**ACKNOWLEDGEMENT**

We thank the almighty for giving us the courage and perseverance in completing the main project. This project itself is acknowledgements for all those people who have gave us their heartfelt co-operation in making this project a grand success.

We extend our sincere thanks to **Prof.** **Dr.** **A.** **KANAGARAJ** **PhD,** Chairman of our college, for providing sufficient infrastructure and good environment in the College to complete our course.

We are thankful to our secretary **Mrs.** **K.** **VIJAYAKUMARI** **M.A.,** **B.** **Ed,** for providing the necessary Infrastructure and labs and also permitting to carry out this project.

We are thankful to our principal **Dr.V.SURESHKUMAR** for providing the necessary infrastructure and labs and also permitting to carry out this project.

We are greatly indebted to project guide **Dr.T.MENAKA** Assistant professor**,** Electronics and Communication engineering, for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by him.

Our special thanks to all the faculty of Electronics and Communication Engineering and peers for their valuable advises at every stage of this work.

## ABSTRACT

In the era of rapid urbanization and climate change, sustainable and automated agriculture is crucial. Hydroponics is a promising technique that allows soil-less cultivation of plants using nutrient-rich water. However, maintaining optimal growth conditions and preventing disease in plants requires constant monitoring.

This project proposes an AI-based smart hydroponics system that uses Convolutional Neural Networks (CNN) to detect plant diseases from leaf images. A camera module connected to a Raspberry Pi 3B+ captures plant images daily and classifies them using a trained CNN model. If a disease is detected, alerts are generated for necessary action.

In addition, environmental parameters such as temperature, humidity, and pH level are continuously monitored using DHT11 and pH sensors. The Arduino Uno controls actuators like water pumps, pH up/down valves, NPK solution pumps, and a fan using relay modules to automatically adjust the conditions. A small solar panel is included to partially power the system, promoting green energy usage.

This intelligent setup minimizes the need for human intervention, ensures precise nutrient delivery, and promotes healthy crop growth, making it an ideal solution for smart and sustainable agriculture.

# LIST OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER** **NO.** | **TITLE** | **PAGE** **NO.** |
|  | **ABSTRACT** | IV |
| **LIST** **OF** **CONTENTS** | V |
| **LIST** **OF** **FIGURES**  **LIST OF SYMBOLS**  **LIST OF ABBREVIATIONS** | X  XII  XIII |
| **1.** | **INTRODUCTION** | 1 |
|  | 1.1 OBJECTIVES | 2 |
|  | 1.2 SYSTEM OVERVIEWS | 2 |
| **2.** | **LITERATURE** **SURVEY** | 3 |
| **3.** | **SYSTEM** **ANALYSIS** | 6 |
|  | 3.1 EXISTING SYSTEM | 6 |
|  | 3.1.1 WORKING PRINCIPLE OF EXISTING SYSTEM | 6 |
|  | 3.1.2 DRAWBACKS | 7 |
|  | 3.2 PROPOSED SYSTEMS | 9 |
|  | 3.2.1 HARDWARE REQUIREMENTS | 9 |
|  | 3.2.2 SOFTWARE REQUIREMENTS | 10 |
|  | 3.2.3 BLOCK DIAGRAM | 10 |
| **4.** | **HARDWARE** **DESCRIPTION** | 14 |
|  | 4.1 POWER SUPPLY | 14 |
|  | 4.1.1 GENERAL DESCRIPTION | 14 |
|  | 4.1.2 PRODUCT DESCRIPTION | 14 |
|  | 4.1.3 FEATURES | 15 |
|  | 4.2 MICROCONTROLLER | 16 |

|  |  |  |
| --- | --- | --- |
|  | 4.2.1 ARDUINO UNO | 16 |
| 4.2.2 GENERAL DESCRIPTION | 17 |
| 4.2.3 SPECIFICATIONS OF ARDUINO | 17 |
| 4.3 MICROPROCESSOR | 18 |
| 4.3.1 RASPBERRY PI 3B+ | 18 |
| 4.3.2 GENERAL DESCRIPTION | 19 |
| 4.3.3 SPECIFICATIONS OF RASPBERRY PI 3B+ | 19 |
| 4.4 DHT 11 SENSOR | 20 |
| 4.4.1 GENERAL DESCRIPTION | 20 |
| 4.4.2 PRODUCT DESCRIPTION | 20 |
| 4.4.3 FEATURES | 21 |
| 4.5 PH SENSOR | 21 |
| 4.5.1 GENERAL DESCRIPTION | 22 |
| 4.5.2 PRODUCT DESCRIPTION | 22 |
| 4.5.3 FEATURES | 22 |
| 4.6 CAMERA MODULE | 23 |
| 4.6.1 GENERAL DESCRIPTION | 24 |
| 4.6.2 PRODUCT DESCRIPTION | 24 |
| 4.6.3 FEATURES | 24 |
| 4.7 RELAY MODULE | 25 |
| 4.7.1 GENERAL DESCRIPTION | 26 |
| 4.7.2 PRODUCT DESCRIPTION | 26 |
|  | 4.7.3 FEATURES  4.8 WATERPUMP AND FAN  4.8.1 GENERAL DESCRIPTION  4.8.2 PRODUCT DESCRIPTION  4.8.3 FEATURES  4.9 SOLAR PANEL  4.9.1 GENERAL DESCRIPTION  4.9.2 PRODUCT DESCRIPTION  4.9.3 FEATURES | 26  27  28  28  28  29  29  29  30 |

|  |  |  |
| --- | --- | --- |
| **5.** | **SOFTWARE** **DESCRIPTION** | 31 |
|  | 5.1 PYTHON AND TENSORFLOW | 31 |
|  | 5.1.1 INTRODUCTION TO PYTHON AND TENSORFLOW | 31 |
|  | 5.1.2 DATASET PREPROCESSING | 32 |
|  | 5.1.3 CNN MODEL TRAINING | 33 |
|  | 5.1.4 MODEL CONVERSION AND DEPLOYMENT | 35 |
|  | 5.1.5 LIBRARIES USED | 36 |
|  | 5.2 CNN MODEL FOR PLANT DISEASE DETECTION | 38 |
|  | 5.2.1 CNN ARCHITECTURE OVERVIEW | 38 |
|  | 5.2.2 TRAINING DETAILS | 40 |
|  | 5.2.3 MODEL PERFORMANCE | 40 |
|  | 5.2.4 MODEL OPTIMIZATION  5.3 RASPBERRYPI PROGRAMMING  5.3.1 SYSTEM SETUP & OS  INSTALLATION  5.3.2 CAMERA MODULE INTERFACING  5.3.3 MODEL INFERENCE USING TFLITE  5.3.4 ACTUATOR CONTROL USING  GPIO/ SERIAL PIN  5.3.5 AUTOMATION &SCRIPT  EXECUTION  5.4 ARDUINO PROGRAMMING  5.4.1 SENSOR INTERFACING & DATA  READING  5.4.2 ACTUATOR CONTROL VIA RELAY  MODULE  5.4.3 CODE STRUCTURE AND ARDUINO  CONTROL WORKFLOW      5.4.4 ARDUINO SIMULATION USING  PROTEUS | 42  43  43  44  45  46  46  47  50  52  54  55 |
|  | 5.5 SOFTWARE FLOW DIAGRAM  5.5.1 OVERVIEW  5.5.2 RASPBERRY PI FLOW  5.5.3 ARDUINO FLOW | 59  59  60  61 |
|  | 5.6 SUMMARY | 62 |
|  |  |  |
|  |  |  |
| **6.** | **SMART CONTROL FLOW** | 63 |
|  | 6.1 SMART CONTROL FLOW | 63 |
|  | 6.2 CHANNEL FREQUENCIES | 63 |
|  | 6.3 ACTUATOR RESPONSE BASED ON  CONDITIONS | 64 |
|  | 6.4 ENERGY OPTIMIZATION USING SOLAR  POWER | 64 |
|  | 6.5 APPLICATIONS | 65 |
|  |  |  |
| **7.** | **CONCLUSION**  7.1 FUTURE ENHANCEMENT | 66  67 |
| **8.** | **REFERENCES** | 68 |

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| FIGURE NO | TITLE | PAGE NO |
| 1 | EXISTING BLOCK DIAGRAM | 8 |
| 2(a) | PROPOSED BLOCK DIAGRAM | 10 |
| 2(b) | PROPOSED CIRCUIT DIAGRAM | 13 |
| 3 | ADAPTER | 15 |
| 4 | ARDUINO UNO BOARD | 16 |
| 5 | RASPBERRY PI 3B+ | 18 |
| 6 | DHT 11 SENSOR | 20 |
| 7 | PH SENSOR | 21 |
| 8 | PI CAMERA | 23 |
| 9 | RELAY | 25 |
| 10 | WATER PUMP & FAN | 27 |
| 11 | SOLAR PANEL | 29 |
| 12 | MODEL PERFORMANCE | 41 |
| 13 | SYSTEM SETUP & OS INSTALLATION | 43 |
| 14(a) | ARDUINO PROGRAMMING | 47 |
| 14(b) | COMPONENTS TO INSTALL | 48 |
| 14© | DESTINATION FOLDER | 49 |
| 14(d) | EXTRACTION | 49 |
| 15 | ARDUINO SIMULATION USING PROTEUS | 58 |

|  |  |  |
| --- | --- | --- |
| **SYMBOL** | **DESCRIPTION** | **UNIT** |
| PH | |  | | --- | |  |   Power of hydrogen   |  | | --- | | Power of hydrogen | | -  (unit less) |
| T | Temperature | (°C) |
| H | |  | | --- | |  |  |  | | --- | | Humidity | | % |
| N | Nitrogen concentration | |  | | --- | |  |  |  | | --- | | mg/L | |
| P | Phosphorus concentration | |  | | --- | |  |  |  | | --- | | mg/L | |
| K | Potassium concentration | |  | | --- | |  |  |  | | --- | | mg/L | |
| VCC | Supply voltage | V |
| I | Current | A |
| ACC | Accuracy | % |

**LIST OF SYMBOLS**

|  |  |  |
| --- | --- | --- |
| S.NO | ABBREVIATIONS | FULLFORM |
| 1 | AI | Artificial Intelligence |
| 2 | CNN | Convolutional Neural Network |
| 3 | IOT | Internet of Things |
| 4 | LCD | Liquid Crystal Display |
| 5 | DHT | Digital Humidity and Temperature Sensor |
| 6 | TF LITE | |  | | --- | |  |  |  | | --- | | TensorFlow Lite | |
| 7 | NPK | |  | | --- | |  |  |  | | --- | | Nitrogen, Phosphorus, Potassium | |
| 8 | ML | Machine Learning |
| 9 | API | Application Programming Interface |

**LIST OF ABBREVIATIONS**

**CHAPTER** **1**

**INTRODUCTION**

1. **INTRODUCTION**

Agriculture is the backbone of many economies, and with the growing need for food and sustainable farming methods, smart agriculture is gaining importance. One such innovative approach is hydroponics — a soil-less cultivation method that uses nutrient-rich water to grow plants. Although hydroponics offers benefits such as faster plant growth, reduced water usage, and space efficiency, maintaining plant health and nutrient balance requires constant monitoring and timely intervention.

In traditional systems, farmers manually check the health of plants and adjust temperature, humidity, and nutrient levels. This process is time-consuming and prone to error. To overcome these challenges, this project proposes a **smart automated hydroponics system integrated with AI-based plant disease detection** and sensor-based environmental control.

A **Convolutional Neural Network (CNN)** is used to detect diseases in plant leaves through image processing. The model is trained using the PlantVillage dataset and deployed on a **Raspberry Pi 3B+**, which captures images through a camera module and performs disease classification.

Environmental factors like **temperature, humidity, and pH** are monitored using **DHT11 and pH sensors**. Based on sensor values and AI outputs, the system automatically activates actuators such as water pumps, NPK solution valves, pH control valves, and cooling fans through an **Arduino Uno** connected to relays.. A **solar panel** powers selected components to promote energy efficiency.

This smart hydroponic system aims to reduce human effort, enhance precision farming, and promote healthy crop yield with minimal resource usage. It also aligns with the goal of using technology for smart and sustainable agriculture.

**1.1 OBJECTIVES**

* To design a smart hydroponic system that automates plant care using IoT and AI.
* To detect plant diseases in early stages using a Convolutional Neural Network (CNN) model.
* To monitor environmental parameters such as temperature, humidity, and pH in real time.
* To automate the actuation of fan, water pump, pH control valves, and nutrient supply based on sensor data.
* To deploy the trained CNN model on Raspberry Pi and integrate it with Arduino for system-wide control.
* To reduce manual labor, save resources, and promote precision agriculture using technology.
* To partially power the system using solar energy for sustainability.

**1.2 SYSTEM OVERVIEW**

The proposed system uses a **Raspberry Pi 3B+** with a camera module to capture images of plant leaves and analyze them using a **TensorFlow Lite CNN model** for disease detection. Environmental parameters like **temperature, humidity**, and **pH** are constantly monitored using **DHT11** and **pH sensors**.

An **Arduino Uno** acts as the control unit for actuators, such as the **water pump**, **pH up/down valves**, **NPK nutrient delivery**, and **cooling fan**, which are controlled via a **relay module**. A **16x2 LCD with I2C** is used to display sensor data and alerts. Communication between Raspberry Pi and Arduino is established using **serial or GPIO** interfaces.

Additionally, a **small solar panel** is used to power selected components, reducing energy consumption and making the system eco-friendly. This smart hydroponics setup enables precise plant care with minimal human intervention

**CHAPTER2** **LITERATURE** **SURVEY**

**Literature Survey**

Hydroponics, a method of growing plants in a soil-less environment with a nutrient-rich water solution, has been practiced in various forms for centuries. Though its roots can be traced to the ancient Babylonian Hanging Gardens, the modern hydroponic systems emerged during the 20th century. The growing global concerns over soil degradation, water scarcity, and the need for sustainable agricultural practices have elevated hydroponics as a promising solution. This method offers control over plant growth parameters, providing advantages like faster growth, higher yield, and efficient resource utilization.

