releasing oxygen. [1] However, the ability of these air purifying plants is influenced on proper watering and care, and traditional methods for the caring of these plants have its few disadvantages.

Traditional methods of plant care, such as manually scheduled monitoring of the plants. This inconsistent monitoring can be a challenge and may cause uneven watering distribution of water and care among plants, especially when they are different plants involved. Overwatering can cause the root to rot, while underwatering can cause plants to wither and die. With a consistent, scheduled monitoring solution, the efficiency of automated systems is improved, which can prevent wastage of water and unhealthy plants. and make it difficult for the plant to effectively clean the air. These issues affect both the plant's aesthetic appeal and its ability to purify the air. Therefore, there is a rising curiosity in incorporating modern technologies like computer vision (CV) to improve plant care systems [4].

This project explores the integration of a computer vision-based approach with a self-watering system specifically designed for indoor plants that serve as green air purifiers. The system proposed aims to improve to consistency and efficiency of watering schedules by utilizing real-time image processing to monitor plant health indicators like leaf color, growth patterns, and overall vitality [5]. The use of computer vision combined with IoT sensors allows for a more flexible and accurate method for taking care of plants, ensuring each plant receives the right amount of water tailored to its specific needs and environmental conditions [6].

The purpose of integrating computer vision system with a dashboard monitoring system is to provide real time data of plant health visually. As urban living spaces continue to shrink and the demand for low-maintenance, high-efficiency plant care systems increase, integrating computer vision into these systems offers a promising solution for innovation [7]. By advancing self-watering systems with computer vision capabilities, this project aims to

improve the effectiveness of indoor plants as air purifiers, ultimately leading to healthier indoor environments [8].

## **1.2 Research Objectives**

This project proposes the deployment of advanced computer vision technology to enhance the self-watering system for indoor plants. The goal is to ensure precise and adaptive watering based on real-time plant and environmental data. The key objectives are:

1. To design and train a Plant Recognition for Watering Automation. There is a camera system capable of identifying different plant types and recommending customized watering levels based on real-time soil moisture data. Integrate an automated self-watering mechanism to maintain optimal plant health.
2. To develop a functional prototype of a smart air purifier housed on a three-level trolley, integrating plant compartments, a self-watering mechanism, and an IoT sensor system for real-time environmental data collection.
3. To Incorporate IoT sensors to monitor soil moisture levels in the plant compartments and measure the Air Quality Index (AQI) in the surrounding environment. Stream this data to a centralized dashboard for real-time monitoring.

During the process of image detection, a camera system will be developed to identify different air-purifying plant species with trained datasets. The camera system will continuously capture real-time video feed and pictures. By integrating this system with an automated self-watering mechanism, the prototype will ensure that each plant receives the appropriate amount of water tailored to its specific needs. Traditional manual watering methods often fail to deliver consistent and optimal care. Hence, the first objective aims to utilize computer vision technology to reduce the risk of over or under-watering and to streamline the process of plant care and monitoring. The visual results of the computer vision will continuously monitor plants and capture crucial data, improving plant health and reducing the need for manual intervention. The collected data will be comprehensively visualized and monitored through Grafana, allowing for real-time adjustments and improved management of the plant compartments.

Continuing the process, the end result will be that the plants and their moisture levels are constantly monitored using a computer vision (CV) model and IoT sensors to ensure optimal air quality from the purifying plants. Each plant should consistently maintain moisture within a predefined threshold, suitable for its species and environmental conditions. The CV system, in conjunction with real-time sensor data, will assess the health of the plants based on visual indicators such as leaf colour, texture, and overall appearance. For instance, signs of water stress, such as wilting or discoloration, will be detected early, allowing for immediate corrective actions. The integration of this monitoring system ensures that the self-watering mechanism adjusts accordingly to maintain both plant health and optimal air quality.

### **1.3 Project Scope and Direction**

In this project, the research and development will centre around the implementation and exploration of various computer vision algorithms and models. The primary objective is to develop a plant recognition model that can accurately identify and monitor plant health. The project will investigate different computer vision approaches, evaluating their effectiveness in recognizing plant species and assessing plant health.

The focus will be on optimizing the model's accuracy and reliability, with an emphasis on real-time processing to enable timely monitoring and potential integration with automated systems, such as a self-watering mechanism. This exploration will not only aim to achieve high precision in plant recognition but also explore how these models can be enhanced or customized for specific applications, such as indoor plant care and air quality improvement.

At the conclusion of the project, a prototype system will be delivered, comprising both software and hardware components. The hardware will include a camera system for plant identification and monitoring, while the software will include the plant recognition model, along with tools for analysing plant health. This prototype will serve as a foundational system designed to recognize plants, offering potential applications in smart gardening and automated plant care systems.



## **1.4 Contributions**

The aim of this project is to showcase the practicality and effectiveness of integrating computer vision and IoT technologies to enhance self-watering systems for indoor air-purifying plants. Firstly, the project successfully demonstrates the computer vision system for plant recognition. This sets the foundation for future progress in automating plant care with computer vision and machine learning. By the end of the project, a solution will be provided, including hardware and software elements. The hardware will have cameras to take high-quality images of plants, with the software having a computer vision algorithm for identifying plant types. This project aims to create a prototype system showcasing the capabilities of computer vision technology in plant care and monitoring, leading the path for further developments in the field.

## **1.5 Report Organization**

This report focuses on developing a computer vision algorithm for a plant recognition system to enhance a self-watering mechanism for indoor air purifying plants. Chapter 1 introduces the problem statement and highlights the research objectives, project scope, and contributions. It sets the stage for the project by explaining the significance of integrating computer vision with automated plant care systems and the anticipated impact of this technology. Chapter 2 reviews existing computer vision algorithms pertinent to plant recognition. This chapter explores various approaches and techniques used in the field. Chapter 3 presents the proposed methodology for implementing the computer vision system. This chapter focuses on the design and structure of the system, including the hardware configuration, software requirements, and the procedure for integrating computer vision algorithms with collected datasets. Chapter 4 focuses on the preliminary work done during the project, including initial experiments, setup procedures, and challenges encountered. It discusses about the challenges and issues during the development process and the strategies employed to address these issues. Chapter 5 concludes the report based on the project's progress and evaluating its overall success. It reflects on the achievements, assesses the effectiveness of the system, and offers recommendations for future enhancements. It also discusses potential directions for further research and development to advance the technology and its applications.

# CHAPTER 2

## Literature Review

### 2.1 Previous works on Computer Vision

#### 2.1.1 Performance of YOLOv5 for Plant Detection in Agricultural Fields

Research in object detection using computer vision by [9], which evaluated effectiveness of the YOLOv5 model in identifying the cotton plants within corn fields at various growth stages. The research focused on assessing the accuracy of YOLOv5, under different environmental conditions and plant growth stages, which are crucial for agricultural sectors. The study showed that YOLOv5 could effectively identify cotton plants in corn fields across three distinct growth stages, even under varying lighting and occlusion conditions. The algorithm showed high detection accuracy and speed, which are essential for real-time detections where timely decision-making is needed. The researchers highlighted that YOLOv5's capability to quickly process and analyse images in complex backgrounds made it particularly suitable for agricultural scenarios where multiple plant types co-exist.

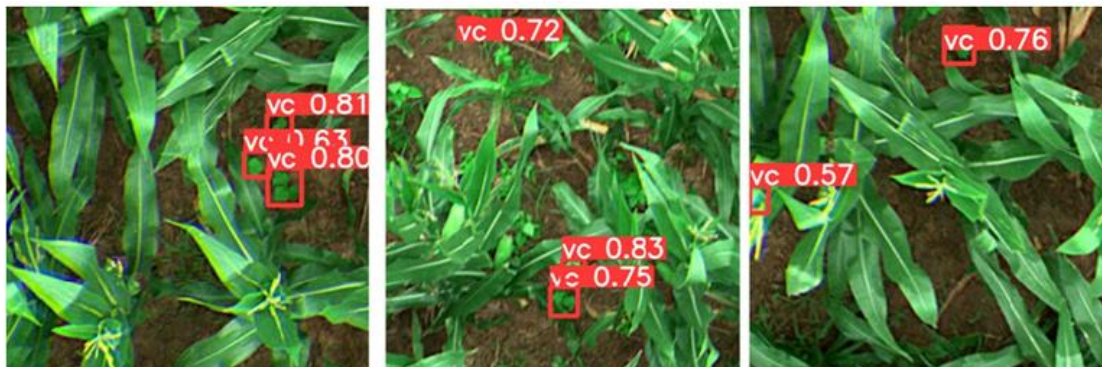


Figure 2.1 YOLOv5 for Plant Detection in Agricultural Fields

#### 2.1.2 Improved YOLOv5-Based Vegetable Disease Detection

[10] The improved method of this research focused on improving YOLOv5's accuracy in real-time detection and classification of different vegetable diseases. They trained the standard YOLOv5 model to better generalize to the distinct features of vegetable leaves, like colours, shapes, and textures, that could affect the accuracy of detection. The improved YOLOv5 model included several innovative adjustments, such as fine-tuning the network layers and integrating a new attention mechanism to enhance feature extraction in complex backgrounds.. The study

demonstrated that these modifications significantly boosted detection accuracy and reduced false positives, particularly in cases with overlapping leaves or where diseases manifested subtly. Additionally, the improved model achieved faster processing speeds, which was critical for real-time agricultural applications where quick and accurate responses are necessary.

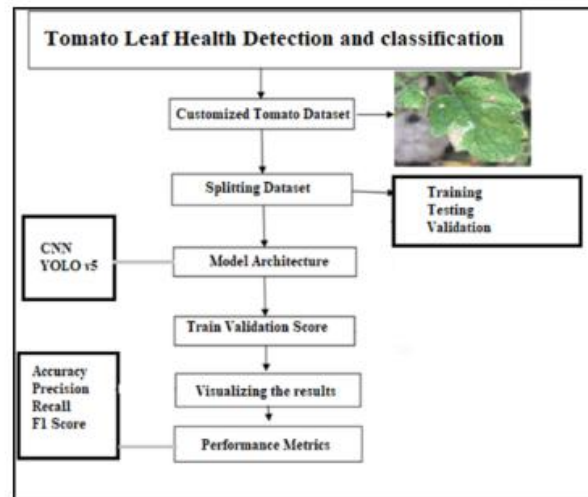


Figure 2.2 Tomato Leaf Classification

## 2.2 Current available computer vision algorithms

### 2.2.1 Computer Vision Algorithm (YOLOv5)

In this section, we explore the YOLOv5 computer vision algorithm for the plant recognition and watering automation system. YOLOv5 is widely recognized for its real-time object detection capabilities, offering a compelling balance between speed and accuracy [11]. This makes it particularly suitable for applications where fast response times are critical, which requires immediate decision-making to adjust watering schedules based on plant identification. [12]

The YOLOv5 model uses convolutional neural networks (CNNs) to rapidly process visual data captured by the camera system, allowing fast and accurate identification of plant species. Its lightweight architecture and optimized performance allow for fast inference, ensuring that the system can operate efficiently in real-time without significant latency. This ability to maintain

high detection speed while providing reliable accuracy makes YOLOv5 a potential choice for our project. Figure 2.3 illustrates the general architecture of the YOLOv5 model. [12], [13]

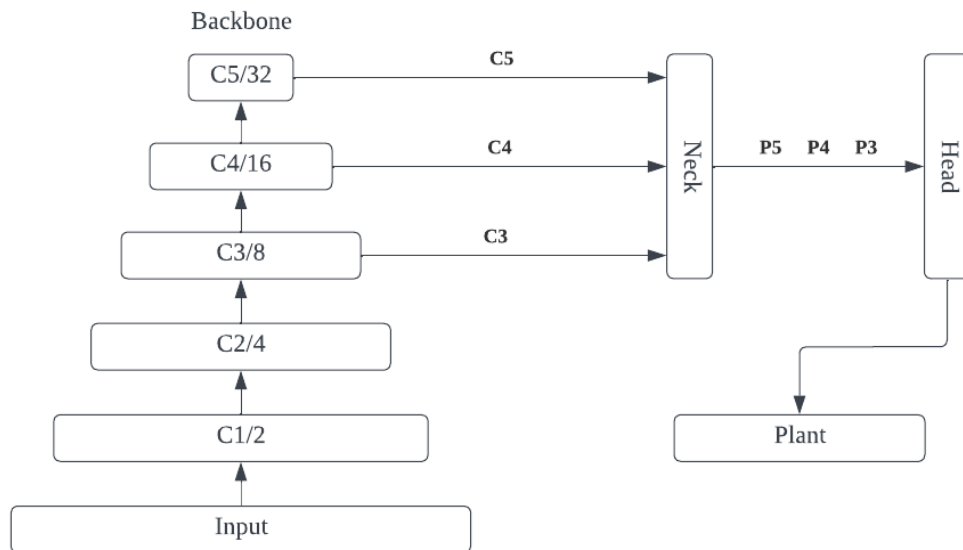


Figure 2.3 General architecture of the YOLOv5

YOLOv5's flexible architecture, with variants (**s**, **m**, **l**, **x**) tailored to different requirements for detection speed and accuracy, makes it ideal for applications that require balancing model complexity with inference speed on limited hardware. the YOLOv5s version was selected due to its lower computational demands and ability to run on devices with limited processing power while achieving high accuracy for plant species recognition [15]. This makes YOLOv5s potentially well-suited for our automated watering system, which requires real-time plant detection to adapt watering schedules dynamically. Table 2.1 below shows the different model of YOLOv5 variants. [11]

Table 2.1 YOLOv5 variants

Model	size (pixels)	mAP <sup>val</sup> 50-95	mAP <sup>val</sup> 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @ 640 (B)
-------	------------------	-----------------------------	--------------------------	-------------------------	--------------------------	------------------------------	---------------	--------------------

YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4

### 2.2.2 Computer Vision Algorithm (YOLOv8)

This section explores the architecture of the YOLOv8 computer vision algorithm, focusing on why it was selected for our plant recognition and watering automation system over its predecessors. YOLOv8, introduced by Glenn Jocher in 2023 [16], is the most recent development in the YOLO (You Only Look Once) lineup, building upon the achievements of YOLOv5 and its previous models, it has improved significantly in terms of speed, accuracy, and versatility for live object detection [17]. Figure 2.4 shows comparison of different model versions of YOLO.

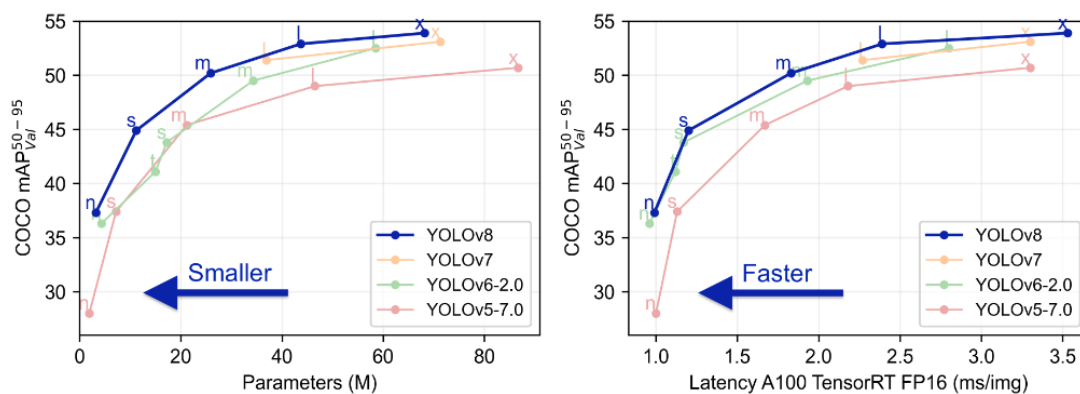


Figure 2.4 Comparison of YOLO models

YOLOv8 features various architectural enhancements that distinguish it from YOLOv5. The model includes an improved backbone network that helps achieve improved feature extraction and more accurate object detection [12]. This progress decreases the number of calculations needed and training time while still achieving excellent results, allowing for quicker predictions. Additionally, YOLOv8 uses innovative anchor-free detection heads to improve localization accuracy and decrease false positives by getting rid of predefined anchor boxes. This leads to enhanced identification of objects in different sizes and forms, making YOLOv8 highly efficient in a variety of challenging settings. The flexible design of YOLOv8 also makes it easy to adapt to different scenarios and deployment environments [19]. Figure 2.5 below is the architecture of YOLOv8.

