# MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

*Techno Complex,. Madurdaha,Beside NRI Complex, Post-Uchhepota, Kolkata 700 150*

MAJOR PROJECT REPORT

ROBOTIC ARDU-VISONAGRO ROBO BOT IOT

MAKAUT ODD SEMESTER 2023-2025



[MASTER OF COMPUTER APPLICATION]



## MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

(Formerly known as WEST BENGAL UNIVERSITY OF TECHNOLOGY)

**UNDER THE SUPERVISION OF**

**MR. SOUMYA CHAKRAVRATY**

**Done By:**

**Rahul Samanta Roll No : - 1427102358 Arghyadip Roy Roll No: - 14271023005**

**Animesh Das Roll No:- 14271023041**

## ACKNOWLEDGEMENT

We would like to express our sincere gratitude to the mentors at Training Organizations “Mr. Soumya Chakravarty”. Without their guidance and constant supervision, our web development project would not have been possible. We are truly thankful for the invaluable information and support they provided throughout the project’s duration.

Additionally, we extend our heartfelt thanks to everyone who willingly contributed their skills and expertise to help us in developing this project. Your assistance has been instrumental in bringing our vision to life.

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **ROLL NO.** | **COLLEGE** | **SIGNETURE** |
| **Rahul Samanta** | **14271023058** | **MSIT** |  |
| **Arghyadip Roy** | **14271023005** | **MSIT** |  |
| **Animesh Das** | **14271023041** | **MSIT** |  |

Thanks & regards

MR. SOUMYA CHAKRAVRATY

# MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

*Techno Complex,. Madurdaha,Beside NRI Complex, Post-Uchhepota, Kolkata 700 150*

**

## CERTIFICATE

We hereby certify that the project named Caliber entitled to be a “  **Robotic Ardu-VisonAgro Robo bot IOT** ” project that is being presented as MCA MAJOR PROJECT 2025 of the requirement for the award of the Master of Computer Application and submitted to the Department of MCA of Meghnad Saha Institute of Technology is an authentic record of my own work carried out during a period from February 2025 to June 2025(4th semester) under the supervision of “ **MR.SOUMYA CHAKRAVRATY**”, MCA Department.

The matter presented in the Minor Project has not been by me for the award of other degree elsewhere.

## Signature Of Students

Rahul Samanta Arghyadip Roy Animesh Das

This is to certify that above statement made by the students is correct to the best of my knowledge.

Date: -

Head of the Department MS. Aparna Datta

Signature of project guide

Mr. Soumya Chakravarty

Signature of external guide Signature of External Examiner

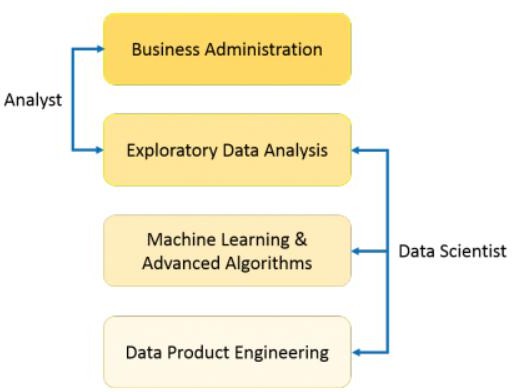
**CONTENTS**

|  |  |
| --- | --- |
| **CHAPTER** | **TOPIC** |
| **1.** | **About Data science and python and IOT** |
| **2.** |  |
| **3.** |  |
| **4.** |  |
| **5.** |  |
| **6.** |  |
| **7.** |  |
| **8.** |  |
| **9.** |  |
| **10.** |  |
| **11.** |  |
| **12.** |  |
| **13.** |  |
| **14.** |  |
| **15.** |  |
| **16.** |  |
| **17.** |  |

***-: ABOUT DATA SCIENCE AND PYTHON AND IOT: -***

**What is Data Science?**

Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data. How is this different from what statisticians have been doing for years?

The answer lies in the difference between explaining and predicting.

In the above image, a Data Analyst usually explains what is going on by processing history of the data. On the other hand, Data Scientist not only does the exploratory analysis to discover insights from it, but also uses various advanced machine learning algorithms to identify the occurrence of a particular event in the future. A Data Scientist will look at the data from many angles, sometimes angles not known earlier.

So, Data Science is primarily used to make decisions and predictions making use of predictive causal analytics, prescriptive analytics (predictive plus decision science) and machine learning.

**Predictive causal analytics: -**

Predictive causal analytics combines **prediction** with **causal inference** to not only forecast future outcomes but also understand the **cause-and-effect** relationships behind them. Unlike regular predictive models that find correlations, causal analytics aims to determine **why** something happens. Techniques such as **causal diagrams (DAGs)**, **propensity score matching**, and **instrumental variables** are used. For example, in healthcare, it helps predict patient outcomes and understand whether a treatment **causes** improvement. This approach supports more informed decision-making by identifying interventions that can **change** future outcomes, not just anticipate them.

**Prescriptive analytics: -**

Prescriptive analytics is the advanced stage of data analytics that recommends actions based on data insights. Unlike descriptive (what happened) and predictive (what might happen) analytics, prescriptive analytics answers “what should be done?” by using techniques like optimization, simulation, and machine learning. It helps in decision-making by suggesting the best course of action under given constraints. For example, in supply chain management, it can recommend optimal inventory levels or delivery routes. Prescriptive analytics enables organizations to make proactive, data-driven decisions for improved efficiency and outcomes.

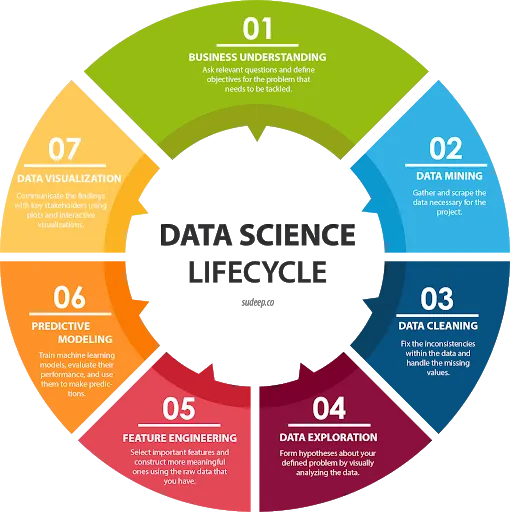
Machine learning for making predictions: -

Machine learning is commonly used to **make predictions** based on data. In supervised learning, algorithms are trained on historical labelled data to learn the relationship between input features and output labels. Once trained, the model can predict outcomes for new, unseen data. For example, ML can predict **house prices**, **stock trends**, or whether a patient has a disease. Algorithms like **linear regression**, **decision trees**, and **neural networks** are often used. Prediction through ML improves accuracy and efficiency in various domains such as healthcare, finance, marketing, and more.

**Machine learning for pattern discovery: -**

Machine learning (ML) is widely used for **pattern discovery**—the process of identifying meaningful structures, trends, or regularities in data. ML algorithms, especially unsupervised techniques like **clustering** (e.g., K-Means) and **association rule mining** , help discover hidden patterns without predefined labels. For example, ML can group customers based on purchase behavior or detect fraud patterns in transactions. This aids in decision-making, prediction, and automation by uncovering insights that are difficult to detect manually.

**-: LIFE CYCLE OF DATA SCIENCE*:* -**

****

**What is python?**

Python is a versatile, high-level programming language renowned for its readability and simplicity, used in various fields like web development, data science, and machine learning Python is a versatile, high-level programming language renowned for its readability and simplicity, used in various fields like web development, data science, and machine learning. It was created by **Guido van Rossum**, and released in 1991.

It is used for:

* web development (server−side),
* software development,
* mathematics,
* system scripting.

## What can Python do?

Here are key points on what Python can do:

* Web development
* Data analysis and visualization
* Machine learning and AI
* Automation/scripting
* Desktop application development
* Game development
* IoT and hardware programming
* Mobile app development
* Cybersecurity and ethical hacking
* API development and consumption
* Database access and management
* Cloud and DevOps tasks
* Scientific computing
* Natural Language Processing
* Blockchain and cryptocurrency applications
* GUI automation and testing

**Why python?**

Python is a popular programming language due to its simplicity, versatility, and extensive libraries, making it suitable for various applications, including data science, web development, and automation. Its beginner-friendly design, readability, and extensive support from the community further contribute to its widespread adoption.

Key Features and Benefits:

* **Simplicity and Readability:**

Python's syntax is designed to be easy to learn and understand, resembling natural language, making it easier for beginners to pick up.

* **Versatility:**

Python is used in a wide range of fields, including web development, data science, machine learning, automation, and more.

* **Extensive Libraries and Frameworks:**

Python boasts a vast ecosystem of libraries and frameworks that simplify complex tasks, such as data analysis (Pandas, NumPy), web development (Django, Flask), and machine learning (Scikit-learn, TensorFlow).

* **Open Source:**

Python is open-source, meaning it's free to use and distribute, fostering a strong community and collaborative development.

* **Strong Community Support:**

Python has a large and active community that provides extensive documentation, tutorials, and support, making it easier to find answers to questions and get help with development.

* **Cross-Platform Compatibility:**

Python is designed to run on various operating systems, making it a portable language for different environments.

* **Fast Prototyping:**

Python's interpreted nature allows for rapid prototyping, enabling developers to quickly test and iterate on ideas.

* **Scalability:**

Python is scalable and can handle complex projects with growing demands.

**Python Installation: -**

* Many PCs and Macs will have python already installed.
* To check if python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

**Packages of Python: -**

Packages are a way of structuring many packages and modules which helps in a well− organized hierarchy of data set, making the directories and modules easy to access. Just like there are different drives and folders in an OS to help us store files, similarly packages help us in storing other sub−packages and modules, so that it can be used by the user when necessary.

There are two types of packages−

1. User Define Packages
2. Import Packages

**1.User Define Packages: -**

**What is a User-Defined Package?**

A user-defined package is a collection of modules (Python files) created by the user to organize related code.

**Structure Example:**

my\_package/

\_\_init\_\_.py # Makes it a package

math\_ops.py # Module 1

string\_ops.py # Module 2

**How to Create a User-Defined Package:**

1. Create a folder – e.g., my\_package
2. Add an \_\_init\_\_.py file (can be empty)
3. Add Python files (modules) – e.g., math\_ops.py

**How to Use (Import) the Package:**

from my\_package import math\_ops

result = math\_ops.add(5, 3)

print(result)

**2.Import Packages:**

Import in python is similar to #include header\_file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way.

**Import module\_name**

When import is used, it searches for the module initially in the local scope by calling

import () function. The value returned by the function are then reflected in the output of the initial code.

**Import module\_name.member\_name**

In the above code module math is imported, and its variables can be accessed by considering it to be a class and pi as its object.

The value of pi is returned by import ().

pi as whole can be imported into our intial code, rather than importing the whole module.

from math import pi

print (pi)

**Some import packages are-**

1. **NumPy: -**

**What is NumPy?**

* NumPy = Numerical Python
* Core library for numerical computing in Python
* Used for arrays, mathematical operations, linear algebra, etc.

**How to Install:**

pip install NumPy

**Basic Import:**

import NumPy as np

**Key Features of NumPy**

| Feature | Description |
| --- | --- |
| ndarray | Fast, multidimensional array object |
| Vectorized operations | No need for loops (faster computation) |
| Broadcasting | Automatic resizing of arrays in operations |
| Mathematical functions | Built-in (e.g., np.sum, np.mean) |
| Linear algebra | e.g., np.dot, np.linalg.inv |
| Random number support | e.g., np.random.rand() |

**Example Code:**

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

print ("Addition:", a + b)

print ("Mean:", np.mean(a))

print ("Reshape:", np.reshape(a, (3, 1)))

1. **Pandas: -**

**What is Pandas?**

Pandas = Python Data Analysis Library

Used for data manipulation, cleaning, and analysis

Works great with tabular data (like Excel or SQL)

**How to Install:**

pip install pandas

**Basic Import:**

import pandas as pd

**Core Data Structures:**

| Structure | Description |
| --- | --- |
| Series | 1D labeled array (like a column) |
| DataFrame | 2D labeled table (like Excel sheet) |

**Example Code:**

import pandas as pd

data = {

'Name': ['Alice', 'Bob', 'Charlie'],

'Age': [25, 30, 35]

}

df = pd.DataFrame(data)

print(df)

print(df['Name'])

print(df.describe())

print(df[df['Age'] > 25])

1. **Sklenar:-**

**What is sklearn?**

Scikit-learn (often shortened to sklearn) is a free, open-source Python library that provides a wide range of tools for machine learning and statistical modeling. It is built upon NumPy, SciPy, and Matplotlib.

Key Features:

* **Comprehensive Algorithms**: Offers a variety of algorithms for classification, regression, clustering, dimensionality reduction, and model selection.
* **User-Friendly Interface:** Designed with a consistent and easy-to-use API.
* **Data Preprocessing:** Includes tools for data cleaning, transformation, and feature engineering.
* **Model Evaluation:** Provides metrics and methods for assessing model performance.
* **Integration with Other Libraries:** Works seamlessly with other Python data science libraries like NumPy, Pandas, and Matplotlib.

Common Use Cases:

* **Classification:**

Identifying categories an object belongs to (e.g., spam detection, image recognition).

* **Regression:**

Predicting continuous values (e.g., house prices, stock market trends).

* **Clustering:**

Grouping similar data points (e.g., customer segmentation, anomaly detection).

* **Dimensionality Reduction:**

Reducing the number of variables in a dataset while preserving important information (e.g., feature selection, visualization).

Installation:

**Scikit-learn can be installed using pip:**

pip install scikit-learn

**Importing in Python:**

import sklearn

1. **Matplotlib: -**

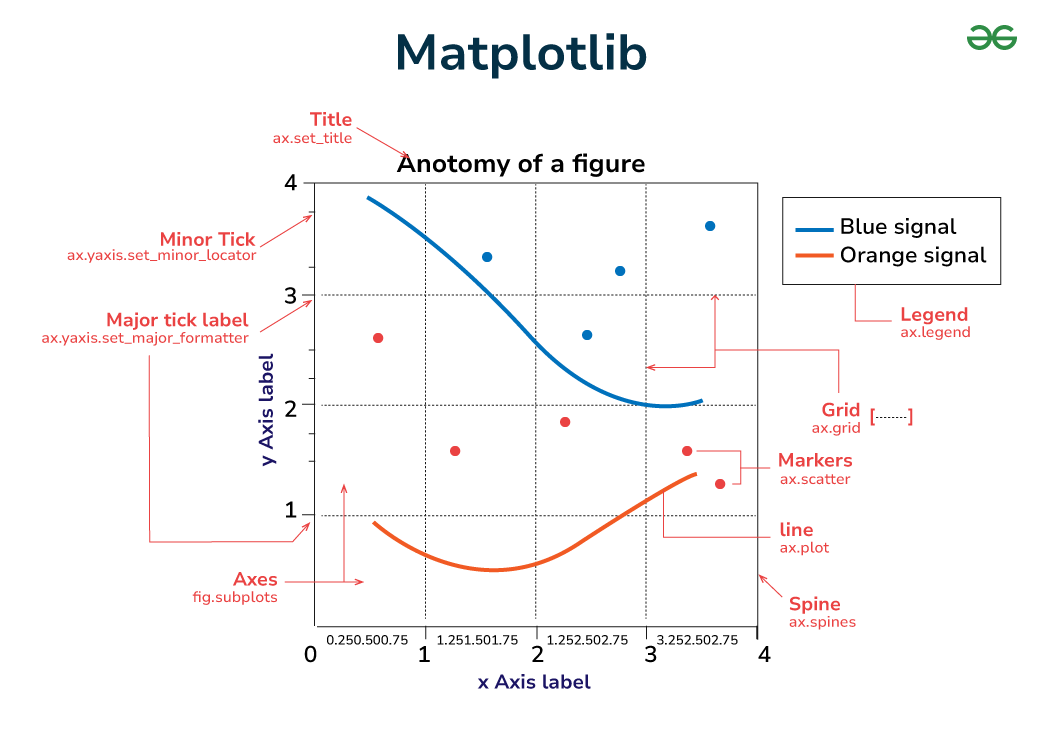
**What is matplotlib?**

Matplotlib is a Python library used for creating static, animated, and interactive visualizations. It is a versatile tool for generating various types of plots, including line plots, scatter plots, bar charts, histograms, and 3D plots.

**Key Features:**

* Diverse Plot Types: Supports a wide range of plots for different visualization needs.
* Customization: Offers extensive control over plot appearance, including colors, labels, styles, and layouts.
* Integration: Works seamlessly with other Python libraries like NumPy and Pandas.
* Subplots: Allows the creation of multiple plots within a single figure.
* Interactive Plots: Enables dynamic visualizations with zooming, panning, and updating capabilities.
* Exporting: Supports exporting plots to various file formats.
* Pyplot Module: Provides a MATLAB-like interface for creating plots with minimal code.
* Figures and Axes: Figures are the overall containers for plots, while axes represent individual plots within a figure.

Matplotlib is a fundamental tool for data scientists, researchers, and engineers, facilitating data analysis, exploration, and presentation.



**how to use OpenCV with Arduino:-**

OpenCV and Arduino do not directly work together because OpenCV is too resource-intensive for Arduino's capabilities. However, you can use Arduino to capture data (like camera images) and send it to a computer (like a Raspberry Pi or laptop) for processing with OpenCV. This allows you to leverage the power of OpenCV while using Arduino to handle peripheral tasks like interacting with sensors and actuators.

**what is OpenCV:-**

OpenCV (Open-Source Computer Vision Library) is a widely used, open-source library for computer vision, machine learning, and image processing. It provides a wide array of tools and algorithms for tasks such as:

* Image and Video Processing: Reading, writing, resizing, filtering, and colour space conversion of images and videos.
* Object Detection and Recognition: Identifying objects, faces, and patterns in images and videos.
* Feature Detection and Matching: Finding key points and descriptors in images for tasks like image stitching and object tracking.
* Machine Learning: Implementing various machine learning algorithms for image classification, object recognition, and more.
* Camera Calibration and 3D Vision: Estimating camera parameters and creating 3D models from images.



**What is Mediapipe:-**

MediaPipe is a cross-platform framework from Google for building machine learning pipelines, especially for real-time multimedia applications. It allows developers to process time-series data like video, audio, and sensor data on various devices. MediaPipe offers pre-trained models, customization options, and tools for evaluation and benchmarking.

**Here's a more detailed breakdown:**

* **Open-Source Framework:**

MediaPipe is an open-source project, meaning its code is freely available for anyone to use, modify, and distribute.

* **Real-time Processing:**

It's designed for real-time applications, enabling fast and efficient processing of multimedia data.

* **Cross-Platform Support:**

MediaPipe works across different platforms, including desktop/server, Android, iOS, and embedded devices.

* **Pre-trained Models:**

It provides a variety of pre-trained models for tasks like object detection, pose estimation, hand tracking, and more.

* **Customization:**

Developers can customize MediaPipe pipelines and models to fit their specific needs.

* **Evaluation and Benchmarking:**

It offers tools for visualizing, evaluating, and benchmarking solutions directly in the browser.

* **MediaPipe Studio:**

MediaPipe Studio is a web-based application that allows users to test and customize MediaPipe solutions in their browser.

* **MediaPipe Tasks:**

MediaPipe Tasks provide cross-platform APIs and libraries for deploying MediaPipe solutions.

* **MediaPipe Framework:**

The low-level component for building efficient on-device machine learning pipelines.

**TENSERFLOE**: -

TensorFlow is an open-source machine learning framework developed by Google. It is used for building and training deep learning and machine learning models.

**Key Features:**

Model Building: Supports both high-level (Keras API) and low-level APIs for flexibility.

Deep Learning: Ideal for neural networks, including CNNs (image), RNNs (text), and transformers.

Deployment: Models can be deployed on web (TensorFlow.js), mobile (TensorFlow Lite), and cloud.

Scalability: Works across CPUs, GPUs, and TPUs for faster training.