As technology advanced, so did hydroponic practices. Today, environmental sensors that monitor parameters like temperature, humidity, pH levels, and nutrient concentrations are incorporated into hydroponic systems. This helps optimize growing conditions, improve plant productivity, and minimize resource waste. Automation tools now allow real-time adjustments to be made based on the data collected from these sensors, making hydroponic systems more scalable and efficient, especially in urban settings (Jones, 2005).

In the field of plant disease detection, various systems have been developed that utilize imaging technologies and machine learning algorithms to detect plant diseases in hydroponic setups. Early identification of plant diseases is critical, as diseases can spread quickly in confined spaces, reducing crop yield. Many systems today use deep learning techniques, particularly Convolutional Neural Networks (CNNs), to detect diseases from images of plant leaves.

One such system, developed by **Gowtham V et al. (2024)**, uses a CNN-based approach for detecting plant diseases in hydroponic systems. Their system, which leverages Raspberry Pi for real-time monitoring and cloud platforms like ThingSpeak, utilizes the PlantVillage dataset for training. Despite its success in detecting diseases, the system's dependence on cloud services and lack ofintegration with environmental sensors is a significant limitation.

Another study by **Smith et al. (2023)** integrates environmental data such as temperature, humidity, and pH levels to improve disease prediction accuracy in hydroponic farming. Their system, while effective in predicting diseases, still lacks a fully automated response system, requiring manual intervention to adjust environmental conditions or treat the plants after disease detection.

**Tan and Lee (2022)** proposed a system for nutrient management optimization in hydroponics, using real-time sensor data to control nutrient flow. Although their work significantly enhanced plant growth, it did not incorporate disease detection capabilities, revealing a gap in the integration of nutrient and disease management in existing systems.

**Similarly, Patel et al. (2021)** focused on developing a machine learning model to predict diseases in hydroponic setups, using data from environmental sensors. Although their system showed promise in disease detection, it also lacked an automated response mechanism, highlighting the need for fully automated systems capable of responding to detected diseases without manual intervention.

Despite these advancements, there are significant gaps in current hydroponic systems. Many systems either focus solely on disease detection or environmental control but rarely address both simultaneously. Furthermore, the reliance on cloud-based services limits their use in areas with unreliable internet access. These gaps emphasize the need for an integrated system that not only detects diseases but also automatically adjusts environmental conditions and treatment measures.

In conclusion, while numerous systems have been developed to detect diseases in hydroponics, there remains a significant gap in creating fully automated systems that respond to detected diseases. The integration of environmental sensors and disease detection could provide more comprehensive monitoring, leading to more efficient and automated systems. Furthermore, the reliance on internet-based cloud services in many existing systems limits their use in remote areas, where connectivity may be unreliable. These gaps highlight the potential for developing a more integrated, automated, and offline-capable hydroponic disease detection system that can manage both disease and environmental factors efficiently.

These studies demonstrate the growing importance of integrating AI-driven solutions in hydroponic systems, enabling real-time monitoring, efficient disease management, and enhanced productivity

Aminu Musa et al. (2022) proposed an intelligent plant disease detection system using CNN for smart hydroponic setups. Their model achieved 98% accuracy by using a comprehensive dataset containing 38 different plant diseases [[1]](https://www.researchgate.net/publication/358281665). The system also integrated IoT sensors for environmental monitoring, highlighting the potential for a fully automated hydroponic farm.

Similarly, Archana Bhamare et al. (2024) developed an optimized AI model to predict abnormal lettuce leaf growth in hydroponic environments. Their work focused on the early detection of plant anomalies to improve crop yield and reduce resource wastage [[2]](https://www.researchgate.net/publication/388129867).

In addition to disease detection, low-power solutions have been explored to reduce the operational cost of smart hydroponic systems. A study by Aminu Musa et al. (2022) introduced knowledge distillation techniques to develop a low-power deep learning model for plant disease detection, which is suitable for resource-constrained environments [3].

These studies demonstrate the growing importance of integrating AI-driven solutions in hydroponic systems, enabling real-time monitoring, efficient disease management, and enhanced productivity.

**CHAPTER** **3**

**SYSTEM** **ANALYSIS**

* 1. **EXISTING** **SYSTEM**

In the current research landscape, various systems have been developed to identify plant diseases using image processing and machine learning techniques. Traditional methods heavily relied on manual inspection, which is time-consuming and prone to human error. With the advancement of deep learning, Convolutional Neural Networks (CNNs) have become a popular approach due to their ability to extract features from leaf images and classify diseases with high accuracy.

Previous systems, such as the one proposed by Gowtham V et al. (2024), focused on hydroponic farming and used CNNs to detect plant diseases from leaf images. Their system used Raspberry Pi and ThingSpeak for real-time monitoring and cloud-based result display. They employed transfer learning and trained models using the PlantVillage dataset, achieving good results in real-time disease detection and system automation. However, their model required internet access for cloud operations and relied primarily on visual disease symptoms, with limited integration of environmental sensor data for decision-making.

Another limitation in many existing systems is the lack of closed-loop automation. While image analysis and alerts are well-handled, the actual physical response—like adjusting nutrient flow, temperature, or humidity—is often manual or semi-automated. Moreover, many setups do not provide detailed remedies or dynamic environmental control based on detected diseases.

**3.1.1 Working Principle of Existing System**

The existing system uses a camera connected to a Raspberry Pi to capture images of plant leaves in a hydroponic setup. These images are passed to a pre-trained Convolutional Neural Network (CNN) model that identifies whether the plant is healthy or affected by a disease. The CNN performs feature extraction from the images and classifies the disease category based on trained patterns.

To support real-time monitoring, the system uses cloud platforms like **ThingSpeak**, where the disease prediction results are uploaded and displayed. It also collects environmental data (like temperature, humidity, etc.) through sensors, which helps provide additional context for the detected disease conditions.

The CNN model is trained using the **PlantVillage dataset**, which contains thousands of images of diseased and healthy leaves. Techniques like **image resizing (224x224)** and **normalization** are applied for effective training. The system uses **transfer learning**, allowing it to adapt pre-trained CNN layers to the specific task of plant disease detection.

**3.1.2 Limitations of Existing System**

1. **Cloud Dependency**: Most systems rely on cloud services for data storage and result visualization. This requires a constant internet connection, which may not be available in all agricultural environments.
2. **Limited Automation**: While disease detection is automated, the physical actions (like spraying fertilizer, adjusting pH, or turning on a fan) are often left for the user to perform manually.
3. **No Real-time Local Display**: In some setups, results are only visible through cloud dashboards, limiting on-site usability without internet.
4. **Lack of Remedy Suggestions**: The system detects diseases but doesn’t actively suggest or apply remedies based on the disease type.
5. **Dataset Constraints**: Although CNN is trained using large datasets, the accuracy of prediction may drop if the real-world image conditions differ (like lighting or camera angle).
6. **High Processing Load**: Some systems are not optimized for lightweight edge devices like
7. Raspberry Pi, which may slow down real-time processing or lead to overheating.

**3.1.3 BLOCK DIAGRAM**

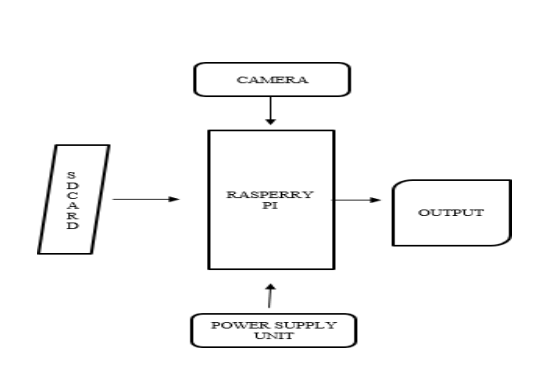


Figure 1 block diagram of Existing system

## 3.2 PROPOSED SYSTEM

The proposed system combines artificial intelligence and IoT to automate plant disease detection and hydroponic nutrient management. A **Raspberry Pi 3B+** is used to run a **TensorFlow Lite CNN model** trained on the **PlantVillage dataset** to detect diseases from real-time images captured by a camera module. The system provides early alerts if a disease is detected.

Environmental parameters like **temperature, humidity**, and **pH level** are continuously monitored using **DHT11** and **pH sensors**. The values are sent to an **Arduino Uno**, which controls actuators such as **water pump**, **NPK solution delivery**, **pH up/down valves**, and **exhaust fan** using a **relay module**. The status is displayed on an **LCD screen**. A **solar panel** is used to partially power the system, making it more energy efficient and eco-friendly.

This setup provides complete automation, improves accuracy in disease detection and nutrient control, and significantly reduces human effort in maintaining the hydroponic system.

**3.2.1 HARDWARE REQUIREMENTS**

* Raspberry Pi 3B+
* Arduino Uno
* Camera Module
* DHT11 Temperature and Humidity Sensor
* pH Sensor
* Relay Module
* Water Pump
* Fan
* Solar Panel

**3.2.2 SOFTWARE REQUIREMENTS**

* Python
* TensorFlow and Keras
* Arduino IDE
* TensorFlow Lite Runtime
* Raspbian OS (on Raspberry Pi)

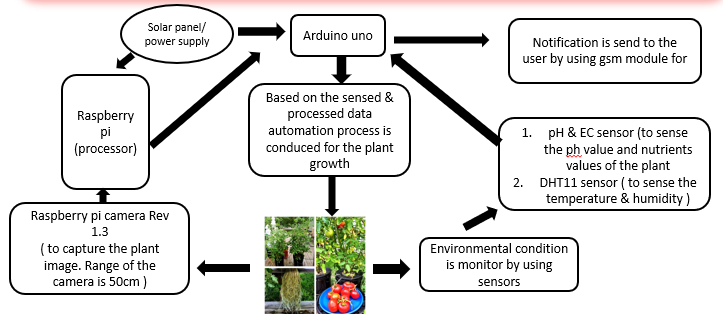
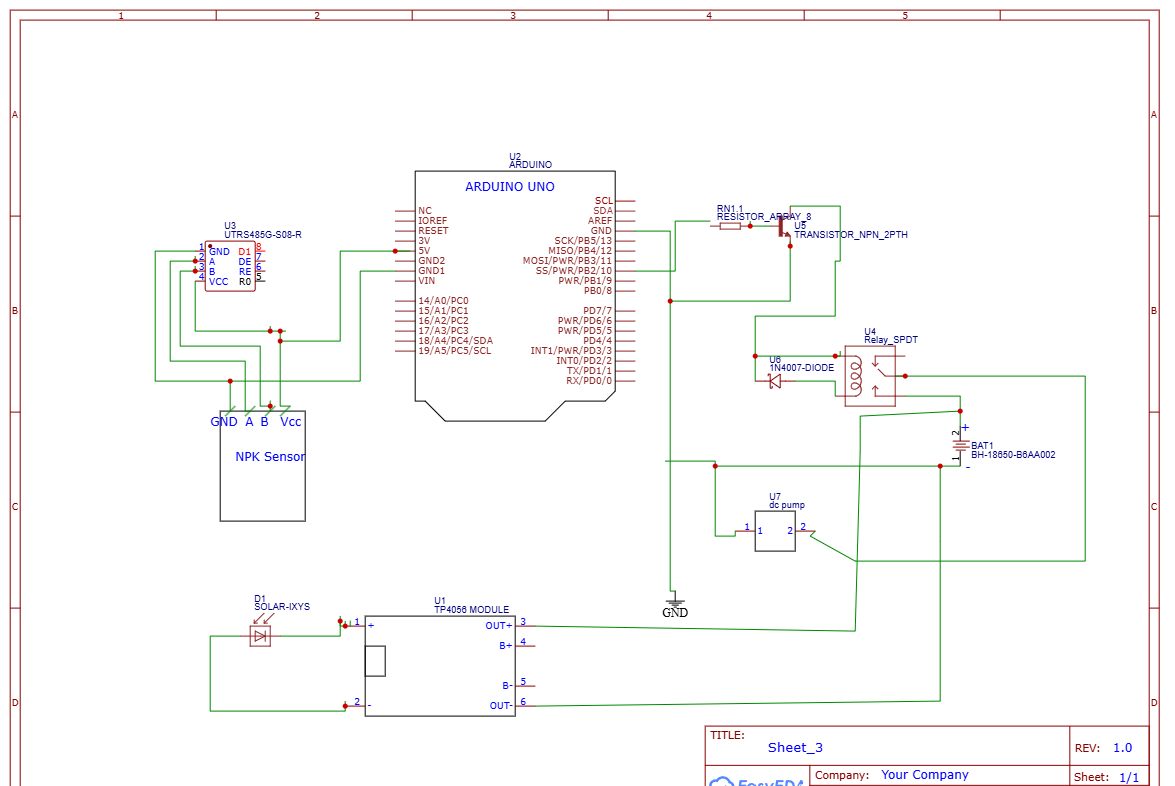
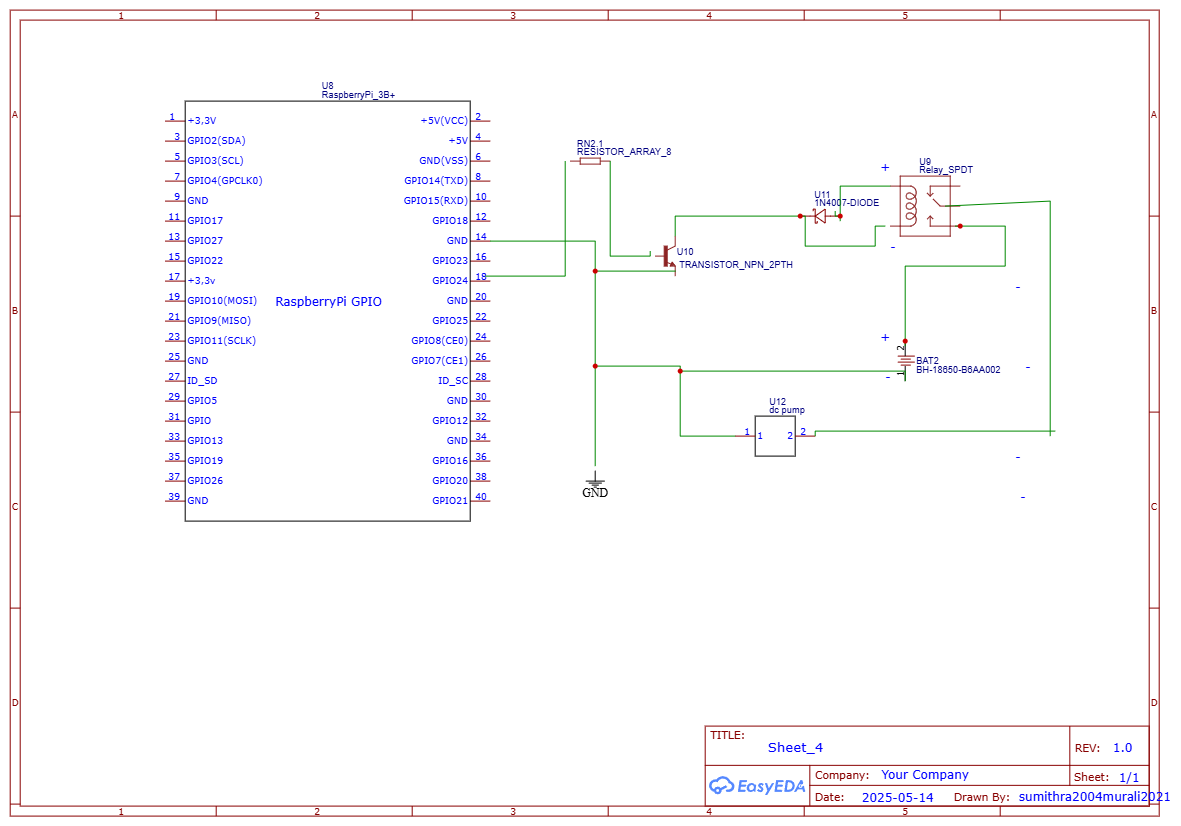
3.2.3 **BLOCK DIAGRAM** **AND** 