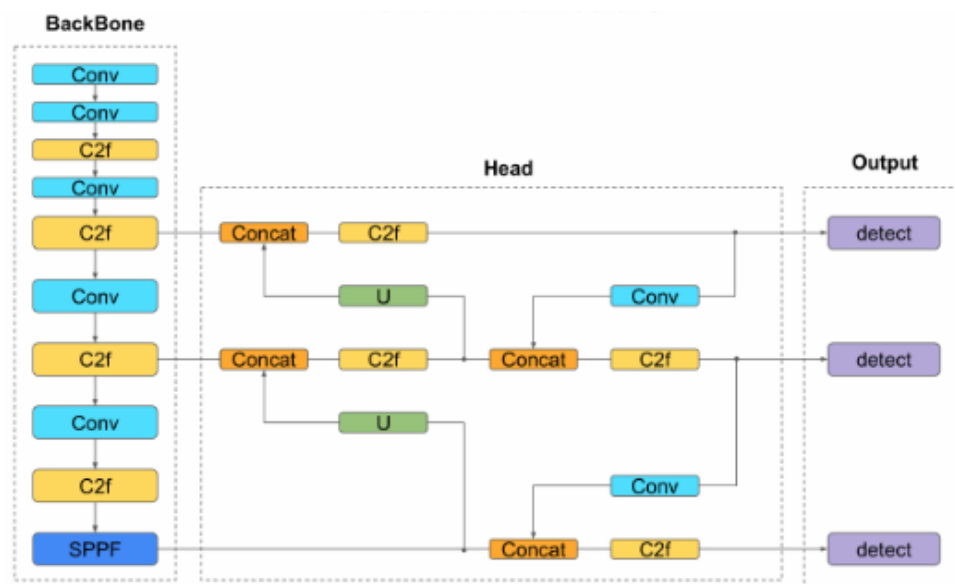


Figure 2.5 Architecture of the YOLOv8 Model

YOLOv8 too comes with a variety of variants (**n, s, m, l, x**) tailored to different requirements for detection speed and accuracy, makes it ideal for applications that require balancing model complexity with inference speed on various hardware setups. The YOLOv8n version was selected due to its lower computational demands and ability to run efficiently on devices with limited processing power while still achieving high accuracy for plant species recognition. This makes YOLOv8n best suited for our automated watering system, which relies on real-time plant detection to dynamically adjust watering schedules. Table 2.2 below shows the different model variants of YOLOv8. [16]

<b>Model</b>	<b>size</b> (pixels)	<b>mAP<sup>val</sup></b> 50-95	<b>Speed</b> CPU ONNX (ms)	<b>Speed</b> A100 TensorRT (ms)	<b>params</b> (M)	<b>FLOPs</b> (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 2.2 YOLOv8 variants

## 2.3 Comparison Table

Based on the literature review on two different types of computer vision models, a comparison table is structured to outline the main points for each model considered in the plant recognition and watering automation system.

Table 2.3: Comparison Table of Reviewed Computer Vision Models

<b>Feature</b>	<b>YOLOv5</b>	<b>YOLOv8</b>
<b>Release Date</b>	June 2020	January 2023
<b>Architecture</b>	CSPDarknet53 with PANet	Improved backbone with enhanced PANet
<b>Anchor Boxes</b>	Uses predefined anchor boxes	Anchor-free detection
<b>Speed and Efficiency</b>	Fast, but less optimized for real-time	Faster with reduced computations
<b>Model Size</b>	Moderate to large	Smaller and more lightweight
<b>Accuracy</b>	High, but struggles with small objects	Improved accuracy, especially for small objects
<b>False Positives</b>	Higher due to anchor boxes	Lower due to anchor-free approach

<b>Real-time Applications</b>	Suitable, but less efficient than YOLOv8	Highly optimized for real-time use
-------------------------------	--	------------------------------------

## 2.4 Limitation of Reviewed Algorithms

YOLOv5 faces several limitations despite its advancements in object detection. One of its primary challenges is limited robustness in diverse environments. While YOLOv5 performs well under normal conditions, it struggles with generalizing on unseen data, or complex backgrounds. This lack of flexibility impacts the model's effectiveness in real-world applications where environmental conditions are less controlled. Additionally, YOLOv5's computational resource requirements, though optimized, it is still a large model as compared YOLOv8. This can be an issue for deployment on resource constraint devices. Additionally, YOLOv5 may struggle with detecting smaller and more specific object, which can be crucial in applications requiring precise object detection, like detailed medical images or complex backgrounds. The exchange between accuracy and speed often means that YOLOv5 might not capture fine details effectively. Finally, YOLOv5's performance can fluctuate depending on the specific object detection task. While it performs well in general object detection, specialized tasks may require extra fine-tuning and additional training, potentially making it less versatile and more resource intensive.

YOLOv8 introduces several advancements over its predecessors but also brings new challenges. Its increased complexity and advanced architecture lead to higher computational and memory overhead during both training and inference but it is still smaller than and faster than YOLOv5 in terms of training time and datasets [11]. Additionally, YOLOv8's enhanced capabilities increase its susceptibility to overfitting, especially when trained on limited datasets. Overfitting can decrease or reduce the model's capability to generalize to new data which was what happened in our training and deployment of YOLOv5 model, which is crucial for real-world scenarios. In our experience, using the same datasets, YOLOv8 performs better than YOLOv5 with lesser training time and generalizes better on unseen data with higher confidence level. Lastly, integrating YOLOv8 into existing systems can be complex. The model's advanced features may require significant adjustments and optimizations during deployment, posing challenges for users who need a more straightforward and easily integrable solution.



# Chapter 3

## System Methodology

### 3.1 System Design Diagram

The objective of this project is to create a specialized computer vision system for monitoring indoor plants with advanced technology. The system will use advanced computer vision techniques to accurately identify and assess the health of indoor plants. By integrating computer vision into indoor plant care, the system aims to automate the monitoring of plant care, resulting in better plant health and improved care. Figure 3.1 below is the proposed system diagram our proposed method.

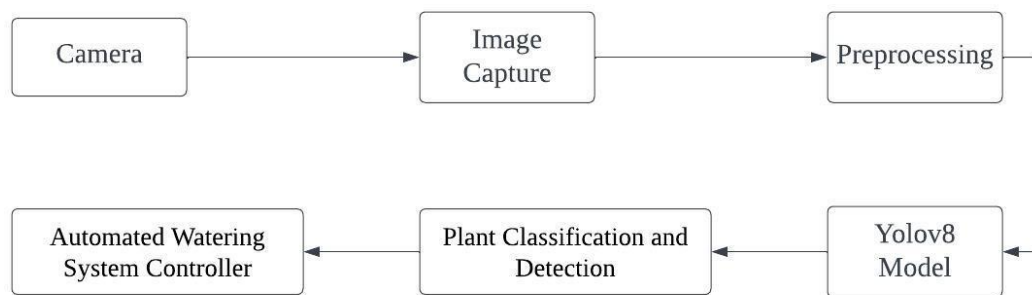


Figure 3.1 System diagram

To achieve this, a wide collection of images of plants will be gathered. This dataset will contain various air purifying plant species, covering different scenarios, angles, and environmental settings to ensure the model's accuracy. The gathering of data will require capturing detailed images of the plants with varying lighting and angles to fully depict their visual attributes. The gathered datasets will be used to train a computer vision model, using advanced models like YOLOv5 or YOLOv8, which are known for their effectiveness in identifying objects.

Preparing the images to improve quality and consistency and labelling the data to provide plant species information will be necessary for training the model. The training will require making changes and fine tuning the model by adding more preprocessing steps to improve the accuracy of the model and the ability to generalize on new data. Once training is complete, the model

will be tested to evaluate its performance in real-world settings, ensuring its ability to accurately detect the plant species.

After successful testing, the model will be deployed into an environment where it will be integrated with a Python-based script. This script will facilitate real-time operation by processing live camera feeds to monitor plant conditions continuously. The Python script will interface with the computer vision model to analyse images, detect plant species. The deployment phase will include setting up the model on a suitable computing platform, ensuring compatibility with the existing hardware and software infrastructure. The Python script will manage the model's execution, handling tasks such as image capture, data processing, and result visualization. The custom model that will be deployed will be based on the custom datasets and weights trained, instead of the pre-trained weights.

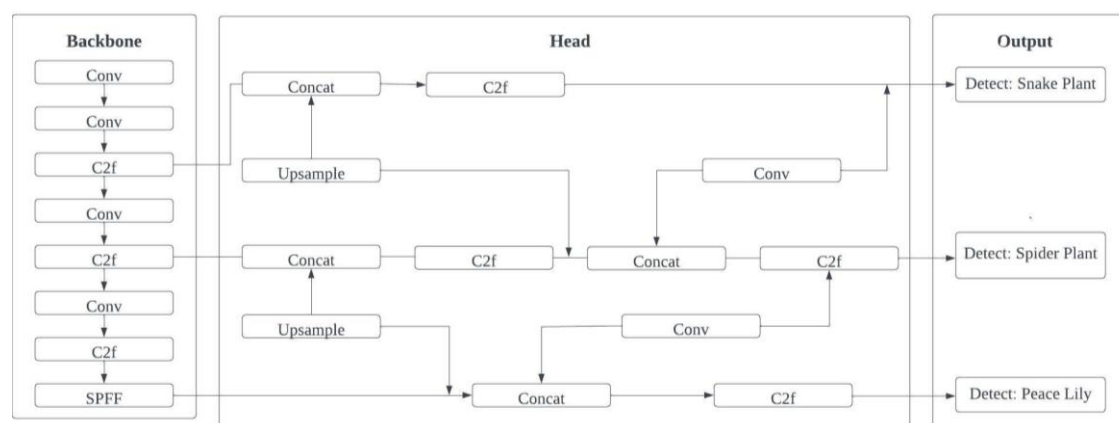


Figure 3.2 Custom YOLOv8 architecture

### 3.1.2 Use Case Diagram and Description

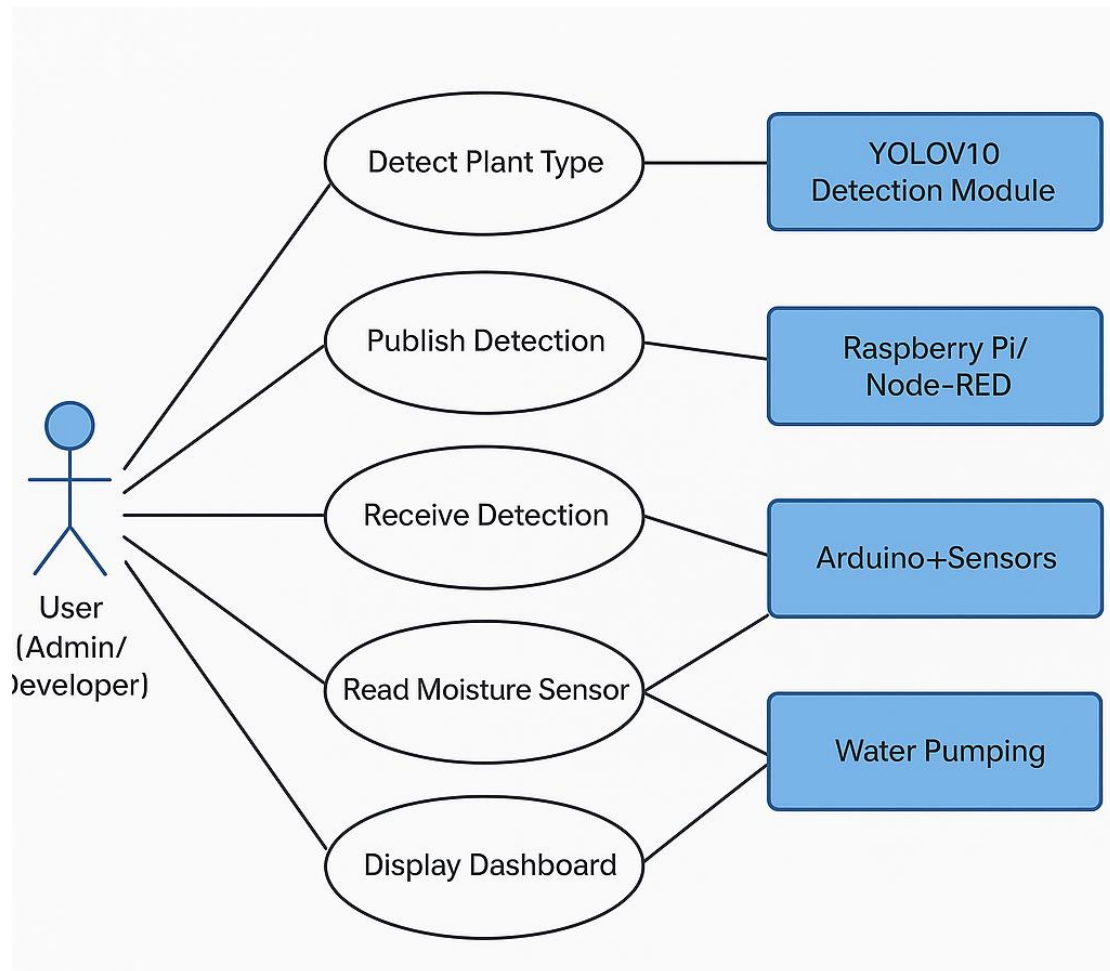


Figure 3.3 Use Case Diagram

The use case diagram illustrates the interaction between the system's primary actors: *User*, *Raspberry Pi*, *Laptop*, *Arduino* and the main functions of the automated plant monitoring and watering system.

#### Description:

- **User:** The user can initiate system monitoring, configure plant type settings, and observe system status via a user interface (Node-RED Dashboard or terminal output).
- **Raspberry Pi:** Acts as the middleware, receiving plant detection results from the laptop via MQTT, processing sensor data, and forwarding commands to the Arduino. It also runs Node-RED for managing logic flows.

- **Laptop:** Runs the YOLOv10-based detection script, captures the video feed streamed from the Raspberry Pi camera, performs inference, and publishes the results to the MQTT broker.
- **Arduino:** Receives watering commands and interfaces with moisture sensors and the water pump. Based on soil conditions and plant type, it executes the watering process accordingly.

Each use case represents a functional requirement of the system. For instance:

- *Capture and Analyse Video Feed* involves real-time processing using YOLOv10.
- *Send Detection Result to MQTT Broker* handles communication to the Raspberry Pi.
- *Trigger Watering Based on Conditions* uses decision-making logic on Node-RED to determine when and how much to water the plant.

# Chapter 4

## System Design

### 4.1 System Block Diagram

This project implemented a hybrid computer vision–IoT-based system to automate the monitoring and care of indoor plants. The core component was a custom-trained YOLOv10 object detection model, executed on a local laptop, which classified indoor air-purifying plants in real time using a video stream captured from a Raspberry Pi Camera. Upon detection, the system published the identified plant types via MQTT protocol to a Node-RED server hosted on the Raspberry Pi, which acted as the communication bridge between the computer vision pipeline and an Arduino-based automated watering system controlled by four moisture sensors and water pumps. This integrated setup allowed dynamic, species-specific irrigation control based on both real-time visual classification and environmental moisture feedback. **Figure 4.1** below is the final system architecture diagram.