**Common Use Cases:**

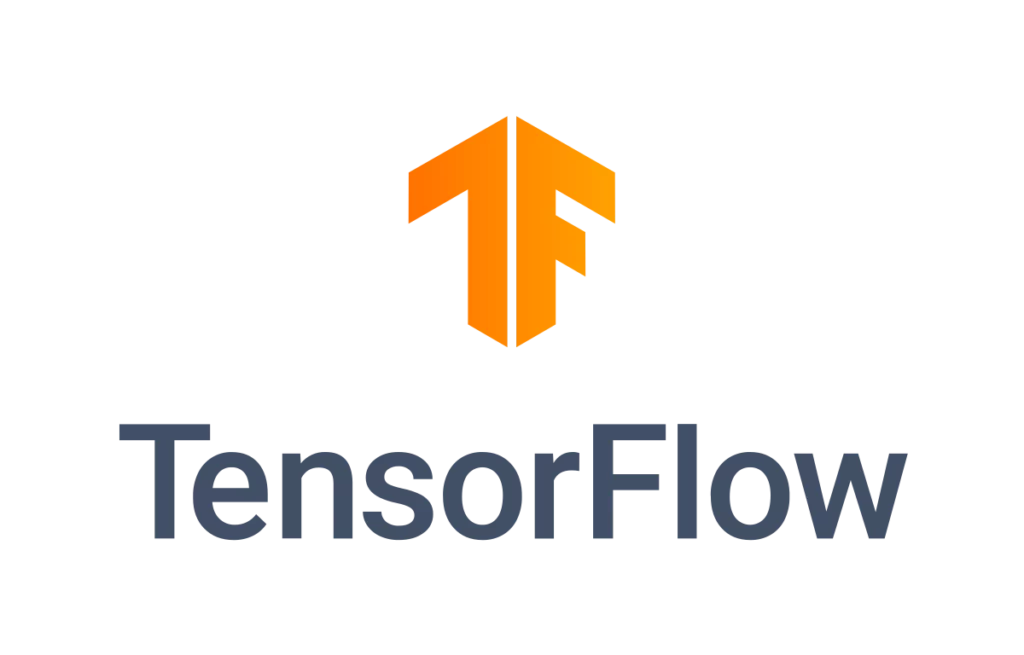
Image classification

Object detection

Speech recognition

Natural language processing

Time-series forecasting



**NEURAL NETWORK: -**

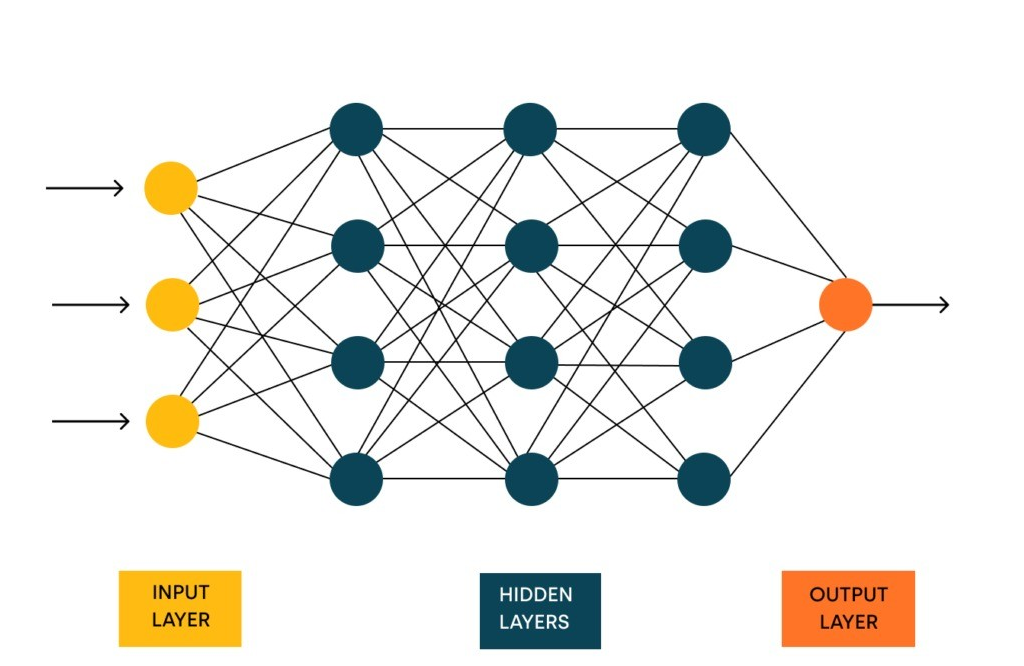
A neural network is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning (ML) process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain. It creates an adaptive system that computers use to learn from their mistakes and improve continuously. Thus, artificial neural networks attempt to solve complicated problems, like summarizing documents or recognizing faces, with greater accuracy.

**Why are neural networks important?**

Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex. For instance, they can do the following tasks.

Make generalizations and inferences

Neural networks can comprehend unstructured data and make general observations without explicit training.



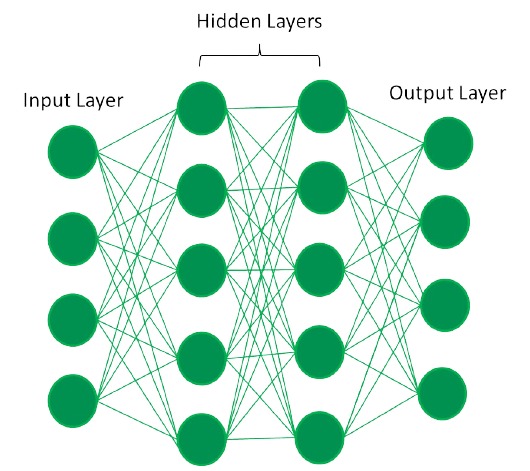
**DEEP LEARNING:-**

Deep Learning is transforming the way machines understand, learn, and interact with complex data. Deep learning mimics neural networks of the human brain, it enables computers to autonomously uncover patterns and make informed decisions from vast amounts of unstructured data.

**How Deep Learning Works?**

Neural network consists of layers of interconnected nodes or neurons that collaborate to process input data. In a fully connected deep neural network data flows through multiple layers where each neuron performs nonlinear transformations, allowing the model to learn intricate representations of the data.

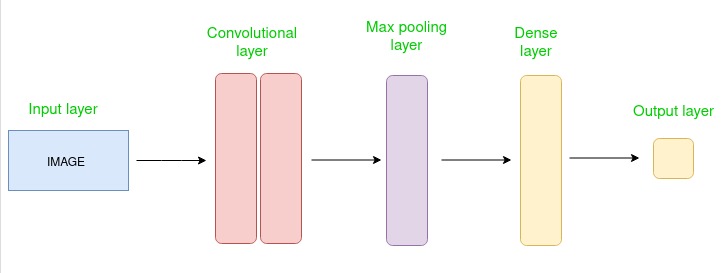
In a deep neural network, the input layer receives data which passes through hidden layers that transform the data using nonlinear functions. The final output layer generates the model’s prediction.



**CNN (Convolutional Neural Network):** -

Convolutional Neural Network (CNN) is an advanced version of artificial neural networks (ANNs), primarily designed to extract features from grid-like matrix datasets. This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role. CNNs are widely used in computer vision applications due to their effectiveness in processing visual data.

CNNs consist of multiple layers like the input layer, Convolutional layer, pooling layer, and fully connected layers. Let us learn more about CNNs in detail.



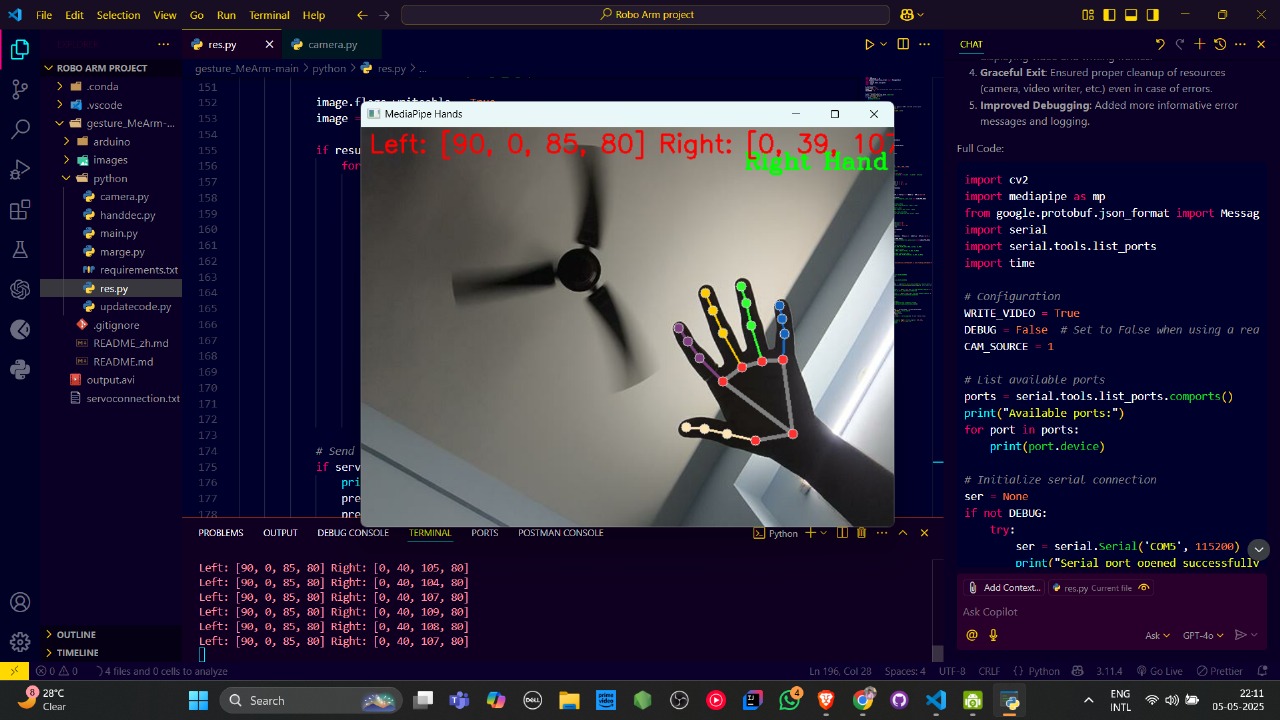
**COMPUTER VISION: -**

Computer vision is the ability of computers to extract information and insights from images and videos. With neural networks, computers can distinguish and recognize images like humans. Computer vision has several applications, such as the following:

Visual recognition in self-driving cars so they can recognize road signs and other road users

Content moderation to automatically remove unsafe or inappropriate content from image and video archives

Facial recognition to identify faces and recognize attributes like open eyes, glasses, and facial hair etc.



**What is IOT?**

The Internet of Things (IoT) is a network of physical objects, or "things," embedded with sensors, software, and other technologies that allow them to connect and exchange data with other devices and systems over the internet.

**How does IOT work?**

1. **Capture Data:**

IoT devices, like sensors, collect data from their surroundings, such as temperature, humidity, light, motion, or pressure.

1. **Share Data:**

This data is then transmitted to a cloud system or another device using network connections like Wi-Fi, Bluetooth, or cellular.

1. **Process Data:**

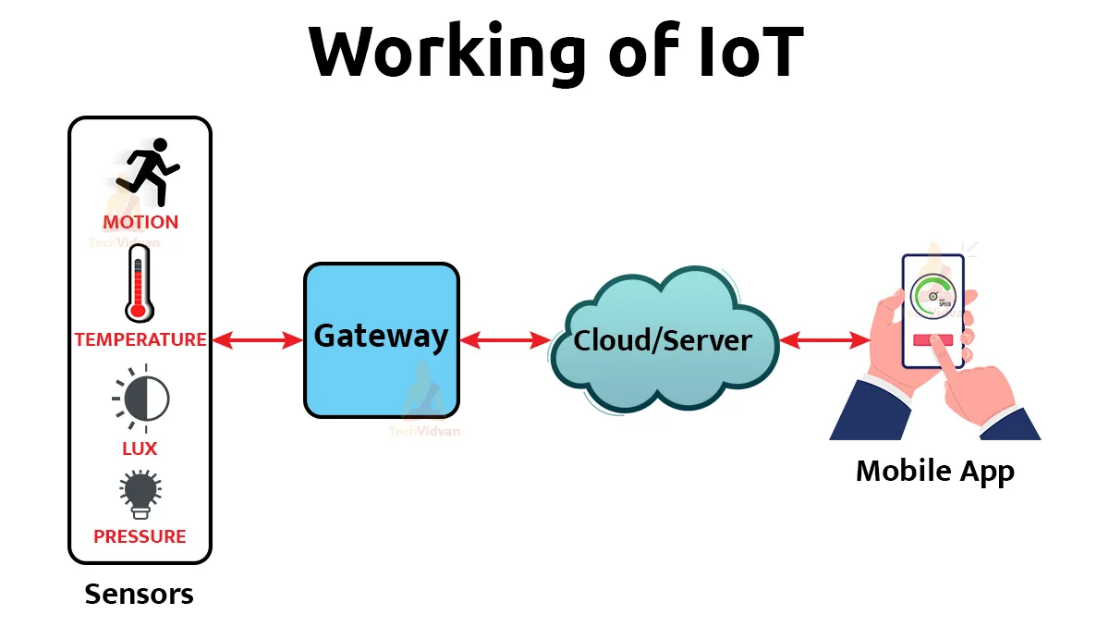
Software on the cloud or the receiving device processes the data and can take actions based on it, like sending alerts, adjusting settings, or making automated decisions**.**

1. **Act on Data:**

The processed data can be analysed to provide insights for better decision-making and business operations.

1. **User Interaction:**

Users can also interact with the system through a user interface to adjust or check on the system's status, and these actions are then sent back to the devices

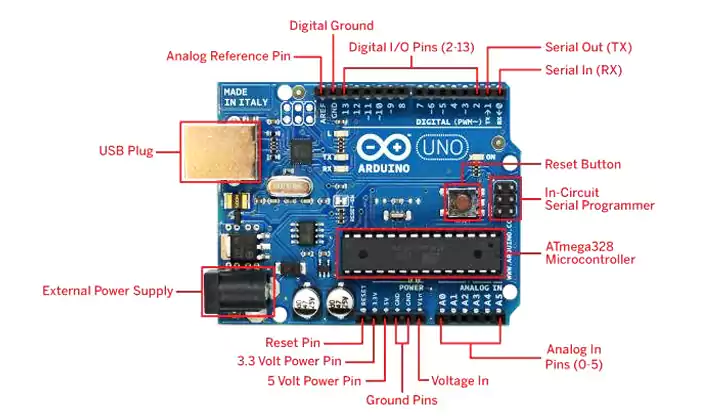


**What is Arduino?**

Arduino is a programmable device that enables interaction with external hardware devices using software programs. Hardware devices that can be connected to Arduino include lights, sensors, actuators, screens, speakers, and other electronic devices.

**Work of Arduino: -**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It allows users to create interactive projects by reading inputs (like sensor data) and controlling outputs (like activating motors or LEDs). Essentially, Arduino acts as a microcontroller, or a small computer, that can be programmed to control various electronic components.



**what is esp32 cam module: -**

The ESP32-CAM is a compact, low-cost camera module that integrates an ESP32 microcontroller with an OV2640 camera and a microSD card slot. It is designed for applications like wireless video monitoring, WIFI image upload, and QR code identification.

**Key Features:**

* **ESP32-S Microcontroller:**

The module is based on the ESP32-S chip, offering Wi-Fi, Bluetooth, and low-power operation.

* **OV2640 Camera:**

It features the OV2640 camera sensor, capable of capturing 2-megapixel images.

* **microSD Card Slot:**

An onboard microSD card slot allows for data storage and offline image capture.

* **Low Power:**

It offers various sleep modes, including deep sleep with a current as low as 6mA.

* **Versatile Connectivity:**

Supports Wi-Fi, Bluetooth 4.2, and multiple communication protocols (UART, SPI, I2C, PWM).

* **Compact Size:**

The module's small size (40 x 27 mm) makes it suitable for various applications.

* **Cost-Effective:**

It's a relatively inexpensive solution for adding camera capabilities to IoT projects.

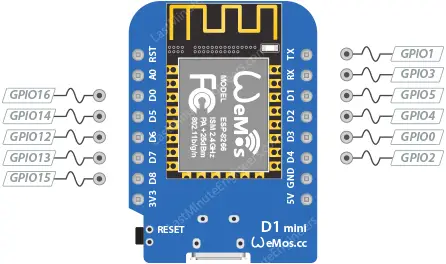


**What is D1 Mini?**

The Wemos D1 Mini is a small, Arduino-compatible development board based on the ESP8266 microcontroller. It is a popular choice for Internet of Things (IoT) projects due to its compact size, built-in WiFi, and ease of use with the Arduino IDE. It can be programmed via USB and features 11 digital I/O pins, 1 analog input, and support for various protocols like I2C, SPI, and UART.

**Key Features:**

* **ESP8266 Microcontroller:** Provides WiFi connectivity for IoT applications.
* **Arduino IDE Compatibility:** Easy to program using the Arduino IDE.
* **NodeMCU Support:** Can also be used with NodeMCU firmware.
* **Compact Size:** Small form factor for easy integration.
* **Versatile I/O:** 11 digital I/O pins, 1 analog input, and support for various protocols.
* **Onboard USB-to-Serial Converter:** No need for a separate programmer.
* **Low Cost:** Relatively inexpensive compared to other development boards.



**SERIAL PORT: -**

A serial port facilitates data transmission between devices, sending bits one after another over a single wire, unlike parallel ports which send multiple bits simultaneously. It's commonly used for connecting devices like modems, printers, and other peripherals to computers, and it's characterized by a D-shaped connector with 9 or 25 pins.

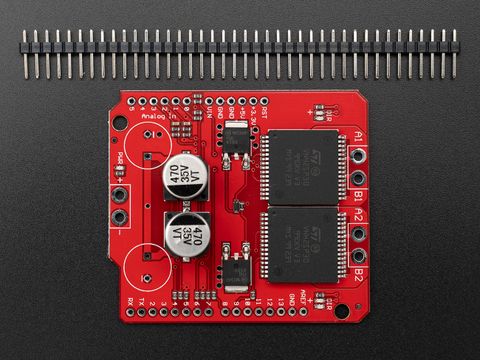
**How it Works:**

1. **Bit-by-Bit Transmission:** Data is converted from parallel form (where multiple bits are sent simultaneously) to serial form (where bits are sent one after another).
2. **Clocking:** A clock signal synchronizes the transmission and reception of data bits.
3. **Data Framing:** Data is organized into frames, which may include a start bit, data bits, an optional parity bit, and stop bits.
4. **Handshaking:** Hardware handshaking (using signals like RTS/CTS) or software handshaking ensures proper data flow between devices.
5. **Baud Rate:** The baud rate determines the speed at which data bits are transmitted.



**What is Moto driver?**

The motor driver is the hidden controller of these electronic components. Depending on the external control input signal, it can provide enough power to manipulate the motor's speed, torque, and direction.



**Breadboard?**

A breadboard, also known as a solderless breadboard or protoboard, is a tool used for building and testing electronic circuits without soldering. It's a plastic board with a grid of holes where electronic components and wires can be inserted and connected to create temporary circuits. This allows for easy modification and experimentation, making it ideal for learning and prototyping.



**SERVO MOTOR: -**

A servo motor is a special kind of electric motor designed to enable precise and accurate movement. It is commonly used in applications such as robots or machines. A servo motor is like a smart motor because it can control its position, speed, and power. This motor works with a sensor that tells it where it is, and a controller makes sure it moves the way we want it to. People use these motors a lot in big machines, especially in places like factories where we need things to move very accurately and smoothly. This article discusses servo motors, and their importance in various fields such as robotics and manufacturing, explaining how they work, how they're built, and where they are commonly used.

Servo motors are like powerful little engines with a high torque-to-inertia ratio, meaning they can generate a lot of twisting force relative to their size and weight. This makes them perfect for tasks that need quick acceleration, deceleration, and precise movements. They come in various sizes and power levels to suit different jobs and can be powered by either AC or DC voltage. What's good is that we can control their speed and force using special signals called pulse width modulation (PWM). Some servo motors have extra features like built-in controllers and communication interfaces, which makes them adaptable and easy to integrate into complex systems.



**-: Introduction of project: -**

**About Robotic Ardu-VisonAgro Robo bot IOT: -**

This project introduces a multipurpose robotic system that combines embedded hardware, computer vision, and machine learning to carry out various real-world tasks in the industrial and agricultural domains. The system's robotic arm, which can be operated with simple hand gestures and AI-driven decision-making, offers a smooth human-machine interface for a variety of applications.

**There are two main sections to the system:**

**Industrial Bots and Surgery**

Here, an Arduino microcontroller is linked to the robotic arm, which is operated by hand gesture recognition enabled by MediaPipe and OpenCV. MediaPipe dynamically maps finger positions to corresponding servo motor actions by capturing and tracking the user's hand landmarks in real-time. Because of this, the robotic arm can imitate human movements, which makes it appropriate for industrial automation and accurate surgical simulations.

**Bot for Farming**

Plant health monitoring is the main focus of the agricultural module. This section's robotic arm is configured to select a leaf from a plant and hold it up to an ESP32-CAM. A pre-trained deep learning model constructed with TensorFlow/Keras processes the leaf image that is captured by the camera. The model helps with early diagnosis and more intelligent farming methods by accurately predicting the presence of common plant diseases.

Along with the robotic arm modules, the project includes a Wi-Fi-operated robotic vehicle intended for either autonomous or manual movement in agricultural and industrial settings. This vehicle is constructed with DC motors, a motor driver module, and a D1 Mini (ESP8266) microcontroller for wireless communication. The vehicle acts as a portable platform for the robotic arm, enabling it to traverse the surroundings for active engagements like finding plants, examining leaves, or executing tasks at various sites. Using Wi-Fi-enabled remote operation, the vehicle's navigation can be managed via a GUI or web interface, allowing for excellent flexibility and adaptability to various terrains and task areas. This mobile robotic platform improves the overall system by facilitating mobility, remote control, and scalability, connecting the divide between static AI models and real-world applications in agriculture, industry, and healthcare simulations.

**Importance of Robotic Ardu-VisonAgro Robo bot: -**

Importance of the System

This project holds significant importance due to its potential applications in critical domains like agriculture, healthcare, and industry. Its core strengths and contributions include:

1. **Multidomain Utility**

The system is not limited to a single use-case—it seamlessly adapts to farming, surgery simulation, and industrial automation.

It offers a modular approach, making it suitable for education, research, and prototyping in robotics and AI.

2. **Touchless and Safe Operation**

Gesture-based control via MediaPipe reduces the need for physical contact, making it ideal for sterile environments like surgical simulations and hazardous industrial zones.

Enhances safety and hygiene, especially in post-pandemic settings.

3. **Precision and Efficiency**

The robotic arm offers accurate servo motor movements driven by real-time hand tracking.

This improves task precision in medical or industrial tasks, which typically require steady and repeatable actions.

4. **AI-Powered Smart Farming**

The system enables early detection of plant diseases using a deep learning model and live ESP32-CAM feed.

Promotes smart agriculture, reducing crop loss and supporting sustainable farming practices.

5. **Wireless Mobility**

The Wi-Fi-controlled robotic car adds mobility and flexibility, allowing the arm to operate across different locations autonomously.

Useful in larger fields, factory floors, or multi-room setups.

6. **Low-Cost and Scalable**

Built using affordable components like Arduino, ESP32-CAM, and D1 Mini, the system is cost-effective and scalable for wider adoption in developing areas.

7. **Educational Value**

Demonstrates integration of AI, IoT, robotics, and embedded systems in one platform.

8. AI-Based Plant Disease Detection

A custom-trained deep learning model accurately classifies plant diseases from leaf images.

Hosted using Streamlit, the model offers a lightweight, browser-accessible interface for fast and user-friendly disease diagnosis.

Enables early detection, helping farmers take timely action and minimize crop losses.

9. Dual UI System for Versatile Interaction

Streamlit Web UI is used for model interaction and real-time disease prediction.