Figure 2(a) proposed block diagram

**CIRCUIT DIAGRAM**





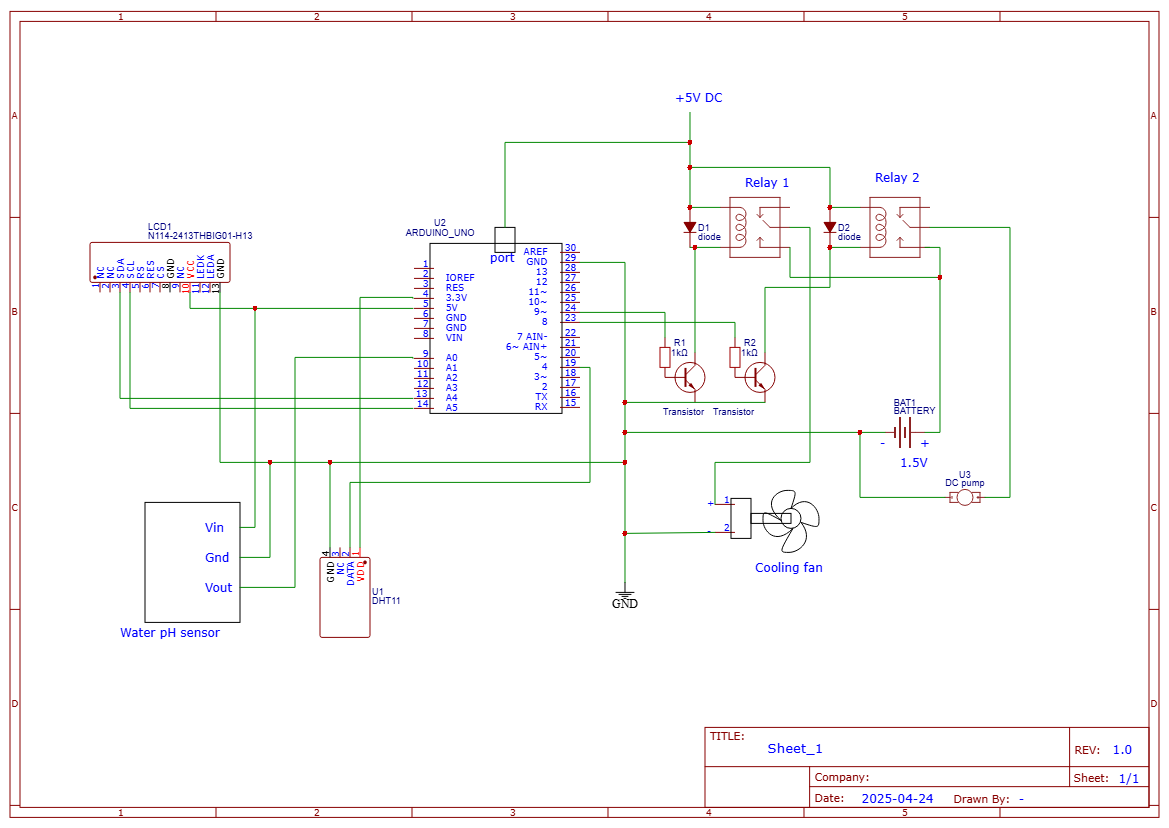


Figure 2 (b) circuit diagram

**CHAPTER** **4**

**HARDWARE** **DESCRIPTIONS**

* 1. **POWER** **SUPPLY**
     1. **GENERAL** **DESCRIPTION**

An adapter is a device that converts attributes of one electrical device or system to those of an otherwise incompatible device or system. Some modify power or signal attributes, while others merely adapt the physical form of one electrical connector to another. In a computer, an adapter is often built into a card that can be inserted into a slot on the computer's motherboard. The card adapts information that is exchanged between the computer's microprocessor and the devices that the card supports.

## PRODUCT DESCRIPTION

An electric power adapter may enable connection of a power plug, sometimes called, used in one region to a AC power socket used in another, by offering connections for the disparate contact arrangements, while not changing the voltage. An AC adapter, also called a "recharger", is a small power supply that changes household electric current from distribution voltage) to low voltage DC suitable for consumer electronics. Some modify power or signal attributes, while others merely adapt the physical form of one electrical connector to another. For computers and related items, one kind of serial port adapter enables connections between 25-contact and nine-contactconnectors, but does not affect electrical power- and signalling-related attributes.

 Figure 3 ADAPTER

## FEATURES

* + - * Output current:1A
      * Supply voltage: 220-230VAC
      * Output voltage: 5VDC
      * Reduced costs.
      * Increased value across front-office and back-office functions.
      * Access to current, accurate, and consistent data.
      * It generates adapter metadata as WSDL files with J2CA extension.

## MICROCONTROLLER

Arduino is a great platform for beginners into the World of Microcontrollers and Embedded Systems. With a lot of cheap sensors and modules, you can make several projects either as a hobby or even commercial.As technology advanced, new project ideas and implementations came into play and one particular concept is the Internet of Things or IoT. It is a connected platform, where several “things” or devices are connected over internet for exchange of information.

**4.2.1 ARDUINO UNO**

Arduino Uno is one of the most widely used development boards in the Arduino family. It is based on the **ATmega328P microcontroller** and is known for its simplicity, ease of programming, and rich support for sensor/actuator interfacing.

Arduino Uno is suitable for projects involving real-time data acquisition, automation, and hardware control. It supports serial communication, analog/digital I/O, PWM, and multiple protocols like UART, SPI, and I2C. It can be programmed via the **Arduino IDE** using C/C++.

In this smart hydroponics system, Arduino Uno is used for:

* Reading data from the **DHT11** and **pH sensor**
* Controlling actuators like **water pump**, **NPK solenoid**, **pH control valves**, and **fan** via a **relay module**
* Updating the **LCD** with live values also send to the user via gsm module



Figure 4 Arduino Uno Board

**4**.**2.2 GENERAL DESCRIPTION**

The Arduino Uno operates at 5V and comes with 14 digital I/O pins (of which 6 provide PWM output), 6 analog input pins, and a 16 MHz crystal oscillator. It includes an onboard USB interface, a reset button, and headers for connecting shields and modules. It is ideal for automation and control-based applications due to its real-time response and reliable operation.

The board is programmed using a USB cable and the open-source **Arduino IDE**. Its GPIO pins allow it to interface with various modules like relays, LCDs, motor drivers, and sensors. It is suitable for beginner-friendly experimentation as well as industrial-level prototypes.

**4.2.3 SPECIFICATIONS OF ARDUINO UNO**

* Microcontroller: ATmega328P
* Operating Voltage: 5V
* Input Voltage (recommended): 7–12V
* Input Voltage (limits): 6–20V
* Digital I/O Pins: 14 (6 PWM outputs)
* Analog Input Pins: 6
* Flash Memory: 32 KB (0.5 KB used by bootloader)
* SRAM: 2 KB
* EEPROM: 1 KB
* Clock Speed: 16 MHz
* USB Interface: Type-B USB port
* Communication Protocols: UART, I2C, SPI
* Dimensions: 68.6 mm × 53.4 mm
* Weight: ~25 grams

**4.3 SINGLE BOARD COMPUTER**

The Raspberry Pi is a low-cost, credit card-sized computer developed by the Raspberry Pi Foundation. It is capable of functioning as a standalone computer, supporting Linux-based operating systems, and is widely used in robotics, IoT, and AI-based applications. Due to its GPIO pins and compatibility with Python, it is often used as a smart controller for real-time monitoring and automation tasks.

**4.3.1 RASPBERRY PI 3B+**

The **Raspberry Pi 3 Model B+** is an upgraded version of Raspberry Pi 3, offering better speed, connectivity, and thermal performance. It features a **64-bit quad-core ARM Cortex-A53 processor** running at **1.4 GHz**, with **1 GB RAM**, integrated **Wi-Fi (802.11 b/g/n/ac)**, **Bluetooth 4.2**, and **Gigabit Ethernet (over USB 2.0)**.

Raspberry Pi 3B+ includes 40 GPIO pins, 4 USB ports, HDMI, and CSI/DSI ports for camera and display connections. It runs on **Raspbian OS** and supports programming in Python, C/C++, Java, and more. In this project, it is used for running the **TensorFlow Lite CNN model** to detect plant diseases and send control commands to the Arduino Uno based on the classification result.

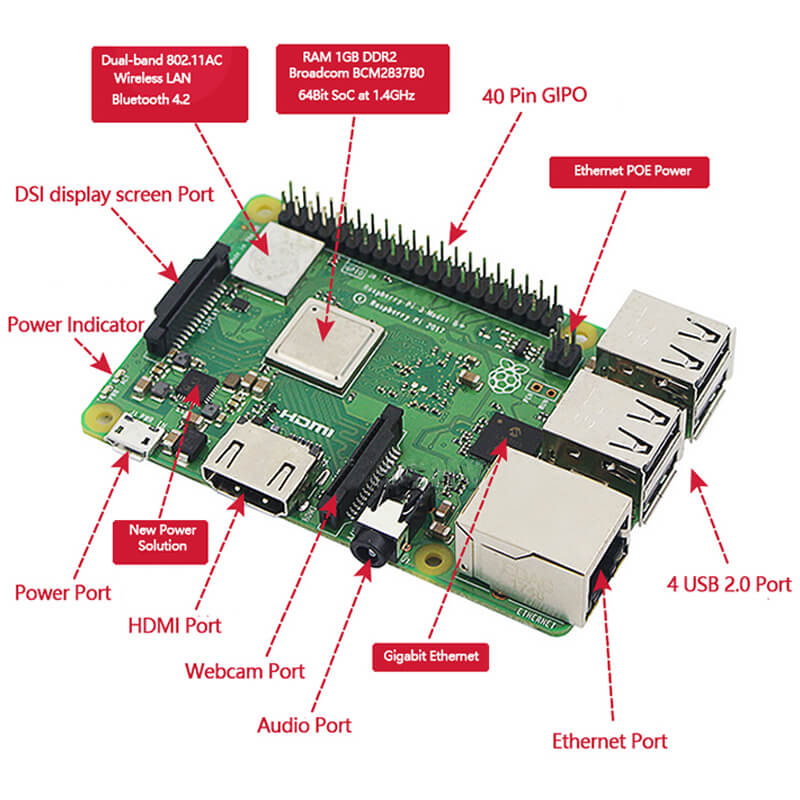


Fig 5 Raspberry pi 3B+ board

**4.3.2 GENERAL DESCRIPTION**

Raspberry Pi 3B+ acts as the brain of the intelligent hydroponics system. It is responsible for:

* Capturing leaf images using the camera module
* Running the CNN-based disease detection model using TensorFlow Lite
* Sending alerts or control signals to Arduino for actuating components
* Logging data and handling image classification logic

Its small size, low power consumption, and compatibility with AI/ML models make it perfect for smart agriculture applications where real-time decision-making is essential.