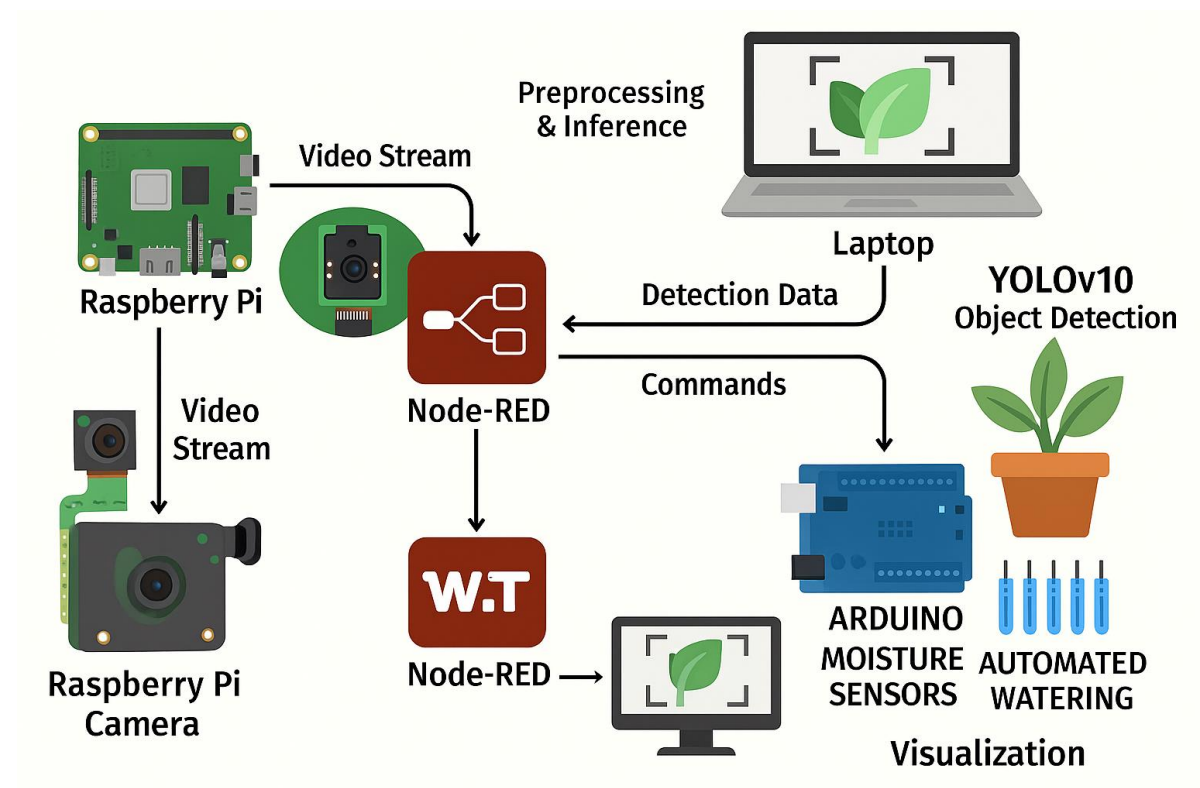


Figure 4.1 System Design

## 4.2 System Components Specifications

Component	Specification
YOLOv10	Real-time object detection model (PyTorch-based)
Camera	Raspberry Pi Camera v2, 8MP, 1080p video
Laptop	Runs Python script for detection; uses OpenCV, Flask
Raspberry Pi 4 Model B	Hosts Node-RED and MQTT broker
Node-RED	Visual programming tool to control logic flow
MQTT	Lightweight protocol for IoT communication
Arduino Leonardo	Microcontroller for sensor/pump control
Moisture Sensors	Analog sensors to read soil moisture
Water Pumps	3V–6V DC pumps, controlled via relays

## 4.3 System Components Interaction Operations

Operation	Involved Components	Description
Capture & Detect	Camera, Laptop, YOLOv10	Camera sends video to YOLOv10 for classification
Publish Detection	Laptop, MQTT Broker	YOLOv10 sends result (e.g., { "plant": "Aloe Vera" }) via MQTT
Receive & Process	Node-RED, Raspberry Pi	Node-RED receives MQTT data and maps plant to pump
Check Moisture	Arduino, Sensors	Arduino checks moisture level from mapped sensor
Watering Action	Arduino, Relay, Pump	If needed, pump is triggered for a specific duration
Dashboard Update	Node-RED UI	System displays sensor data and action logs in real-time

### System Components and Flow:

#### 1. Raspberry Pi with Pi Camera

- Captured live video of indoor plants.
- Streamed video to a connected laptop via Wi-Fi using a lightweight streaming protocol (e.g., MJPEG or GStreamer).

#### 2. Laptop (Model Inference Host)

- Handled video preprocessing and real-time inference using the **YOLOv10 model**.
  - Published detection results (plant type and confidence score) via MQTT to the Raspberry Pi.
3. **Raspberry Pi with Node-RED**
- Hosted a **Node-RED server** acting as MQTT subscriber and control intermediary.
  - Detected plant types are **published via MQTT protocol** to a **Node-RED instance running on the Raspberry Pi**, which acts as an **intermediary controller** between the AI model and the actuator system.
  - Routed the information to the Arduino based on predefined species-to-sensor mappings.
4. **Arduino Leonardo (ATMega32u4) with Moisture Sensors and Relay-Controlled Watering Module**
- Received commands from Node-RED through serial communication.
  - Measured moisture levels for four individual plants.
  - Activated water pumps only when the detected plant's assigned soil sensor reported values below its moisture threshold.

Initially, object detection models such as YOLOv5 and YOLOv8 were explored due to their strong performance in similar tasks. However, to improve both detection speed and accuracy, the project eventually shifted to YOLOv10, the latest version in the YOLO series. YOLOv10 offers improved efficiency, particularly in handling high frame rates and small object features, which made it an ideal choice for real-time video processing.

Once the computer vision pipeline was finalized, it was integrated into an automated plant watering system using **Node-RED** as the central orchestration platform. Hosted on a **Raspberry Pi 4 Model B**, Node-RED manages both data flow and command execution between the detection system and hardware actuators, including **Arduino Leonardo**, moisture sensors, and water pumps.

```

stream.py x
1 from flask import Flask, Response
2 from picamera2 import Picamera2
3 import cv2
4 import io
5 app = Flask(__name__)
6 camera = Picamera2()
7
8 camera.configure(camera.create_preview_configuration(main={"size": (720, 720)}))
9 camera.start()
10
11 def generate_frames():
12     while True:
13         frame = camera.capture_array()
14         ret, buffer = cv2.imencode('.jpg', frame)
15         frame = buffer.tobytes()
16         yield (b'--frame\r\n'
17              b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
18
19 @app.route('/video_feed')
20 def video_feed():
21     return generate_frames()

```

Shell x  
Python 3.11.2 (/usr/bin/python3)

Figure 4.2 Flask Video Stream Code

Following successful testing, the model was integrated into a Python-based script and **deployed on a laptop**. This script handles the core computer vision tasks such as **capturing frames from the live video feed**, performing inference with the YOLOv10 model, and determining the detected plant species. The **live video stream** is **captured** using a **Raspberry Pi Camera Module**, which **streams the feed** to the **laptop** via a local network connection. The laptop then processes this input and identifies the plant species in real time.

The detection model runs on a **laptop**, which acts as the **model inference host**. A live video feed is captured using the **Raspberry Pi Camera Module V2**, which streams video over Wi-Fi using a lightweight protocol such as MJPEG. The Python script on the laptop handles frame capturing, preprocessing, and real-time inference using the **YOLOv10** model. Once a plant is detected, the identified species (e.g., "Moisture Sensor 1": "Snake Plant") published as an **MQTT message** to the Raspberry Pi.

```

Published to MQTT: {"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum",
Published to MQTT: {"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum",
Published to MQTT: {"Moisture Sensor 1": "None", "Moisture Sensor 2": "None", "Moisture Sensor 3": "None", "Moisture Sensor 4": "None",
Published to MQTT: {"Moisture Sensor 1": "None", "Moisture Sensor 2": "None", "Moisture Sensor 3": "None", "Moisture Sensor 4": "None",
Published to MQTT: {"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum",
Published to MQTT: {"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum",

```

Figure 4.3 Published MQTT message

On the Raspberry Pi, **Node-RED** acts as an **MQTT subscriber**, receiving detection results and initiating a logic sequence. Each plant species is mapped to a specific soil moisture sensor.



Upon receiving a detection result, Node-RED forwards a command to the **Arduino Leonardo** via USB serial communication. The Arduino reads the corresponding sensor's moisture level, compares it against a predefined threshold, and activates the **water pump** if watering is required.

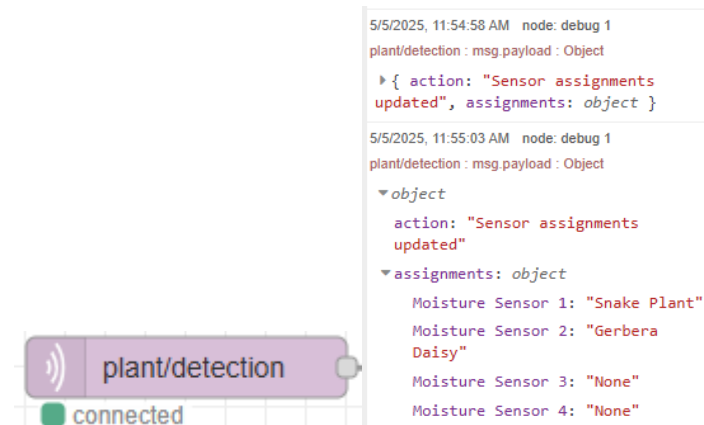


Figure 4.4 Detection Results

An additional enhancement to the system is the integration of a real-time dashboard, built using Node-RED's Dashboard module.

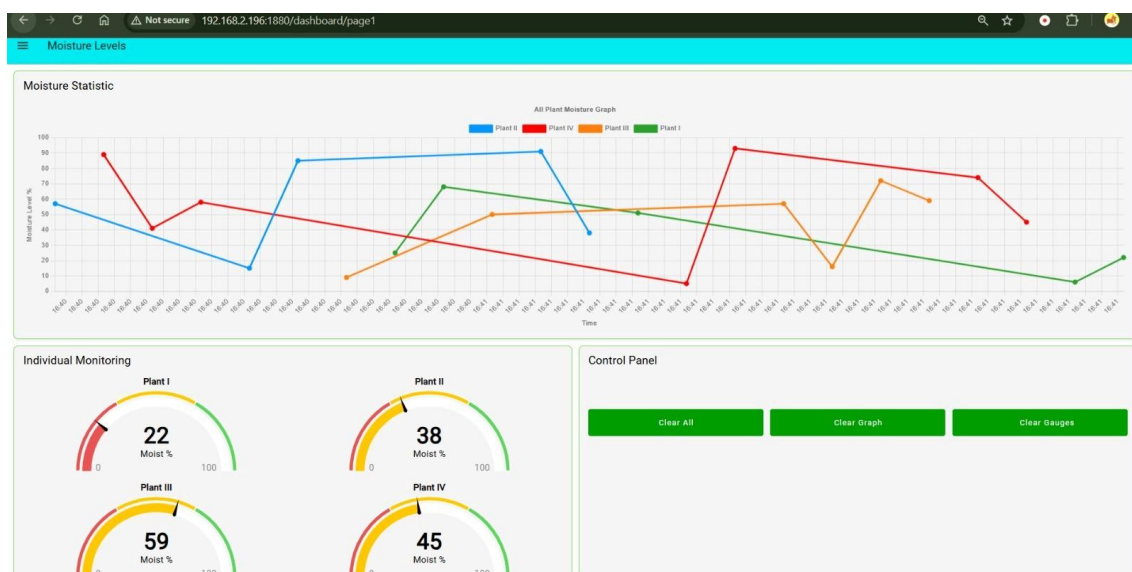


Figure 4.5 Dashboard Visualization

This dashboard serves multiple functions:

- Displays live updates of soil moisture levels from each sensor.
- Shows logs of detected moisture and watering actions taken.
- Visualizes MQTT messages and system health metrics (e.g., online/offline devices, data timestamps).

This interface enables users to monitor the system remotely through a web browser and provides clear transparency into system decisions and environmental conditions.

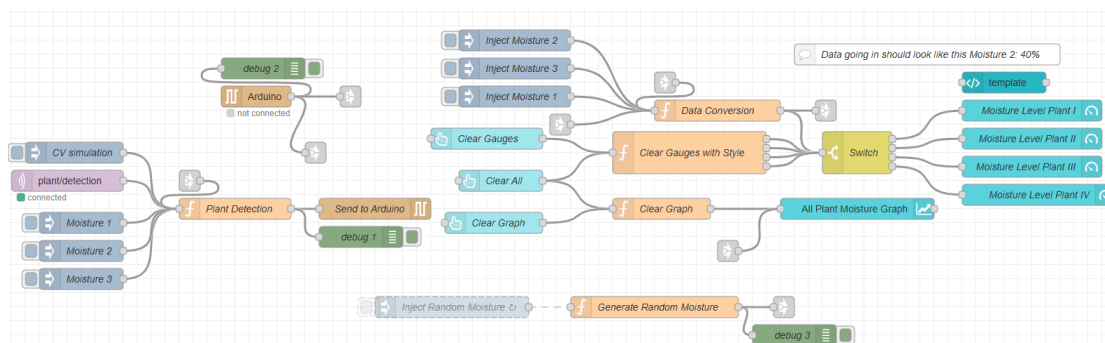
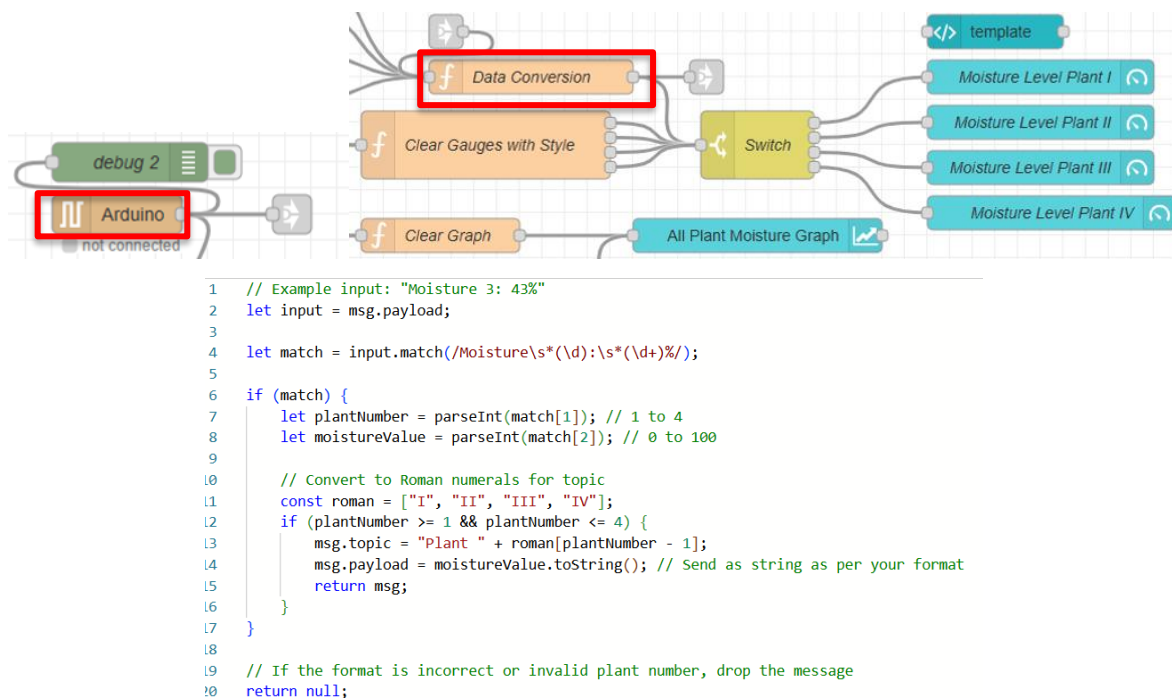


Figure 4.6 System Node Flow



To enable **visualization on UI Chart and Gauge nodes** in Node-RED, a **Function Node** is used to **process raw moisture data received from Arduino** (e.g., "Moisture 3: 43%"). This node parses the string, extracts the plant number and moisture value, and assigns the correct **msg.topic** for each plant. This is crucial because both the **Chart** and **Gauge nodes** rely on consistent topics and numeric payloads for correct plotting.

# Chapter 5

## System Implementation

### 5.1 Hardware Setup

The hardware that will be involved in this project are laptop for development and testing of programs and webcam. The laptop will also be the main device that runs the CV algorithm.