A Python-based Tkinter desktop GUI provides physical control of the robotic hardware, with clearly labeled buttons for:

Switching between bots (Farming, Surgery, Industrial)

Starting ESP32-CAM feed

Triggering servo movement based on gestures

Controlling the mobile robotic car

**THE DEEP LEARNING ADVANTAGES**: -

1. **Precise Identification of Plant Diseases**

In leaf images, the trained CNN model can identify subtle visual patterns that the human eye might miss.

It supports quicker and more intelligent agricultural decisions by achieving high accuracy in the classification of numerous plant diseases.

2. **Visual Analysis Automation**

By automating the process of evaluating plant health, deep learning eliminates the need for agricultural specialists to perform manual inspections.

guarantees accurate, timely, and consistent forecasts—even in extensive farming operations.

5. **Combining Robotic Action with Integration**

An intelligent closed-loop system is made possible by the direct correlation between the model's predictions and robotic actions (such as positioning the arm or marking infected leaves).

supports autonomous operation and lessens the need for human intervention.

6. **Gaining Knowledge from Data**

Deep learning gets better with more data, in contrast to rule-based systems.

Continued training and updates make the system more reliable and accurate over time.

**OBJECTIVES: -**

This project's main objective is to create a multipurpose intelligent robotic system that combines computer vision, artificial intelligence, and embedded hardware for practical uses. The particular goals are:

1. **To Create a Robotic Arm with Gestures**

Create a robotic arm with hand gesture control that can be tracked in real time using MediaPipe and OpenCV for precise industrial and surgical tasks.

2**. To Put Plant Disease Detection into Practice Utilizing Deep Learning** To help with early diagnosis and intelligent farming, train and implement a CNN-based model to categorize plant diseases from ESP32-CAM leaf images.

3. **To Construct a Robotic Platform With Wi-Fi Control**

Create a robotic car with a D1 Mini, motor drivers, and DC motors to give the arm mobility so it can move and function in a variety of settings.

4. **To Create a Control System with Two Interfaces Make two user interfaces:**

A Streamlit web application for identifying illnesses.

a desktop GUI built with Tkinter that controls hardware elements such as the mobile base and robotic arm.

5. **To Combine Physical Robotic Actions with AI Predictions**

Make sure that physical actions (like servo movements) and AI decisions (like disease prediction) integrate seamlessly to create a robotic system that is responsive and self-sufficient.

**-: SCOPE OF THE PROJECT: -**

The goal of this project is to create an integrated robotic system that uses deep learning, computer vision, embedded systems, and the Internet of Things to solve real-world issues in simulations related to industry, healthcare, and agriculture. The scope consists of:

1. Arm with Gestures Control

Create a robotic arm with MediaPipe and OpenCV that is controlled by real-time hand tracking.

Use this system to carry out industrial tasks like lifting and positioning and to replicate surgical precision.

2. Plant Disease Diagnosis Using AI

Take pictures of leaves with the ESP32-CAM.

Use a trained CNN model to forecast plant diseases, then present the findings via a Streamlit web interface.

3. Integration of Mobile Robots

Create a robotic car with Wi-Fi by utilizing D1 Mini, DC motors, and motor drivers.

Give the robot the ability to move and carry the arm for dynamic, mobile interaction in a variety of settings.

4. Development of User Interfaces

Create two user interfaces:

Streamlit for disease detection in real time.

ESP32 feed, servo motors, and robot movement can all be controlled via buttons on the Tkinter GUI.

5. Integration of Real-Time Systems

Create a closed-loop automation system by connecting robotic hardware and AI predictions.

To control motors using gesture inputs, use serial communication with an Arduino.

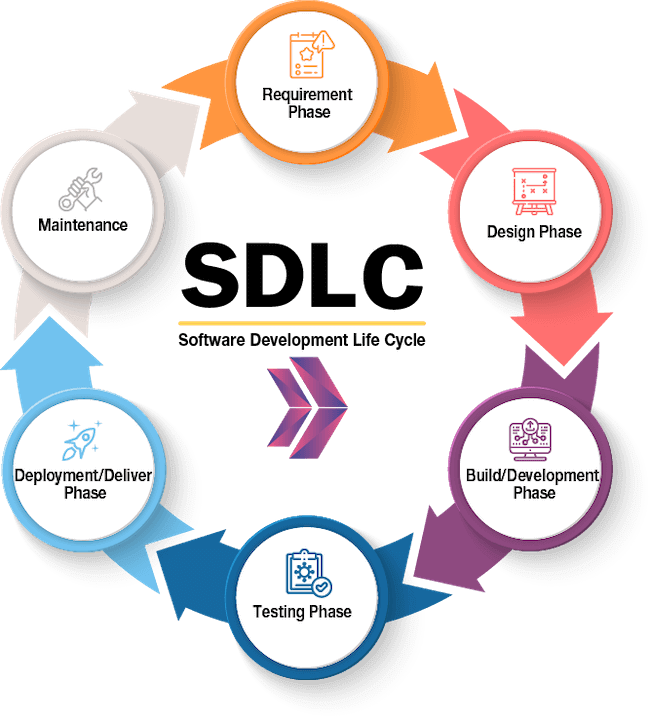
6. Use in Education and Prototyping

Give a working model that can be applied to research on AI-embedded systems, robotics education, and academic projects.

**-: SOFTWARE DEVELOPMENT LIFE CYCLE: -**

A software life cycle model (also known as a process model) is a visual and diagrammatic depiction of the software life cycle. A life cycle model depicts all of the steps required to move a software product through its life cycle stages. It also captures the structure in which these techniques will be implemented.

In other words, a life cycle model depicts the numerous actions carried out on a software product from inception until retirement. Different life cycle models may organize the necessary development activities into phases in different ways. Thus, regardless of the life cycle model used, the key activities are present in all life cycle models, even if the actions are performed in various orders. Multiple activities may be carried out at any point of the life cycle.



**REQUIRMENT PHASE:** -

**Hardware Requirements**

Robotic Arm System

Arduino UNO / Nano – Microcontroller to control servo motors.

Servo Motors (SG90/MG996R) – Used for robotic arm movement.

Robotic Arm Frame – Structural support for the arm and servo mounting.

Power Supply (Battery or Adapter) – Powers the servo motors and Arduino.

Connecting Wires, Breadboard – For circuit assembly.

Plant Monitoring System

ESP32-CAM – For capturing and streaming real-time leaf images

Custom Mount/Stand – To hold leaves in front of the ESP32-CAM.

Mobile Robotic Car Platform

D1 Mini (ESP8266) – Wi-Fi-enabled microcontroller for wireless control.

L298N Motor Driver Module – Controls speed and direction of DC motors.

DC Geared Motors (x2 or x4) – Provide forward, reverse, and turning motion.

Wheels (x2 or x4) – Attached to motors to enable movement.

Caster Wheel (optional) – For added balance and turning support.

Chassis Frame (Acrylic/Metal) – Base structure for mounting all components.

Battery Pack / Rechargeable Li-ion Batteries – Powers motors and control board.

Jumper Wires, Switches, Mounting Screws – For electrical connections and assembly.

**Software Requirements**

Python 3.x – Core development language.

Arduino IDE – For programming microcontrollers.

Streamlit – Web-based interface for plant disease prediction.

Tkinter – Desktop GUI for hardware control and module selection.

Non-Functional Requirements

Processor: Multi-core CPU (Intel i5 / AMD Ryzen 5 or better recommended for smooth inference and execution)

RAM: Minimum 16 GB to handle deep learning model loading, video processing, and simultaneous UI operations.

Storage: At least 100 GB free for storing model files, project data, captured images, and logs.

Graphics: Dedicated GPU (preferably NVIDIA with CUDA support) is optional for inference but essential for local model training.

Operating System: Compatible with Windows 10/11, macOS Ventura+, or Linux (Ubuntu 20.04+).

Python Libraries Used

GUI and Image Handling

tkinter, ttk, messagebox – Used to build the GUI, handle button events, and display alerts.

PIL (Image, ImageTk, ImageDraw) – For image loading, manipulation, and displaying images in the GUI.

Computer Vision and MediaPipe

cv2 (OpenCV) – Used for capturing video streams, processing frames, and integrating with MediaPipe for gesture recognition.

mediapipe – Google's library for real-time hand tracking and gesture recognition.

google.protobuf.json\_format.MessageToDict – Converts MediaPipe detection outputs into dictionary format for easier use.

Concurrency and Time Handling

threading – To run camera feeds and processing in parallel with the UI.

time, datetime – For timestamps and time-controlled operations (e.g., delays).

Communication serial – Enables serial communication between Python and Arduino for motor control.

requests – Used for sending HTTP requests, possibly between local servers (like from Tkinter to Streamlit).

webbrowser – Opens the Streamlit UI or other web pages from the Tkinter interface.

System Utilities

os, sys – Handle file paths, environment variables, and system-level operations.

numpy – Handles array processing and image manipulation where needed.

**SYSTEM DESIGN**: -

Streamlit: Likely for a web-based interface, perhaps for displaying real-time plant disease detection results, bot status, or offering more complex controls that can be accessed from any device with a browser. The design would involve laying out widgets (buttons, sliders, image displays) and structuring the data flow for updates.

Tkinter: Potentially for a desktop-based, more direct control application, perhaps for initial setup, calibration, or situations where a simpler, embedded GUI is preferred. The design would involve the layout of windows, buttons, text fields, and how they trigger actions on the robot.

**IMPLEMENTATION PHASE:** -

Python installation:

 **Download Installer:**

• Go to the official Python website: https://www.python.org/downloads/

• Click the "Download Python X.X.X" button (choose the latest version).

 **Run the Installer:**

• Double-click the downloaded .exe file.

• Check the box that says "Add Python to PATH" at the bottom of the installer window.

• Click "Install Now".

 **Verify Installation:**

• Open Command Prompt (Win + R, type cmd, and press Enter).

• Type: python –version

Anaconda Installation:

 **Download Installer**:

• Go to https://www.anaconda.com/products/distribution

• Click "Download" and select the Windows version (choose 64-bit Graphical Installer for most systems).

** Run the Installer:**

• Double-click the .exe installer you downloaded.

• Click "Next" → Agree to the license.

• Choose "Just Me" (recommended).

• Select the installation location (default is fine).

• Check the box for “Add Anaconda to my PATH environment variable” (optional but helpful).

• Click "Install".

 **Verify Installation:**

• Open Anaconda Navigator from the Start menu.

• Or open Anaconda Prompt and type: conda –version

How to set environment in Anaconda:

1. Create a New Environment

You can create an environment with a specific Python version or leave it to default.

Command: conda create --name myenv

2. With a specific Python version (e.g., Python 3.10):

Command: conda create --name myenv python=3.10

3. Activate the Environment

Command: conda activate myenv

4. Install Packages into the Environment

Once the environment is activated, you can install packages like this:

Command: conda install numpy pandas matplotlib

Or install using pip inside the environment:

Command: pip install flask

During the project's implementation phase, the software and hardware components were developed concurrently in order to translate the intended system architecture into a functional prototype. Because the system incorporates a number of technologies, such as robotic hardware, AI-based disease prediction, gesture control, and user interfaces, implementation was done in stages, with each module being created, tested, and improved before complete integration.

The robotic arm was first linked to an Arduino microcontroller, and OpenCV and MediaPipe in Python were used to recognize gestures. The arm was able to replicate human hand movements by mapping the real-time coordinates of hand landmarks to particular servo motor angles. For the industrial and surgical modules, where accuracy and responsiveness were essential, this configuration was essential. Concurrently, a convolutional neural network (CNN) was trained on a dataset of plant leaf photos in order to detect diseases as part of the farming module. After that, Streamlit was used to host this model, offering a straightforward and engaging web interface.

An ESP32-CAM was set up to stream video in order to provide real-time leaf images. This module created a closed-loop system between AI and hardware by programming the robotic arm to pick a leaf and place it in front of the camera for analysis. Additionally, the team used a D1 Mini microcontroller, DC motors, and a motor driver to construct a mobile robotic base.

The robotic arm was able to move and carry out tasks in various physical locations thanks to this mobile platform. The car was controlled remotely via Wi-Fi, which was seamlessly integrated with the rest of the system.

A dual-interface strategy was used for user interaction: a desktop GUI based on Tkinter handled the hardware controls, while the Streamlit web application was utilized for disease prediction. In order to ensure user-friendly operation, this interface featured buttons for controlling servos, starting the camera feed, and switching between different bots. In order to guarantee solid integration and dependable performance.

**TESTING PHASE:** -

To guarantee the performance and dependability of the integrated robotic system, the testing phase was essential. Each component was tested separately prior to complete system integration because the project included several interconnected modules, including servo motor control, AI-based disease prediction, mobile robot movement, and gesture recognition. The accuracy and real-time responsiveness of the gesture tracking module were assessed by examining how well the robotic arm replicated hand movements. Servo motors were adjusted to react accurately and without jitter or delay to various gestures.

To make sure the CNN model could accurately classify leaf conditions, testing for the plant disease model concentrated on prediction accuracy using both static and live images taken from the ESP32-CAM. The robotic car's ability to move without obstacles, maintain Wi-Fi connectivity, and control direction was tested. Integration testing was done to evaluate the modules' interactions after each one passed its own tests. For example, the system was seen to execute a full cycle in real-time, including identifying a gesture, initiating arm movement, taking a picture of a leaf, forecasting illness, and displaying the results. Iteratively resolving any delays, misunderstandings, or hardware irregularities discovered during this phase produced a system that was well-synchronized, operational, and effective.

**DEPLOYMENT PHASE: -**

Setting up the entire system to function properly on local hardware and software environments was the main task of the project's deployment phase. The user interfaces were successfully hosted on local computers, enabling complete system functionality during testing and demonstration, despite not being deployed on public platforms or cloud servers. A local server was used to launch the Streamlit-based plant disease prediction model, allowing users to view the web interface in a browser on the same network. This enabled smooth communication between the locally hosted deep learning model and the ESP32-CAM, which streamed live leaf images.

The desktop GUI based on Tkinter was simultaneously installed on the host computer as a Python application. It functioned as the main control center for the mobile robot interface, gesture tracking module, and robotic arm. Because the deployment was made to run offline, it can be used for development, classroom settings, and lab demonstrations without requiring internet access.

The system is designed for future scalability even though it hasn't been put online yet. To enable remote control, cloud-based model updates, and real-time worldwide accessibility, plans call for turning the local setup into a mobile or web-based application and deploying it on a distant server.

**MAINTENANCE PHASE**: -

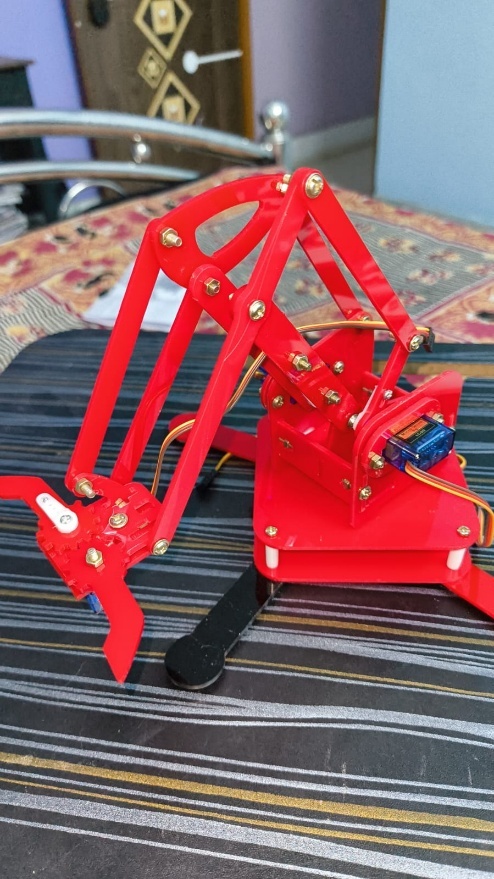
After deployment, the maintenance phase starts to make sure the system keeps working well over time. It entails resolving hardware problems like servo failures or sensor malfunctions, updating the software (such as enhancing the plant disease model), and repairing bugs. Performance adjustments and UI improvements may also be implemented as user input is obtained. Future objectives for this project, such as upgrading components or launching the user interface online, will be covered by adaptive maintenance. All things considered, this stage guarantees the integrated robotic and AI system's long-term dependability, scalability, and improvement.

System Working and Visual Representation:-

* System Workflow Overview

Robotic arm workflow:

The project creates a smooth system by combining several modules, including mobility, AI prediction, and robotic control. One of the three primary bots—Farming Bot, Surgery Bot, or Industrial Bot—is chosen by the user as they interact with a Tkinter-built GUI. The system turns on the appropriate functionality based on the selection.



The aforementioned image displays the specially constructed robotic arm that uses hand gestures to precisely control motion. The arm is made of sturdy yet lightweight acrylic parts that are fastened together with metal bolts and fixed to a sturdy base. Its numerous SG90 servo motors allow for movement in a number of axes, such as gripper control, elbow lift, and base rotation.

An Arduino UNO, which serves as the main motor controller for this robotic arm, is integrated into it (not seen in the picture). It is connected to a laptop through serial communication. Two essential programs are running on the laptop:

**MediaPipe Python Code**: Uses Google's MediaPipe library to detect both hands by capturing real-time webcam input. In order to map particular gestures or positions to servo motor angles, the hand landmarks are processed.

**How Arduino Connects to and Controls Servo Motors:** Signal and Power Wiring Three wires make up each servo motor: signal, ground (GND), and power (VCC).

The Arduino's 5V and GND pins (or an external power source if necessary) are connected to the power and ground.

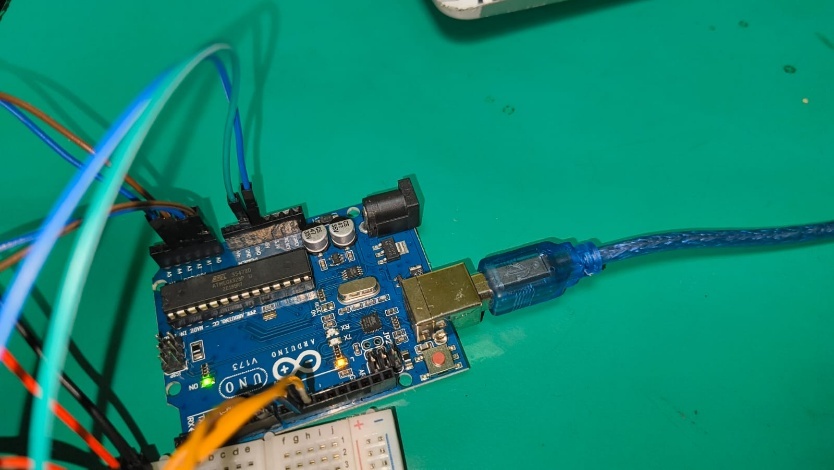
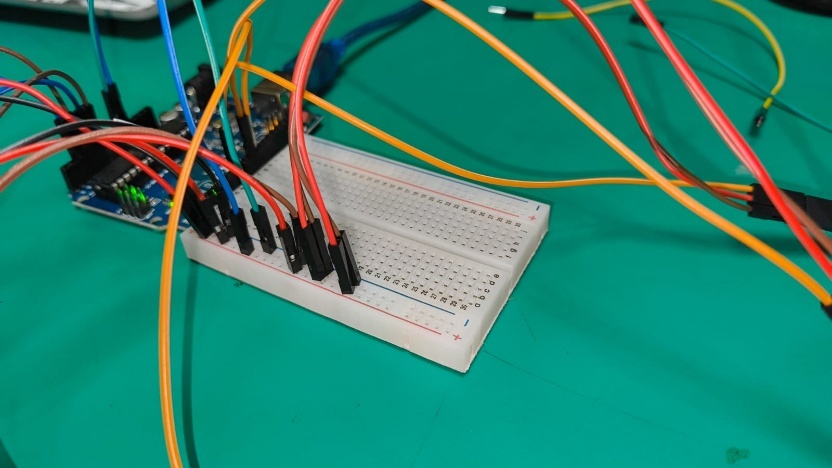
A PWM-capable pin on the Arduino, such as D3, D5, D6, etc., is where the signal wire is connected.Python-Based Serial Communication

For incoming data, the Arduino listens to its serial port (COM5, for example).

Python uses serial bytes to transmit servo angles (such as [90, 45, 120, 180]) over USB.

Getting and Interpreting Data

The serial data is continuously read by the Arduino.It parses the entire set of angles after receiving them (four angles for four servos, for example).



**Connections Overview:**

Power (Red wires):

* 5V pin of Arduino → Breadboard + rail
* Servo VCC (red wire) → Breadboard + rail

**Ground (Brown/Black wires):**

GND pin of Arduino → Breadboard – rail

Servo GND (black/brown wire) → Breadboard – rail

**Signal (Orange/Yellow wires):**

Each servo has a signal wire (usually orange or yellow), connected to specific digital pins on the Arduino.

Orange wires from servos go to:

Pin 3

Pin 5

Pin 6

Pin 9

Pin 10

These are commonly used PWM-capable pins for servo control.

**What Takes Place in Arduino Code**: The Servo library is used in the code to declare each pin (such as pin 3, pin 5, etc.) as a servo signal pin.  
Through those pins, the Arduino transmits PWM signals, which the servo motors decipher as instructions for angle.