**4.3.3 SPECIFICATIONS OF RASPBERRY PI 3B+**

* Processor: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
* RAM: 1 GB LPDDR2 SDRAM
* Wi-Fi: 802.11 b/g/n/ac wireless LAN
* Bluetooth: Bluetooth 4.2, BLE
* Ethernet: Gigabit Ethernet over USB 2.0 (max 300 Mbps)
* USB Ports: 4 × USB 2.0
* GPIO: 40-pin header (fully backward-compatible)
* Display Interface: HDMI and MIPI DSI
* Camera Interface: MIPI CSI
* Operating System: Raspbian (Raspberry Pi OS), Linux-based
* Power Input: 5V/2.5A via micro USB
* Storage: microSD card (for OS and data storage)
* Dimensions: 85.6 mm × 56.5 mm
* Weight: ~50 grams

* 1. **DHT11 Temperature and Humidity Sensor**

The DHT11 sensor is a basic, ultra-low-cost digital sensor used to measure temperature and humidity. It is widely used in environmental monitoring, smart farming, weather stations, and indoor automation systems due to its simple design and reliable performance.

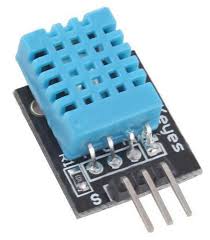


Figure 6 DHT11 sensor

**4.4.1 General Description**

The DHT11 is a basic, low-cost digital sensor used to measure ambient temperature and humidity. It plays a crucial role in monitoring the environmental conditions surrounding the hydroponic plants. Maintaining optimal temperature and humidity is essential for healthy plant growth, especially in an indoor or controlled farming environment. The DHT11 provides real-time environmental data, which is processed by the Arduino or Raspberry Pi to trigger actions like turning ON/OFF a fan, activating a pump, or alerting users.

**4.4.2 Product Description**

The DHT11 sensor consists of a capacitive humidity sensor and a thermistor to measure temperature. It has a built-in ADC (Analog-to-Digital Converter) that outputs digital signals, making it easy to interface with microcontrollers like Arduino. The sensor is factory-calibrated and provides reliable readings with decent accuracy for basic environmental monitoring applications. In this project, it helps in automating climate control inside the hydroponic setup.

**4.4.3 Features:**

* Measures both Temperature and Humidity
* Temperature range: 0 to 50°C with ±2°C accuracy
* Humidity range: 20% to 90% RH with ±5% accuracy
* Operating voltage: 3.3V to 5.5V
* Low power consumption – ideal for long-term monitoring
* Digital output – easy to interface with microcontrollers
* Compact size – easy to mount in small enclosures
* Single wire communication protocol
* Sampling rate: 1Hz (one reading per second)

**4.5 pH Sensor**

The pH sensor is an essential component used to measure the acidity or alkalinity of a solution. In hydroponics, maintaining the correct pH level is crucial because it directly affects nutrient absorption by plants.

****

Figure 7 pHsensor

**4.5.1 General Description:**

The pH sensor is a critical component in hydroponics that measures the acidity or alkalinity of the nutrient solution. For optimal plant growth, the pH level must be maintained within a specific range (usually 5.5 to 6.5 for most hydroponic plants). If the pH level is too high or too low, plants cannot absorb essential nutrients, leading to nutrient deficiencies and poor yield. This sensor continuously monitors the water solution and provides real-time data to the Arduino, which can automatically trigger the addition of pH up or pH down solutions to restore balance.

**4.5.2 Product Description:**

This sensor consists of a pH probe and a signal conditioning circuit. The probe is made of a special glass electrode that produces a small voltage depending on the hydrogen ion concentration in the solution. The signal board converts this analog voltage into a usable digital or analog output that can be interpreted by the Arduino or Raspberry Pi. In this project, the sensor is placed inside the hydroponic tank and communicates with the Arduino to maintain the ideal growing environment.

## 4.5.3 Features:

## Measurement Range: 0 to 14 pH

## Accuracy: ±0.1 pH at 25°C

## Operating Voltage: 5V

## Output Type: Analog voltage signal

## Easy to interface with microcontrollers

## Durable glass probe for continuous water immersion

## Can be calibrated using standard buffer solutions

## Long life and stable performance in hydroponic condition

## Operating voltage: (3.3v-5v) DC

## Operating current: 15mA

## Output: analog (0-5v) DC

the **Raspberry Pi 3B+** and works in coordination with the system to capture images periodically or on-demand. These images are then processed to classify whether the plant is healthy or affected by a disease.

## 4.6.2 PRODUCT DESCRIPTION

This project uses the **Raspberry Pi Camera Module v2**, which is an **8-megapixel** high-definition camera. It is specially designed to work with Raspberry Pi boards via the **CSI (Camera Serial Interface)** port. It uses the **Sony IMX219 sensor**, providing high-quality image capture suitable for both image processing and machine learning tasks.

The camera can be programmed using Python libraries such as **Picamera** or **OpenCV**, and can be triggered to take photos at fixed intervals or in response to sensor data.

**4.6.3 FEATURES**

* **Sensor:** Sony IMX219 8MP sensor
* **Resolution:** 3280 x 2464 pixels
* **Video Support:** 1080p30, 720p60, 640x480p90
* **Interface:** CSI interface, connects directly to the Raspberry Pi
* **Lens:** Fixed focus lens
* **Compact and lightweight**, ideal for embedded applications
* Compatible with **OpenCV** and **TensorFlow Lite** for AI-based image classification
* Low power consumption, suitable for solar-assisted systems

**4.7 RELAY MODULE**

A **relay module** is an electrically operated switch used to control high-voltage devices using a low-voltage microcontroller like an Arduino or Raspberry Pi. It acts as an interface between the low-power control circuit and high-power loads such as water pumps, fans, or lights. The relay works by energizing a coil inside the module, which changes the state of its internal contacts to open or close the circuit. In this project, a **5V single-channel relay** is used to automatically turn ON or OFF the **fan and water pump** based on sensor values like temperature and pH. The module has three main output terminals: **Normally Open (NO)**, **Normally Closed (NC)**, and **Common (COM)**. It also includes an **opto-isolator** for electrical isolation, an LED indicator to show its status, and a transistor driver circuit for switching. This makes it highly reliable, safe, and ideal for automation in hydroponic agriculture systems.



Figure 9 relay (5v Dc)

**4.7.1 GENERAL DESCRIPTION**

A **relay module** is an electromechanical switch that allows a low-power microcontroller like Arduino to control high-power devices such as motors, fans, and water pumps. It works by using a small electrical signal to open or close a separate electrical circuit, allowing automation of AC or DC appliances safely.

**4.7.2 PRODUCT DESCRIPTION**

In this project, a **5V single-channel SPDT (Single Pole Double Throw)** relay module is used. It consists of a relay coil, driver circuit, flyback diode, and an LED indicator. It also includes an **opto-isolator**, which provides electrical isolation between the microcontroller and the high-voltage part of the circuit for protection. The relay has three output terminals: **Normally Open (NO), Normally Closed (NC),** and **Common (COM)**. When triggered, the relay connects COM to NO, allowing current to flow through the load.

This module is mainly used in this project to **automatically control the fan and water pump** based on sensor values like temperature and pH.

**4.7.3 FEATURES**

* Operates on 5V DC input
* Can switch up to 250V AC or 30V DC, 10A current
* Built-in opto-isolator for safe microcontroller connection
* LED indicator shows relay status (ON/OFF)
* Easy interface with Arduino or other controllers
* Provides electrical isolation between control and load circuit
* Compact and suitable for automation and smart agriculture projects

**4.8 WATER PUMP AND FAN**

The **water pump and fan** are important components in this smart hydroponic system. The **water pump** is used to circulate nutrient-rich water to the plant roots, ensuring proper hydration and healthy growth. It is a **mini DC submersible pump** that operates at 5V to 12V and is suitable for small-scale applications. The **fan** is used to maintain airflow and control the temperature inside the plant environment. It is a **DC-powered cooling fan**, similar to a CPU fan, and helps prevent overheating, especially during hot weather. Both the water pump and fan are connected to a **relay module**, which turns them ON or OFF automatically based on the sensor data (like temperature and pH). This automation reduces human effort and improves the efficiency of the plant-growing system.

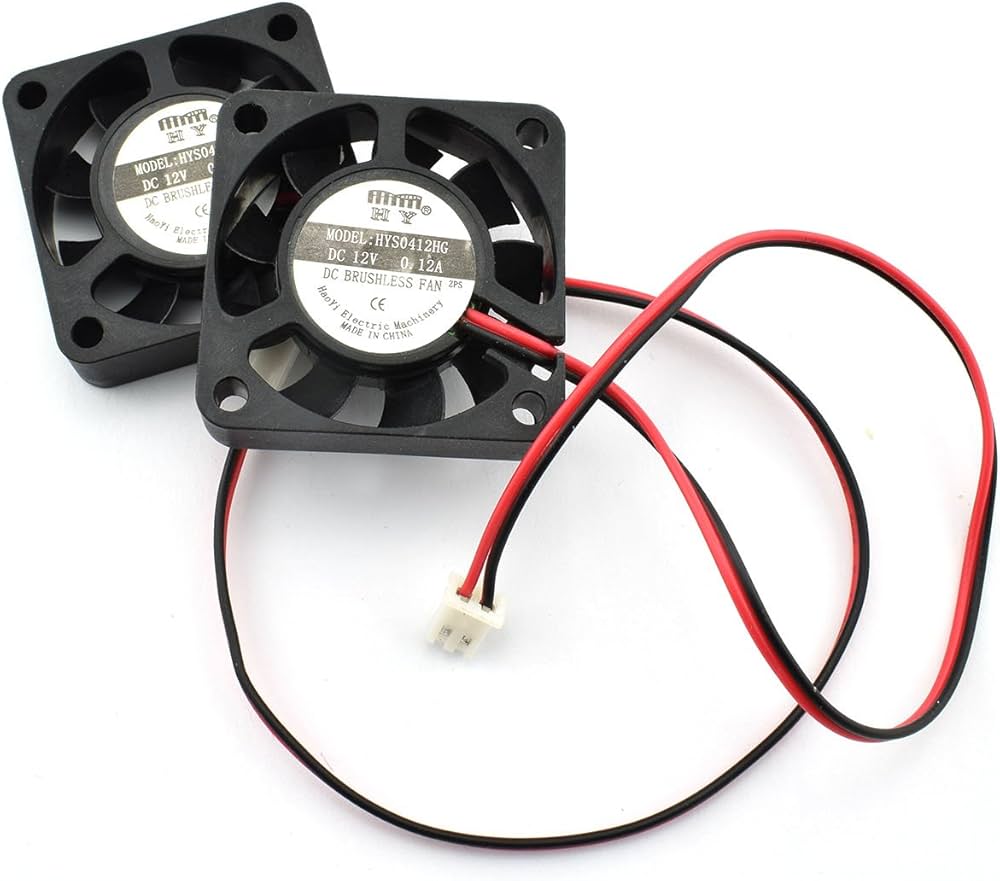
 

Figure 10 water pump and Fan

## 4.8.1 GENERAL DESCRIPTION

## The water pump and fan are essential actuators used in smart agriculture and hydroponic systems. The water pump is used to deliver nutrient-rich water to the plant roots, ensuring proper growth and hydration. The fan is used to maintain proper air circulation and regulate the temperature inside the setup. Both devices are controlled automatically using a relay module based on sensor readings like temperature and pH value

## 4.8.2 PRODUCT DESCRIPTION

## The mini submersible water pump used in this project operates at 5V–12V DC and can be easily powered by a battery or adapter. It is compact, energy-efficient, and suitable for small-scale hydroponic setups. The fan used is a Zebronics 5V/12V DC fan, similar to those used in computer cooling systems. It helps maintain optimal airflow and reduces excessive heat inside the growing area. Both the pump and fan are connected to relay modules, allowing automatic ON/OFF control using the Arduino board based on real-time environmental conditions.

## 4.8.3 FEATURES

## Water Pump:

## Operates on 5V to 12V DC

## Compact and lightweight

## Suitable for continuous water circulation

## Low power consumption

## Easy to install and maintain

## Fan:

## Operates on 5V or 12V DC

## Helps maintain ideal temperature for plant growth

## Low noise and efficient performance

## Lightweight and durable

### **4.9 SOLAR PANEL**

The **solar panel** is a renewable energy source that converts sunlight into electrical energy using photovoltaic (PV) cells. In this project, it is used to **partially power low-energy components** like sensors and fans, reducing the dependence on traditional power supply and promoting eco-friendly operation. The panel used is a **6V–12V polycrystalline type**, suitable for small embedded systems. It is compact, lightweight, and can generate sufficient current during daylight to support basic system functions. By using solar energy, the system becomes more energy-efficient and suitable for use in outdoor or remote agricultural environments.

## 

## Figure 11 solar panel

## 4.9.1 GENERAL DESCRIPTION

## A solar panel is a renewable energy device that converts sunlight into electrical energy using photovoltaic (PV) cells. It is an eco-friendly and sustainable power source commonly used in remote or energy-saving systems. In this project, the solar panel helps to partially power small components like sensors or fans, reducing dependency on conventional power sources and promoting green energy usage.

## 4.9.2 PRODUCT DESCRIPTION

## The solar panel used in this project is a small polycrystalline panel rated at 6V–12V with a current output of 100mA to 500mA, depending on sunlight intensity. It is lightweight, portable, and suitable for charging batteries or powering low-power devices. It includes diodes to prevent reverse current, and it can be connected through a charge controller or directly to a regulated circuit to power devices safely. It is ideal for low-power embedded systems like those used in smart farming or hydroponics.

## 4.9.3 FEATURES

## Converts solar energy into electrical energy

## Output Voltage: 6V to 12V (varies with light)

## Lightweight and easy to install

## Environmentally friendly and sustainable

## Suitable for outdoor or semi-outdoor use

## Ideal for powering sensors, fans, or small loads in smart agriculture systems

## CHAPTER 5

## SOFTWARE DESCRIPTION

**5.1 PYTHON AND TENSORFLOW**

In our project, Python and TensorFlow play a central role in implementing the artificial intelligence-based plant disease detection system. Python is a high-level programming language widely adopted for machine learning applications due to its simplicity, vast library ecosystem, and ease of integration with hardware platforms such as the Raspberry Pi. It is used to write the complete code for data preprocessing, model training, image handling, and hardware control.