Table 5.1 Specifications of laptop

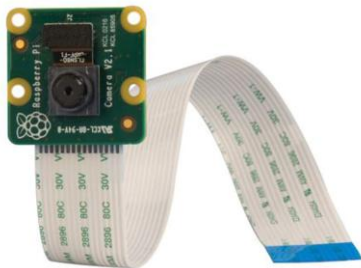
Description	Specifications
Model	HP Pavilion Gaming Laptop
Processor	Intel Core i5-9300H / i7-9750H or equivalent
GPU	NVIDIA GeForce GTX 1650 / GTX 1050 or equivalent
RAM	8GB / 16GB DDR4
Storage	256GB / 512GB SSD
Operating System	Windows 10 / Windows 11
Display	15.6" Full HD (1920 x 1080) Anti-Glare IPS Display
Connectivity	Wi-Fi 5 (802.11ac), Bluetooth 4.2
Ports	1 x USB Type-C, 3 x USB Type-A, 1 x HDMI, 1 x Ethernet, 1 x Audio Combo Jack

## Raspberry Pi 4 Model B



- **Function:** Acts as the edge computing device responsible for streaming live video from the camera and handling automation commands via Node-RED and MQTT.
- **Processor:** Quad-core ARM Cortex-A72 @ 1.5GHz
- **RAM:** 4GB LPDDR4 (expandable up to 8GB)
- **Connectivity:** Dual-band 2.4GHz and 5GHz Wi-Fi, Bluetooth 5.0, Gigabit Ethernet
- **Ports:** 2 × USB 3.0, 2 × USB 2.0, 2 × micro-HDMI, GPIO Header
- **Operating System:** Raspberry Pi OS (Debian 12 Bookworm, 64-bit)


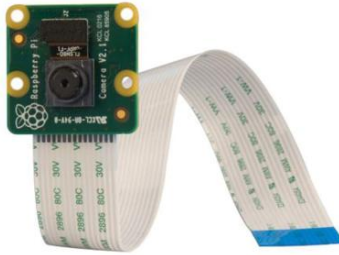

## Raspberry Pi Camera Module V2



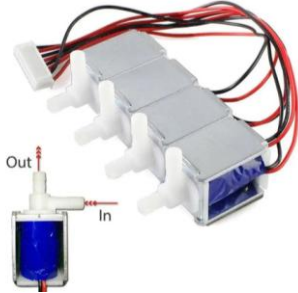



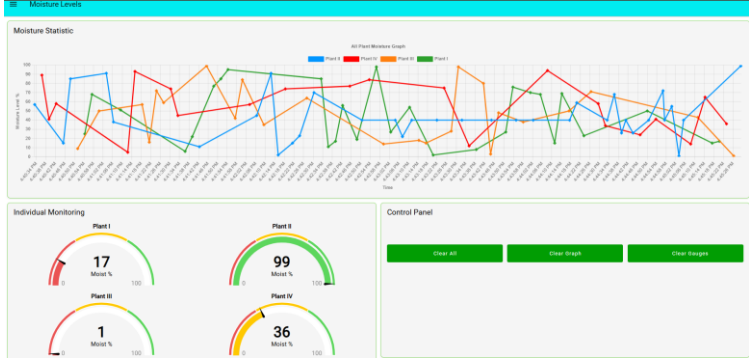
- **Function:** Captures continuous image feed of the plants for real-time analysis.
- **Resolution:** 8 Megapixels
- **Frame Rate:** Up to 1080p at 30fps
- **Interface:** CSI (Camera Serial Interface)

## Components of Computer Vision and Automation System

Table 5.2 Components of Computer Vision and Automation System

Components	Descriptions
<b>Raspberry Pi 4 Model B</b>	<p>Figure 5.1 Raspberry Pi 4 Model B</p>  <p>Acts as the core edge computing device. It runs a local Flask server for video feed streaming and hosts the Node-RED server for controlling and visualizing the self-watering system. It also handles MQTT communications.</p>
<b>Raspberry Pi Camera Module V2</b>	<p>Figure 5.2 Raspberry Pi Camera Module V2</p>  <p>Captures real-time images of plants and streams them for computer vision analysis. This module uses an 8MP Sony IMX219 sensor, ideal for high-definition detection tasks.</p>
<b>Raspberry Pi 15W USB-C Power Supply</b>	<p>Figure 5.3 Raspberry Pi Power Supply</p>  <p>Provides stable 5V/3A power to the Raspberry Pi 4, ensuring consistent performance for real-time image processing, sensor data handling, and overall system stability.</p>
<b>Laptop (Host Computer)</b>	<p>My laptop runs the YOLOv10 object detection model to classify plant species and determine watering needs. The computer receives live feed via Flask and returns water requirement results to the Raspberry Pi.</p>

<b>Arduino Leonardo (ATmega32U4)</b>	<p>Figure 5.4 Arduino Leonardo</p>  <p>Connected via USB serial to the Raspberry Pi. It reads moisture data from four soil moisture sensors and triggers the watering system. It acts as the actuator control unit.</p>
<b>Soil Moisture Sensors (x4)</b>	<p>Figure 5.5 Soil Moisture Sensors</p>  <p>Placed in different plant pots, these analog sensors read real-time soil moisture levels and transmit data to the Arduino for decision-making</p>
<b>4-Way Water Valve with Pump</b>	<p>Figure 5.6 Water Valve</p>  <p>Controlled by the Arduino. Dispenses water only to the plants that require it, based on sensor input and visual classification output.</p>
<b>USB Serial Cable</b>	<p>Figure 5.7 USB Serial Cable</p>

	 <p>Connects the <b>Arduino Leonardo</b> to the <b>Raspberry Pi</b> for continuous data transfer of sensor readings and execution of watering commands.</p>
<b>Node-RED Dashboard</b>	<p>Figure 5.8 Node-RED Dashboard</p> 

## 5.2 Software Setup

Software such as Thonny, Visual Studio Code (VS Code), Roboflow, YOLOv10, Google Collab, Node-RED, and Grafana will be used to implement the proposed smart plant care and watering system. Each software tool contributes to specific stages of the development lifecycle, from programming and model training to automation and visualization. The details are as follows:

### Thonny

- A beginner-friendly Python IDE that will be used **on the Raspberry Pi 4** to handle the **live streaming of the Pi camera feed**.
- **Reason for Use:**
  - Its **lightweight nature** and minimal configuration requirements make it ideal for running real-time scripts directly on the Raspberry Pi.
- **Features:**
  - **Simple Interface:** Straightforward and intuitive, making it suitable for embedded or resource-limited environments.

- **Integrated Debugger:** Useful for debugging streaming scripts and sensor-based triggers.
- **Virtual Environment Support:** Enables clean Python environment management for modular script execution.

### Visual Studio Code (VS Code)

- A powerful and extensible code editor from Microsoft that will be used **on the laptop** to run the **computer vision model using YOLOv10**.
- **Reason for Use:**
  - VS Code offers **robust support for Python, integrated terminal, and resource management tools**, making it well-suited for handling the heavier computational tasks involved in running and testing the trained model.
- **Features:**
  - **Rich Extension Support:** Enhances development with tools like Python linting, Docker, and Jupyter Notebooks.
  - **Built-in Version Control:** Integrates Git for project tracking and collaborative development.
  - **Cross-platform Compatibility:** Runs smoothly across Windows, macOS, and Linux, ensuring development flexibility.

### Roboflow

- A web-based platform used for managing and annotating computer vision datasets.
- **Features:**
  - **Annotation & Augmentation:** Provides tools for labeling plant images and enhancing dataset diversity.
  - **Dataset Versioning:** Supports tracking of different dataset iterations and their respective training performances.
  - **Seamless Exports:** Easily exports datasets in YOLOv10-compatible formats.

### YOLOv10

- The most recent iteration of the YOLO (You Only Look Once) real-time object detection model.
- **Features:**



- **Enhanced Accuracy & Speed:** Delivers high-performance object detection ideal for identifying plant types and stress indicators.
- **Resource-Efficient:** Supports deployment on edge devices like Raspberry Pi and high-end machines alike.
- **Modular Architecture:** Easier to integrate with various detection and automation workflows.

### Google Colab

- A cloud-hosted Jupyter Notebook platform primarily used for training the YOLOv10 model.
- **Features:**
  - **Free GPU/TPU Access:** Accelerates training, especially for deep learning models.
  - **Drive Integration:** Simplifies dataset management and result storage in the cloud.

### Node-RED

- A flow-based development tool installed on the Raspberry Pi to visually orchestrate the automation logic.
- **Features:**
  - **Visual Programming:** Allows creating automation flows using a simple drag-and-drop interface.
  - **IoT Integration Ready:** Supports protocols such as MQTT, HTTP, and WebSocket for seamless IoT device communication.
  - **Logic and Control Flows:** Automates watering decisions based on moisture readings and detection results.

### Node-RED Dashboard 2.0

- A dashboard extension for Node-RED used to **visualize and monitor real-time sensor data**, such as soil moisture levels.
- **Features:**
  - **Real-Time Data Visualization:** Displays live updates from soil moisture sensors through gauges, charts, and indicators.

- **Customizable Widgets:** Allows the creation of tailored dashboards to monitor different aspects of the smart plant care system.
- **Mobile-Friendly Interface:** Accessible via web browsers on desktops, tablets, and smartphones, providing flexibility in system monitoring.
- **Lightweight and Responsive:** Optimized for smooth performance even on low-power devices like Raspberry Pi.

### 5.2.1 Python Environment

**Platform:** Installed on Laptop (VS Code)

**Purpose:** Library requirements to handle real-time plant detection using a YOLOv10 model and processing detection results to the IoT control system.

Table 5.3 Libraries Requirements

Library	Version	Description
<b>Ultralytics</b>	8.3+ >	Provides the implementation of <b>YOLOv10</b> used for real-time object detection and inference. Enables loading of the best.pt model, frame-by-frame detection, and output parsing.
<b>opencv-python</b>	4.11.0.86	Used to <b>capture video frames</b> from the webcam or camera stream, draw bounding boxes, and preprocess images for model input.
<b>torch</b>	2.6.0+cu126	Core deep learning framework powering the YOLOv10 model. Used for loading the model, running inference, and accessing <b>GPU acceleration</b> (CUDA).
<b>torchvision</b>	0.21.0+cu126	Provides utilities for <b>image transformations</b> , pretrained models, and handling vision-related tasks (used internally by YOLO/Ultralytics).
<b>torchaudio</b>	2.6.0+cu126	Not directly used in my project but often bundled with PyTorch installs; can be safely ignored if unused.
<b>numpy</b>	2.1.1	Essential for <b>array operations</b> and image frame manipulation during OpenCV and PyTorch operations.

<b>requests</b>	(standard lib)	Typically used for making HTTP requests; may be used to send logs or image data to remote servers or REST APIs (optional in your system).
<b>paho.mqtt.client</b>	Latest (2.1.0)	Enables the system to <b>publish detection results</b> via the <b>MQTT protocol</b> to the Raspberry Pi running Node-RED.
<b>json</b>	(standard lib)	Used to format detection results (e.g., { "plant": "Aloe Vera" }) before sending them via MQTT.
<b>time</b>	(standard lib)	Used for <b>timestamping events</b> , introducing delays (e.g., sleep()), and time-based operations.
<b>pandas</b>	(required for dashboard/logging)	While not in the main detection loop, pandas can help structure sensor logs, detection results, or generate summary reports.
<b>serial (pyserial)</b>	(on Arduino communication side)	Used for <b>serial communication with the Arduino</b> via USB (e.g., sending watering commands or receiving moisture data).
<b>socket</b>	(optional)	May be used for <b>low-level network communication</b> , such as streaming the camera feed or interacting with IoT modules over TCP/UDP.
<b>RPi.GPIO</b>	(on Raspberry Pi only)	Used to <b>interface with GPIO pins</b> when the Python code is running on the Raspberry Pi (e.g., manual pump activation via script). Not used in the detection laptop.

Table 5.4 Library Summary Contributions by Category

<b>Function</b>	<b>Libraries Involved</b>
<b>Real-time Detection</b>	ultralytics, torch, torchvision, opencv-python, numpy
<b>MQTT Communication</b>	paho.mqtt.client, json
<b>Camera Streaming &amp; Frame Capture</b>	opencv-python, numpy, time
<b>Hardware/IoT Integration</b>	serial, RPi.GPIO, socket
<b>Data Handling/Visualization</b>	pandas, requests, time

### 5.2.2 Raspberry Pi OS Configuration

The system uses **Raspberry Pi OS based on Debian 12 (Bookworm)** due to its modern software support, enhanced security, and compatibility with the latest Python and AI libraries. Bookworm enables seamless integration with critical packages such as *paho-mqtt*, *serial*, and *GPIO* libraries, which are essential for moisture sensor reading, relay control, and MQTT-based communication. Additionally, the OS ensures reliable performance when running Node-RED and supports future upgrades to more advanced AI models or IoT functionalities.

## 5.3 Setting and Configuration

The development of the updated computer vision (CV) algorithm for plant detection using **YOLOv10** began with refining the testing environment to handle the increased complexity of a larger number of plant classes. Due to the higher computational demands of YOLOv10 compared to previous models, a high-performance laptop was once again chosen over the Raspberry Pi. The laptop's enhanced GPU acceleration and memory were critical in managing the more intensive training and inference tasks associated with a 10-class detection system.

For real-time video feeds, flask code streams the video feed through pi camera. The software setup included installing **Python 3.10.11**, along with updated versions of key libraries such as **torch** for deep learning (PyTorch framework), **OpenCV-python** for image handling, **NumPy** for numerical computation, and **Ultralytics** for seamless YOLOv10 integration.

The development environment was configured using **Visual Studio Code (VS Code)**, benefiting from its robust extension ecosystem, integrated terminal, and virtual environment management. **CUDA 12.6** was utilized for GPU acceleration, significantly boosting the training and inference speeds.

The project began by collecting and curating an extensive dataset featuring **10 different types of indoor plants**, considerably expanding from the previous 3 classes. Training used a combination of pre-trained YOLOv10 weights from the Ultralytics repository, which were fine-tuned on the custom plant dataset. The resulting model checkpoint (best.pt) was optimized for high-precision, real-time detection tasks. Table 5.5 summarizes the new dataset.