**In our project**:

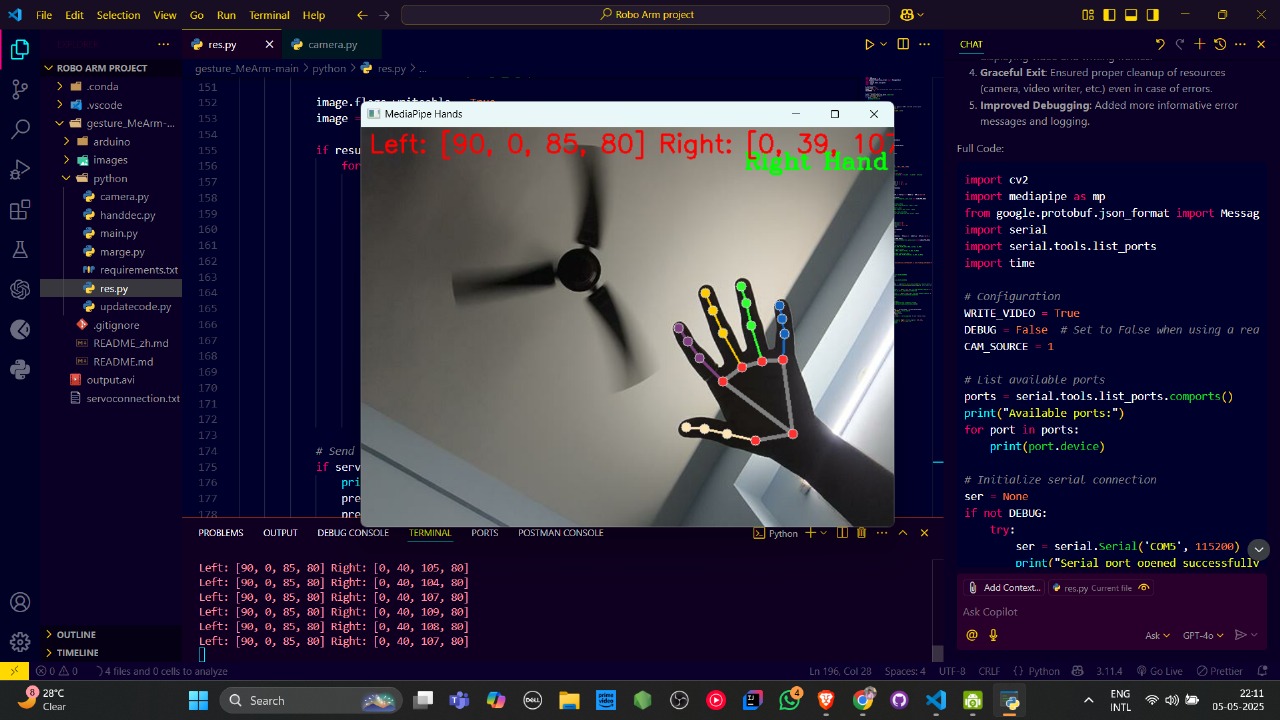
The Arduino's pin 5 is linked to the Low Base Servo (X). This regulates the robotic arm's base tilt or rotation close to the bottom.  
Pin 6 is connected to the left servo (Y). The left-side joint or movement is probably controlled by this.  
Pin 7 is connected to the right servo (Z), which is in charge of the right-side joint or comparable motion**.**

Pin 8 is connected to the Hand Servo (C), which is usually used to move the robot's hand portion or open and close the gripper.  
Every servo gets:  
a command from the designated digital pin on the Arduino (5, 6, 7, 8).  
The Arduino's 5V and GND pins supply power (5V) and ground (GND) to the breadboard's power rails.

**Interpretation of hand gestures to generate servo commands:**

Python, OpenCV, and MediaPipe are used to create a full hand-tracking-based servo control system for a dual-arm robotic setup. It specifies different angles and configurations for two sets of servos, which stand in for the left and right robotic arms. In order to identify hand landmarks for both hands, the system uses a webcam to record live video and MediaPipe to process it. In order to estimate servo angles for the X, Y, Z and C axis and to regulate the opening or closing of the claws based on fist detection, it computes wrist positions and palm sizes. To ensure exact control, the servo angles are proportionately mapped from the user's hand position and movements.

If the system is not in debug mode, a serial connection is established, and commands for the servo angle are transmitted to the Arduino through this serial port. Additionally, the video feed is flipped, shown with landmarks drawn on it, and can be saved as a video file if desired. It scales the output image for GUI display in a Tkinter label and has logic to refresh the video capture if the frame is empty. The interface updates the video label without stopping the main thread, and the servo control only makes changes when they are detected. It is appropriate for real-world robotic arm applications because it employs threading and graceful error handling for serial communication.



Car workflow through wifi and esp32 cam module integration:

**1.Motor Driver (Red Board with Heat Sink)**

Purpose: Drives the DC motors by controlling direction and speed.

Connections:

* Blue terminals (left/right) for motor outputs (M1 and M2).
* Middle blue terminals: power (VCC), ground (GND), and 12V (motor power).
* Input pins connected via jumper wires (blue ribbon cable) to d1 mini.

**2.D1 mini (Blue Microcontroller Board)**

Receives Commands (via Wi-Fi)

It connects to Wi-Fi and receives control signals — either from a mobile app, web interface, or programmed logic (like obstacle avoidance or line following).

Controls the L298N Motor Driver

Based on the commands, it sends digital HIGH/LOW signals to the IN1–IN4 pins of the L298N motor driver to control.

Power & Communication It manages logic-level power (3.3V), and can also be used to print data on serial monitor (for debugging via USB).

**3.Yellow Gear Motors (4x)**

Purpose: DC motors connected to wheels; convert electrical energy to mechanical motion.

Voltage: Typically 3–12V.

Connected to: Motor output terminals of L298N.

Wheels (Black with Yellow Center)

Attached to gear motors to move the robot.

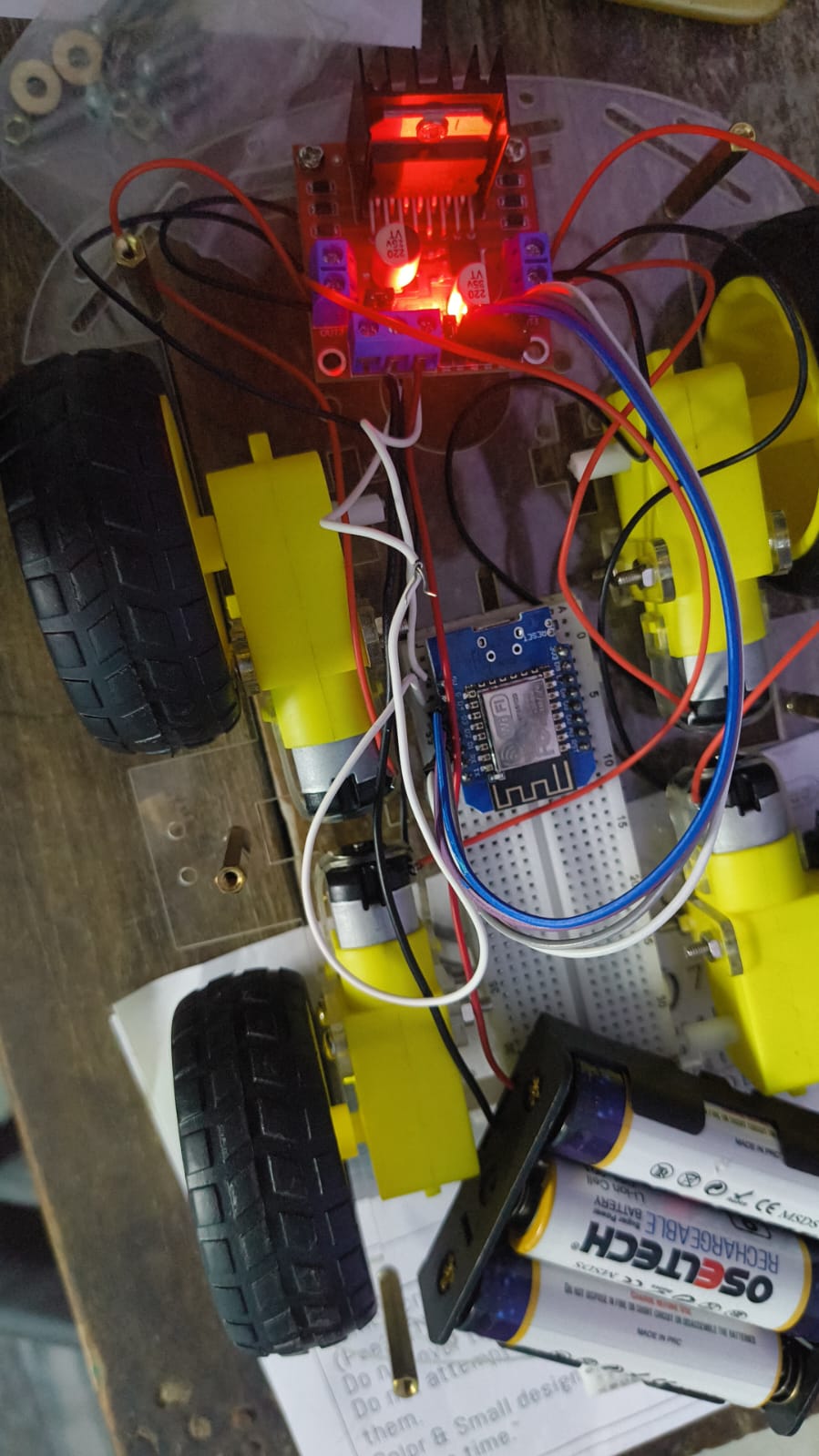
**4.Battery Pack (Black Holder with 4x AA Batteries)**

Purpose: Powers the motors (via L298N) and possibly the ESP8266.

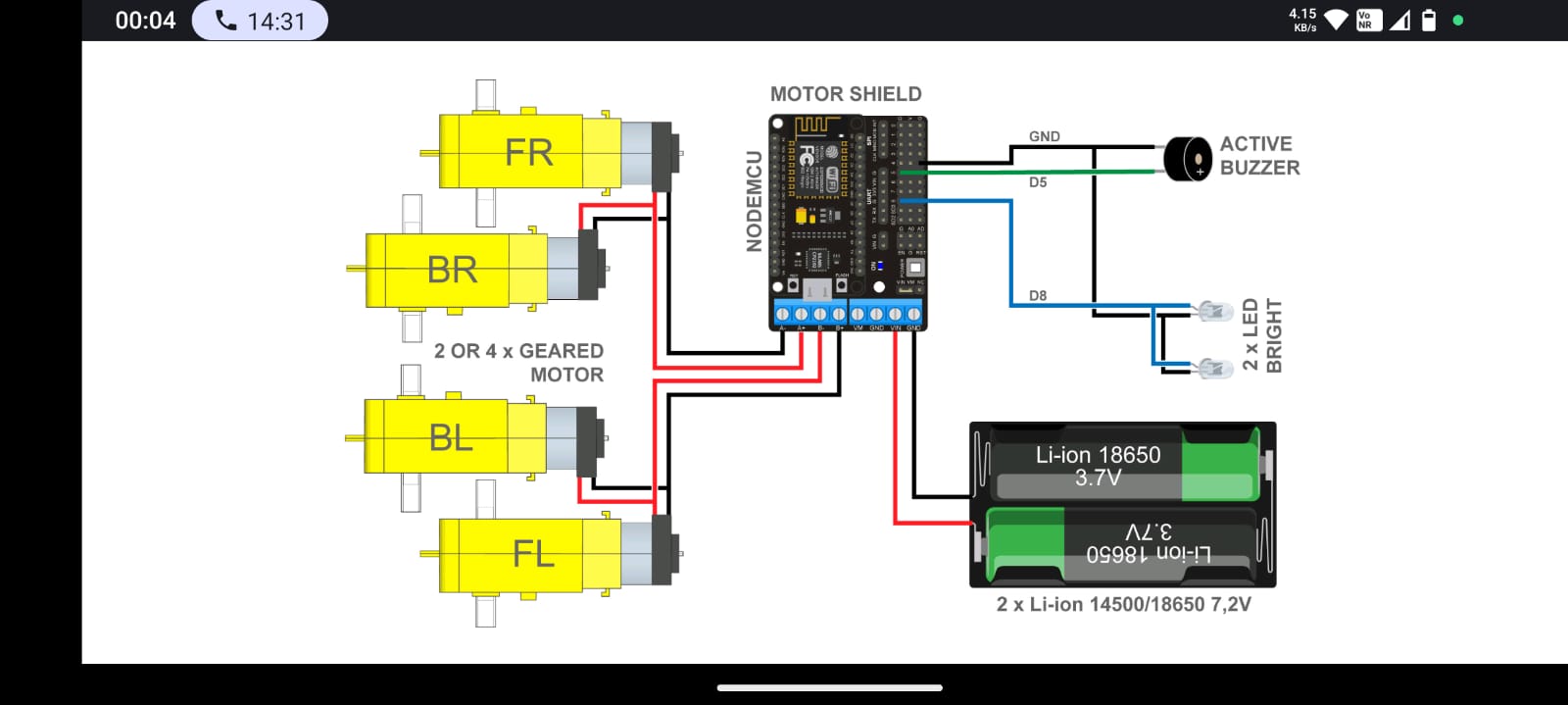
Batteries used: Rechargeable AA 1.2V each (likely giving ~4.8V total).

Chassis Plate (Transparent Acrylic)

Purpose: Base structure that holds all components.



**The Circuit Diagram**



**Live video streaming through Esp32 cam module:**

**Hardware Requirements for ESP32-CAM Live Streaming**

* ESP32-CAM module (AI-Thinker)
* ESP32-CAM MB programmer (USB adapter board)
* Micro-USB cable (for programming via laptop)
* Laptop or PC with Arduino IDE
* Wi-Fi network (2.4 GHz only, no 5GHz support)

***Step-by-Step Procedure for ESP32-CAM Live Streaming***

**1. Hardware Connections**

* Insert the ESP32-CAM module into the ESP32-CAM MB board (as shown in your image).
* Connect the MB board to your laptop via a micro-USB cable

**2. Install the ESP32 Board in Arduino IDE**

Open Arduino IDE.

* Go to:

File > Preferences > Additional Board Manager URLs

Add this URL:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\_esp32\_index.json

* Then go to:

Tools > Board > Board Manager

Search and install: "esp32" by Espressif Systems

**3. Load the Camera Web Server Example**

* Go to:

File > Examples > ESP32 > Camera > CameraWebServer

* In the code:

Uncomment the line: #define CAMERA\_MODEL\_AI\_THINKER

Comment out all other #define CAMERA\_MODEL\_\* lines

* Set your WiFi credentials:

const char\* ssid = "YOUR\_WIFI\_NAME";

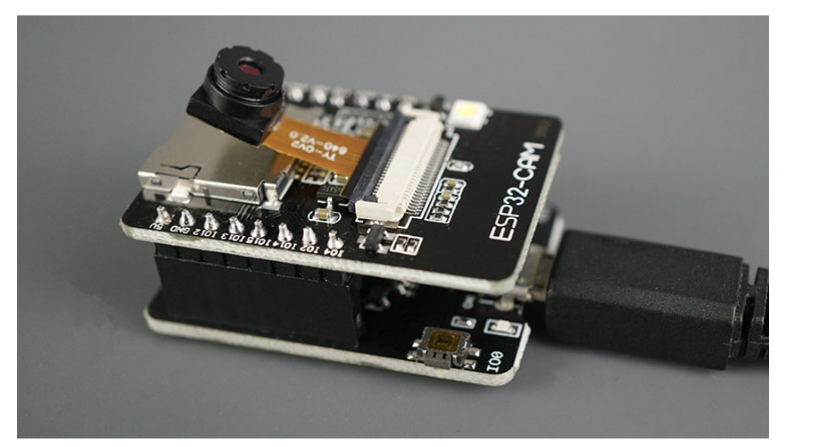
const char\* password = "YOUR\_WIFI\_PASSWORD";

**4. Configure Board and Port**

Go to:

Tools > Board > ESP32 Arduino > AI Thinker ESP32-CAM

Select the correct COM port under Tools > Port.

**5. Put ESP32-CAM into Flash Mode**

Press and hold the IO0 (BOOT) button.

While holding IO0, press and release the RST button.

Then release IO0.

**6. Upload the Code**

Click the Upload button in Arduino IDE.

Wait for "Done Uploading".

**7. Reboot to Run Mode**

After uploading, press only the RST button (don’t press IO0).

**8. Get the IP Address**

Open Serial Monitor (set baud rate to 115200).

You will see a line like:

Camera Ready! Use the following URL to connect: http://192.168.1.x

**9. View Live Video Stream**

Open a browser and enter the IP (e.g., http://192.168.1.10)

Click "Start Stream"

UI PART AND STREAMLIT WEB APP WORKFLOW:

The "Bot Selector" is a desktop application that provides a graphical user interface (GUI) for selecting and managing different robotic systems. Upon launching, users are presented with a main selection screen where they can choose which type of bot they wish to interact with. Each bot type (Farming, Surgery, Industrial) leads to a dedicated control panel with specific functionalities tailored to its domain.

**Tkinter:**

This Ui is built by Tkinter a python built in module it is used to create graphical desktop applications. You can create interactive windows with ease thanks to its widgets, which include buttons, labels, text boxes, menus, and more.

**Essential Elements of Tkinter:**Easy to use and lightweight: It doesn't require external installations and is simple for novices.  
Cross-Platform: Compatible with Linux, macOS, and Windows.  
Label, Button, Entry, Text, Frame, Canvas, Checkbutton, Radiobutton, Listbox, Menu, and more are among the available widgets.  
Programming that reacts to user events, such as keystrokes or clicks, is known as event-driven programming.

**Common Use Cases:**

* Small desktop apps
* GUI frontends for automation scripts
* Educational tools
* Robotics control panels
* Image capture interfaces (e.g., with OpenCV)

**Basic Structure**

import tkinter as tk

root = tk.Tk()

root.title("My Tkinter App")

label = tk.Label(root, text="Hello, Tkinter!")

label.pack()

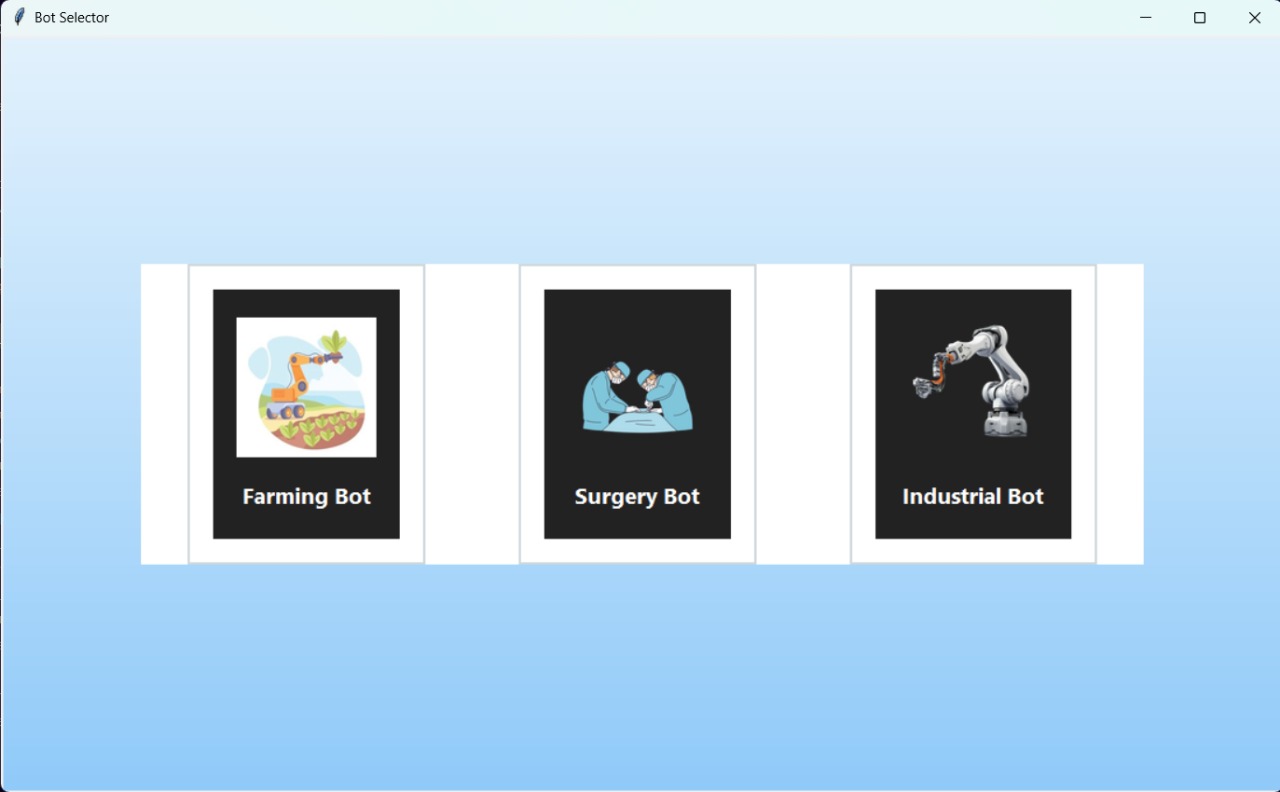
button = tk.Button(root, text="Click Me", command=lambda: print("Button Clicked"))

button.pack()

root.mainloop()

**Key features of our UI Interface**

* *Easy Bot Selection*: Each type of bot has its own buttons on a neat, eye-catching main screen.
* *Modular Control Panels*: Distinct, customized user interface designs for every bot, guaranteeing a targeted user experience.
* *Real-time Feedback*: To give visual feedback on bot operations, live camera streams (ESP32-CAM and webcam) are integrated.
* *Interactive Control*: Robotic arm manipulation using hand gesture recognition as a novel input technique.
* The smooth launch of associated web-based applications (like Streamlit dashboards) straight from the user interface is known as external application integration.
* *Modern Aesthetics*: To improve the visual appeal, use gradient backgrounds and custom button styling.
* *Tooltips*: Educative tooltips on buttons to help users understand how they work.

**Main Bot Selection Screen:**

This interface serves as the "Bot Selector" application's primary selection screen. It acts as the first point of contact where the user selects the kind of robotic system they want to monitor or control.

This is our main interface where three sections are present

* Farming bot section
* Surgery bot section
* Industrial bot section

**Farming bot Section:**

Advanced features are available in the "Farming Bot" control panel, with a special emphasis on remote monitoring and control.

*Layout* *Divided into three main sections*: Action buttons are located in the left pane (controls).A button to open an external web application is displayed in the Center Pane (Bot Arm Stream & Website Link), along with the live feed from the bot arm.  
Right Pane (ESP32-CAM & Captured Images): Displays both a still image display and the live stream from an ESP32 camera.