TensorFlow, developed by Google, is a powerful open-source deep learning framework that provides tools and APIs for building, training, and deploying machine learning models. In this project, TensorFlow is used to develop a Convolutional Neural Network (CNN) that can classify different plant diseases based on leaf images. The trained model is later converted into TensorFlow Lite format for optimized deployment on Raspberry Pi, ensuring fast and efficient prediction in real time.

The combination of Python and TensorFlow offers a robust and scalable platform for AI-based smart farming, allowing disease detection, decision-making, and automation to happen locally on the device, without needing constant internet access or external processing support.

**5.1.1 INTRODUCTION TO PYTHON AND TENSORFLOW**

**Python** is a high-level, general-purpose programming language that has become the cornerstone of modern machine learning and artificial intelligence (AI) applications. It is well-known for its simplicity, readability, and extensive ecosystem of libraries and frameworks, which significantly reduce development time. Python’s ability to interface with numerous platforms, devices, and libraries makes it a preferred language for both academic research and industrial applications. Its broad range of data manipulation and visualization libraries, such as **NumPy**, **Pandas**, **Matplotlib**, and **Scikit-learn**, makes it highly effective for scientific computing and data analysis.

**TensorFlow**, developed by Google, is an open-source deep learning framework that provides comprehensive tools and libraries to build, train, and deploy machine learning models. It has become one of the most widely adopted frameworks for creating neural networks and conducting complex computations in both research and industry. TensorFlow’s flexibility and scalability make it suitable for a wide range of machine learning tasks, from simple regression to more advanced deep learning models. It supports multiple platforms, including mobile, embedded, and edge devices, making it ideal for real-time AI applications.

In this project, **Python** is used to write the complete code for the machine learning pipeline, including data preprocessing, training, and hardware interaction. **TensorFlow** is employed to design and implement a **Convolutional Neural Network (CNN)** that can classify plant leaf images into healthy or diseased categories. The trained model is then converted into the **TensorFlow Lite** format, enabling efficient deployment on edge devices like **Raspberry Pi**. This integration allows real-time disease detection and decision-making, all within a resource-constrained environment.

The synergy between Python and TensorFlow provides a powerful and flexible development environment for building AI-based applications, and it forms the foundation for this plant disease detection system, which demonstrates the potential of AI in precision agriculture.

**5.1.2 DATASET PREPROCESSING**

Dataset preprocessing is a crucial step in any machine learning pipeline. It involves preparing raw data into a clean and suitable format that can be effectively used to train a model. In this project, the **PlantVillage dataset** is used, which contains images of healthy and diseased tomato plant leaves categorized into various classes.

The preprocessing steps include:

* **Image Resizing:** All input images are resized to a uniform dimension of **224x224 pixels** to match the input shape expected by the CNN model.
* **Normalization:** Pixel values are scaled to the range [0, 1] by dividing by 255, This helps in faster and more stable training.
* **Data Augmentation:** Techniques like horizontal/vertical flipping, zooming, rotation, and brightness adjustments are applied to artificially increase the size and variability of the dataset. This helps the model generalize better and prevents overfitting.
* **Label Encoding:** The class labels (e.g., "Tomato\_Late\_blight", "Tomato\_healthy") are converted into numeric format using one-hot encoding for multiclass classification.
* **Train-Test Split:** The dataset is split into training, validation, and testing sets to evaluate the model's performance on unseen data.

These preprocessing steps ensure that the CNN model receives high-quality, balanced data and can learn relevant features effectively. Proper preprocessing directly impacts the model’s accuracy and reliability in detecting plant diseases in real-time.

**5.1.3 CNN MODEL TRAINING**

The **Convolutional Neural Network (CNN)** is a specialized deep learning algorithm highly effective in image classification tasks due to its ability to automatically extract spatial and temporal patterns in image data. In this project, a custom CNN model is developed using **TensorFlow and Keras APIs in Python** to classify tomato plant leaves into various disease categories and healthy leaves.

It is the critical phase in this project as it directly impacts the model's ability to accurately detect and classify plant diseases. CNNs are a class of deep learning models particularly effective for image-based tasks due to their ability to automatically learn spatial hierarchies of features from input images. For this project, the goal is to train the CNN to classify various plant leaf images into different disease categories or as healthy.

The training process starts by defining the CNN architecture, which typically includes:

* **Convolutional layers** that scan the input image using filters to extract important features like edges, textures, or disease spots.
* **Activation functions** like **ReLU (Rectified Linear Unit)** are applied to introduce non-linearity, helping the model to learn complex patterns.
* **Pooling layers** (like MaxPooling2D) reduce the dimensionality of the feature maps, preserving essential features while minimizing computation.
* **Dropout layers** are introduced to reduce overfitting by randomly disabling some neurons during training.
* **Fully connected (dense) layers** are used at the end to make final classification decisions based on the extracted features.

The model is compiled using the **Adam optimizer**, known for fast and adaptive convergence, and the **categorical cross-entropy loss function** since it is a multiclass classification task. The training is performed for multiple **epochs** (iterations over the dataset), and during each epoch, the model adjusts its internal weights to minimize the prediction error.

Performance metrics such as **accuracy**, **precision**, **recall**, and **loss** are monitored using the **validation dataset**. If the model shows signs of overfitting, adjustments such as early stopping or learning rate reduction are applied.

After training, the final model is saved in **.h5 format**, and graphs of accuracy/loss over epochs are plotted to visualize training progress.

**Data Preparation and Augmentation**

Before the model can be trained, the dataset undergoes extensive preprocessing to ensure the images are in a suitable format. The images are resized to a uniform size of **224x224 pixels**, as required by the model. The pixel values are normalized to the range [0, 1] to help speed up the convergence of the model during training. To prevent overfitting and improve the model’s generalization ability, **data augmentation** techniques such as **random rotation**, **flipping**, **zooming**, and **shearing** are applied to artificially expand the dataset.

**Model Architecture**

The architecture of the CNN model typically consists of multiple **convolutional layers**, followed by **max-pooling layers** to extract features from the input images. These layers are followed by **fully connected layers** that are used for the classification task. The model architecture can be designed in several ways, but in this project, the network is composed of several convolutional layers to capture fine-grained features of plant leaves, followed by dense layers that map the features to the final disease categories.

**Training the Model**

The model is trained using a **categorical cross-entropy loss function** for multiclass classification, with the **Adam optimizer** used to adjust the model weights during training. The model is trained on the prepared dataset, with training and validation sets split to monitor the model's performance on unseen data. During training, several performance metrics such as **accuracy**, **precision**, **recall**, and **F1 score** are evaluated to ensure the model is learning effectively. The model is trained for a specified number of epochs, typically ranging from 10 to 50, depending on the dataset and model complexity.

**Model Evaluation**

After training, the model is evaluated on a separate **test set** to assess its ability to generalize to new, unseen data. If the model performs well on the test set, it is considered ready for deployment. The final model is stored and later used for making real-time predictions on the Raspberry Pi system.

By utilizing CNNs for image classification, this project leverages one of the most powerful deep learning techniques for solving complex image recognition tasks such as plant disease detection.

**5.1.4 MODEL CONVERSION AND DEPLOYMENT**

Once the Convolutional Neural Network (CNN) model has been successfully trained and evaluated, the next step is to prepare the model for deployment on an embedded device like the **Raspberry Pi**. Since a standard TensorFlow model may be too large and resource-intensive for such devices, it is essential to **optimize and convert** the model using **TensorFlow Lite (TFLite)**. TensorFlow Lite is specifically designed for running machine learning models on low-power, edge devices.

**Model Conversion**

To deploy the trained CNN model on Raspberry Pi, it is first converted into a .tflite format using TensorFlow’s TFLite Converter. This conversion compresses the model by reducing its size and memory usage, without significantly sacrificing accuracy. The conversion process may also include **quantization**, which further reduces model size by converting floating-point weights to 8-bit integers. This helps the model run faster and with lower power consumption, making it ideal for real-time applications in smart agriculture systems.

**Deployment on Raspberry Pi**

After conversion, the .tflite model is transferred to the Raspberry Pi device. The deployment setup involves using a lightweight TFLite interpreter in Python that allows the Raspberry Pi to load the model and run **inference** on input images captured by a connected **camera module**.

The Raspberry Pi is programmed to:

* Continuously capture real-time plant images.
* Preprocess the images (resize and normalize).
* Pass the images to the deployed model.
* Interpret the output of the model to identify the disease or confirm healthy condition.

Based on the inference result, actions are triggered:

* If the plant is **healthy**, the system takes no action.
* If the plant is **diseased**, the system:
  + Displays the result on the **I2C LCD Display**,
  + Turns on the **water pump and sprayer**,
  + Optionally sends an alert using a **GSM module** (if connected).

**Benefits of Edge Deployment**

Deploying the model on Raspberry Pi has multiple advantages:

* **No need for constant internet connectivity**, as predictions happen locally.
* **Faster decision-making**, ideal for real-time agricultural applications.
* **Energy-efficient operation**, especially when powered by a **solar panel**.

This deployment phase transforms the project from just a machine learning experiment into a **fully functional, automated, AI-based plant monitoring system** that operates independently in real-world conditions.

**5.1.5 LIBRARIES USED**

In this project, various Python libraries have been used to simplify machine learning model development, data preprocessing, deployment, and hardware interfacing. Each library serves a unique purpose in the workflow, from training the CNN model to controlling hardware components connected to the Raspberry Pi. Below is an overview of the key libraries used:

**1. TensorFlow**

* **Purpose**: Deep learning framework used to build, train, and convert the CNN model.
* **Use in Project**:
  + Designing the CNN architecture.
  + Training the model using image data.
  + Converting the model to TensorFlow Lite for deployment.
* **Key Functions**: Sequential(), Conv2D, Dense, compile(), fit(), tflite.TFLiteConverter.

**2. NumPy**

* **Purpose**: Handling numerical operations and array manipulation.
* **Use in Project**:
  + Image data reshaping and normalization.
  + Performing mathematical calculations.
* **Key Functions**: array(), reshape(), astype().

**3. OpenCV (cv2)**

* **Purpose**: Image processing and real-time image capture.
* **Use in Project**:
  + Capturing images from the Raspberry Pi camera.
  + Resizing and preprocessing input images for CNN.
* **Key Functions**: VideoCapture(), resize(), imread(), imshow().

**4. Keras**

* **Purpose**: High-level API for building and training deep learning models (built into TensorFlow).
* **Use in Project**:
  + Simplifying the CNN model definition.
  + Monitoring accuracy and loss during training.
* **Key Functions**: ImageDataGenerator, ModelCheckpoint, EarlyStopping.

**5. Matplotlib**

* **Purpose**: Visualizing training performance and dataset.
* **Use in Project**:
  + Plotting model accuracy and loss.
  + Displaying sample disease images.
* **Key Functions**: plot(), imshow(), xlabel(), ylabel().

**6. TFLite Interpreter (from TensorFlow Lite)**

* **Purpose**: Running the trained model on Raspberry Pi using low resources.
* **Use in Project**:
  + Loading and performing inference on the .tflite model.
* **Key Functions**: Interpreter(), allocate\_tensors(), set\_tensor(), invoke().

**7. RPi.GPIO**

* **Purpose**: Controlling GPIO pins on Raspberry Pi.
* **Use in Project**:
  + Turning on/off the water pump, fan, or sprayer based on disease detection.
* **Key Functions**: setup(), output(), IN, OUT.

**8. smbus**

* **Purpose**: Communication with I2C devices like LCD display.
* **Use in Project**:
  + Sending data to the 16x2 LCD via I2C protocol.
* **Key Functions**: write\_byte\_data(), write\_i2c\_block\_data().

## ****5.2 CNN MODEL FOR PLANT DISEASE DETECTION****

In this project, a Convolutional Neural Network (CNN) is used to automatically detect and classify various plant diseases from leaf images. CNNs are a type of deep learning model particularly effective in image recognition tasks. By learning spatial hierarchies of features from input images, CNNs can distinguish between healthy and diseased leaves with high accuracy, making them ideal for agricultural applications.

**5.2.1 CNN ARCHITECTURE OVERVIEW**

The CNN architecture used in this project is designed to balance accuracy and computational efficiency, as it will be deployed on a Raspberry Pi. The model includes:

* **Input Layer**: Accepts 224x224 pixel RGB images.
* **Convolutional Layers**: Multiple layers with filters (kernels) that extract features such as edges, colors, spots, and textures.
* **Activation Function**: ReLU (Rectified Linear Unit) is applied after each convolution to introduce non-linearity.
* **Pooling Layers**: MaxPooling2D layers reduce the spatial size of the feature maps, helping to reduce computation and prevent overfitting.
* **Flatten Layer**: Converts the 2D feature maps into a 1D vector.
* **Dense Layers**: Fully connected layers interpret the extracted features and perform classification.
* **Output Layer**: Uses Softmax activation for multi-class classification (for example, 10 classes in tomato disease detection).

This structure allows the model to progressively learn important visual features and classify diseases accurately.