Attribute	Details
Number of Images	3090
Image Dimensions	640 x 640 pixels
Color Mode	RGB
Image Size	~200 KB per image
Number of Classes	10
Plant Classes	Aloe Vera, Areca Palm, Chinese Evergreen, Chrysanthemum, Gerbera Daisy, Golden Pothos, Peace Lily, Rhapis Palm, Snake Plant, Spider Plant
Total Size	~1.5 GB
Data Augmentation	Yes (90° Rotation, Flipping, Exposure, Brightness/Contrast Adjustments)
Image Format	JPEG, PNG

Table 5.5 Dataset Summary

### 5.3.1 Preparation of Datasets

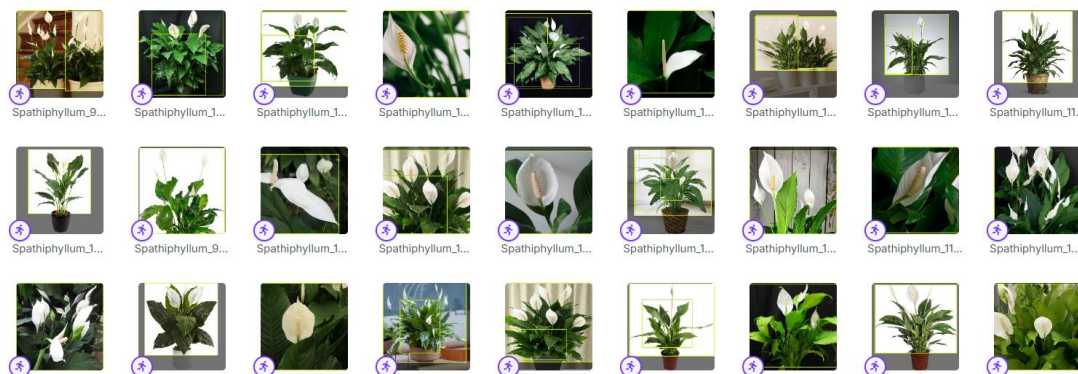


Figure 5.9 Collection of datasets

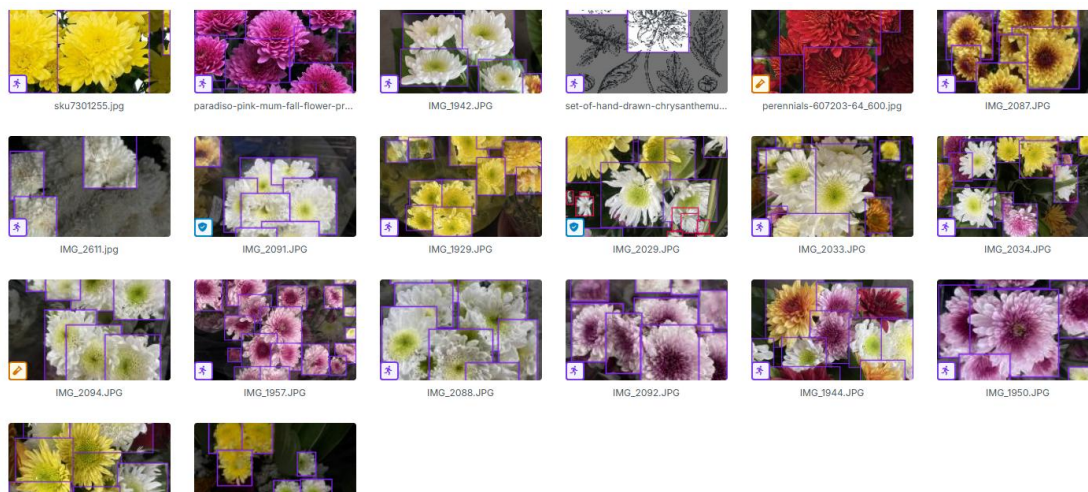


Figure 5.10 Collection of datasets

The preparation of the datasets was an important step in developing the computer vision model. A diversified dataset was compiled, capturing plant images under varying conditions including different lighting scenarios, angles, occlusions, and backgrounds. Annotation was performed using Roboflow, where bounding boxes were drawn around each plant instance and accurately labelled according to their class. Data preprocessing included standardizing image resolutions and applying augmentation techniques such as rotations, flipping, brightness modifications, and exposure adjustments to artificially increase dataset diversity. This approach ensured that the YOLOv10 model could generalize better to unseen real-world conditions. Labelling examples are shown in Figure 5.11.

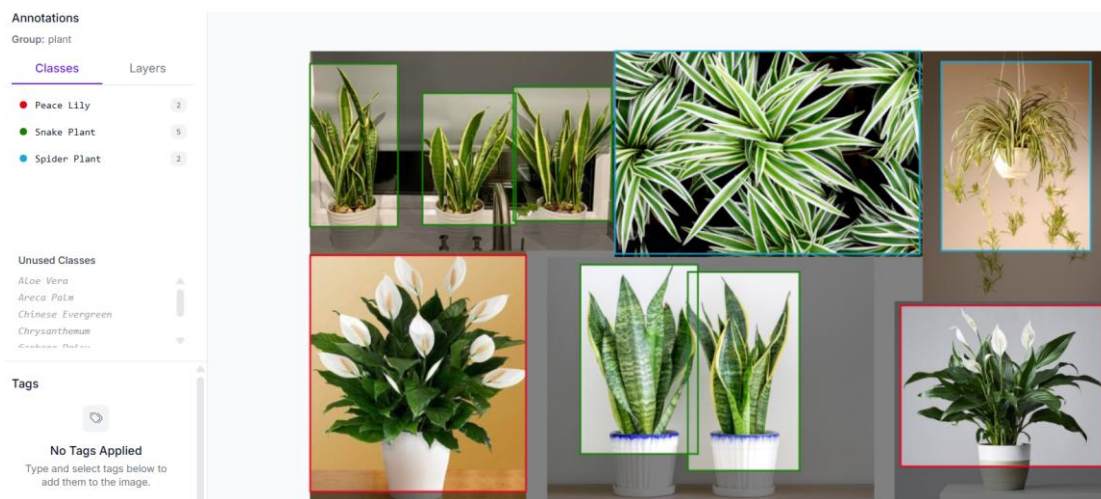


Figure 5.11 Annotation of datasets

## 5.3.2 Training of the Model


+ Code	+ Text	Copy to Drive									
 ...	1/25	6.16G	1.923	5.442	2.344	23	800:	100%	10/10	[00:06<00:00, 1.52it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:03<00:00, 3.49s/it]			
	all		30	34	0.398	0.412	0.381	0.143			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	2/25	6.07G	0.6463	1.51	1.367	23	800:	100%	10/10	[00:04<00:00, 2.19it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 1.79it/s]			
	all		30	34	0.04	0.559	0.0312	0.0126			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	3/25	5.84G	0.4978	0.9673	1.25	23	800:	100%	10/10	[00:03<00:00, 2.57it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 2.10it/s]			
	all		30	34	0.114	0.412	0.0833	0.0442			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	4/25	5.83G	0.4616	0.7606	1.221	24	800:	100%	10/10	[00:04<00:00, 2.14it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 1.04it/s]			
	all		30	34	0.0346	0.618	0.0345	0.0171			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	5/25	5.86G	0.4083	0.6368	1.189	20	800:	100%	10/10	[00:03<00:00, 2.57it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 2.29it/s]			
	all		30	34	0.209	0.235	0.0808	0.0226			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	6/25	5.85G	0.4825	0.6822	1.252	16	800:	100%	10/10	[00:03<00:00, 2.61it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 2.68it/s]			
	all		30	34	0.213	0.412	0.16	0.0643			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	7/25	5.83G	0.4348	0.6045	1.187	21	800:	100%	10/10	[00:04<00:00, 2.05it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 1.12it/s]			
	all		30	34	0.0261	0.0882	0.00946	0.00208			
	Epoch	GPU_mem	box_loss	cls_loss	df_l_loss	Instances	Size				
	8/25	6.08G	0.442	0.5724	1.181	18	800:	100%	10/10	[00:03<00:00, 2.59it/s]	
	Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 1/1 [00:00<00:00, 2.41it/s]			
	all		30	34	0.0258	0.147	0.00944	0.00174			

Figure 5.12 Model Training for YOLOv10

The model was trained using the annotated plant dataset, with the training process optimized for both speed and performance. The annotated images were fed into YOLOv10, where the network iteratively updated its parameters using **backpropagation** and **stochastic gradient descent**.

Training hyperparameters were carefully selected, including a learning rate of **0.01**, batch size of **16**, and a total of **100 epochs**. The use of GPU acceleration (CUDA) allowed significant reductions in training time while improving the model's convergence.

### 5.3.3 Preprocessing and MQTT Integration:

The initial detection script was modified to include MQTT publishing functionality. Upon detecting a plant species, the system formats the result into a JSON payload and publishes it to the MQTT broker hosted on the Raspberry Pi (identified by its static IP).

```
10 # MQTT Setup
11 MQTT_BROKER = "192.168.2.196"
12 MQTT_PORT = 1883
13 MQTT_TOPIC = "plant/detection"
14
15 def connect_mqtt():
16     client = mqtt.Client()
17     client.connect(MQTT_BROKER, MQTT_PORT, 60)
18     return client
19
20 mqtt_client = connect_mqtt()
21 print(f"Connecting to MQTT broker at {MQTT_BROKER}:{MQTT_PORT}")
```

Figure 5.13 MQTT Integration

### 5.3.4 Sensor Assignment Function

A key software feature was the implementation of a logic mapping function that **assigns detected plant species to specific moisture sensors**. This mapping was necessary for targeting irrigation to the correct plant and was based on predefined pairings of plant types to sensor numbers.

```
160 # Compute sensor assignments
161 sensor_assignments = {
162     "Moisture Sensor 1": "None",
163     "Moisture Sensor 2": "None",
164     "Moisture Sensor 3": "None",
165     "Moisture Sensor 4": "None"
166 }
167 for det in current_detections:
168     xyxy = det['xyxy']
169     center_x = (xyxy[0] + xyxy[2]) / 2
170     center_y = (xyxy[1] + xyxy[3]) / 2
171     classname = labels[det['cls']]
172     if center_x < imgW / 2 and center_y < imgH / 2:
173         sensor_assignments["Moisture Sensor 1"] = classname
174     elif center_x >= imgW / 2 and center_y < imgH / 2:
175         sensor_assignments["Moisture Sensor 2"] = classname
176     elif center_x < imgW / 2 and center_y >= imgH / 2:
177         sensor_assignments["Moisture Sensor 3"] = classname
178     elif center_x >= imgW / 2 and center_y >= imgH / 2:
179         sensor_assignments["Moisture Sensor 4"] = classname
```

Figure 5.14 Sensor Assignment Function



### 5.3.5 Flask Server Setup

A Flask-based web server was written and executed via **Thonny and Visual Studio** to facilitate **live video streaming** from the Raspberry Pi Camera to the laptop for frame capture and preprocessing. This allowed the YOLOv10 model to continuously receive frames for inference.

### 5.3.6 Setting Up the Pi Camera

- The Raspberry Pi Camera Module was configured using terminal commands such as “*sudo raspi-config*” to enable the camera interface.
- Video stream was transmitted over the network using **MJPEG** or **GStreamer** protocols, ensuring lightweight, real-time delivery to the laptop’s detection script.

### 5.3.7 YOLOv10 Threshold Configuration

```
# YOLO model and settings
model_path = 'Best.pt'
min_thresh = 0.70
stream_url = 'http://192.168.43.154:5000/video_feed'
imgW, imgH = 640, 640 # Match YOLO input
input_size = 640 # Standard YOLO input size
inference_interval = 0 # Balance accuracy and performance
```

Figure 5.15 Threshold Configuration

The **confidence threshold** for the YOLOv10 model was set to **0.7** to improve detection accuracy by filtering out low-confidence predictions. Additionally, an **inference interval** was introduced to **balance performance** and **processing load**, especially on lightweight CPUs. This helped maintain real-time processing without overloading the system or causing detection lag.

### 5.3.8 MQTT publish rate

To reduce unnecessary traffic and prevent MQTT message spamming, the publish rate was configured to **once every 5 seconds**. This ensures efficient communication without overwhelming the broker or causing message queue delays.

### 5.3.9 AutoStart Scripts

For system automation, **Node-RED** was configured to start automatically on Raspberry Pi boot using “*systemctl*”.

The following command was used: “*sudo systemctl enable nodered.service*”

This ensures that the control logic (hosted on Node-RED) is always active after reboot or power loss

### 5.3.10 Network Setup

To ensure reliable **video streaming**, **VNC viewer access**, and **Node-RED dashboard availability**, the Raspberry Pi was assigned a **static IP address**. This was configured in: “*/etc/dhcpd.conf*”

And Wi-Fi credentials were managed in: “*/etc/wpa\_supplicant/wpa\_supplicant.conf*”

This configuration guarantees consistent connectivity and easier remote monitoring from the laptop.

### 5.3.11 Setup of Node-RED on Raspberry Pi

#### MQTT Subscription:

Node-RED was configured to subscribe to a specific topic “*plant/detection*” via the MQTT input node to receive detection results.

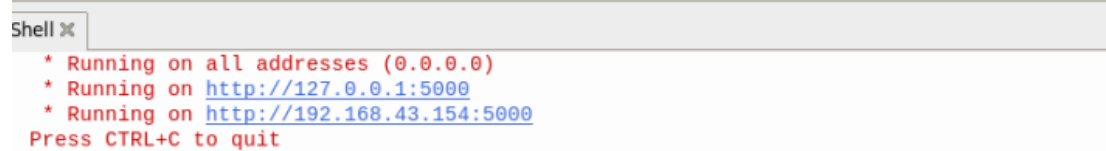
## 5.4 System Operation (with Screenshot)

### 5.4.1 Startup Sequence

#### 1. Raspberry Pi Flask Stream:

The Raspberry Pi camera begins capturing video, and a Flask-based video streaming server is launched. This stream is made accessible at: *http://<raspi\_ip>:5000/video\_feed*

```
11 def generate_frames():
12     while True:
13         frame = camera.capture_array()
14         ret, buffer = cv2.imencode('.jpg', frame)
15         frame = buffer.tobytes()
16         yield (b'--frame\r\n'
17               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
18
19 @app.route('/video_feed')
20 def video_feed():
21     return Response(generate_frames(),
22                     mimetype='multipart/x-mixed-replace; boundary=frame')
23
24 if __name__ == "__main__":
25     app.run(host='0.0.0.0', port=5000, debug=False)
26
27 |
```



```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.43.154:5000
Press CTRL+C to quit
```

Figure 5.16 Raspberry Pi Flask Stream

#### 2. Laptop CV Model Execution:

The laptop, running a Python script, accesses the video stream. The YOLOv10 model (previously trained and exported as best.pt) is loaded, and real-time detection begins. Each frame is pre-processed, and objects (plants) are classified.

```

# MQTT Setup
MQTT_BROKER = "192.168.43.154"
MQTT_PORT = 1883
MQTT_TOPIC = "plant/detection"

def connect_mqtt():
    client = mqtt.Client()
    client.connect(MQTT_BROKER, MQTT_PORT, 60)
    return client

mqtt_client = connect_mqtt()
print(f"Connecting to MQTT broker at {MQTT_BROKER}:{MQTT_PORT}")

# YOLO model and settings
model_path = 'Best.pt'
min_thresh = 0.70
stream_url = 'http://192.168.43.154:5000/video_feed'
imgW, imgH = 640, 640 # Match YOLO input
input_size = 640 # Standard YOLO input size
inference_interval = 0 # Balance accuracy and performance

```

Figure 5.17 CV Model Execution

#### 5.4.2 Detection Demo

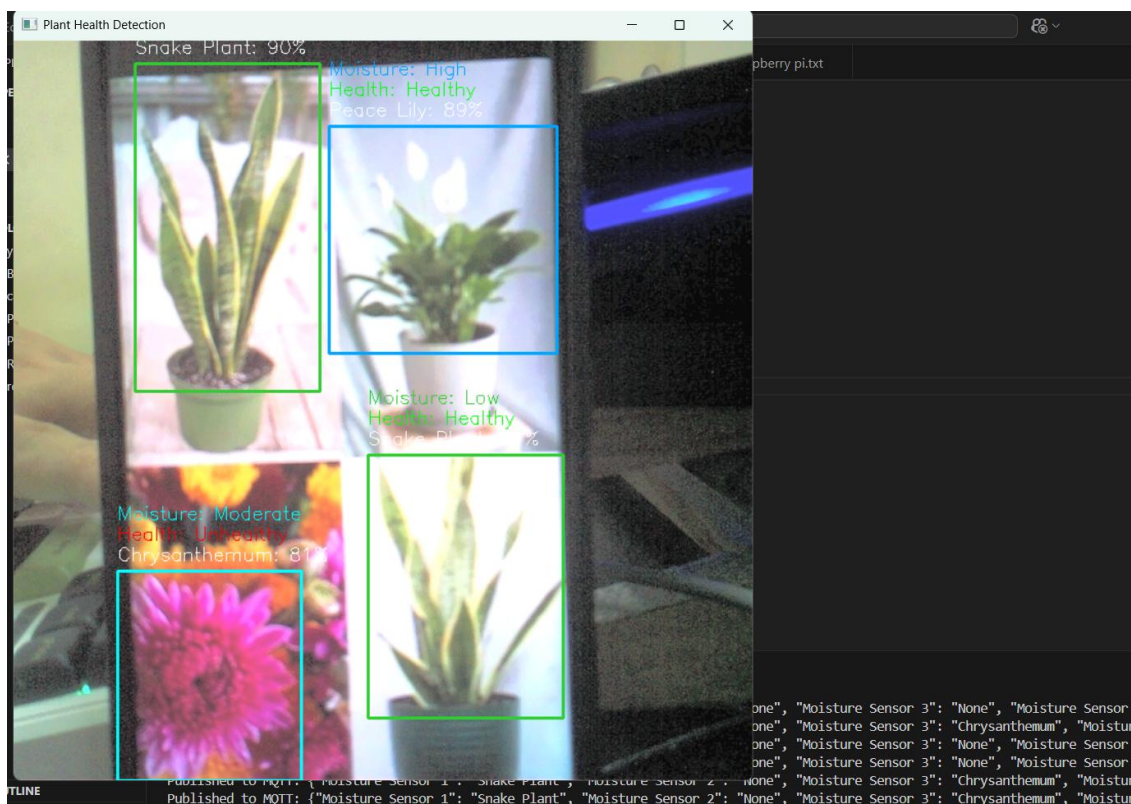


Figure 5.18 Detection Demo

```

{"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum", "Moisture Sensor 4": "Snake Plant"}
{"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "None", "Moisture Sensor 4": "None"}
{"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "None", "Moisture Sensor 4": "Snake Plant"}
{"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum", "Moisture Sensor 4": "Snake Plant"}
{"Moisture Sensor 1": "Snake Plant", "Moisture Sensor 2": "None", "Moisture Sensor 3": "Chrysanthemum", "Moisture Sensor 4": "Snake Plant"}

```

Figure 5.19 Terminal Output

### 5.4.3 Watering Action Triggered

Once a plant is identified and its corresponding soil moisture value is obtained:

- Node-RED passes the detection result (via MQTT) to the Arduino. The Arduino compares the current soil moisture value with the predefined threshold for that plant type.