**Control Buttons (Left Pane):**

* "Start Cam": Initiates a real-time video stream from an ESP32-CAM module (configured via ESP32\_CAM\_URL).
* "Stop Cam": Halts the ESP32-CAM video stream.
* "Capture Img": Captures the current frame from the ESP32-CAM stream, saves it to the user's "Downloads/CapturedImages" directory, and displays it in the "Captured Image" area.
* "Start Handcam": Activates the hand gesture recognition system, allowing users to control the robotic arm using hand movements.
* "Stop Handcam": Deactivates the hand gesture recognition system.

**Bot Arm Stream (Center Pane):**

* A large display area labeled "Bot Arm Stream" where the live video feed from the camera used for hand tracking is shown. This stream visually represents the environment the robotic arm is operating in or the arm itself.

**Website Integration (Center Pane):**

* "Go to Website": A button that opens a predefined Streamlit application URL (STREAMLIT\_URL) in the user's default web browser. This is intended for accessing external dashboards or ML model prediction interfaces relevant to the farming bot.

**ESP32 Live Cam (Right Pane):**

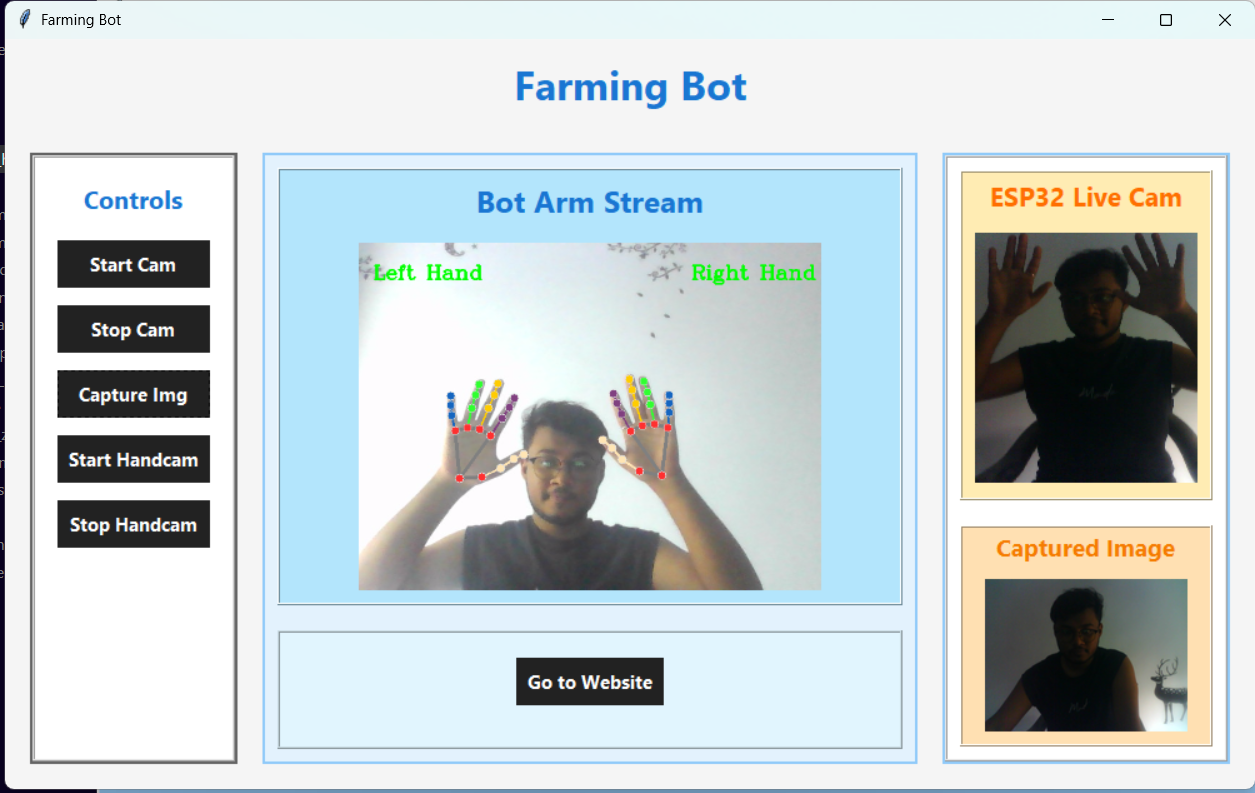
* Displays the real-time video feed from the ESP32-CAM module, providing remote visual monitoring of the farming environment.

**Captured Image (Right Pane):**

Shows the most recently captured image from the ESP32-CAM, allowing for quick review of snapshots.

When the Farming Bot window is closed, it automatically stops both the ESP32-CAM stream and the hand tracking system to conserve resources.

This is the Farming bot Interface.



**Surgery Bot & Industrial Bot Control Panels**

The control panels for the "Surgery Bot" and "Industrial Bot" have a similar simplified user interface, with a primary focus on hand gestures for controlling robotic arms.

Divided into two main sections:

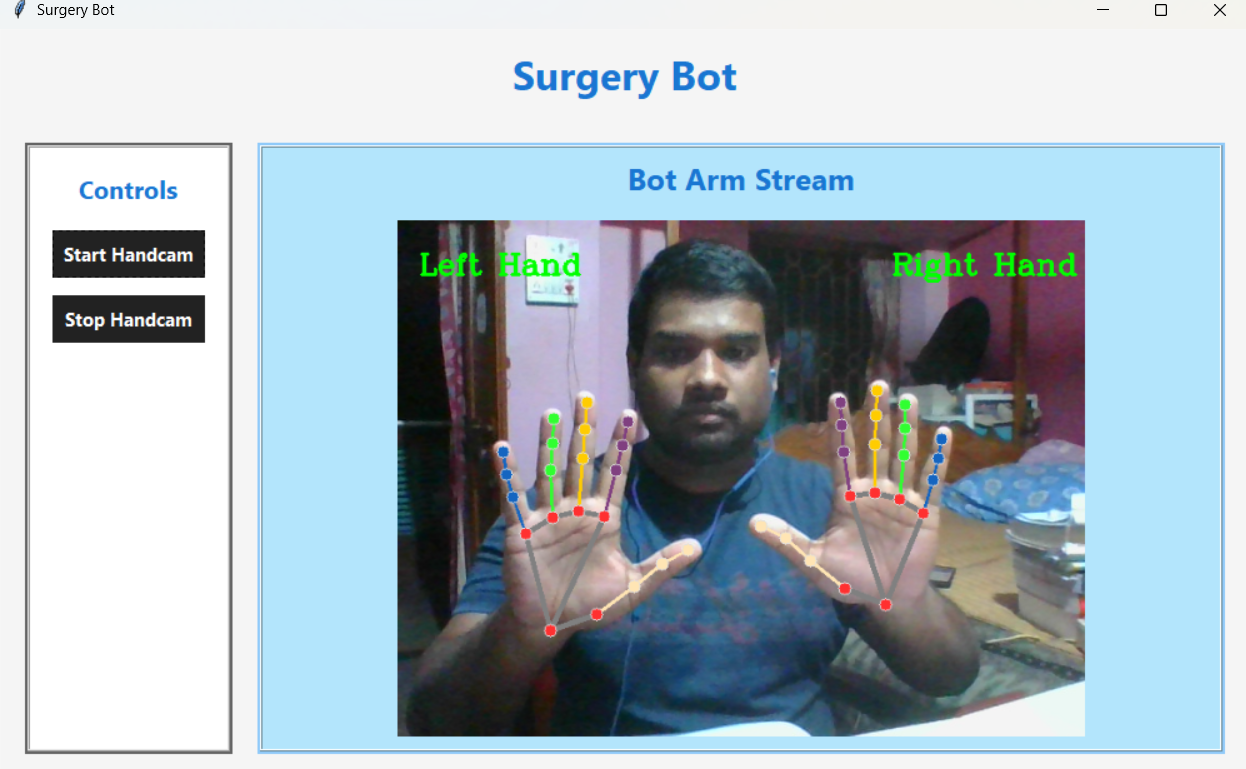
* Left Pane (Controls): Similar to the Farming Bot, this section houses action buttons.
* Right Pane (Bot Arm Stream): Displays the bot arm's live feed.

Control Buttons (Left Pane):

* "Start Handcam": Activates the hand gesture recognition system for robotic arm control.
* "Stop Handcam": Deactivates the hand gesture recognition system.

Bot Arm Stream (Right Pane):

* A large display area labeled "Bot Arm Stream" that shows the live camera feed used for hand tracking, providing visual feedback on the robotic arm's movements.



**Machine Learning For plant disease prediction and Streamlit Web app Integrated with Tkinter UI:**

Using a dataset, we trained a Convolutional Neural Network (CNN) to identify illnesses of plant leaves. The dataset was separated into folders for training and validation. To improve learning, images were normalized and shrunk to 128x128 pixels.

Three convolutional layers and max-pooling layers make up the model architecture. Important characteristics including leaf color, shape, and texture are extracted with the aid of these layers. Following feature extraction, the model makes predictions using a fully linked (dense) layer. To lessen overfitting, a dropout layer is applied before the final output. The Adam optimizer and categorical cross-entropy as the loss function were used to create the model. Training and validation accuracy increased steadily over a number of epochs (10).

**Training Phase**

**1.Dataset Loading:**

Loads images from train/ and valid/ directories. Likely uses ImageDataGenerator for preprocessing and augmentation.

**2.Image Preprocessing**:

Resize to a fixed shape (e.g., 128x128).

Normalize pixel values to [0, 1].

**3.Model Definition:**

A Convolutional Neural Network (CNN).

Layers: Conv2D → MaxPooling → Dropout → Dense → Output (Softmax).

Output: One class per disease.

**4.Compilation**:

Optimizer: e.g., adam.

Loss: categorical\_crossentropy or sparse\_categorical\_crossentropy.

Metrics: accuracy.

**5.Model Training**:

Runs for X epochs (e.g. 10).

Shows training and validation accuracy/loss per epoch.

**6.Model Saving:**

Saves the trained model, likely as:

plant\_disease\_recog\_model\_pwp.keras

**Testing Phase**

***1.*Model Load:**

model = tf.keras.models.load\_model("plant\_disease\_recog\_model\_pwp.keras")

***2.* Load Test Image:**

A single test image (or multiple from a folder).

Resize to (128, 128).

Convert to array, normalize, expand dimensions:

image = tf.keras.preprocessing.image.load\_img("test.jpg", target\_size=(128, 128))

input\_arr = tf.keras.preprocessing.image.img\_to\_array(image)

input\_arr = np.expand\_dims(input\_arr, axis=0) / 255.0

**3*.* Image Preprocessing**

The test image is first loaded and resized to 128x128 pixels, which matches the input shape used during training. It is then converted into a numerical array and expanded to form a batch of one image:

image = tf.keras.preprocessing.image.load\_img(image\_path, target\_size=(128, 128))

input\_arr = tf.keras.preprocessing.image.img\_to\_array(image)

input\_arr = np.array([input\_arr])

**4. Model Prediction**

The preprocessed image is passed to the trained CNN model using the predict() function. This returns a probability distribution over all disease classes:

predictions = cnn.predict(input\_arr)

**5. Result Interpretation**

The index of the highest probability value is identified using:

result\_index = np.argmax(predictions)

This index corresponds to a specific plant disease name, which is mapped using a list class\_name:

model\_predictions = class\_name[result\_index]

**6. Result Visualization**

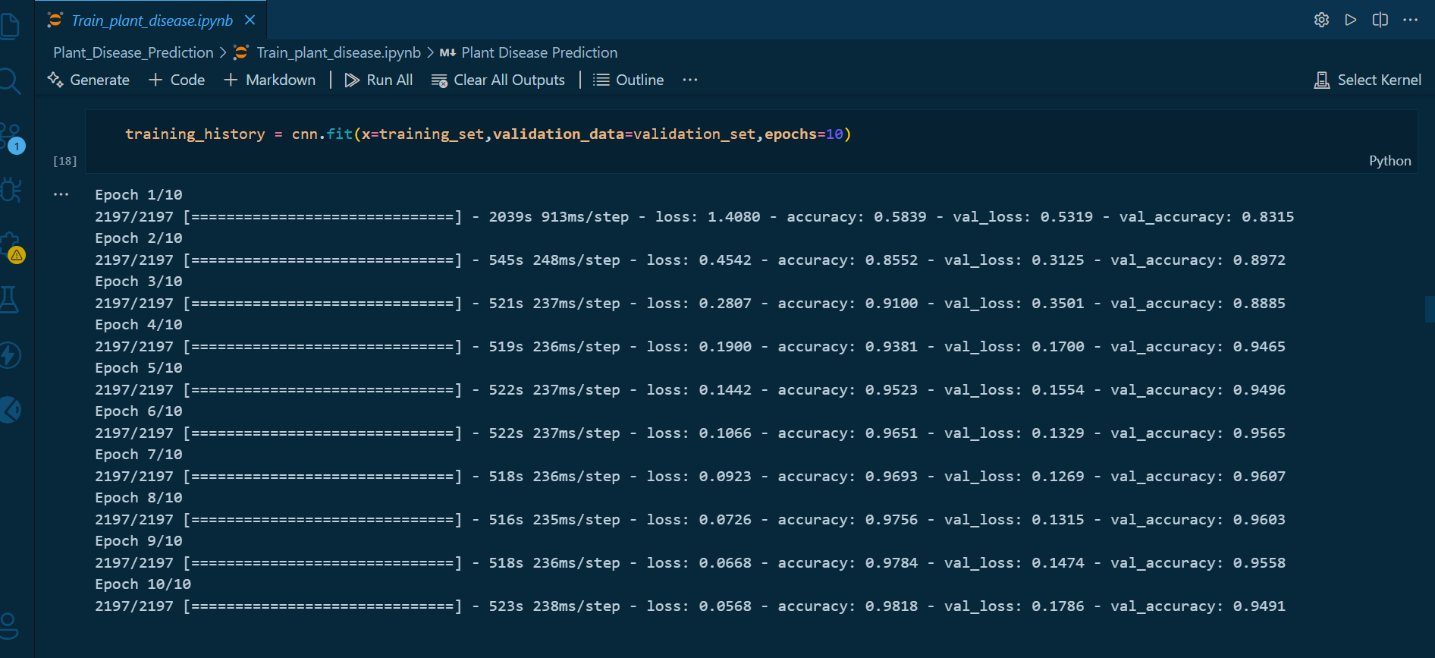
Finally, the original image is displayed using matplotlib, along with the predicted disease name:

plt.imshow(img)

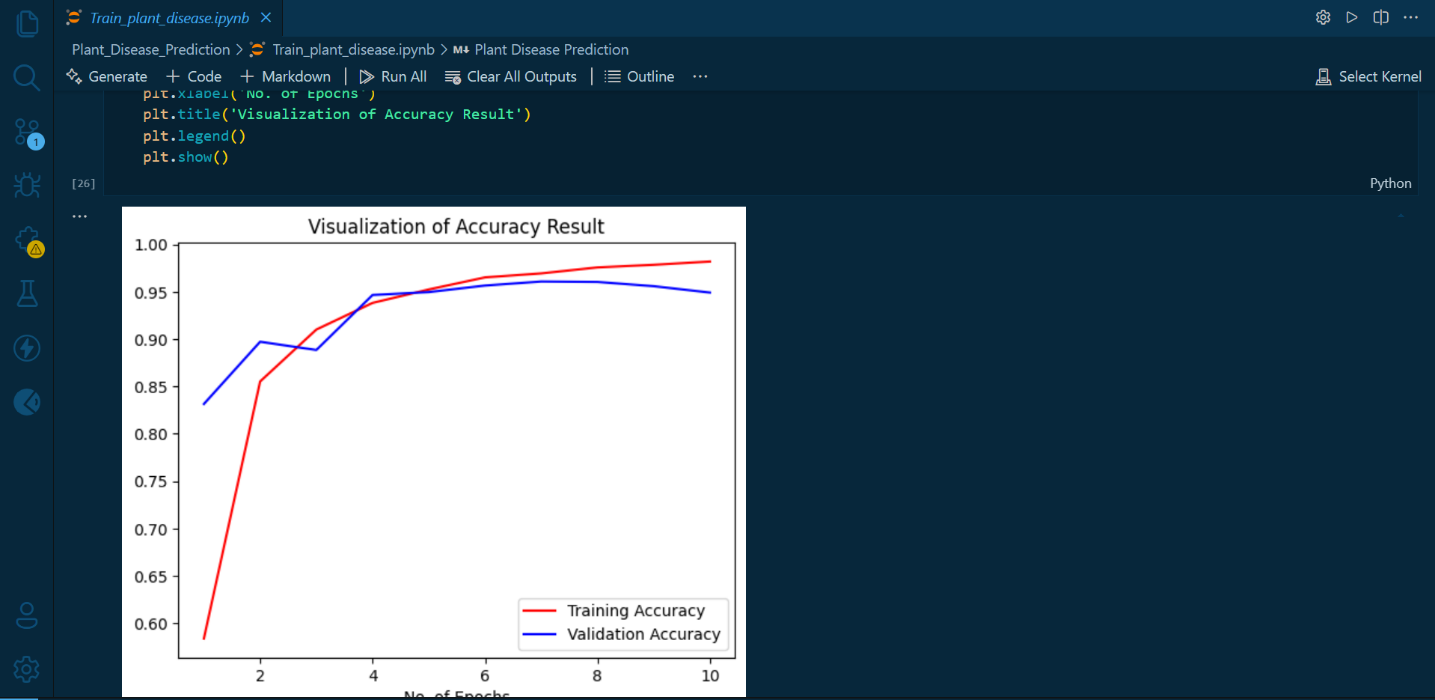
plt.title(f'Predicted: {model\_predictions}')

plt.xticks([]); plt.yticks([])

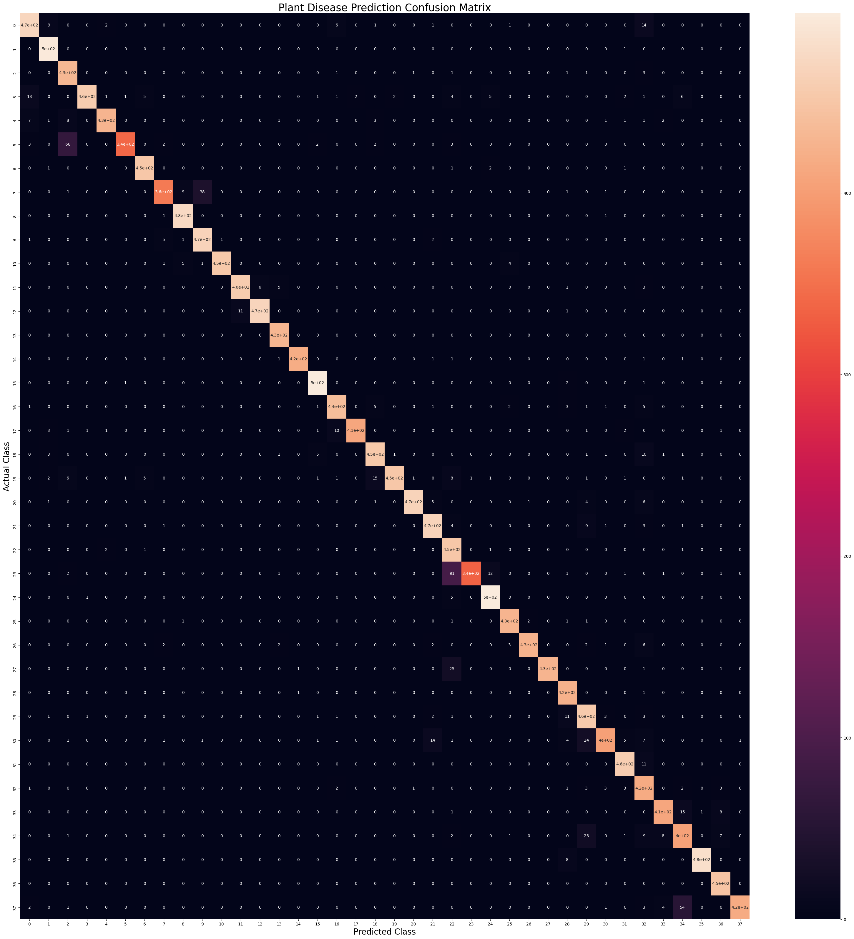
plt.show()



Training Accuracy of Our Model

Accuracy Result Visualization

Confusion Matrix



**Streamlit Web app for integrate with Tkinter User Interface:**

An open-source Python library called Streamlit enables programmers to easily and rapidly build unique web applications for data science and machine learning projects. By converting Python scripts into interactive web applications, it makes it possible to quickly prototype, visualize, and share data-driven insights and models.

The integration of Streamlit into our "Bot Selector" project offers several significant advantages, particularly for the Farming Bot given its explicit mention of a "Go to Website" button.

Streamlit excels at "turning data scripts into shareable web apps." This makes it an ideal platform for deploying and demonstrating machine learning models.

*Project Context*: A Streamlit application (STREAMLIT\_URL) is specifically referenced by the "Go to Website" button. This suggests that a machine learning model is probably going to be used in a crucial use case.

*Benefit*: Anyone with a web browser can access a Streamlit application once it has been deployed (for example, on a local network or cloud service), and no installation of Python or particular dependencies is required. This makes sharing and cooperation easier.

A Streamlit web application for advanced predictions (such as crop health and disease detection) can be accessed with ease by clicking the "Go to Website" button in the Farming Bot section.

The "Capture Img" button in the Tkinter UI does two things: it takes a live picture of the ESP32-CAM stream, saves it locally, and then automatically uploads it to the running Streamlit web application. This enables instantaneous AI-powered examination of the gathered visual data, fusing potent predictive models with in-the-moment physical observations for effective agricultural management.

Clicking "Go to Website" in the Farming Bot section takes users to a Streamlit web application that makes sophisticated predictions (like crop health and disease detection).