CNN ALGORITHM

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense, Dropout

model = Sequential([Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(224, 224, 3)),

BatchNormalization(),

MaxPooling2D(2, 2),

Dropout(0.25),

Conv2D(64, (3, 3), activation='relu', padding='same'),

BatchNormalization(),

MaxPooling2D(2, 2),

Dropout(0.25),

Conv2D(128, (3, 3), activation='relu', padding='same'),

BatchNormalization(),

MaxPooling2D(2, 2),

Dropout(0.3),

Flatten(),

Dense(256, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax') # 10 classes

])

**5.2.2 TRAINING DETAILS**

* **Dataset Used**: The PlantVillage dataset, consisting of thousands of healthy and diseased plant leaf images.
* **Classes**: 10 classes (e.g., Tomato\_Bacterial\_spot, Tomato\_Early\_blight, Tomato\_healthy, etc.)
* **Image Preprocessing**:
  + Resized all images to 224x224.
  + Normalized pixel values to [0, 1].
  + Applied data augmentation (rotation, zoom, shift, flip) to avoid overfitting.
* **Split**: 80% for training, 20% for validation.
* **Epochs**: 25–50 (depending on early stopping).
* **Batch Size**: 32
* **Optimizer**: Adam optimizer.
* **Loss Function**: Categorical Crossentropy.
* **Callbacks**: ModelCheckpoint and EarlyStopping used to save best model and stop when overfitting begins.

**5.2.3 MODEL PERFORMANCE**

After training, the CNN model was evaluated using validation and test data. Key performance metrics include:

* **Accuracy**: Achieved above 95% accuracy on the validation set.
* **Loss**: Training and validation loss remained low, indicating good generalization.
* **Confusion Matrix**: Showed that the model correctly classified most disease types with minimal confusion.
* **Precision and Recall**: High precision and recall scores across all classes indicate that the model is reliable for real-time detection.

Sample Output:

* Tomato\_Leaf\_Mold detected – 97% confidence
* Tomato\_healthy detected – 99% confidence

These results confirm that the CNN model can be confidently used in the field for plant health monitoring.

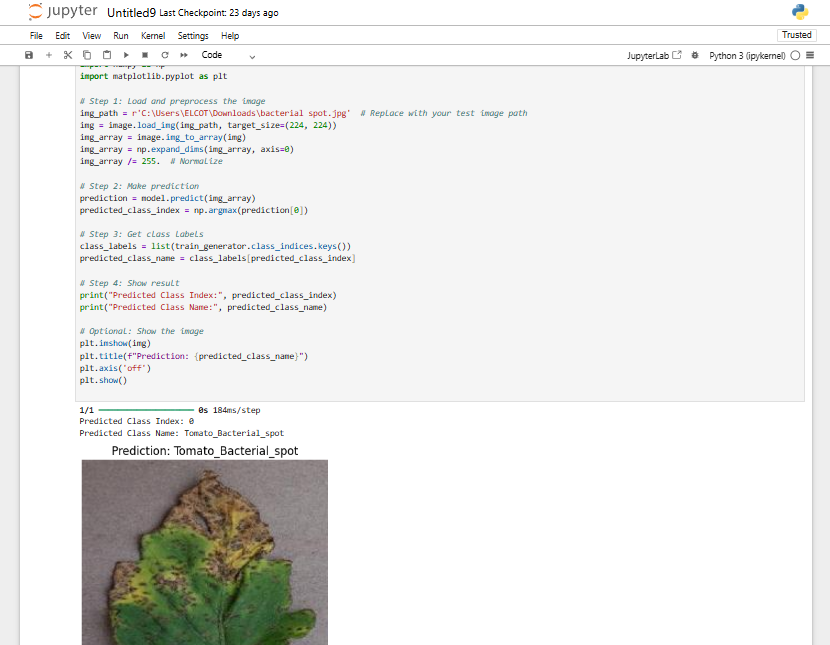


Figure 12 model performance (output of prediction)

**5.2.4 MODEL OPTIMIZATION**

To make the model suitable for deployment on the Raspberry Pi, several optimization techniques were applied:

* **Model Conversion**: The trained model was converted to TensorFlow Lite format (.tflite), which is optimized for mobile and embedded devices.
* **Quantization**: Reduced the model size by converting weights to 8-bit integers, improving speed and memory usage.
* **Pruning (optional)**: Removed unnecessary connections in the model to reduce complexity without affecting accuracy.
* **Lightweight CNN Architecture**: Instead of very deep networks, a compact model like MobileNet or a custom CNN with fewer layers was chosen to ensure real-time inference on edge devices.

These steps ensure the model runs smoothly on Raspberry Pi, even with limited processing power and memory.

**CODE:**

import tensorflow as tf

# Load the model

model = tf.keras.models.load\_model("tomato\_disease\_model.h5")

# Convert the model to TensorFlow Lite

converter = tf.lite.TFLiteConverter.from\_keras\_model(model)

tflite\_model = converter.convert()

# Specify the path to save the model in your Documents folder

save\_path = r"C:\Users\ELCOT\Documents\tomato\_disease\_model.tflite" # Update this path if necessary

# Save the TensorFlow Lite model

with open(save\_path, 'wb') as f:

f.write(tflite\_model)

print(f"Model saved to {save\_path}")

**5.3 RASPBERRY PI PROGRAMMING**

The Raspberry Pi acts as the core processing unit in this project. It runs the AI model for disease detection, controls hardware components like the water pump and fan, and manages automation through GPIO pins. Efficient programming of the Raspberry Pi is essential for real-time performance, reliable detection, and responsive actuator control.

**5.3.1 SYSTEM SETUP AND OS INSTALLATION**

The Raspberry Pi 3B+ is equipped with a Broadcom processor and 1GB RAM, suitable for lightweight AI applications. The system was set up as follows:

* **Operating System**: Raspberry Pi OS (32-bit) was installed using the Raspberry Pi Imager tool.
* **Storage**: A high-speed 32GB microSD card was used to store the OS, Python scripts, and trained models.
* **Initial Setup**:
  + Enabled SSH and VNC for remote access.
  + Configured Wi-Fi and regional settings.
  + Updated the system using sudo apt update && sudo apt upgrade.

Python 3.9 and required libraries were pre-installed or added manually using pip.

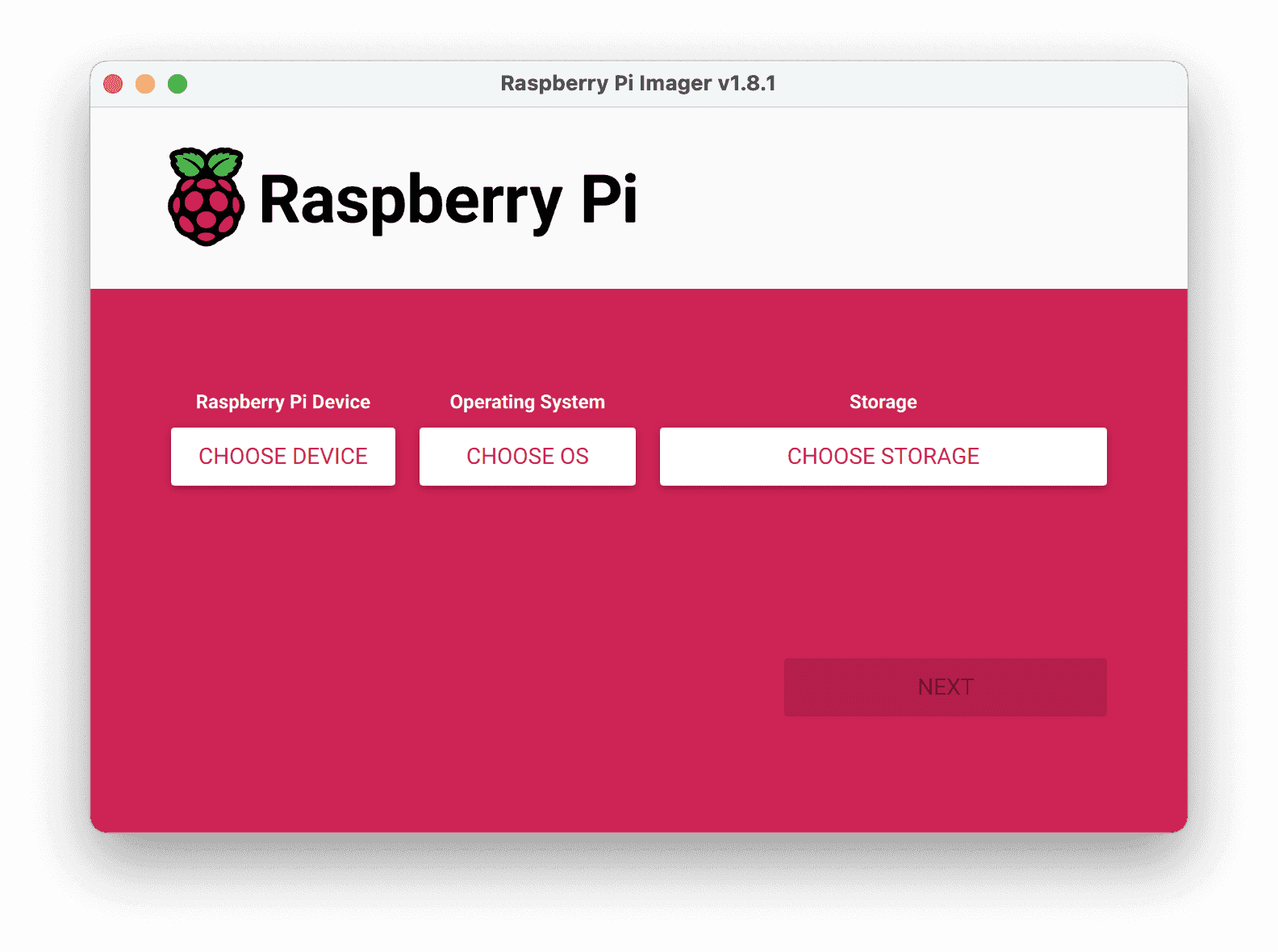


Figure 13 SYSTEM SETUP AND OS INSTALLATION

**5.3.2 CAMERA MODULE INTERFACING**

A Raspberry Pi Camera Module was used to capture real-time images of the plant for disease detection. The steps include:

* **Hardware Connection**: The camera is connected via the CSI (Camera Serial Interface) port.
* **Enable Camera**: Activated using sudo raspi-config > Interface Options > Enable Camera.
* **Python Libraries Used**: cv2 (OpenCV) or picamera for image capture.

are defined by the compiler-specific, named-register, storage class for each supported processor. The processor registers are declared and used like conventional C variables (in many cases volatile variables). Developers using Embedded C can now develop their applications, including direct access to the condition code register and other processor-specific status flags, in a high-level language, instead of inline assembly code. Named address spaces and full processor access reduces application dependency on assembly code and shifts the responsibility for computing data types, array and structure offsets, and all those things that C compilers routinely and easily do from developers to compilers.

Sample code:

**import cv2**

**cam = cv2.VideoCapture(0)**

**ret, frame = cam.read()**

**cv2.imwrite("plant\_image.jpg", frame)**

Captured images are saved and passed to the AI model for inference**.**

**5.3.3 MODEL INFERENCE USING TFLITE**

The trained TensorFlow model was converted into a TensorFlow Lite (.tflite) format for lightweight execution on Raspberry Pi.

* Library Used: tflite-runtime (installed using pip).

Model Loading:

**import tflite\_runtime.interpreter as tflite**

**interpreter = tflite.Interpreter(model\_path="model.tflite")**

**interpreter.allocate\_tensors()**

**Image Processing**:

* Resize image to 224x224
* Normalize to [0,1]
* Convert to NumPy array

**Inference**:

**interpreter.set\_tensor(input\_index, image\_data)**

**interpreter.invoke()**

**output = interpreter.get\_tensor(output\_index)**

The model returns probabilities for each disease class, and the class with the highest probability is displayed.

**5.3.4 ACTUATOR CONTROL USING GPIO/SERIAL PIN**

Once the disease is identified, the system triggers appropriate actions using GPIO pins.

* GPIO Library: RPi.GPIO for controlling components.
* Relay Interfacing:
  + Water pump and fan are connected to relay modules.
  + GPIO pins send HIGH/LOW signals to turn components ON/OFF

**import RPi.GPIO as GPIO**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(17, GPIO.OUT) # Pin connected to relay**

**GPIO.output(17, GPIO.HIGH) # Turn ON pump**

Based on the model output:

* If the plant is healthy: No action needed.
* If diseased: Activate the relay for water pump and solution spraying.

**5.3.5 AUTOMATION AND SCRIPT EXECUTION**

The entire detection and control process is automated using Python scripts.

* Automation Steps:
  + Image captured at a scheduled time (e.g., once per day).
  + Image processed and passed to the model.
  + Output displayed on LCD.
  + If disease is detected, relay is triggered to treat the plant.
* Script Execution:
  + Scripts are set to run automatically on boot using crontab or by placing them in/etc/rc.local.

**Example**

**@reboot python3 /home/pi/plant\_disease\_detector.py**

**5.4 ARDUINO PROGRAMMING (IDE)**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It is widely used for building digital devices and interactive objects that can sense and control the physical world. In this project, the Arduino Uno plays a vital role in reading sensor data, controlling actuators (such as fans and water pumps), and communicating with the Raspberry Pi for decision-making.

The Arduino Integrated Development Environment (IDE) is the primary platform used to write and upload programs (sketches) to the Arduino board. It supports C/C++-based code and provides a simple interface for code editing, compiling, and uploading.