If the soil is dry:

- The **relay is activated**
- The **water pump turns on** according to predefined moisture requirements.

```

5/5/2025, 9:04:42 PM  node: debug 1
plant/detection : msg.payload : Object
▼ object
  action: "Sensor assignments
  updated"
▼ assignments: object
  Moisture Sensor 1: "Snake Plant"
  Moisture Sensor 2: "Peace Lily"
  Moisture Sensor 3:
  "Chrysanthemum"
  Moisture Sensor 4: "Snake Plant"

```

Figure 5.20 Moisture Assignments

### 5.4.4 Visualization Using Node-RED Dashboard 2.0

UI Chart and Gauge Nodes were configured to:

- Display real-time **moisture readings** for each plant.
- Indicate which plant was currently being watered and track past activity.
- Control panel was implemented for clearing of visual charts and gauges for clearer updated history.

The **Dashboard Layout** was customized to present data in a clean, user-friendly interface, enabling at-a-glance monitoring of plant conditions and system activity.

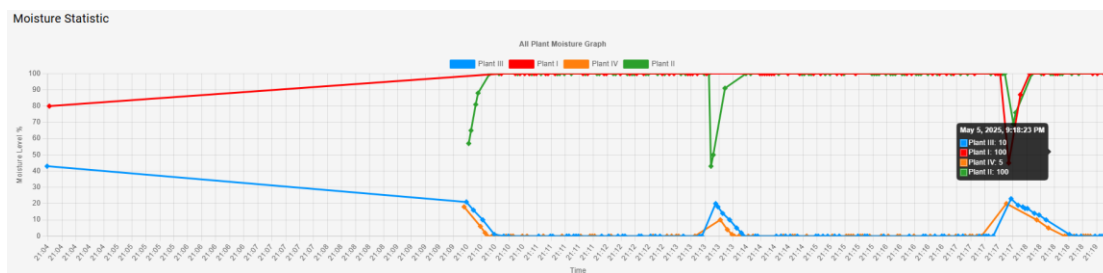


Figure 5.21 Dashboard Visualization

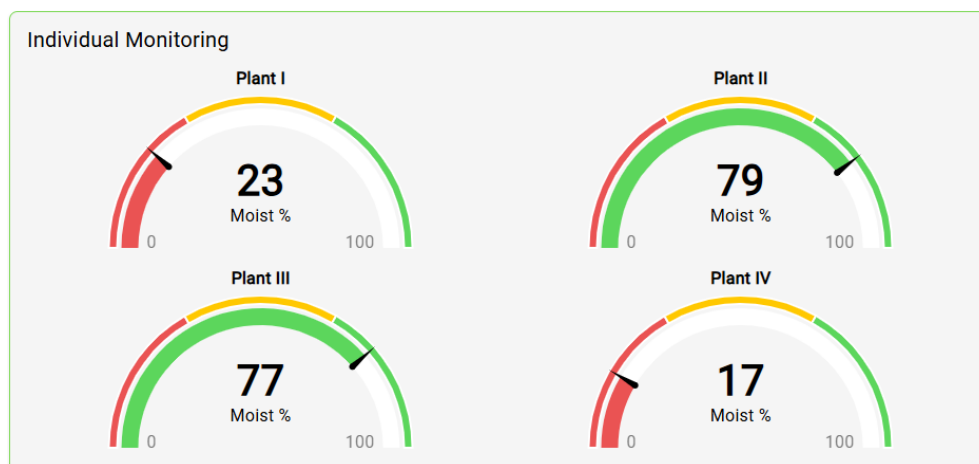


Figure 5.22 Gauge Visualization

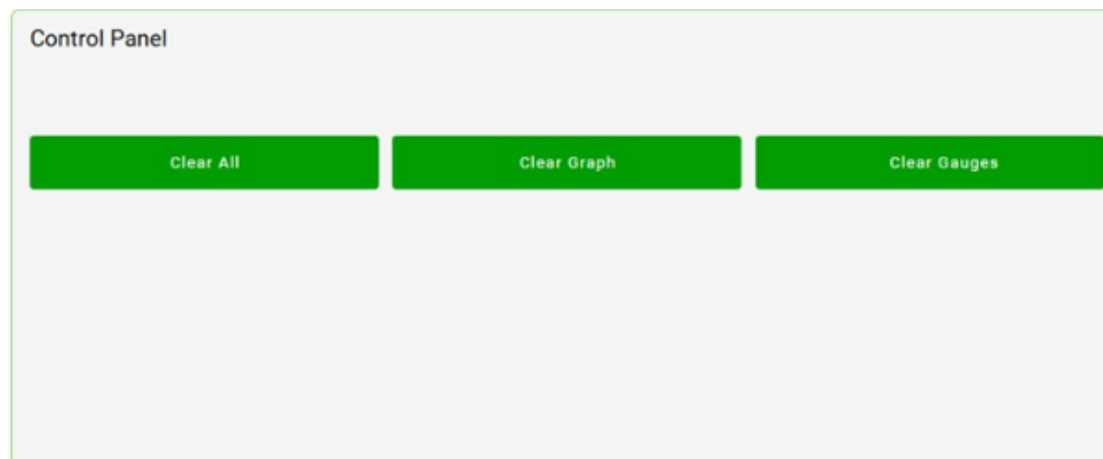


Figure 5.23 Control Panel

### 5.5.5 Function Node Configuration in Node-RED

To process logic and condition-based outputs in the smart watering system, a **Function Node** was used within Node-RED. This node played a critical role in interpreting sensor data and determining when to activate watering actions.

The function node was scripted to:

- Analysed incoming soil moisture readings.
- Compare them to the predefined moisture threshold (e.g., below 30%).
- Trigger a corresponding message payload to activate or deactivate the water pump.
- This allowed the system to perform conditional automation in real-time based on live sensor input.

```
// Handle plant detection messages from MQTT
if (typeof msg.payload === "object" && !Array.isArray(msg.payload)) {
    let sensorAssignments = msg.payload;

    // Store the assignments directly (no deduplication)
    flow.set("sensorAssignments", sensorAssignments);
    msg.payload = {
        action: "Sensor assignments updated",
        assignments: sensorAssignments
    };
    return msg;
}

// Handle moisture sensor data from Arduino
} else if (typeof msg.payload === "string" && msg.payload.includes("Plant") && msg.payload.includes("Moisture")) {
    let sensorAssignments = flow.get("sensorAssignments") || {};
    let sensorMoisture = msg.payload.trim(); // Remove \r\n

    // Parse format: "Plant X Moisture (raw): Y -> Z%"
    const match = sensorMoisture.match(/Plant (\d+) Moisture \(raw\): (\d+) -> (\d+)\%/);
    if (!match) {
        msg.payload = { error: "Invalid moisture data format." };
        return msg;
    }

    // Extract sensor number, raw value, and percentage
    let sensorNumber = match[1]; // e.g., "4"
    let rawMoisture = parseInt(match[2]); // e.g., 592
    let currentMoisture = parseInt(match[3]); // e.g., 0
```

Figure 5.24 Function Node Configuration

## 5.5 Implementation Issues and Challenges

### 5.5.1 Hardware Limitations

One of the critical issues faced was attempting to run the **YOLOv10 model on a lightweight device**, specifically the Raspberry Pi. While YOLOv10 offers improved performance and speed, the limited processing power and thermal management of the Raspberry Pi made it unsuitable for direct model inference. During prolonged tests, the excessive CPU load from running YOLOv10 locally **caused overheating**, which led to hardware instability and ultimately **damaged the camera module**, as illustrated in Figure 5.23.

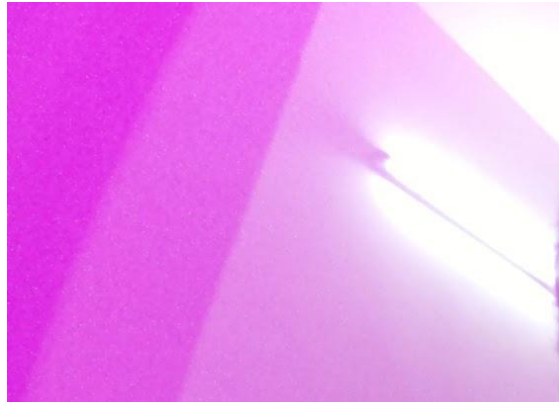


Figure 5.25 Test Image of Camera Module

### 5.5.2 Integration with Arduino Leonardo

Another major challenge arose from using **Arduino Leonardo**, which lacks built-in Wi-Fi capability. As a result, it **could not directly connect to MQTT brokers** or communicate wirelessly. This limitation required the implementation of a **wired serial connection between the Arduino and Raspberry Pi**, complicating the hardware layout and increasing dependency on physical connections for data transmission.

### 5.5.3 Network Communication Delay

The system depends on **real-time MQTT messaging** between the laptop (running the CV model) and Raspberry Pi (managing sensors and pumps). However, occasional **latency issues** were observed during command transmission, especially under heavy network load or during video streaming. These delays affected the promptness of watering actions and data feedback loops.

### 5.5.4 Camera Positioning

The **accuracy of plant detection** was highly sensitive to the **camera's height, angle, and lighting**. Improper positioning resulted in:

- Incomplete plant capture
- Missed detections or bounding box errors
- False positives due to background noise or shadow interference



This required repeated **physical adjustments** and fine-tuning of the webcam placement during system setup to ensure consistent and accurate detection.

## 5.6 Concluding Remark

The development and deployment of the smart plant monitoring and self-watering system proved that computer vision, IoT sensors, and automation into a functional real-time solution. Using the YOLOv10 model and OpenCV for plant detection, alongside MQTT for communication and Raspberry Pi to Arduino integration, the system successfully achieved its intended goal.

The project offered important insights into real-world system implementation despite a number of technical obstacles, including hardware constraints, integration complexity, and calibration irregularities. Important tactics that guaranteed overall system performance and stability included stream processing using Flask, model offloading to laptops, and the application of error-handling techniques.

In summary, the project not only demonstrated the potential of AI-driven plant care, but it also brought to light crucial factors to take into account when implementing such solutions in settings with limited resources. These include enhancing hardware configuration for long-term operation, guaranteeing dependable sensor communication, and optimising model inference for edge devices. Future iterations and advancements in automated smart agriculture systems will be built upon the lessons learnt.

# Chapter 6

## System Evaluation and Discussion

### 6.1 System Testing and Performance Metrics

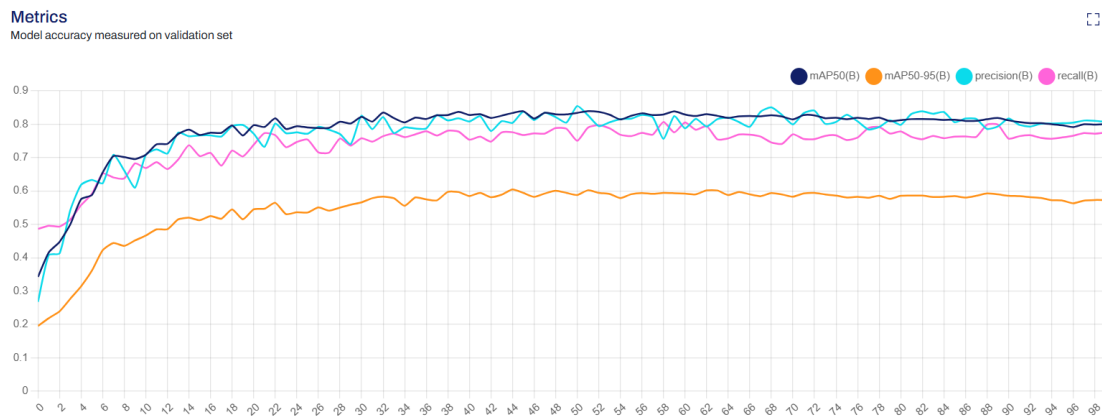


Figure 6.1 Metrics Curve

During the training process, the performance of the model was continuously monitored and retrained, until metrics such as **loss values, precision, recall, and mean Average Precision (mAP)** being used to determine how well it could generalize to unseen data. The precision-recall curve played a crucial role by showing a complete overview of the balance between precision and recall at different threshold levels. By analyzing the form of this curve, potential problems such as overfitting or underfitting can be recognized. Training stopped when the precision-recall curve showed a satisfactory balance of high precision and recall values, indicating that the model had effectively learned to detect plants without showing a preference for either metric. This method guaranteed that the model maintained a strong level of effectiveness in practical situations, where both precision and recall are essential for accurate plant identification.

The outcomes of the inference testing phase show the key metrics and trends observed during the training of the YOLOv10 model. These metrics include training losses (**train/box\_loss, train/cls\_loss, and train/dfl\_loss**), validation losses (**val/box\_loss, val/cls\_loss, and**

val/dfl\_loss), and performance metrics such as **precision**, **recall**, and **mean Average Precision (mAP)**.

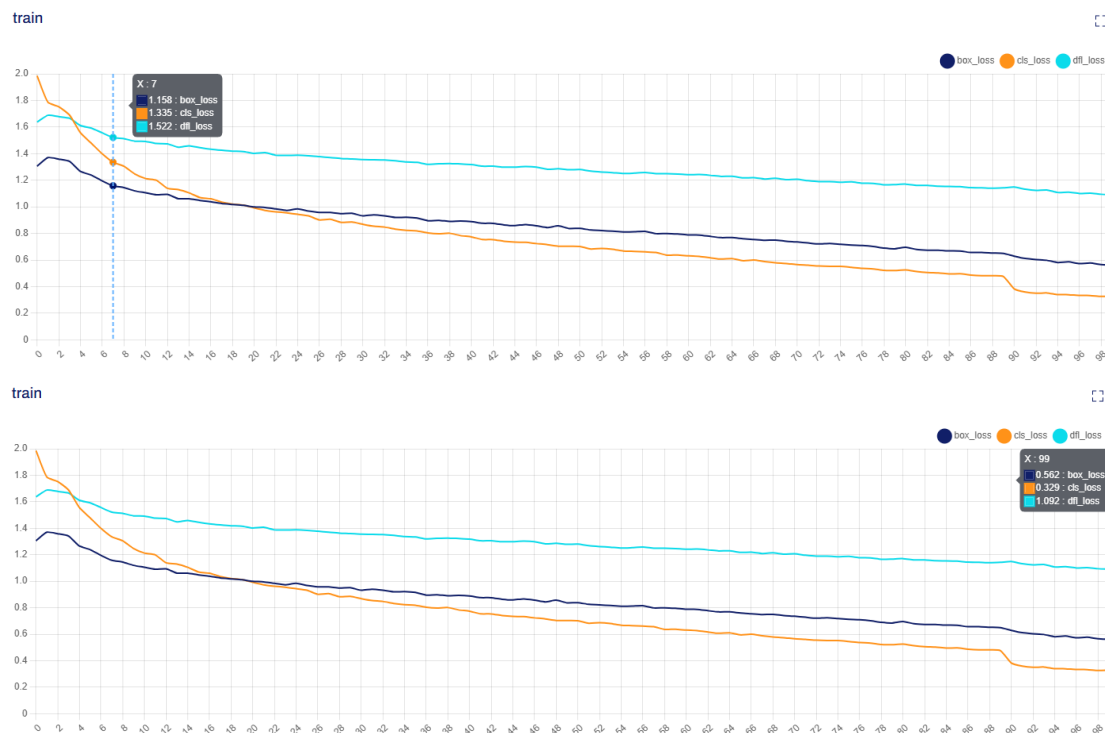


Figure 6.2 Training Loss Metrics

## Training Loss Metrics

The training results indicate a consistent decrease in the training loss metrics beginning at 7<sup>th</sup> epoch, reflecting the model's improving accuracy and efficiency in plant detection tasks. Specifically, the **train/box\_loss**, which represents the error in predicting the bounding boxes around detected plants, **decreased** steadily from an initial value of **1.158** in the 7<sup>th</sup> epoch to **0.562** by the of epochs 99. This reduction suggests that the model's predictions of plant locations became more precise over time. Similarly, the **train/cls\_loss**, which measures the error associated with classifying detected objects correctly, decreased significantly from **1.335** to **0.327**, indicating enhanced accuracy in identifying different plant species. The **train/dfl\_loss**, which evaluates the distribution of the predicted bounding boxes relative to the ground truths, also showed a downward trend, reducing from **1.522** to approximately **1.092** by the end of the training period. This overall reduction in training losses illustrates that the model is learning to minimize errors effectively, enhancing its ability to correctly identify and classify plants.