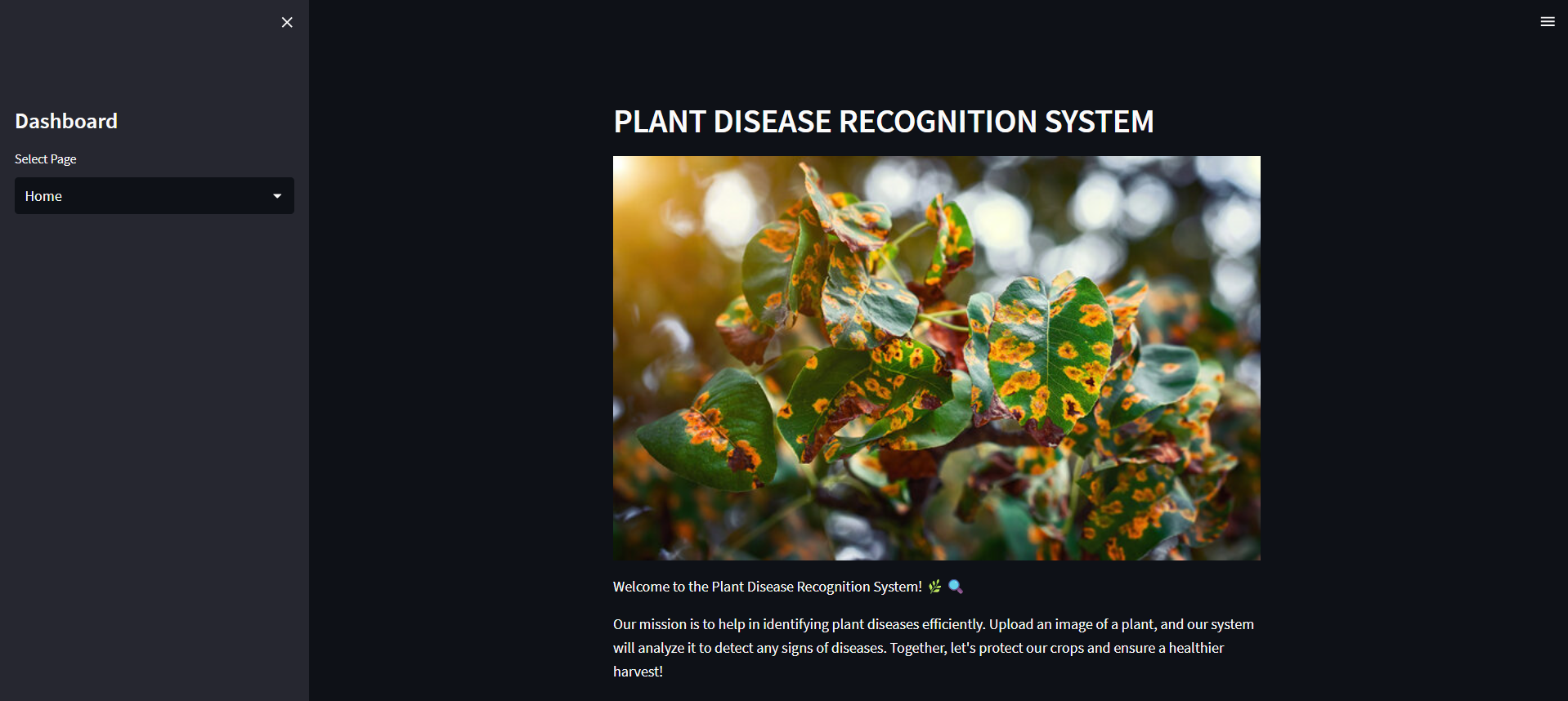
Simultaneously, a "Capture Img" button in the Tkinter user interface enables users to take a live picture of the ESP32-CAM stream, store it locally, and then easily upload it to the same Streamlit web application for instant model inference. Real-time visual data collection and immediate AI-driven analysis are made possible by this integration, which offers complete control and predictive capabilities for agricultural management straight from the user interface.

* The web app loads the trained model trained\_plant\_disease\_model.keras.
* Images are resized to 128x128 before prediction.

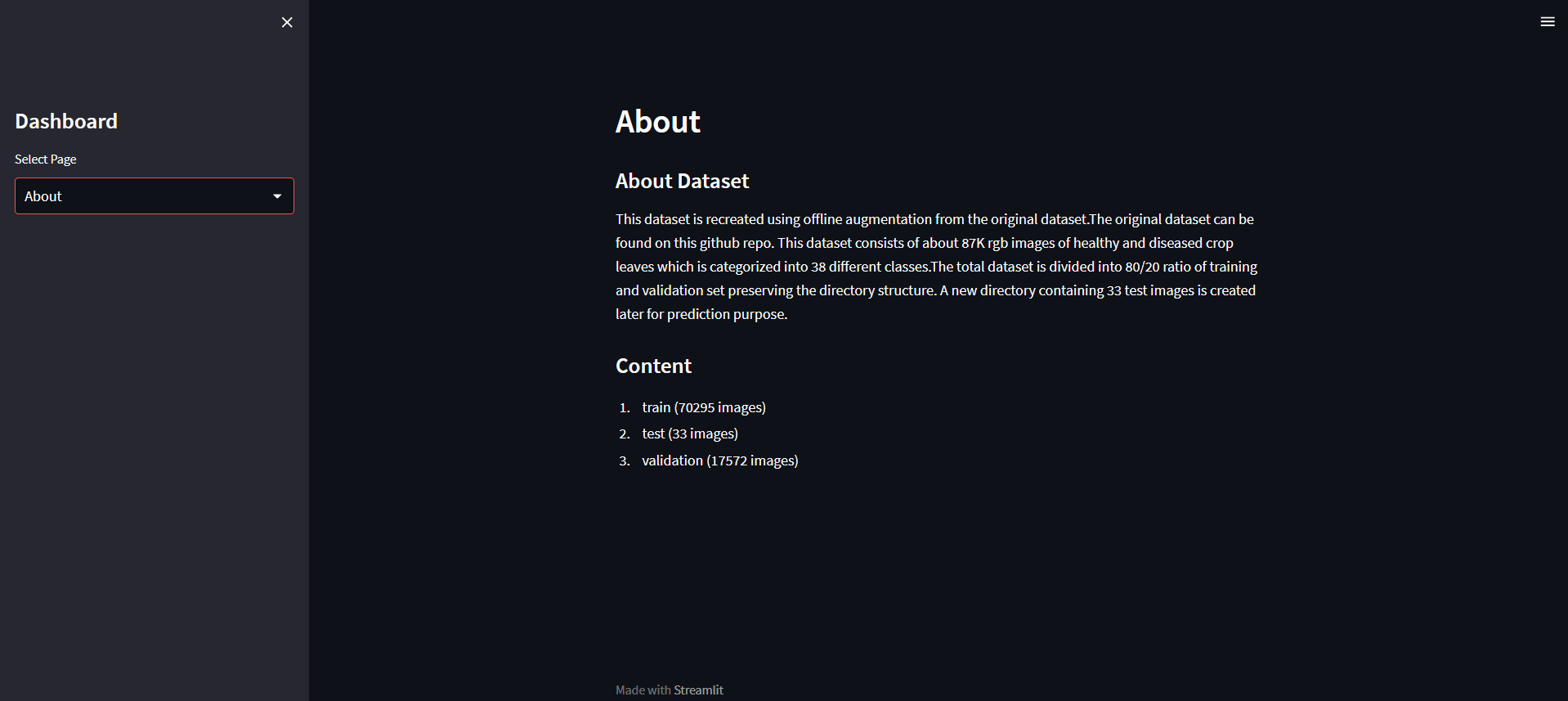
**Features of the Web App:**

*Homepage*

* shows the banner image and project title.
* explains the benefits, goal, and operation of the system.
* gives users precise instructions on how to use the app efficiently.

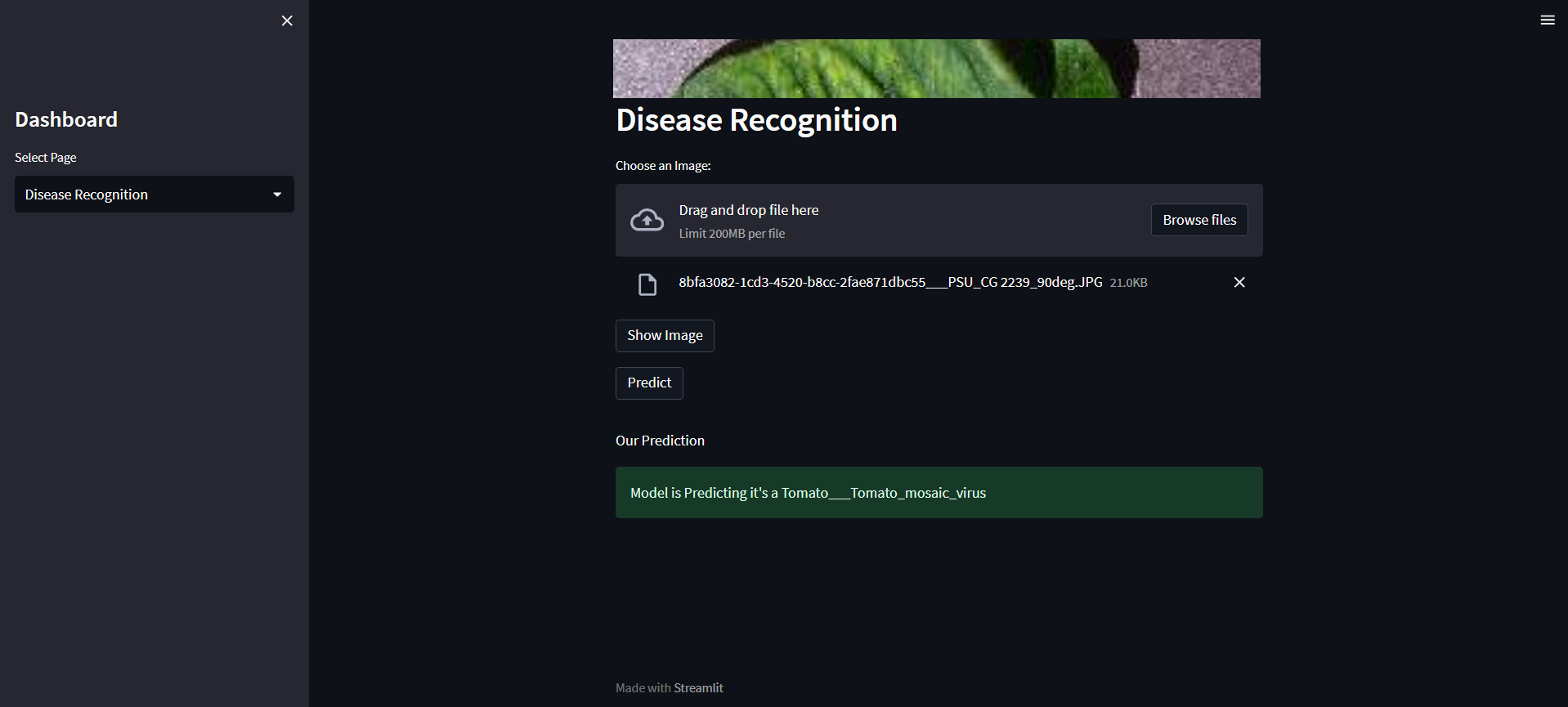


*About Page:*

* explains the dataset that was used to train the model.
* explains the distribution of the data:
* Instruction: 70,295 pictures
* Verification: 17,572 photos
* 33 pictures were tested.
* Brings up 38 plant disease classes were identified from the dataset, which was produced using offline image augmentation techniques.

**Disease Recognition Page**

* Allows users to **upload an image** of a diseased plant leaf.
* On clicking **“Predict”**, the image is preprocessed and passed to the trained CNN model.
* The model returns a prediction index, which is mapped to a readable disease name.
* The predicted class is displayed with a snow animation and success message.



**FEATURES OF THIS PROJECT:**

This project combines an articulated arm with a robotic car base to create a multipurpose smart robotic system. The car can move in all directions thanks to the ESP8266 NodeMCU, which is controlled by a WiFi-based Android app called the ESP8266 WiFi Robot Car app. An ESP32-CAM module, which offers live video streaming via WiFi, is mounted on the vehicle. The robotic arm has several servo motors and uses MediaPipe on a networked system to recognize hand gestures to carry out motion tasks. The ESP32-CAM is used to scan the leaf that the arm has acquired. To predict plant diseases, a trained machine learning model processes the captured image. The arm is also made to be versatile for industrial and surgical applications where visual inspection and precise movements are crucial. With the software handling the ML model, camera feed, and user interaction, the arm and vehicle can be controlled using both gesture input and an app-based user interface.

**1.Wi-Fi Controlled Smart Car**

The robotic car is powered by NodeMCU (ESP8266) and controlled via the ESP8266 WiFi Robot Car App.

Users can move the car forward, backward, left, right, and stop using smartphone controls over Wi-Fi.

**2. ESP32-CAM Live Video Streaming**

The robotic arm is equipped with an ESP32-CAM module.

It can be incorporated into the GUI or accessed via IP in a browser, streaming live camera feeds over Wi-Fi. used to take pictures of surgical sites or plant leaves in real time.

**3. Multipurpose multifunctional robotic arm**

* The car has an arm that is powered by a servo motor.
* Managed through the application and able to:
* selecting and positioning items (such as leaves for scanning).
* simulating tasks requiring surgical precision.
* carrying out industrial tasks like manipulating or sorting items.

**4. Control of Hand Gestures**

OpenCV and MediaPipe Hand Tracking were integrated in Python. The arm movements are wirelessly controlled by gestures. provides a touchless, user-friendly interaction model.

**5. Machine Learning-Based Disease Detection**

* Captured leaf images from the ESP32-CAM are fed into a trained ML model.
* The model detects and classifies plant diseases.
* This aids in smart farming applications and automated disease surveillance.

**6. A Web Application Based on Streamlight**

* With a Streamlit GUI, users can:
* Add photos that have been taken.
* See the ESP32-CAM feed in real time.
* Get results for disease prediction in real time.
* Visually observe how your arms move.

**ARDUINO IDE CODES: -**

**CAR-D1-CODES:**

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include <ArduinoOTA.h>

// connections for drive Motors

// Define GPIO pins for motor control using Wemos D1 Mini

const int PWM\_A = 14; // GPIO5

const int PWM\_B = 12; // GPIO4

const int DIR\_A = 0; // GPIO0

const int DIR\_B = 2; // GPIO2

const int buzPin = D5; // set digital pin D5 as buzzer pin (use active buzzer)

const int ledPin = D8; // set digital pin D8 as LED pin (use super bright LED)

const int wifiLedPin = D0; // set digital pin D0 as indication, the LED turn on if NodeMCU connected to WiFi as STA mode

String command; // String to store app command state.

int SPEED = 1023; // 330 - 1023.

int speed\_Coeff = 3;

ESP8266WebServer server(80); // Create a webserver object that listens for HTTP request on port 80

unsigned long previousMillis = 0;

String sta\_ssid = ""; // set Wifi networks you want to connect to

String sta\_password = ""; // set password for Wifi networks

void setup(){

Serial.begin(115200); // set up Serial library at 115200 bps

Serial.println();

Serial.println("WiFi Robot Remote Control Mode");

Serial.println("--------------------------------------");

pinMode(buzPin, OUTPUT); // sets the buzzer pin as an Output

pinMode(ledPin, OUTPUT); // sets the LED pin as an Output

pinMode(wifiLedPin, OUTPUT); // sets the Wifi LED pin as an Output

digitalWrite(buzPin, LOW);

digitalWrite(ledPin, LOW);

digitalWrite(wifiLedPin, HIGH);

// Set all the motor control pins to outputs

pinMode(PWM\_A, OUTPUT);

pinMode(PWM\_B, OUTPUT);

pinMode(DIR\_A, OUTPUT);

pinMode(DIR\_B, OUTPUT);

// Turn off motors - Initial state

digitalWrite(DIR\_A, LOW);

digitalWrite(DIR\_B, LOW);

digitalWrite(PWM\_A, LOW);

digitalWrite(PWM\_B, LOW);

// set NodeMCU Wifi hostname based on chip mac address

String chip\_id = String(ESP.getChipId(), HEX);

int i = chip\_id.length()-4;

chip\_id = chip\_id.substring(i);

chip\_id = "wificar-" + chip\_id;

String hostname(chip\_id);

Serial.println();

Serial.println("Hostname: "+hostname);

// first, set NodeMCU as STA mode to connect with a Wifi network

WiFi.mode(WIFI\_STA);

WiFi.begin(sta\_ssid.c\_str(), sta\_password.c\_str());

Serial.println("");

Serial.print("Connecting to: ");

Serial.println(sta\_ssid);

Serial.print("Password: ");

Serial.println(sta\_password);

// try to connect with Wifi network about 10 seconds

unsigned long currentMillis = millis();

previousMillis = currentMillis;

while (WiFi.status() != WL\_CONNECTED && currentMillis - previousMillis <= 10000) {

delay(500);

Serial.print(".");

currentMillis = millis();

}

// if failed to connect with Wifi network set NodeMCU as AP mode

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("");

Serial.println("WiFi-STA-Mode");

Serial.print("IP: ");

Serial.println(WiFi.localIP());

digitalWrite(wifiLedPin, LOW); // Wifi LED on when connected to Wifi as STA mode

delay(3000);

} else {

WiFi.mode(WIFI\_AP);

WiFi.softAP(hostname.c\_str());

IPAddress myIP = WiFi.softAPIP();

Serial.println("");

Serial.println("WiFi failed connected to " + sta\_ssid);

Serial.println("");

Serial.println("WiFi-AP-Mode");

Serial.print("AP IP address: ");

Serial.println(myIP);

digitalWrite(wifiLedPin, HIGH); // Wifi LED off when status as AP mode

delay(3000);

}

server.on ( "/", HTTP\_handleRoot ); // call the 'handleRoot'

function when a client requests URI "/"

server.onNotFound ( HTTP\_handleRoot ); // when a client requests an unknown URI (i.e. something other than "/"), call function "handleNotFound"

server.begin(); // actually start the server

ArduinoOTA.begin(); // enable to receive update/uploade firmware via Wifi OTA

}

void loop() {

ArduinoOTA.handle(); // listen for update OTA request from clients

server.handleClient(); // listen for HTTP requests from clients

command = server.arg("State"); // check HTPP request, if has arguments "State" then saved the value

if (command == "F") Forward(); // check string then call a function or set a value

else if (command == "B") Backward();

else if (command == "R") TurnRight();

else if (command == "L") TurnLeft();

// else if (command == "G") TurnLeft();

// else if (command == "H") TurnLeft();

// else if (command == "I") TurnRight();

// else if (command == "J") TurnRight();

else if (command == "S") Stop();

else if (command == "V") BeepHorn();

else if (command == "W") TurnLightOn();

else if (command == "w") TurnLightOff();

else if (command == "0") SPEED = 330;

else if (command == "1") SPEED = 400;

else if (command == "2") SPEED = 470;

else if (command == "3") SPEED = 540;

else if (command == "4") SPEED = 610;

else if (command == "5") SPEED = 680;

else if (command == "6") SPEED = 750;

else if (command == "7") SPEED = 820;

else if (command == "8") SPEED = 890;

else if (command == "9") SPEED = 960;

else if (command == "q") SPEED = 1023;

}

// function prototypes for HTTP handlers

void HTTP\_handleRoot(void){

server.send ( 200, "text/html", "" ); // Send HTTP status 200 (Ok) and send some text to the browser/client

if( server.hasArg("State") ){

Serial.println(server.arg("State"));

}

}

void handleNotFound(){

server.send(404, "text/plain", "404: Not found"); // Send HTTP status 404 (Not Found) when there's no handler for the URI in the request

}

// function to move forward

void Forward(){

digitalWrite(DIR\_A, HIGH);

digitalWrite(DIR\_B, LOW);

digitalWrite(PWM\_A, HIGH);

digitalWrite(PWM\_B, LOW);

}

// function to move backward

void Backward(){

digitalWrite(DIR\_A, LOW);

digitalWrite(DIR\_B, HIGH);

digitalWrite(PWM\_A, LOW);

digitalWrite(PWM\_B, HIGH);

}

// function to turn right

void TurnRight(){

digitalWrite(DIR\_A, LOW);

digitalWrite(DIR\_B, HIGH);

digitalWrite(PWM\_A, HIGH);

digitalWrite(PWM\_B, LOW);

}

void TurnLeft(){

digitalWrite(DIR\_A, HIGH);

digitalWrite(DIR\_B, LOW);

digitalWrite(PWM\_A, LOW);

digitalWrite(PWM\_B, HIGH);

}

void Stop(){

digitalWrite(DIR\_A, LOW);

digitalWrite(DIR\_B, LOW);

digitalWrite(PWM\_A, LOW);

digitalWrite(PWM\_B, LOW);

}

void BeepHorn(){

digitalWrite(buzPin, HIGH);

delay(150);

digitalWrite(buzPin, LOW);

delay(80);

}

// function to turn on LED

void TurnLightOn(){

digitalWrite(ledPin, HIGH);

}

// function to turn off LED

void TurnLightOff(){

digitalWrite(ledPin, LOW);

}

**ESP32-CAM-CODE**: -

#include "esp\_camera.h"

#include <WiFi.h>

#include "esp\_timer.h"

#include "img\_converters.h"

#include "Arduino.h"

#include "fb\_gfx.h"

#include "soc/soc.h" //disable brownout problems

#include "soc/rtc\_cntl\_reg.h" //disable brownout problems

#include "esp\_http\_server.h"

//Replace with your network credentials

const char\* ssid = "Turtle-4g";

const char\* password = "DHOLAKPUR";

#define PART\_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM Model and M5STACK WITHOUT PSRAM

#define CAMERA\_MODEL\_AI\_THINKER

//#define CAMERA\_MODEL\_M5STACK\_PSRAM

//#define CAMERA\_MODEL\_M5STACK\_WITHOUT\_PSRAM

// Not tested with this model

//#define CAMERA\_MODEL\_WROVER\_KIT

#if defined(CAMERA\_MODEL\_WROVER\_KIT)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 21

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 19

#define Y4\_GPIO\_NUM 18

#define Y3\_GPIO\_NUM 5

#define Y2\_GPIO\_NUM 4

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

#elif defined(CAMERA\_MODEL\_M5STACK\_PSRAM)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM 15