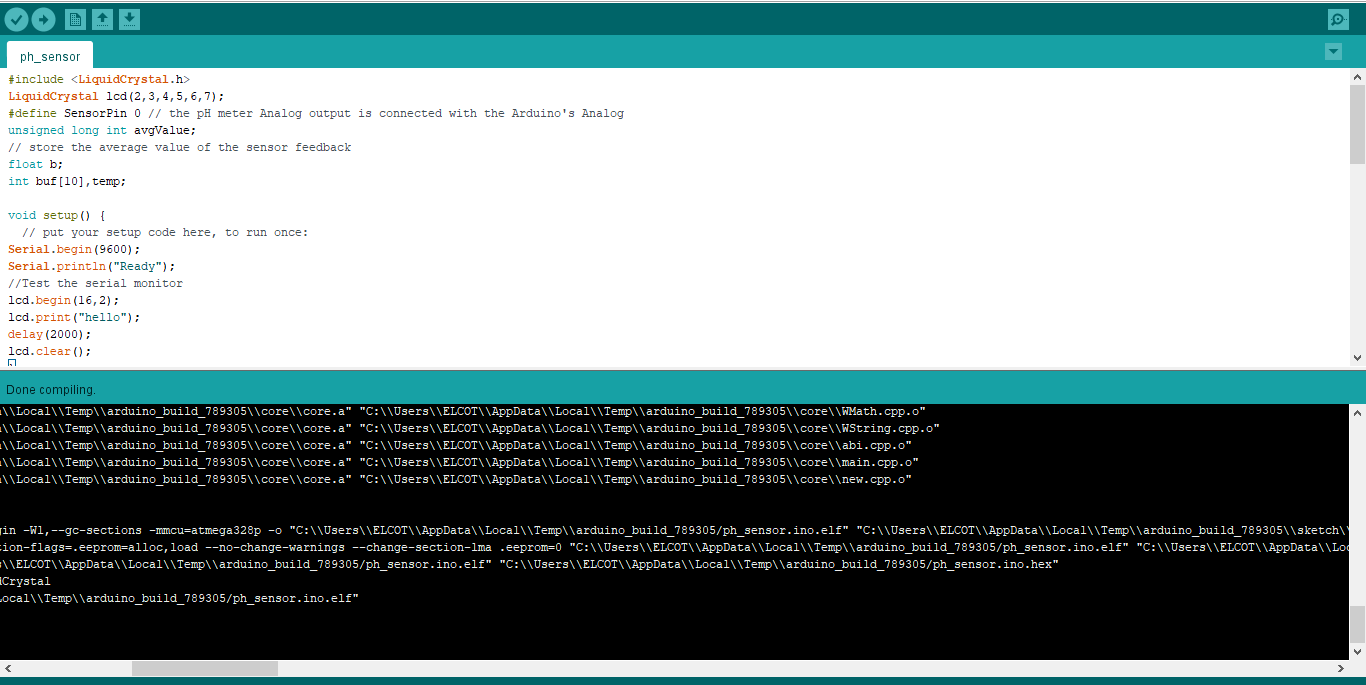


Figure 14(a) Arduino programming(IDE)

## ARDUINO SOFTWARE (IDE)

Get the latest version from the download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a [portable](https://www.arduino.cc/en/Guide/PortableIDE) [installation](https://www.arduino.cc/en/Guide/PortableIDE). When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system

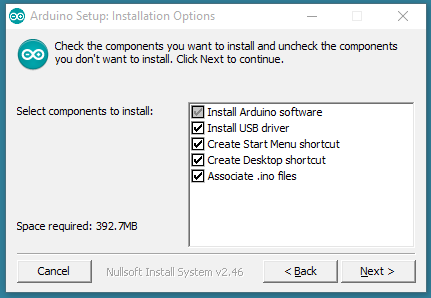
.

Figure 14( b) COMPONENTS TO INSTALL

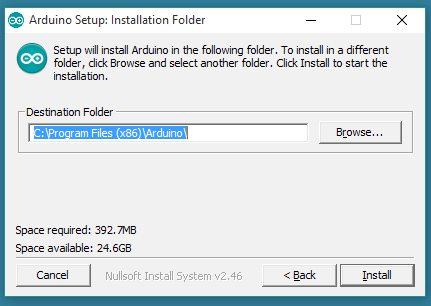
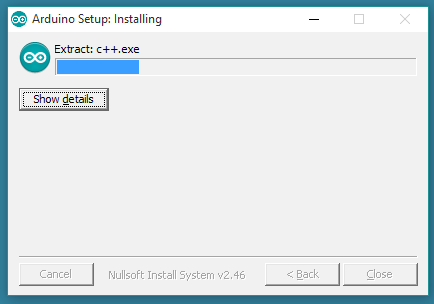


Figure 14© DESTINATION FOLDER



Choose the installation directory (we suggest to keep the default one)

Figure 14(d) EXTRACTION

The process will extract and install all the required files to execute properly the Arduino Software (IDE)

**5.4.1 SENSOR INTERFACING AND DATA READING**

Sensor interfacing is one of the most important tasks in embedded systems, especially in smart agriculture and hydroponic systems. In this project, various sensors are used to monitor the plant’s environment and nutrient solution parameters. These sensors are connected to the Arduino Uno, which continuously reads and processes the data to make intelligent decisions.

**1. DHT11 – Temperature and Humidity Sensor:**

The DHT11 sensor is a low-cost digital sensor that measures two important environmental conditions: temperature and relative humidity. It has three pins: VCC (power), GND (ground), and Data.

* **Connection**:
  + VCC → 5V pin of Arduino
  + GND → GND pin of Arduino
  + Data → Digital pin (e.g., D2) of Arduino
* **Working**:  
  The sensor provides calibrated digital output. It sends data using a proprietary one-wire protocol. We use the DHT.h library in Arduino IDE to simplify the process of reading data.
* **Usage in project**:
  + If the temperature exceeds a threshold (e.g., 30°C), the Arduino turns ON the fan via relay.
  + The humidity reading can also be displayed on the LCD to monitor plant environment health.

**2. pH Sensor:**

The pH sensor measures the acidity or alkalinity of the nutrient solution. It is an analog sensor that provides voltage output corresponding to the pH level (usually from 0 to 14 scale).

* **Connection**:
  + VCC → 5V
  + GND → GND
  + Analog Output → A0 pin of Arduino
* **Working**:  
  The output voltage varies depending on the pH of the solution. Calibration is usually done using known buffer solutions (pH 4.0, 7.0, 10.0). The voltage is mapped using a formula or lookup table in the Arduino code to get the approximate pH value.
* **Usage in project**:
  + If pH is too low, Arduino activates the “pH up” pump.
  + If pH is too high, Arduino activates the “pH down” pump.
  + Ideal pH is maintained automatically for optimal plant growth.

**3. Optional Sensors (if used):**

* **Soil Moisture Sensor / EC Sensor (in hydroponics)**:  
  These can measure the presence of water or nutrient concentration. Similar to pH, these are also analog sensors and are read using analogRead().

**Reading Sensor Data in Arduino Code:**

The Arduino reads sensor data using built-in functions:

* analogRead(pin) for analog sensors like pH
* digitalRead(pin) for digital sensors (if any)
* dht.readTemperature() and dht.readHumidity() for DHT11

The data is processed, stored in variables, and used for further decisions (like triggering a relay or sending info to Raspberry Pi).

**Benefits of Sensor Integration:**

* Enables **real-time environmental monitoring**
* Ensures **automatic control** of actuators
* Forms the **foundation of precision agriculture**
* Data can be **logged or sent to cloud** in advanced versions

**5.4.2 ACTUATOR CONTROL VIA RELAY MODULE**

**Introduction:**

A relay module is an electromechanical switch used to control high-voltage devices like fans, pumps, and lights using low-power control signals from a microcontroller like Arduino. In this project, the relay module acts as an interface between the Arduino board and external devices such as water pumps and exhaust fans used for controlling plant environment conditions and spraying solutions automatically when a disease is detected.

**Why Use Relay Modules?**

* Arduino operates on low voltages (5V logic), which is not sufficient to drive high-voltage devices directly.
* Relays provide **electrical isolation** using an internal electromagnetic coil.
* They act as **switches**, opening or closing circuits based on the signal received from Arduino.

**Working Principle of a Relay:**

A relay typically consists of:

* An **electromagnet (coil)**
* A **common (COM)** terminal
* A **normally closed (NC)** terminal
* A **normally open (NO)** terminal

When Arduino sends a LOW signal to the relay:

* The coil energizes
* The contact shifts from NC to NO
* Thus, the device connected to NO gets power and turns ON

When the signal is HIGH:

* The coil de-energizes
* Contact returns to NC
* The device turns OFF

**Relay in This Project:**

**Connected Devices:**

* **Water pump:** Sprays water/NPK solution when disease is detected
* **Fan:** Helps in maintaining temperature/humidity levels when they exceed ideal values

**How It Works:**

1. Arduino receives data from Raspberry Pi via serial port, indicating whether the plant is healthy or diseased.
2. Based on this input, Arduino controls the relay connected to the water pump or fan.
3. If a disease is detected, the relay is triggered to turn ON the water pump and spray the required treatment.
4. Similarly, if the temperature is too high, the fan is turned ON through the relay.

### **Wiring the Relay to Arduino:**

| **Relay Pin** | **Arduino Pin / Connection** |
| --- | --- |
| IN1 (Control Pin) | Digital Pin (e.g., D8) |
| GND | GND |
| VCC | 5V |
| COM | Live Wire from Power Source |
| NO (Normally Open) | Device live wire (e.g., pump) |

A separate **power supply** (12V or 230V) is used for the high-power devices, and the relay acts as a gate controlled by Arduino.

**Application in This Project**

In the hydroponics system, the relay module allows Arduino to:

* **Turn ON a water pump** to spray disease treatment solution when a plant disease is detected by the Raspberry Pi
* **Activate a fan** to cool down the system when temperature crosses a threshold

This automatic control **reduces human effort** and ensures plants get immediate car

**Advantages of Relay Control**

* Provides **electrical isolation** between Arduino and high-power loads
* Can control **multiple AC/DC appliances**
* **Simple to program and integrate**
* Safe and cost-effective solution for automation

**5.4.3 CODE STRUCTURE AND ARDUINO CONTROL WORKFLOW**

The Arduino control workflow represents the step-by-step logic that the microcontroller follows during operation. It is a crucial part of embedded system development, helping in understanding how the program interacts with sensors, actuators, and external communication interfaces.

At the heart of every Arduino program are two primary functions:

* **setup() Function**: Executes once when the system is powered on or reset. This is where initial configurations are made — such as setting pin modes, initializing serial communication, and starting sensors or displays.
* **loop() Function**: Continuously executes after setup() and contains the main logic that runs repeatedly as long as the system is powered.

In this project, the Arduino reads data from sensors (like pH, DHT11, soil moisture), evaluates the conditions, and takes necessary actions such as turning ON the water pump, fan, or pH control system. It also sends the data to Raspberry Pi via serial communication and displays output on the I2C LCD.

**Control Workflow Steps:**

1. **Initialize components** – Set pin modes, begin serial communication, initialize LCD.
2. **Read sensor values** – pH sensor, temperature and humidity (DHT11), and soil moisture.
3. **Display readings on LCD** – Sensor outputs are shown in real-time.
4. **Send data to Raspberry Pi** – Using UART/Serial communication.
5. **Receive command from Pi (if any)** – To activate pump or spray.
6. **Based on sensor values, take actions:**
   * If temperature is high → turn on fan.
   * If pH is not in ideal range → operate relay to adjust pH.
   * If Raspberry Pi confirms disease detection → turn ON spray pump.
7. **Repeat the loop** – Continuously monitors and responds to conditions.

**Advantages of this Workflow:**

* Efficient decision-making at microcontroller level.
* Enables real-time automation.
* Easy to debug and modify.
* Supports smart agriculture use cases like disease response and environmental control.

**5.4.4 Arduino Simulation Using Proteus**

**Introduction**

Proteus Design Suite is used to simulate embedded systems and microcontroller-based circuits before physically implementing them. In this project, we simulated the entire Arduino-based monitoring and control system using Proteus 8 Professional. This helped in verifying the logic, testing sensor readings, and controlling outputs without hardware.

**Tools Used**

* Proteus 8 Professional
* Arduino IDE
* Arduino Proteus Library
* HEX file exported from Arduino IDE

Simulation Procedure

1. Code Development  
   The Arduino code was written and compiled using the Arduino IDE. The option "Export Compiled Binary" was used to generate the HEX file required for simulation.
2. Circuit Design in Proteus  
   The Proteus simulation circuit was designed by placing the required Arduino board (e.g., Arduino UNO), sensors, and output modules such as LEDs, fans, pumps, etc. Wiring was done according to the actual project connections.
3. Library Integration  
   Arduino libraries were added to Proteus to support microcontroller simulation. These include .LIB and .IDX files, which were placed in the Proteus library folder.
4. Loading the HEX File  
   The compiled HEX file was loaded into the Arduino module in Proteus by editing the Arduino properties.
5. Running the Simulation  
   After connecting all components, the simulation was executed. Sensor values were virtually input, and the output was observed to verify functionality.

**Advantages of Using Proteus Simulation**

* No need for physical components initially
* Easy debugging and visualization of logic
* Saves time and cost during development
* Helpful for verifying sensor and actuator response

### **Components Used in Simulation**

| **S.No** | **Component Name** | **Purpose** |
| --- | --- | --- |
| 1 | Arduino UNO | Microcontroller for control logic |
| 2 | DHT11 Sensor | Measures Temperature and Humidity |
| 3 | pH Sensor Module | Monitors water pH level |
| 4 | LCD Display (16x2) | Displays pH, temperature, and humidity |
| 5 | DC Fan | Turns ON when temperature is high |
| 6 | Water Pump (Motor) | Turns ON when pH level is too low or high |
| 7 | Relay Module | Used to switch fan and pump automatically |
| 8 | Power Supply | Provides power for all components |

**Simulation Setup**

* The Arduino UNO is programmed to take input from the **pH sensor** and **DHT11 sensor**.
* The **LCD display** shows:
  + pH value
  + Temperature in °C
  + Humidity in percentage
* Based on the readings:
  + If **temperature > 30°C**, the **fan** is automatically turned ON.
  + If **pH < 5.5**, the **pH up solution pump** (motor) is turned ON.
  + If **pH > 7.5**, the **pH down solution pump** is activated.