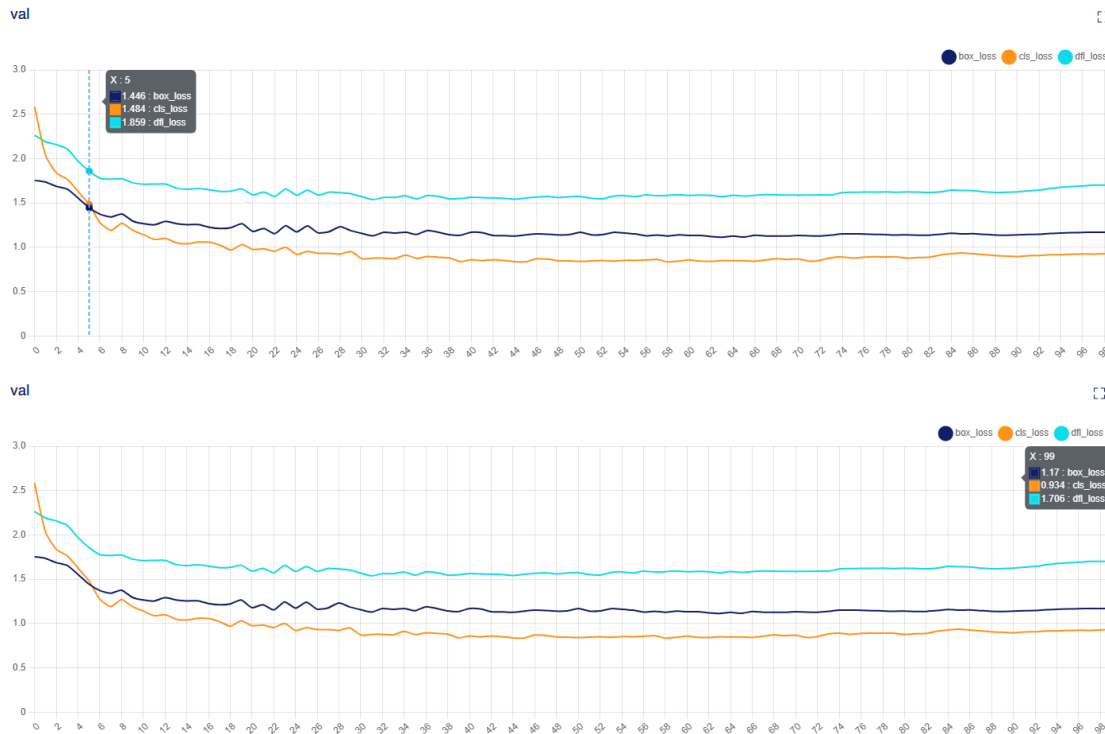


Figure 6.3 Validation Loss Metrics

### Validation Loss Metrics

The validation loss metrics also displayed a decreasing trend, show the ability of the model to generalize well to new, unseen data. Moreover, the **val/box\_loss** began to drop rapidly from **1.446** at epoch 5 to **1.17** by the end of epoch 99, demonstrating that the model is capable of maintaining high accuracy in plant localization even on the validation set. The **val/cls\_loss** also showed significant improvement, reducing from **1.484** to around **0.934**, which suggests that the model's classification performance on new data is becoming increasingly reliable. The consistent decrease in **val/dfl\_loss** further supports these results, reflecting the model's enhanced ability to produce accurate bounding boxes that closely match the ground truth.

## metrics

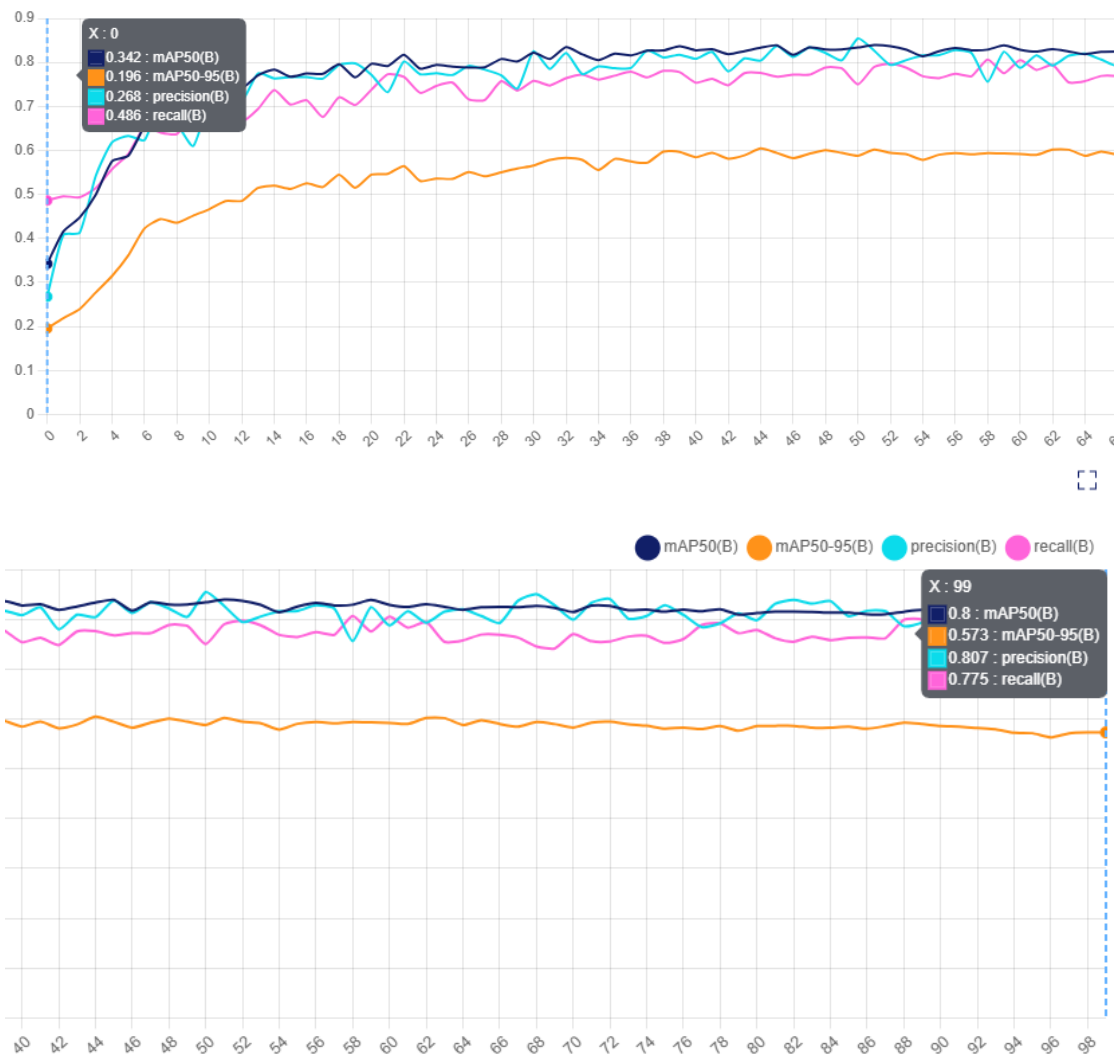


Figure 6.4 Performance Metrics

## Performance Metrics: Precision, Recall, and mAP

The performance metrics, including precision, recall, and mean Average Precision (mAP), further underscore the model's progress. The **metrics/precision(B)**, which measures the proportion of **true positive detections** among all detections made, improved markedly from **0.268** in epoch 0 to **0.807** in epoch 99. This improvement indicates that the model is effectively reducing the number of false positives and increasing the accuracy of its predictions. The **metrics/recall(B)**, which represents the proportion of **actual positive samples correctly identified**, also showed substantial growth, rising from **0.486** to **0.778**. This trend suggests that the model is becoming more proficient at detecting all relevant objects in the images. The **metrics/mAP50(B)** and **metrics/mAP50-95(B)**, which evaluate the accuracy of the model's detections across different Intersection over Union (IoU) thresholds, also demonstrated

positive trends. Metrics/mAP50(B) increased from **0.342** to approximately **0.8**, while **metrics/mAP50-95(B)** increased from **0.196** to around **0.573**, indicating that the model is achieving higher detection accuracy across varying degrees of overlap between the predicted and actual bounding boxes.

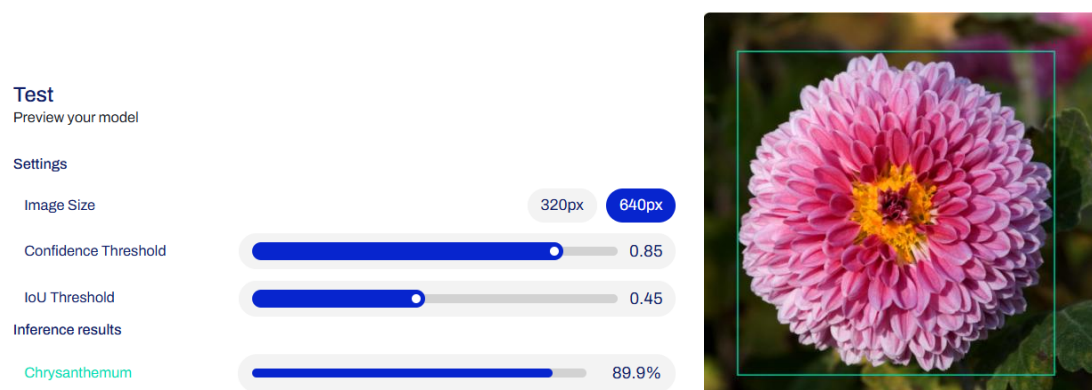
## 6.2 Testing Setup and Result

Following training, inference testing was performed with a live webcam feed, replicating real-world deployment conditions. A Python script processed video frames in real-time, ran inference through the YOLOv10 model, and drew bounding boxes with class labels around detected plants.

The performance was assessed based on two key metrics:

- **Precision:** measuring the percentage of correctly predicted positive instances.
- **Recall:** measuring the percentage of actual positives that were correctly identified.

Testing results showed the model could reliably identify all 10 classes with high confidence under different lighting and background conditions. Fine-tuning of parameters such as **confidence thresholds** and **non-maximum suppression (NMS)** was conducted to optimize real-time performance.



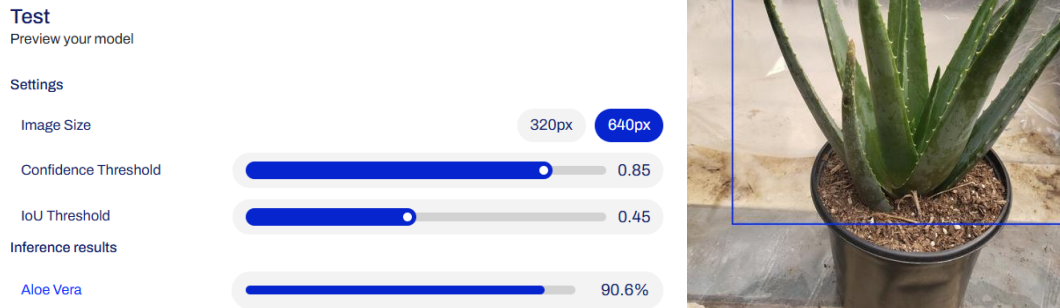


Figure 6.5 Inference testing of Model

### Performance Metrics Across Epochs

A detailed comparison of performance metrics at epochs 50, 80, and 100 is shown in Table 6.1. The table illustrates how extended training boosts the model's detection precision, recall, and overall accuracy.

Epoch	Precision (B)	Recall (B)	mAP@0.5 (B)	mAP@0.5:0.95 (B)
50	0.614	0.594	0.583	0.512
80	0.713	0.626	0.68	0.535
100	0.807	0.778	0.80	0.573

Table 6.1 Performance Metrics table

## Testing Procedures and Results

Test Case	Procedure	Expected Result	Actual Result	Status
<b>Plant Detection Accuracy</b>	Ran live video feed with multiple known plant types under various lighting conditions.	Model correctly identifies plant types with at least <b>85%</b> confidence.	Detection confidence ranged from <b>85%–92%</b> in controlled lighting.	Passed
<b>Plant Classification Consistency</b>	Repeated tests on the same plant species from different angles and distances.	Consistent classification across multiple test scenarios.	YOLOv10 consistently classified plants with minimal variation.	Passed
<b>Real-Time Inference Speed</b>	Measured the time between camera input and bounding box display on screen.	Inference delay should be under 1 second for smooth real-time detection.	Average delay observed: <b>~0.5 seconds</b> .	Passed
<b>Detection–Irrigation Integration</b>	Tested whether detection of a plant triggers correct moisture threshold retrieval and irrigation.	Detected plant type fetches correct profile, system applies watering rules.	Plant-specific profile correctly loaded; irrigation activated accordingly.	Passed
<b>False Positive Filtering</b>	Introduced non-plant objects (e.g., furniture, tools) into the frame.	Model should not detect or misclassify non-plant objects as plants.	No false positives detected in non-plant objects.	Passed

Table 6.2 Testing Procedures and Results



## 6.3 Project Challenges

The main challenge addressed by this project is by integrating multiple advanced technologies such as computer vision, IoT sensing, automated watering, and air purification into a single, reliable smart indoor plant care system. Even if separate parts, such air purifiers or soil moisture sensors, are well-known, integrating them into a coherent system that reacts sensibly to data in real time added a significant amount of complexity.

One major challenge was achieving accurate plant type recognition with a camera system, particularly under shifting lighting conditions and with seemingly identical species. Computer vision models required extensive training and testing to achieve acceptable accuracy for species-specific watering profiles. Inconsistent lighting and camera angles further increased the likelihood of misclassification.

On the hardware side, synchronizing the data flow between the Raspberry Pi, Arduino Leonardo, and multiple sensors presented integration difficulties, particularly when managing simultaneous operations like video streaming, sensor readings, and actuation of pumps. Resource limitations on edge devices also constrained model deployment, requiring efficient offloading of inference tasks to the laptop while maintaining real-time responsiveness. Another challenge was in watering and moisture visualization. Developing a reliable dashboard to display soil moisture data in real time required precise sensor calibration and MQTT-based communication handling. Achieving stability in data transmission and ensuring dashboard updates without latency were key issues to solve.

In summary, this project addresses the multiple task of creating an intelligent, plant-aware environment that is adaptable, automated, and user-friendly by combining various technologies to create a fully functional and scalable indoor plant care system.