#define XCLK\_GPIO\_NUM 27

#define SIOD\_GPIO\_NUM 25

#define SIOC\_GPIO\_NUM 23

#define Y9\_GPIO\_NUM 19

#define Y8\_GPIO\_NUM 36

#define Y7\_GPIO\_NUM 18

#define Y6\_GPIO\_NUM 39

#define Y5\_GPIO\_NUM 5

#define Y4\_GPIO\_NUM 34

#define Y3\_GPIO\_NUM 35

#define Y2\_GPIO\_NUM 32

#define VSYNC\_GPIO\_NUM 22

#define HREF\_GPIO\_NUM 26

#define PCLK\_GPIO\_NUM 21

#elif defined(CAMERA\_MODEL\_M5STACK\_WITHOUT\_PSRAM)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM 15

#define XCLK\_GPIO\_NUM 27

#define SIOD\_GPIO\_NUM 25

#define SIOC\_GPIO\_NUM 23

#define Y9\_GPIO\_NUM 19

#define Y8\_GPIO\_NUM 36

#define Y7\_GPIO\_NUM 18

#define Y6\_GPIO\_NUM 39

#define Y5\_GPIO\_NUM 5

#define Y4\_GPIO\_NUM 34

#define Y3\_GPIO\_NUM 35

#define Y2\_GPIO\_NUM 17

#define VSYNC\_GPIO\_NUM 22

#define HREF\_GPIO\_NUM 26

#define PCLK\_GPIO\_NUM 21

#elif defined(CAMERA\_MODEL\_AI\_THINKER)

#define PWDN\_GPIO\_NUM 32

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 0

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 21

#define Y4\_GPIO\_NUM 19

#define Y3\_GPIO\_NUM 18

#define Y2\_GPIO\_NUM 5

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

#else

#error "Camera model not selected"

#endif

static const char\* \_STREAM\_CONTENT\_TYPE = "multipart/x-mixed-replace;boundary=" PART\_BOUNDARY;

static const char\* \_STREAM\_BOUNDARY = "\r\n--" PART\_BOUNDARY "\r\n";

static const char\* \_STREAM\_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

httpd\_handle\_t stream\_httpd = NULL;

static esp\_err\_t stream\_handler(httpd\_req\_t \*req){

camera\_fb\_t \* fb = NULL;

esp\_err\_t res = ESP\_OK;

size\_t \_jpg\_buf\_len = 0;

uint8\_t \* \_jpg\_buf = NULL;

char \* part\_buf[64];

res = httpd\_resp\_set\_type(req, \_STREAM\_CONTENT\_TYPE);

if(res != ESP\_OK){

return res;

}

while(true){

fb = esp\_camera\_fb\_get();

if (!fb) {

Serial.println("Camera capture failed");

res = ESP\_FAIL;

} else {

if(fb->width > 400){

if(fb->format != PIXFORMAT\_JPEG){

bool jpeg\_converted = frame2jpg(fb, 80, &\_jpg\_buf, &\_jpg\_buf\_len);

esp\_camera\_fb\_return(fb);

fb = NULL;

if(!jpeg\_converted){

Serial.println("JPEG compression failed");

res = ESP\_FAIL;

}

} else {

\_jpg\_buf\_len = fb->len;

\_jpg\_buf = fb->buf;

}

}

}

if(res == ESP\_OK){

size\_t hlen = snprintf((char \*)part\_buf, 64, \_STREAM\_PART, \_jpg\_buf\_len);

res = httpd\_resp\_send\_chunk(req, (const char \*)part\_buf, hlen);

}

if(res == ESP\_OK)

{

res = httpd\_resp\_send\_chunk(req, (const char \*)\_jpg\_buf, \_jpg\_buf\_len);

}

if(res == ESP\_OK){

res = httpd\_resp\_send\_chunk(req, \_STREAM\_BOUNDARY, strlen(\_STREAM\_BOUNDARY));

}

if(fb){

esp\_camera\_fb\_return(fb);

fb = NULL;

\_jpg\_buf = NULL;

} else if(\_jpg\_buf){

free(\_jpg\_buf);

\_jpg\_buf = NULL;

}

if(res != ESP\_OK){

break;

}

//Serial.printf("MJPG: %uB\n",(uint32\_t)(\_jpg\_buf\_len));

}

return res;

}

void startCameraServer(){

httpd\_config\_t config = HTTPD\_DEFAULT\_CONFIG();

config.server\_port = 80;

httpd\_uri\_t index\_uri = {

.uri = "/",

.method = HTTP\_GET,

.handler = stream\_handler,

.user\_ctx = NULL

};

//Serial.printf("Starting web server on port: '%d'\n", config.server\_port);

if (httpd\_start(&stream\_httpd, &config) == ESP\_OK)

{

httpd\_register\_uri\_handler(stream\_httpd, &index\_uri);

}

}

void setup() {

WRITE\_PERI\_REG(RTC\_CNTL\_BROWN\_OUT\_REG, 0); //disable brownout detector

Serial.begin(115200);

Serial.setDebugOutput(false);

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

config.pin\_d1 = Y3\_GPIO\_NUM;

config.pin\_d2 = Y4\_GPIO\_NUM;

config.pin\_d3 = Y5\_GPIO\_NUM;

config.pin\_d4 = Y6\_GPIO\_NUM;

config.pin\_d5 = Y7\_GPIO\_NUM;

config.pin\_d6 = Y8\_GPIO\_NUM;

config.pin\_d7 = Y9\_GPIO\_NUM;

config.pin\_xclk = XCLK\_GPIO\_NUM;

config.pin\_pclk = PCLK\_GPIO\_NUM;

config.pin\_vsync = VSYNC\_GPIO\_NUM;

config.pin\_href = HREF\_GPIO\_NUM;

config.pin\_sccb\_sda = SIOD\_GPIO\_NUM;

config.pin\_sccb\_scl = SIOC\_GPIO\_NUM;

config.pin\_pwdn = PWDN\_GPIO\_NUM;

config.pin\_reset = RESET\_GPIO\_NUM;

config.xclk\_freq\_hz = 20000000;

config.pixel\_format = PIXFORMAT\_JPEG;

if(psramFound()){

config.frame\_size = FRAMESIZE\_UXGA;

config.jpeg\_quality = 10;

config.fb\_count = 2;

} else

{

config.frame\_size = FRAMESIZE\_SVGA;

config.jpeg\_quality = 12;

config.fb\_count = 1;

}

// Camera init

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK) {

Serial.printf("Camera init failed with error 0x%x", err);

return;

}

// Wi-Fi connection

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");

Serial.print(WiFi.localIP());

// Start streaming web server

startCameraServer();

}

void loop() {

delay(1);

}

**HAND-TRAKING-ARDUINO-UNO CODE**: -

#include "esp\_camera.h"

#include <WiFi.h>

#include "esp\_timer.h"

#include "img\_converters.h"

#include "Arduino.h"

#include "fb\_gfx.h"

#include "soc/soc.h" //disable brownout problems

#include "soc/rtc\_cntl\_reg.h" //disable brownout problems

#include "esp\_http\_server.h"

//Replace with your network credentials

const char\* ssid = "Turtle-4g";

const char\* password = "DHOLAKPUR";

#define PART\_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM Model and M5STACK WITHOUT PSRAM

#define CAMERA\_MODEL\_AI\_THINKER

//#define CAMERA\_MODEL\_M5STACK\_PSRAM

//#define CAMERA\_MODEL\_M5STACK\_WITHOUT\_PSRAM

// Not tested with this model

//#define CAMERA\_MODEL\_WROVER\_KIT

#if defined(CAMERA\_MODEL\_WROVER\_KIT)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 21

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 19

#define Y4\_GPIO\_NUM 18

#define Y3\_GPIO\_NUM 5

#define Y2\_GPIO\_NUM 4

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

#elif defined(CAMERA\_MODEL\_M5STACK\_PSRAM)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM 15

#define XCLK\_GPIO\_NUM 27

#define SIOD\_GPIO\_NUM 25

#define SIOC\_GPIO\_NUM 23

#define Y9\_GPIO\_NUM 19

#define Y8\_GPIO\_NUM 36

#define Y7\_GPIO\_NUM 18

#define Y6\_GPIO\_NUM 39

#define Y5\_GPIO\_NUM 5

#define Y4\_GPIO\_NUM 34

#define Y3\_GPIO\_NUM 35

#define Y2\_GPIO\_NUM 32

#define VSYNC\_GPIO\_NUM 22

#define HREF\_GPIO\_NUM 26

#define PCLK\_GPIO\_NUM 21

#elif defined(CAMERA\_MODEL\_M5STACK\_WITHOUT\_PSRAM)

#define PWDN\_GPIO\_NUM -1

#define RESET\_GPIO\_NUM 15

#define XCLK\_GPIO\_NUM 27

#define SIOD\_GPIO\_NUM 25

#define SIOC\_GPIO\_NUM 23

#define Y9\_GPIO\_NUM 19

#define Y8\_GPIO\_NUM 36

#define Y7\_GPIO\_NUM 18

#define Y6\_GPIO\_NUM 39

#define Y5\_GPIO\_NUM 5

#define Y4\_GPIO\_NUM 34

#define Y3\_GPIO\_NUM 35

#define Y2\_GPIO\_NUM 17

#define VSYNC\_GPIO\_NUM 22

#define HREF\_GPIO\_NUM 26

#define PCLK\_GPIO\_NUM 21

#elif defined(CAMERA\_MODEL\_AI\_THINKER)

#define PWDN\_GPIO\_NUM 32

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 0

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 21

#define Y4\_GPIO\_NUM 19

#define Y3\_GPIO\_NUM 18

#define Y2\_GPIO\_NUM 5

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

#else

#error "Camera model not selected"

#endif

static const char\* \_STREAM\_CONTENT\_TYPE = "multipart/x-mixed-replace;boundary=" PART\_BOUNDARY;

static const char\* \_STREAM\_BOUNDARY = "\r\n--" PART\_BOUNDARY "\r\n";

static const char\* \_STREAM\_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

httpd\_handle\_t stream\_httpd = NULL;

static esp\_err\_t stream\_handler(httpd\_req\_t \*req){

camera\_fb\_t \* fb = NULL;

esp\_err\_t res = ESP\_OK;

size\_t \_jpg\_buf\_len = 0;

uint8\_t \* \_jpg\_buf = NULL;

char \* part\_buf[64];

res = httpd\_resp\_set\_type(req, \_STREAM\_CONTENT\_TYPE);

if(res != ESP\_OK){

return res;

}

while(true){

fb = esp\_camera\_fb\_get();

if (!fb) {

Serial.println("Camera capture failed");

res = ESP\_FAIL;

} else {

if(fb->width > 400){

if(fb->format != PIXFORMAT\_JPEG)

{

bool jpeg\_converted = frame2jpg(fb, 80, &\_jpg\_buf, &\_jpg\_buf\_len);

esp\_camera\_fb\_return(fb);

fb = NULL;

if(!jpeg\_converted){

Serial.println("JPEG compression failed");

res = ESP\_FAIL;

}

} else {

\_jpg\_buf\_len = fb->len;

\_jpg\_buf = fb->buf;

}

}

}

if(res == ESP\_OK){

size\_t hlen = snprintf((char \*)part\_buf, 64, \_STREAM\_PART, \_jpg\_buf\_len);

res = httpd\_resp\_send\_chunk(req, (const char \*)part\_buf, hlen);

}

if(res == ESP\_OK){

res = httpd\_resp\_send\_chunk(req, (const char \*)\_jpg\_buf, \_jpg\_buf\_len);

}

if(res == ESP\_OK){

res = httpd\_resp\_send\_chunk(req, \_STREAM\_BOUNDARY, strlen(\_STREAM\_BOUNDARY));

}

if(fb){

esp\_camera\_fb\_return(fb);

fb = NULL;

\_jpg\_buf = NULL;

} else if(\_jpg\_buf){

free(\_jpg\_buf);

\_jpg\_buf = NULL;

}

if(res != ESP\_OK){

break;

}

//Serial.printf("MJPG: %uB\n",(uint32\_t)(\_jpg\_buf\_len));

}

return res;

}

void startCameraServer(){

httpd\_config\_t config = HTTPD\_DEFAULT\_CONFIG();

config.server\_port = 80;

httpd\_uri\_t index\_uri = {

.uri = "/",

.method = HTTP\_GET,

.handler = stream\_handler,

.user\_ctx = NULL

};

//Serial.printf("Starting web server on port: '%d'\n", config.server\_port);

if (httpd\_start(&stream\_httpd, &config) == ESP\_OK) {

httpd\_register\_uri\_handler(stream\_httpd, &index\_uri);

}

}

void setup() {

WRITE\_PERI\_REG(RTC\_CNTL\_BROWN\_OUT\_REG, 0); //disable brownout detector

Serial.begin(115200);

Serial.setDebugOutput(false);

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

config.pin\_d1 = Y3\_GPIO\_NUM;

config.pin\_d2 = Y4\_GPIO\_NUM;

config.pin\_d3 = Y5\_GPIO\_NUM;

config.pin\_d4 = Y6\_GPIO\_NUM;

config.pin\_d5 = Y7\_GPIO\_NUM;

config.pin\_d6 = Y8\_GPIO\_NUM;

config.pin\_d7 = Y9\_GPIO\_NUM;

config.pin\_xclk = XCLK\_GPIO\_NUM;

config.pin\_pclk = PCLK\_GPIO\_NUM;

config.pin\_vsync = VSYNC\_GPIO\_NUM;

config.pin\_href = HREF\_GPIO\_NUM;

config.pin\_sccb\_sda = SIOD\_GPIO\_NUM;

config.pin\_sccb\_scl = SIOC\_GPIO\_NUM;

config.pin\_pwdn = PWDN\_GPIO\_NUM;

config.pin\_reset = RESET\_GPIO\_NUM;

config.xclk\_freq\_hz = 20000000;

config.pixel\_format = PIXFORMAT\_JPEG;

if(psramFound())

{

config.frame\_size = FRAMESIZE\_UXGA;

config.jpeg\_quality = 10;

config.fb\_count = 2;

} else {

config.frame\_size = FRAMESIZE\_SVGA;

config.jpeg\_quality = 12;

config.fb\_count = 1;

}

// Camera init

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK) {

Serial.printf("Camera init failed with error 0x%x", err);

return;

}

// Wi-Fi connection

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");

Serial.print(WiFi.localIP());

// Start streaming web server

startCameraServer();

}

void loop() {

delay(1);

}

**PYTHON HAND TRACKING CODE: -**

import tkinter as tk

from tkinter import messagebox, ttk

from PIL import Image, ImageTk, ImageDraw

import os

import cv2

import threading

import time

import datetime

import mediapipe as mp

from google.protobuf.json\_format import MessageToDict

import serial

import sys

import requests

import numpy as np

import webbrowser # Import the webbrowser module

# === Global Variables ===

camera\_thread = None

camera\_running = False

camera\_label = None # This will now only be used for Farming Bot's ESP32 stream

captured\_image\_label = None # This will now only be used for Farming Bot's captured image

latest\_frame = None # For saving/capturing current frame

handcam\_thread = None

ESP32\_CAM\_URL = "http://192.168.29.46/" # Replace with your ESP32-CAM IP

# Define your Streamlit application URL here

STREAMLIT\_URL = "http://localhost:8501"

# Global for bot arm stream label (Updated to be global)

bot\_stream\_img\_label = None

# === Hand Tracking Section ===

# Configuration

WRITE\_VIDEO = True

DEBUG = True # Set to False when using real Arduino

CAM\_SOURCE = 0

if not DEBUG:

try:

ser = serial.Serial('COM5', 115200)

except serial.SerialException as e:

print(f"Could not open serial port COM5: {e}")

DEBUG = True

# Servo ranges

X\_MIN, X\_MID, X\_MAX = 0, 75, 150

PALM\_ANGLE\_MIN, PALM\_ANGLE\_MID = -50, 20

Y\_MIN, Y\_MID, Y\_MAX = 0, 90, 180

WRIST\_Y\_MIN, WRIST\_Y\_MAX = 0.3, 0.9

Z\_MIN, Z\_MID, Z\_MAX = 10, 90, 180

PALM\_SIZE\_MIN, PALM\_SIZE\_MAX = 0.1, 0.3

CLAW\_OPEN\_ANGLE, CLAW\_CLOSE\_ANGLE = 50, 180

# Right hand

X1\_MIN, X1\_MID, X1\_MAX = 0, 75, 150

PALM1\_ANGLE\_MIN, PALM1\_ANGLE\_MID = -50, 20

Y1\_MIN, Y1\_MID, Y1\_MAX = 0, 90, 180

WRIST1\_Y\_MIN, WRIST1\_Y\_MAX = 0.3, 0.9

Z1\_MIN, Z1\_MID, Z1\_MAX = 10, 90, 180

PALM1\_SIZE\_MIN, PALM1\_SIZE\_MAX = 0.1, 0.3

CLAW1\_OPEN\_ANGLE, CLAW1\_CLOSE\_ANGLE = 50, 180

# Initial angles

servo\_angle = [X\_MID, Y\_MID, Z\_MID, CLAW\_OPEN\_ANGLE]

prev\_servo\_angle = servo\_angle.copy()

FIST\_THRESHOLD = 7

servo1\_angle = [X1\_MID, Y1\_MID, Z1\_MID, CLAW1\_OPEN\_ANGLE]

prev1\_servo\_angle = servo1\_angle.copy()

FIST1\_THRESHOLD = 7

# MediaPipe setup

mp\_drawing = mp.solutions.drawing\_utils

mp\_drawing\_styles = mp.solutions.drawing\_styles

mp\_hands = mp.solutions.hands

# Stop flag for thread control

stop\_handcam = False

def clamp(value, min\_value, max\_value):

return max(min(max\_value, value), min\_value)

def map\_range(value, in\_min, in\_max, out\_min, out\_max):

if in\_min == in\_max:

return out\_min if value <= in\_min else out\_max

return abs(int((value - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min))

def is\_fist(hand\_landmarks, palm\_size):

WRIST = hand\_landmarks.landmark[0]

distance\_sum = sum(

((WRIST.x - hand\_landmarks.landmark[i].x) \*\* 2 +

(WRIST.y - hand\_landmarks.landmark[i].y) \*\* 2 +

(WRIST.z - hand\_landmarks.landmark[i].z) \*\* 2) \*\* 0.5

for i in [7, 8, 11, 12, 15, 16, 19, 20]

)

if palm\_size == 0:

return False

return distance\_sum / palm\_size < FIST\_THRESHOLD

def landmark\_to\_servo\_angle(hand\_landmarks):

WRIST = hand\_landmarks.landmark[0]

MCP = hand\_landmarks.landmark[5]

palm\_size = ((WRIST.x - MCP.x)\*2 + (WRIST.y - MCP.y)2 + (WRIST.z - MCP.z)2)\*0.5

angles = [X\_MID, Y\_MID, Z\_MID, CLAW\_OPEN\_ANGLE]

angles[3] = CLAW\_CLOSE\_ANGLE if is\_fist(hand\_landmarks, palm\_size) else CLAW\_OPEN\_ANGLE

if palm\_size != 0:

angle = (WRIST.x - MCP.x) / palm\_size

angle = int(angle \* 180 / 3.1415926)

else:

angle = PALM\_ANGLE\_MID

angle = clamp(angle, PALM\_ANGLE\_MIN, PALM\_ANGLE\_MID)

angles[0] = map\_range(angle, PALM\_ANGLE\_MIN, PALM\_ANGLE\_MID, X\_MAX, X\_MIN)

wrist\_y = clamp(WRIST.y, WRIST\_Y\_MIN, WRIST\_Y\_MAX)

angles[1] = map\_range(wrist\_y, WRIST\_Y\_MIN, WRIST\_Y\_MAX, Y\_MAX, Y\_MIN)

palm\_size = clamp(palm\_size, PALM\_SIZE\_MIN, PALM\_SIZE\_MAX)

angles[2] = map\_range(palm\_size, PALM\_SIZE\_MIN, PALM\_SIZE\_MAX, Z\_MAX, Z\_MIN)

return [int(i) for i in angles]

def is\_fist1(hand\_landmarks1, palm\_size1):

WRIST1 = hand\_landmarks1.landmark[0]

distance\_sum1 = sum(

((WRIST1.x - hand\_landmarks1.landmark[i].x) \*\* 2 +

(WRIST1.y - hand\_landmarks1.landmark[i].y) \*\* 2 +

(WRIST1.z - hand\_landmarks1.landmark[i].z) \*\* 2) \*\* 0.5

for i in [7, 8, 11, 12, 15, 16, 19, 20]

)

if palm\_size1 == 0:

return False

return distance\_sum1 / palm\_size1 < FIST1\_THRESHOLD

def landmark\_to\_servo\_angle1(hand\_landmarks1):

WRIST1 = hand\_landmarks1.landmark[0]

MCP1 = hand\_landmarks1.landmark[5]

palm\_size1 = ((WRIST1.x - MCP1.x)\*2 + (WRIST1.y - MCP1.y)2 + (WRIST1.z - MCP1.z)2)\*0.5

angles1 = [X1\_MID, Y1\_MID, Z1\_MID, CLAW1\_OPEN\_ANGLE]

angles1[3] = CLAW1\_CLOSE\_ANGLE if is\_fist1(hand\_landmarks1, palm\_size1) else CLAW1\_OPEN\_ANGLE

if palm\_size1 != 0:

angle1 = (WRIST1.x - MCP1.x) / palm\_size1

angle1 = int(angle1 \* 180 / 3.1415926)

else:

angle1 = PALM1\_ANGLE\_MID

angle1 = clamp(angle1, PALM1\_ANGLE\_MIN, PALM1\_ANGLE\_MID)

angles1[0] = map\_range(angle1, PALM1\_ANGLE\_MIN, PALM1\_ANGLE\_MID, X1\_MIN, X1\_MAX)

wrist\_y1 = clamp(WRIST1.y, WRIST1\_Y\_MIN, WRIST1\_Y\_MAX)

angles1[1] = map\_range(wrist\_y1, WRIST1\_Y\_MIN, WRIST1\_Y\_MAX, Y1\_MAX, Y1\_MIN)

palm\_size1 = clamp(palm\_size1, PALM1\_SIZE\_MIN, PALM1\_SIZE\_MAX)

angles1[2] = map\_range(palm\_size1, PALM1\_SIZE\_MIN, PALM1\_SIZE\_MAX, Z1\_MAX, Z1\_MIN)

return [int(i) for i in angles1]

def run\_hand\_tracking():

global stop\_handcam, servo\_angle, prev\_servo\_angle, servo1\_angle, prev1\_servo\_angle, bot\_stream\_img\_label

cap = cv2.VideoCapture(CAM\_SOURCE)

if WRITE\_VIDEO:

out = None

else:

out = None

with mp\_hands.Hands(model\_complexity=0, min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as hands:

while cap.isOpened() and not stop\_handcam:

success, image = cap.read()

if not success:

print("Empty camera frame or camera not available.")