**Simulation Process**

1. **Code Writing and HEX Exporting**  
   Code was written in Arduino IDE and exported as a HEX file using the “Export compiled binary” option.
2. **Circuit Design in Proteus**  
   The components were placed in the Proteus workspace. Wiring was done as per the actual hardware connections.
3. **Library Addition**  
   Custom libraries for Arduino and sensors were added into Proteus (e.g., DHT11, LCD with I2C).
4. **Running the Simulation**  
   The HEX file was loaded into the Arduino UNO module. Simulation was run, and the sensor values were tested. Output devices (fan, pump) responded as expected based on sensor readings.

**Observations**

From the LCD

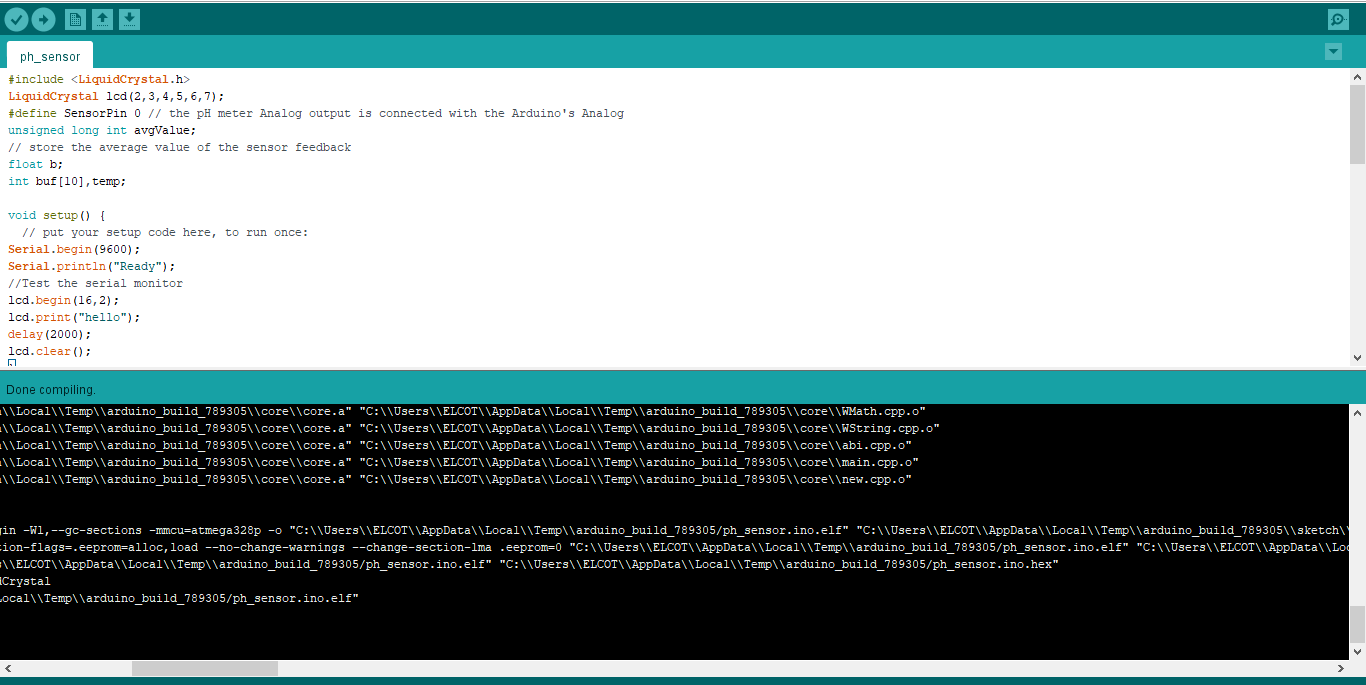
pH: 5.00 Temp: 102

Hum: RH=0043%

* pH is **5.00**, which is **slightly acidic**, so the system would trigger **pH Up pump**.
* Temperature is **102°C** (seems like test data), so **fan would turn ON**.
* Humidity is **43%**, which is acceptable.

This proves that the system responds correctly to real-time conditions.

### 

**Figure** 15Arduino Simulation Using Proteus

### Using Proteus, we successfully simulated our smart hydroponics system, validated sensor integration, output control, and display logic. This step ensured our physical circuit would work as expected**.**

### **5.5 SOFTWARE FLOW DIAGRAM**

#### **5.5.1 Overview**

The software flow diagram represents the logical flow of operations within the system, showcasing how different components such as Raspberry Pi, camera module, sensors, and actuators interact with each other. This structured approach ensures clarity in understanding the sequence of processes, data handling, and decision-making for automation.

**5.5.2 Raspberry Pi Flow**

Trigger Water

If

**5.5.3 Arduino Flow**

**5.6 Summary of Software Description**

The software system in this project integrates two main computing units — **Raspberry Pi** and **Arduino Uno** — each with clearly defined roles, ensuring modularity and efficiency in system operations.

The **Raspberry Pi** handles the **intelligent decision-making tasks**. It captures real-time images of plant leaves using a camera module and processes them using a trained **Convolutional Neural Network (CNN)** model built with **Python** and **TensorFlow Lite**. The software preprocesses the input image, resizes it to 224x224 pixels, and feeds it into the model. Based on the prediction result:

* If the plant is **healthy**, a message “Healthy Plant” is displayed on the **I2C LCD display**, and no action is taken.
* If a **disease** is detected, the disease name is displayed on the LCD, and the Raspberry Pi triggers a **water pump** via GPIO pins to spray pesticide or nutrient solution.

In parallel, the **Arduino Uno** continuously runs its own embedded program (written using **Arduino IDE**). It interfaces with environmental sensors like the **DHT11** (for temperature and humidity), **pH sensor**, and **soil moisture sensor**. Based on the sensor readings:

* If temperature is high, a **fan** is turned on via a **relay module**.
* If the pH is out of the ideal range, **pH up/down solution valves** are activated.
* When the plant needs watering, the Arduino controls the **water delivery system**.

The two systems operate independently but complement each other — Raspberry Pi focuses on **disease detection and treatment**, while Arduino manages **environmental monitoring and maintenance**. This dual-software architecture allows both automation and precision control.

Technologies used include:

* **Python** for image processing, inference, and GPIO control on Raspberry Pi.
* **TensorFlow Lite** for running the optimized CNN model.
* **Arduino IDE** and embedded C/C++ for sensor interfacing and actuator control.
* **Serial print statements** and LCD modules for real-time feedback to users.

**CHAPTER 6**

**6. SMART CONTROL FLOW**

**6.1 SMART CONTROL FLOW**

The smart control flow in this project refers to the automated decision-making and control mechanism that manages both plant health treatment and environmental condition regulation. The entire system functions based on real-time data analysis and model prediction.

The process flow:

* Raspberry Pi captures an image daily and runs a CNN model to detect plant disease.
* Based on the result, if disease is detected, it triggers the pesticide water pump through GPIO.
* Simultaneously, Arduino monitors sensor values (temperature, humidity, pH, soil moisture).
* According to set thresholds, Arduino activates corresponding actuators like fan, pH valve, and water pump.
* LCD modules display both sensor readings and disease status for transparency.

This flow ensures that both health monitoring and environment maintenance work **in parallel and independently** — creating a fully **smart and automated control system**.

**6.2 CHANNEL FREQUENCIES**

The system does not use wireless communication but operates through **dedicated GPIO (General Purpose Input Output) and I2C communication** channels.

* **Raspberry Pi to LCD**: Uses I2C communication at ~100 kHz frequency.
* **Sensor Data** (on Arduino): Collected at regular intervals (1s to 10s depending on sensor type).
* **GPIO Control Signals**: Triggered instantly based on condition checks.

Since this system is wired and not wireless, interference and delay are minimal, allowing real-time and reliable actuator response.

**6.3 ACTUATOR RESPONSE BASED ON CONDITIONS**

| **Parameter** | **Condition** | **Response** |
| --- | --- | --- |
| Disease Detection | Disease Detected (via CNN) | Raspberry Pi turns ON water pump |
| Temperature | > 30°C | Arduino turns ON exhaust fan |
| Humidity | < 40% | Arduino triggers humidifier |
| Soil Moisture | < threshold | Arduino activates water pump |
| pH Value | Not in 5.5 – 6.5 range | Arduino triggers pH up/down valve |

Actuator responses are based on clearly defined conditions

Each actuator is controlled by either Raspberry Pi or Arduino depending on the nature of the input (image-based or sensor-based).

**6.4 ENERGY OPTIMIZATION USING SOLAR POWER**

To reduce energy consumption and support sustainability, solar panels are used to power selected low-energy components:

* A **small solar panel** charges a battery that powers:
  + Arduino Uno
  + Sensors (like DHT11, pH sensor)
  + LCD display module
* Raspberry Pi is powered using a separate power supply or battery bank due to its higher current requirement.

Solar-based optimization helps in rural and off-grid farming setups by ensuring autonomous operation without full dependency on electricity.

**6.5 APPLICATIONS**

This smart plant health monitoring and control system has wide real-world applications:

* **Hydroponic Farming**: For precise disease control and nutrient delivery.
* **Greenhouses**: Automates environmental control, reducing manpower.
* **Urban Gardening**: Helpful in home gardens for disease alerts and auto-spraying.
* **Smart Agriculture Research**: Used as a prototype for precision farming and AI-integrated agriculture.
* **Educational Projects**: Demonstrates integration of AI, IoT, and automation.

## ****CHAPTER 7****

## ****7. CONCLUSION****

The proposed project successfully demonstrates the development and deployment of an **AI-based smart hydroponic monitoring system** using advanced technologies such as **Convolutional Neural Networks (CNN)**, **IoT**, and **embedded automation**. The system is designed to detect plant diseases in real-time using image processing and simultaneously monitor environmental conditions critical for hydroponic plant growth.

The **Raspberry Pi 3B+** serves as the intelligent brain of the system, performing image capture using a camera module, analyzing the plant leaf image using a trained CNN model, and taking necessary action based on the prediction. If a disease is detected, the Raspberry Pi automatically triggers the **water pump** using its GPIO pin to spray the appropriate pesticide or solution. The result of the disease detection is also displayed on the **I2C LCD display** connected to the Pi, providing a clear and direct interface for users.

On the other hand, an **Arduino Uno board** is utilized to monitor the **environmental parameters** such as temperature, humidity, soil pH, and moisture using appropriate sensors like **DHT11**, **pH sensor**, and **soil moisture sensor**. Based on these sensor readings, the Arduino autonomously controls devices like **fans**, **pumps**, and **pH up/down solution valves** through relays, maintaining optimal conditions for plant health and growth. This dual-system approach ensures that both internal plant health and external environmental conditions are simultaneously managed.

An additional benefit of the system is the use of **solar power**, which supports energy optimization and makes the solution suitable for rural and off-grid agricultural areas. The partial use of renewable energy improves sustainability and reduces dependency on conventional power sources.

Overall, this system reduces human effort, enhances precision in agricultural practices, and provides a scalable, smart, and cost-effective solution for modern farming. By integrating machine learning with embedded automation, this project paves the way for more advanced, real-time, and eco-friendly farming practices in the future.

**7.1 Future Enhancement**

In the future, this system can be enhanced with multiple features to improve its efficiency and scalability. A **robotic arm** can be introduced to perform precision tasks such as spraying medicine or nutrients directly on the infected leaves based on disease detection. The system can be extended to support **multiple plant monitoring**, allowing a single unit to handle several hydroponic plants simultaneously. For easier user interaction, **voice command control** can be added using a voice recognition module or virtual assistant integration. **Cloud connectivity** can be established to store and access sensor data and plant health reports remotely. A dedicated **mobile application** can be developed to monitor real-time data, control devices, and receive alerts from anywhere. Additionally, **NPK detection and automatic mixing** can be implemented to ensure balanced nutrient supply for plant growth. Further, detecting **NPK levels in the water** will help in maintaining proper nutrient concentration and preventing deficiencies or excesses in the hydroponic system.

**CHAPTER 8**

**References**

1. Musa, Aminu & Hamada, Mohamed & Aliyu, Farouq & Hassan, Mohammed. (2021). An Intelligent Plant Dissease Detection System for Smart Hydroponic Using Convolutional Neural Network. 10.1109/MCSoC51149.2021.00058.
2. A. Bhamare, V. Upadhyaya, P. Bansal, “Optimized Hydroponic System for Predicting Abnormal Lettuce Leaf using AI Techniques,” *2024*. [Online]. Available: <https://www.researchgate.net/publication/388129867>.
3. Musa, A., Hassan, M., Hamada, M., & Aliyu, F. (2022). Low-Power Deep Learning Model for Plant Disease Detection for Smart-Hydroponics Using Knowledge Distillation Techniques. *Journal of Low Power Electronics and Applications*, *12*(2), 24. <https://doi.org/10.3390/jlpea12020024>
4. Bhamare, Archana & Upadhyaya, Vivek & Bansal, Payal. (2024). Optimized Hydroponic System for Predicting Abnormal Lettuce Leaf using AI Techniques. 1428-1433. 10.1109/ICACRS62842.2024.10841511.
5. Musa, A., Hassan, M., Hamada, M., & Aliyu, F. (2022). Low-Power Deep Learning Model for Plant Disease Detection for Smart-Hydroponics Using Knowledge Distillation Techniques. *Journal of Low Power Electronics and Applications*, *12*(2), 24. <https://doi.org/10.3390/jlpea12020024>
6. Banerjee, A., Lal, E. and Berlin Hency, V. (2023). IoT-Based Plant Health Monitoring System Using CNN and Image Processing. In Integrated Green Energy Solutions Volume 1 (eds M.S. Dangate, W.S. Sampath, O.V. Gnana Swathika and P. Sanjeevikumar). <https://doi.org/10.1002/9781119847564.ch18>
7. Gowtham V et al., “A SMART HYDROPONIC PLANT DISEASE DETECTION SYSTEM USING CNN,” 2024. [Online]. Available: <https://www.psvpec.in/jcres/2024_2/A12.pdf>