## 6.4 Objectives Evaluation

### 1. Design and Train a Plant Recognition System for Watering Automation (Achieved)

The project successfully integrates a YOLOv10-based computer vision model trained to identify multiple types of indoor plants through a live camera feed. The camera, mounted on a Raspberry Pi 4, streams video to a laptop for detection, where plant types are recognized in real-time. Each detected plant is linked to a pre-defined moisture profile to ensure species-specific hydration. When paired with moisture readings from sensors in the soil, the system executes a custom watering action through relays and water pumps, ensuring each plant receives only the amount of water needed to maintain optimal health.

### 2. Develop a Functional Prototype of a Smart Air Purifier with Self-Watering Plant Integration (Achieved)

A fully operational prototype has been developed, assembled on a three-tier trolley frame. This prototype integrates multiple components, including:

- A live camera for plant detection on the top level.
  - Plant compartments with embedded moisture sensors and watering systems on the middle level.
  - Air-purifying plants functioning alongside moisture automation, all housed within a compact, mobile frame.
- The design offers a holistic indoor environment solution that improves both air quality and plant health through smart automation. The combination of air-purifying plants and intelligent watering control helps maintain humidity and freshness in the space.

### 3. Incorporate IoT Sensors for Monitoring and Dashboard Visualization (Achieved)

The system effectively incorporates IoT soil moisture sensors within each plant pot and an AQI sensor module to monitor the surrounding air quality. All sensor data is transmitted using MQTT protocol to a centralized Node-RED dashboard, which presents real-time updates on:

- Soil moisture levels of each plant.
- Detected plant species.
- Current AQI values.
- Watering history and system status.

The dashboard allows users to visualize and interpret environmental data, ensuring informed decision-making and enhancing the overall usability of the system. Alerts and visual indicators provide additional support for proactive plant care and air quality monitoring.

## 6.5 Concluding Remark

The completion of this project signifies a meaningful advancement in smart indoor plant care by successfully integrating computer vision, IoT sensing, and automation technologies into a unified, functional system. Through the deployment of a plant based recognition model, a soil moisture-triggered watering mechanism, and real-time environmental monitoring via IoT sensors, the system demonstrates an intelligent, adaptive approach to maintaining plant health and improving air quality.

The solution achieved all its core objectives such as accurate plant type detection, species-specific watering automation, and real-time environmental data visualization through an interactive dashboard. Despite technical challenges such as hardware-software integration, sensor calibration, and edge computing constraints, strategic decisions like offloading model inference to a laptop and using Flask and Node-RED ensured system performance and stability.

Overall, this project not only proves the feasibility of AI-driven plant care but also opens new opportunities for smart home applications. It provides a strong foundation for future improvements, such as mobile app integration, wireless sensor expansion, and predictive watering using machine learning. The success of this prototype shows how data-driven automation can significantly enhance plant maintenance, reduce human effort, and contribute to sustainable indoor environments.

# Chapter 7

## Conclusion and Recommendation

### 7.1 Conclusion

This project successfully developed the Oasis system, an automated indoor plant care solution that integrates computer vision, IoT sensors, and smart watering mechanisms. By combining real-time plant detection using YOLOv10 with soil moisture monitoring and automated irrigation, the system reduces manual effort while promoting healthier indoor air-purifying plants. Despite some technical challenges, such as processing limitations and occasional detection errors, the project met its objectives and demonstrated the potential of using affordable, intelligent systems to support sustainable living and smart home applications.

### 7.2 Recommendation

Based on the observations and lessons learnt from this project, the following recommendations are made to improve the performance, scalability, and usability of the Oasis system in future iterations:

#### 1. Upgrade to Edge AI Accelerators

To address the Raspberry Pi's computing constraints, edge AI accelerators such as the Google Coral TPU or NVIDIA Jetson Nano are recommended. These devices provide increased processing power for machine learning inference, allowing for faster and more sophisticated model execution. As a result, the system could handle higher-resolution photos, real-time multiple-object identification, and even more advanced analytics with little delay.

#### 2. Expand the Plant Dataset and Retrain the Model

The latest YOLOv10 model was trained on a small but particular dataset of air-purifying plants. To improve accuracy and robustness, future work should concentrate on gathering a larger and

more diverse dataset that includes both indoor and outdoor plant species. Continuous retraining and model validation using new data can improve generalisation across plant kinds, lighting situations, and background noise.

### **3. Develop a User-Centric Interface**

Implementing a dedicated user interface, such as a web or mobile application, can greatly increase user engagement. This dashboard might show current plant detection status, moisture levels, watering history, and system alerts. It may also allow users to customise watering thresholds, schedule system operations, or remotely regulate the pump, making the system more accessible and manageable to non-technical users.

### **4. Integrate Plant Health Monitoring**

The system's capabilities can be expanded beyond presence detection to include plant health assessments. Using powerful image processing and deep learning techniques, the system may be trained to detect disease, discolouration, wilting, and pest infestations. These findings could serve as early alerts for users, encouraging proactive and preventive plant care.

### **5. Incorporate Environmental and Weather Data**

For installations near windows or in semi-outdoor locations, ambient temperature, humidity, and light intensity may all have an impact on watering requirements. Integrating external sensors or APIs that provide weather forecasts could help the system make more informed decisions, such as deferring watering during wet or humid weather.

### **6. Plan for Scalability and Integration**

The system architecture should be scaleable in order to facilitate widespread adoption. This includes support for several plant modules, connection with various home automation ecosystems (such as Google Home and Amazon Alexa), and modular hardware designs that can be readily upgraded or customised.

By implementing these recommendations, the Oasis system can develop into a full smart gardening solution that not only fits current user needs but also adapts to future technical improvements and environmental concerns.

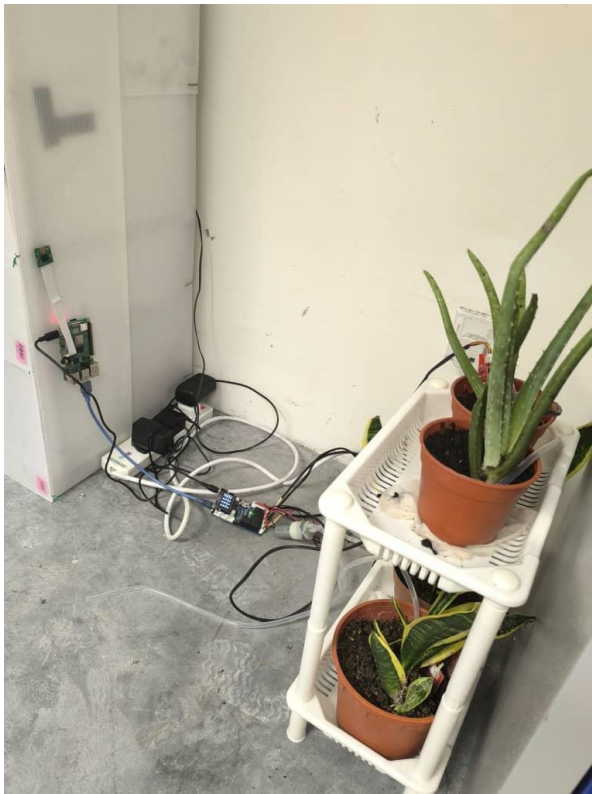
## REFERENCES

- [1] Brilli, F., Fares, S., Ghirardo, A., De Visser, P., Calatayud, V., Muñoz, A., Annesi-Maesano, I., Sebastiani, F., Alivernini, A., Varriale, V., & Menghini, F. (2018). Plants for sustainable improvement of indoor air quality. *Trends in Plant Science*, 23(6), 507–512. <https://doi.org/10.1016/j.tplants.2018.03.004>
- [2] M. L. K. et al., "Effects of Plant Health on Air Quality," *Journal of Indoor Air*, vol. 22, no. 3, pp. 215-227, 2012.
- [3] T. H. C. et al., "The Impact of Watering Regimes on Plant Health and Performance," *Horticultural Science*, vol. 54, no. 1, pp. 23-32, 2019.
- [4] A. R. Smith, "Computer Vision for Plant Monitoring: Current Technologies and Future Directions," *Computers and Electronics in Agriculture*, vol. 169, p. 105211, 2020.
- [5] J. A. Wang, L. J. Lee, "Real-Time Image Processing for Monitoring Plant Health," *Journal of Agricultural Engineering Research*, vol. 86, no. 1, pp. 72-80, 2003.
- [6] M. J. Lee, S. K. Kim, "Integration of IoT and Computer Vision for Smart Agriculture," *Sensors*, vol. 20, no. 8, p. 2335, 2020.
- [7] B. R. Miller, "Urbanization and the Need for Efficient Plant Care Systems," *Urban Ecosystems*, vol. 22, no. 2, pp. 299-310, 2019.
- [8] K. H. Lee and S. J. Park, "Challenges in Traditional Plant Monitoring Systems," *Horticultural Technology*, vol. 29, no. 4, pp. 467-476, 2019.
- [9] Yadav, P. K., Thomasson, J. A., Searcy, S. W., Hardin, R. G., Braga-Neto, U., Popescu, S. C., Martin, D. E., Rodriguez, R., Meza, K., Enciso, J., Diaz, J. S., & Wang, T. (2022). Assessing the performance of YOLOv5 algorithm for detecting volunteer cotton plants in corn fields at three different growth stages. *Artificial Intelligence in Agriculture*, 6, 292–303. <https://doi.org/10.1016/j.aiia.2022.11.005>
- [10] Li, J., Qiao, Y., Liu, S., Zhang, J., Yang, Z., & Wang, M. (2022). An improved YOLOv5-based vegetable disease detection method. *Computers and Electronics in Agriculture*, 202, 107345. <https://doi.org/10.1016/j.compag.2022.107345>
- [11] Ultralytics. (n.d.-a). GitHub - ultralytics/ultralytics: NEW - YOLOv8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite. GitHub. <https://github.com/ultralytics/ultralytics>
- [12] YOLOv8 vs. YOLOv5: Choosing the Best Object Detection Model. (n.d.-b). <https://www.augmentedstartups.com/blog/yolov8-vs-yolov5-choosing-the-best-object-detection-model>
- [13] Bie, M., Liu, Y., Li, G., Hong, J., & Li, J. (2023). Real-time vehicle detection algorithm based on a lightweight You-Only-Look-Once (YOLOv5n-L) approach. *Expert Systems With Applications*, 213, 119108. <https://doi.org/10.1016/j.eswa.2022.119108>

- [14] Rajamohanan, R., & Latha, B. C. (2023). An Optimized YOLO v5 Model for Tomato Leaf Disease Classification with Field Dataset. *Engineering Technology & Applied Science Research*, 13(6), 12033–12038. <https://doi.org/10.48084/etasr.6377>
- [15] Sozzi, M., Cantalamessa, S., Cogato, A., Kayad, A., & Marinello, F. (2022). Automatic bunch detection in white grape varieties using YOLOV3, YOLOV4, and YOLOV5 deep learning algorithms. *Agronomy*, 12(2), 319. <https://doi.org/10.3390/agronomy12020319>
- [16] Ultralytics. (n.d.-a). GitHub - ultralytics/ultralytics: NEW - YOLOv8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite. GitHub. <https://github.com/ultralytics/ultralytics>
- [17] Torres, J. (2024a, September 2). What is New in YOLOv8; Deep Dive into its Innovations-Yolov8. YOLOv8. [https://yolov8.org/what-is-new-in-yolov8/#google\\_vignette](https://yolov8.org/what-is-new-in-yolov8/#google_vignette)
- [18] Yadav, P. K., Thomasson, J. A., Searcy, S. W., Hardin, R. G., Braga-Neto, U., Popescu, S. C., Martin, D. E., Rodriguez, R., Meza, K., Enciso, J., Diaz, J. S., & Wang, T. (2022). Assessing the performance of YOLOv5 algorithm for detecting volunteer cotton plants in corn fields at three different growth stages. *Artificial Intelligence in Agriculture*, 6, 292–303. <https://doi.org/10.1016/j.aiia.2022.11.005>
- [19] Sohan, M., Ram, T. S., & Reddy, C. V. R. (2024). A review on YOLOV8 and its advancements. *Algorithms for Intelligent Systems*, 529–545. [https://doi.org/10.1007/978-981-99-7962-2\\_39](https://doi.org/10.1007/978-981-99-7962-2_39)
- [20] Montaluisa-Mantilla, M. S., García-Encina, P., Lebrero, R., & Muñoz, R. (2023). Botanical filters for the abatement of indoor air pollutants. *Chemosphere*, 345, 140483. <https://doi.org/10.1016/j.chemosphere.2023.140483>



## APPENDIX



### Test

Preview your model

#### Settings

Image Size

320px

640px

Confidence Threshold

0.85

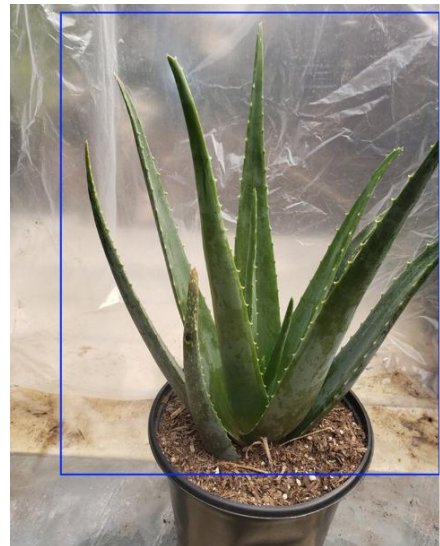
IoU Threshold

0.45

#### Inference results

Aloe Vera

90.6%





## Test

Preview your model

### Settings

Image Size

320px

640px

Confidence Threshold

0.85

IoU Threshold

0.45

### Inference results

Chrysanthemum

89.9%



Confidence Threshold

0.82

IoU Threshold

0.52

### Inference results

Areca Palm

82.6%



Peace Lily

82.5%



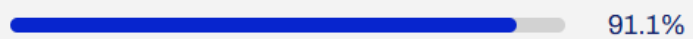
Rhaps Palm

91.2%





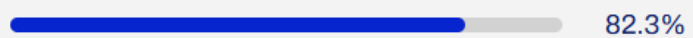
Snake Plant

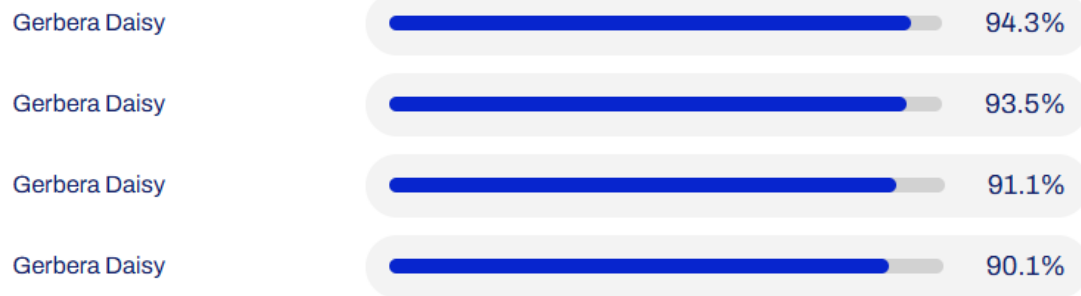


Snake Plant



Snake Plant






Spider Plant



# POSTER

**FACULTY OF INFORMATION COMMUNICATION & TECHNOLOGY**

**UTAR**  
UNIVERSITI TUNKU ABDUL RAHMAN

**OASIS: A COMPUTER VISION  
APPROACH TO SELF-WATERING  
SYSTEM FOR GREEN AIR PURIFIER**

**PROJECT DEVELOPER: TAN WEI JUN 2103410**  
**PROJECT SUPERVISOR: DR AUN YICHIE**

**01 Introduction**

Leveraging computer vision to automate plant care, enhancing both air quality and promoting green air purifiers.

**02 Objective**

To create an intelligent self-watering system that uses computer vision to monitor and manage plant care, ensuring optimal air purifier performance

**03 Proposed Method**

- Real-time object detection of indoor plants
- Automated monitoring via camera system

**04 Advantages**

Implements effective automated monitoring to enhance plant health

**05 Conclusion**

A scalable and integrable computer system through optimized plant care environment