cap.release()

cap = cv2.VideoCapture(CAM\_SOURCE)

time.sleep(1)

continue

if WRITE\_VIDEO and out is None:

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

height, width, \_ = image.shape

out = cv2.VideoWriter('output.avi', fourcc, 30.0, (width, height))

image = cv2.flip(image, 1)

image.flags.writeable = False

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

results = hands.process(image\_rgb)

image.flags.writeable = True

image = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2BGR)

if results.multi\_hand\_landmarks:

for hand\_landmarks, handedness in zip(results.multi\_hand\_landmarks, results.multi\_handedness):

label = MessageToDict(handedness)['classification'][0]['label']

if label == 'Left':

cv2.putText(image, label + ' Hand', (20, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.9, (0, 255, 0), 2)

servo\_angle = landmark\_to\_servo\_angle(hand\_landmarks)

elif label == 'Right':

cv2.putText(image, label + ' Hand', (460, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.9, (0, 255, 0), 2)

servo1\_angle = landmark\_to\_servo\_angle1(hand\_landmarks)

mp\_drawing.draw\_landmarks(

image,

hand\_landmarks,

mp\_hands.HAND\_CONNECTIONS,

mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

mp\_drawing\_styles.get\_default\_hand\_connections\_style()

)

if servo\_angle != prev\_servo\_angle or servo1\_angle != prev1\_servo\_angle:

print("Left:", servo\_angle, "Right:", servo1\_angle)

prev\_servo\_angle = servo\_angle.copy()

prev1\_servo\_angle = servo1\_angle.copy()

if not DEBUG:

try:

if ser.is\_open:

ser.write(bytearray(servo\_angle + servo1\_angle))

else:

print("Serial port is not open. Skipping serial write.")

except serial.SerialException as e:

print(f"Serial write error: {e}")

if bot\_stream\_img\_label and bot\_stream\_img\_label.winfo\_exists():

label\_width = bot\_stream\_img\_label.winfo\_width()

label\_height = bot\_stream\_img\_label.winfo\_height()

if label\_width < 100 or label\_height < 100:

label\_width, label\_height = 500, 350 # Make bigger

original\_height, original\_width, \_ = image.shape

aspect\_ratio = original\_width / original\_height

if label\_width / aspect\_ratio <= label\_height:

new\_width = label\_width

new\_height = int(label\_width / aspect\_ratio)

else:

new\_height = label\_height

new\_width = int(label\_height \* aspect\_ratio)

new\_width = max(1, new\_width)

new\_height = max(1, new\_height)

img\_resized = cv2.resize(image, (new\_width, new\_height))

img\_rgb = cv2.cvtColor(img\_resized, cv2.COLOR\_BGR2RGB)

img\_pil = Image.fromarray(img\_rgb)

imgtk = ImageTk.PhotoImage(img\_pil)

def update\_label\_in\_main\_thread():

if bot\_stream\_img\_label and bot\_stream\_img\_label.winfo\_exists():

bot\_stream\_img\_label.imgtk = imgtk

bot\_stream\_img\_label.configure(image=imgtk)

bot\_stream\_img\_label.after(0, update\_label\_in\_main\_thread)

if WRITE\_VIDEO and out is not None:

out.write(image)

if stop\_handcam:

break

if WRITE\_VIDEO and out is not None:

out.release()

cap.release()

if bot\_stream\_img\_label and bot\_stream\_img\_label.winfo\_exists():

bot\_stream\_img\_label.after(0, lambda: bot\_stream\_img\_label.configure(image=None))

cv2.destroyAllWindows()

def start\_handcam():

global stop\_handcam, handcam\_thread

if handcam\_thread and handcam\_thread.is\_alive():

print("Handcam thread is already running.")

return

stop\_handcam = False

handcam\_thread = threading.Thread(target=run\_hand\_tracking, daemon=True)

handcam\_thread.start()

return handcam\_thread

def stop\_handcam\_func():

global stop\_handcam, handcam\_thread

if handcam\_thread and handcam\_thread.is\_alive():

stop\_handcam = True

handcam\_thread.join(timeout=1.0)

if handcam\_thread.is\_alive():

print("Handcam thread did not terminate in time.")

else:

print("Handcam thread stopped.")

else:

print("Handcam thread is not running.")

def load\_image(path, size=(120, 120)):

try:

if not os.path.exists(path):

print(f"Image not found: {path}")

return None

img = Image.open(path)

img = img.resize(size, Image.Resampling.LANCZOS)

return ImageTk.PhotoImage(img)

except Exception as e:

print(f"Error loading {path}: {e}")

return None

root = tk.Tk()

root.title("Bot Selector")

root.geometry("1100x650")

farming\_img = load\_image("gesture\_MeArm-main\\images\\FarmBot.png")

surgery\_img = load\_image("gesture\_MeArm-main\\images\\surgery.png")

industrial\_img = load\_image("gesture\_MeArm-main\\images\\industry.png")

if not all([farming\_img, surgery\_img, industrial\_img]):

messagebox.showerror("Image Load Error", "One or more bot images could not be loaded. Please check paths.")

global\_images = {

"farming": farming\_img,

"surgery": surgery\_img,

"industrial": industrial\_img

}

canvas = tk.Canvas(root, width=1100, height=650)

canvas.pack(fill="both", expand=True)

def draw\_vertical\_gradient(canvas, color1, color2):

width, height = 1100, 650

gradient = Image.new("RGB", (1, height), "#000")

draw = ImageDraw.Draw(gradient)

r1, g1, b1 = canvas.winfo\_rgb(color1)

r2, g2, b2 = canvas.winfo\_rgb(color2)

r1, g1, b1 = [val >> 8 for val in (r1, g1, b1)]

r2, g2, b2 = [val >> 8 for val in (r2, g2, b2)]

for y in range(height):

ratio = y / height

r = int(r1 + (r2 - r1) \* ratio)

g = int(g1 + (g2 - g1) \* ratio)

b = int(b1 + (b2 - b1) \* ratio)

draw.point((0, y), fill=(r, g, b))

gradient = gradient.resize((width, height), Image.Resampling.LANCZOS)

bg\_img = ImageTk.PhotoImage(gradient)

canvas.create\_image(0, 0, anchor="nw", image=bg\_img)

canvas.bg\_img = bg\_img

draw\_vertical\_gradient(canvas, "#e3f2fd", "#90caf9")

def start\_esp32\_cam\_stream(esp32\_url=ESP32\_CAM\_URL):

global camera\_running, camera\_thread, latest\_frame, camera\_label

if camera\_label is None or not camera\_label.winfo\_exists():

print("ESP32 Live Cam label is not available. This stream is only for Farming Bot.")

return

if camera\_thread and camera\_thread.is\_alive():

print("ESP32 camera stream is already running.")

return

def stream():

global camera\_running, latest\_frame, camera\_label

bytes\_data = bytes()

stream\_request = None

try:

stream\_request = requests.get(esp32\_url, stream=True, timeout=5)

for chunk in stream\_request.iter\_content(chunk\_size=1024):

if not camera\_running:

break

bytes\_data += chunk

a = bytes\_data.find(b'\xff\xd8')

b = bytes\_data.find(b'\xff\xd9')

if a != -1 and b != -1:

jpg = bytes\_data[a:b+2]

bytes\_data = bytes\_data[b+2:]

img\_np = np.frombuffer(jpg, dtype=np.uint8)

frame = cv2.imdecode(img\_np, cv2.IMREAD\_COLOR)

if frame is not None:

latest\_frame = frame.copy()

# --- Remove window size adjustment animation ---

# Always use a fixed size for the ESP32 live cam image

fixed\_width, fixed\_height = 300, 200

frame\_resized = cv2.resize(frame, (fixed\_width, fixed\_height))

frame\_rgb = cv2.cvtColor(frame\_resized, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(frame\_rgb)

imgtk = ImageTk.PhotoImage(image=img)

if camera\_label and camera\_label.winfo\_exists():

camera\_label.imgtk = imgtk

camera\_label.configure(image=imgtk)

time.sleep(0.01)

except requests.exceptions.ConnectionError as ce:

print(f"ESP32 stream connection error: {ce}. Is the ESP32-CAM on and reachable at {esp32\_url}?")

messagebox.showerror("Connection Error", f"Could not connect to ESP32-CAM at {esp32\_url}. Please ensure it's powered on and on the same network.\nError: {ce}")

except Exception as e:

print(f"ESP32 stream error: {e}")

messagebox.showerror("Stream Error", f"An error occurred during ESP32 stream: {e}")

finally:

if stream\_request:

stream\_request.close()

if camera\_label and camera\_label.winfo\_exists():

camera\_label.after(0, lambda: camera\_label.configure(image=None))

print("ESP32 stream stopped.")

camera\_running = True

camera\_thread = threading.Thread(target=stream, daemon=True)

camera\_thread.start()

def stop\_esp32\_cam\_stream():

global camera\_running, camera\_thread

if camera\_thread and camera\_thread.is\_alive():

camera\_running = False

camera\_thread.join(timeout=1.0)

if camera\_thread.is\_alive():

print("ESP32 camera thread did not terminate in time.")

else:

print("ESP32 camera thread stopped.")

else:

print("ESP32 camera stream is not running.")

if camera\_label and camera\_label.winfo\_exists():

camera\_label.after(0, lambda: camera\_label.configure(image=None))

def capture\_image():

global latest\_frame, captured\_image\_label

if captured\_image\_label is None or not captured\_image\_label.winfo\_exists():

print("Captured Image label is not available. This feature is only for Farming Bot.")

return

if latest\_frame is not None:

downloads\_dir = os.path.join(os.path.expanduser("~"), "Downloads", "CapturedImages")

os.makedirs(downloads\_dir, exist\_ok=True)

filename = f"captured\_{datetime.datetime.now().strftime('%Y%m%d\_%H%M%S')}.jpg"

filepath = os.path.join(downloads\_dir, filename)

try:

cv2.imwrite(filepath, latest\_frame)

except Exception as e:

messagebox.showerror("Save Error", f"Failed to save image: {e}")

print(f"Error saving image: {e}")

return

img\_rgb = cv2.cvtColor(latest\_frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(img\_rgb)

label\_width = captured\_image\_label.winfo\_width()

label\_height = captured\_image\_label.winfo\_height()

if label\_width < 100 or label\_height < 100:

label\_width, label\_height = 300, 200

original\_width, original\_height = img.size

aspect\_ratio = original\_width / original\_height

if label\_width / aspect\_ratio <= label\_height:

new\_width = label\_width

new\_height = int(label\_width / aspect\_ratio)

else:

new\_height = label\_height

new\_width = int(label\_height \* aspect\_ratio)

new\_width = max(1, new\_width)

new\_height = max(1, new\_height)

img = img.resize((new\_width, new\_height), Image.Resampling.LANCZOS)

imgtk = ImageTk.PhotoImage(img)

if captured\_image\_label and captured\_image\_label.winfo\_exists():

captured\_image\_label.imgtk = imgtk

captured\_image\_label.configure(image=imgtk)

print(f"Image captured and saved to {filepath}")

messagebox.showinfo("Image Captured", f"Image saved to: {filepath}")

else:

print("No frame available to capture.")

messagebox.showwarning("No Frame", "No live camera frame available to capture.")

def open\_streamlit\_website():

try:

webbrowser.open(STREAMLIT\_URL)

print(f"Opened Streamlit website: {STREAMLIT\_URL}")

messagebox.showinfo("Website Opened", f"Your Streamlit website should now be open in your default browser at {STREAMLIT\_URL}")

except Exception as e:

print(f"Error opening Streamlit website: {e}")

messagebox.showerror("Error", f"Could not open website. Please ensure your Streamlit app is running at {STREAMLIT\_URL} and you have a web browser installed.\nError: {e}")

def add\_tooltip(widget, text):

tooltip = tk.Toplevel(widget)

tooltip.withdraw()

tooltip.overrideredirect(True)

label = tk.Label(tooltip, text=text, background="#263238", foreground="white",

font=("Segoe UI", 9), padx=8, pady=4, borderwidth=1, relief="solid")

label.pack()

def enter(event):

x = event.x\_root + 10

y = event.y\_root + 10

tooltip.geometry(f"+{x}+{y}")

tooltip.deiconify()

def leave(event):

tooltip.withdraw()

widget.bind("<Enter>", enter)

widget.bind("<Leave>", leave)

def style\_modern\_buttons():

style = ttk.Style()

style.theme\_use('clam')

style.configure("Modern.TButton",

font=("Segoe UI", 11, "bold"),

padding=8,

borderwidth=0,

background="#222",

foreground="white")

style.map("Modern.TButton",

background=[("active", "#444")],

foreground=[("active", "white")])

style\_modern\_buttons()

def setup\_farming\_bot\_ui(parent\_frame, title):

global camera\_label, captured\_image\_label, bot\_stream\_img\_label

header\_font = ("Segoe UI", 24, "bold")

label\_font = ("Segoe UI", 13)

button\_font = ("Segoe UI", 11, "bold")

tk.Label(parent\_frame, text=title, font=header\_font, bg="#F5F5F5", fg="#1976D2").pack(pady=10)

content\_frame = tk.Frame(parent\_frame, bg="#F5F5F5")

content\_frame.pack(expand=True, fill="both", padx=10, pady=10)

controls\_frame = tk.Frame(content\_frame, width=170, bg="#FFFFFF", relief="ridge", bd=2, highlightbackground="#90caf9", highlightthickness=2)

controls\_frame.pack(side="left", fill="y", padx=10, pady=10)

tk.Label(controls\_frame, text="Controls", font=("Segoe UI", 15, "bold"), bg="#FFFFFF", fg="#1976D2").pack(pady=(15, 10))

status\_var = tk.StringVar(value="Ready")

def dummy\_action(name):

global handcam\_thread

status\_var.set(f"{name.replace('\_', ' ').capitalize()} clicked")

if name == "start cam":

start\_esp32\_cam\_stream()

elif name == "stop cam":

stop\_esp32\_cam\_stream()

elif name == "capture img":

capture\_image()

elif name == "start handcam":

if handcam\_thread is None or not handcam\_thread.is\_alive():

handcam\_thread = start\_handcam()

elif name == "stop handcam":

stop\_handcam\_func()

else:

print(f"{name} clicked")

btn\_info = [

("start cam", "Start ESP32 camera stream"),

("stop cam", "Stop ESP32 camera stream"),

("capture img", "Capture image from ESP32"),

("start handcam", "Start hand gesture control"),

("stop handcam", "Stop hand gesture control"),

]

for text, tip in btn\_info:

btn = ttk.Button(controls\_frame, text=text.title(), style="Modern.TButton", command=lambda t=text: dummy\_action(t))

btn.pack(pady=7, padx=18, fill="x")

add\_tooltip(btn, tip)

center\_frame = tk.Frame(content\_frame, bg="#E3F2FD", highlightbackground="#90caf9", highlightthickness=2)

center\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

bot\_stream\_frame = tk.Frame(center\_frame, bg="#B3E5FC", width=500, height=350, relief="ridge", bd=2)

bot\_stream\_frame.pack(padx=10, pady=10)

bot\_stream\_frame.pack\_propagate(False)

tk.Label(bot\_stream\_frame, text="Bot Arm Stream", font=("Segoe UI", 18, "bold"), bg="#B3E5FC", fg="#1976D2").pack(pady=5)

bot\_stream\_img\_label = tk.Label(bot\_stream\_frame, bg="#B3E5FC", width=500, height=350)

bot\_stream\_img\_label.pack(expand=False, padx=10, pady=10)

website\_frame = tk.Frame(center\_frame, height=100, bg="#E1F5FE", relief="ridge", bd=2)

website\_frame.pack(fill="both", expand=True, padx=10, pady=10)

website\_btn = ttk.Button(website\_frame, text="Go to Website", style="Modern.TButton", command=open\_streamlit\_website)

website\_btn.pack(pady=20)

add\_tooltip(website\_btn, "Open the Streamlit website for model prediction")

right\_section = tk.Frame(content\_frame, width=300, bg="#FFFFFF", relief="ridge", bd=2, highlightbackground="#90caf9", highlightthickness=2)

right\_section.pack(side="left", fill="both", expand=True, padx=10, pady=10)

camera\_frame = tk.Frame(right\_section, height=150, bg="#FFECB3", relief="ridge", bd=2)

camera\_frame.pack(fill="both", expand=True, padx=10, pady=10)

tk.Label(camera\_frame, text="ESP32 Live Cam", font=("Segoe UI", 16, "bold"), bg="#FFECB3", fg="#FF6F00").pack()

camera\_img\_label = tk.Label(camera\_frame, bg="#FFECB3")

camera\_img\_label.pack(expand=True, padx=10, pady=10)

camera\_label = camera\_img\_label

picture\_frame = tk.Frame(right\_section, height=150, bg="#FFE0B2", relief="ridge", bd=2)

picture\_frame.pack(fill="both", expand=True, padx=10, pady=10)

tk.Label(picture\_frame, text="Captured Image", font=("Segoe UI", 14, "bold"), bg="#FFE0B2", fg="#F57C00").pack()

captured\_img\_label = tk.Label(picture\_frame, bg="#FFE0B2")

captured\_img\_label.pack(expand=True, padx=10, pady=10)

captured\_image\_label = captured\_img\_label

status\_bar = tk.Label(parent\_frame, textvariable=status\_var, bd=1, relief="sunken", anchor="w", bg="#E3F2FD", font=("Segoe UI", 10))

status\_bar.pack(side="bottom", fill="x")

def on\_close():

stop\_esp32\_cam\_stream()

stop\_handcam\_func()

parent\_frame.destroy()

parent\_frame.protocol("WM\_DELETE\_WINDOW", on\_close)

def setup\_simplified\_bot\_ui(parent\_frame, title):

global bot\_stream\_img\_label

global camera\_label, captured\_image\_label

camera\_label = None

captured\_image\_label = None

header\_font = ("Segoe UI", 24, "bold")

label\_font = ("Segoe UI", 13)

button\_font = ("Segoe UI", 11, "bold")

tk.Label(parent\_frame, text=title, font=header\_font, bg="#F5F5F5", fg="#1976D2").pack(pady=10)

content\_frame = tk.Frame(parent\_frame, bg="#F5F5F5")

content\_frame.pack(expand=True, fill="both", padx=10, pady=10)

controls\_frame = tk.Frame(content\_frame, width=170, bg="#FFFFFF", relief="ridge", bd=2, highlightbackground="#90caf9", highlightthickness=2)

controls\_frame.pack(side="left", fill="y", padx=10, pady=10)

tk.Label(controls\_frame, text="Controls", font=("Segoe UI", 15, "bold"), bg="#FFFFFF", fg="#1976D2").pack(pady=(15, 10))

status\_var = tk.StringVar(value="Ready")

def dummy\_action(name):

global handcam\_thread

status\_var.set(f"{name.replace('\_', ' ').capitalize()} clicked")

if name == "start handcam":

if handcam\_thread is None or not handcam\_thread.is\_alive():

handcam\_thread = start\_handcam()

elif name == "stop handcam":

stop\_handcam\_func()

else:

print(f"{name} clicked")

btn\_info = [

("start handcam", "Start hand gesture control for bot arm"),

("stop handcam", "Stop hand gesture control"),

]

for text, tip in btn\_info:

btn = ttk.Button(controls\_frame, text=text.title(), style="Modern.TButton", command=lambda t=text: dummy\_action(t))

btn.pack(pady=7, padx=18, fill="x")

add\_tooltip(btn, tip)

bot\_stream\_frame = tk.Frame(content\_frame, bg="#B3E5FC", relief="ridge", bd=2, highlightbackground="#90caf9", highlightthickness=2)

bot\_stream\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

tk.Label(bot\_stream\_frame, text="Bot Arm Stream", font=("Segoe UI", 18, "bold"), bg="#B3E5FC", fg="#1976D2").pack(pady=5)

bot\_stream\_img\_label = tk.Label(bot\_stream\_frame, bg="#B3E5FC")

bot\_stream\_img\_label.pack(expand=True, padx=10, pady=10)

status\_bar = tk.Label(parent\_frame, textvariable=status\_var, bd=1, relief="sunken", anchor="w", bg="#E3F2FD", font=("Segoe UI", 10))

status\_bar.pack(side="bottom", fill="x")

def on\_close():

stop\_handcam\_func()

parent\_frame.destroy()

parent\_frame.protocol("WM\_DELETE\_WINDOW", on\_close)

def open\_new\_page(title):

new\_window = tk.Toplevel(root)

new\_window.title(title)

new\_window.geometry("1000x600")

new\_window.configure(bg="#F5F5F5")

new\_window.protocol("WM\_DELETE\_WINDOW", lambda: on\_new\_window\_close(new\_window, title))

if title == "Farming Bot":

setup\_farming\_bot\_ui(new\_window, title)

else:

setup\_simplified\_bot\_ui(new\_window, title)

def on\_new\_window\_close(window, title):

if title == "Farming Bot":

stop\_esp32\_cam\_stream()

stop\_handcam\_func()

window.destroy()

container = tk.Frame(canvas, bg="#FFFFFF")

canvas.create\_window(550, 325, window=container, anchor="center")

def create\_bot\_section(parent, img, name):

frame = tk.Frame(parent, bg="white", bd=0, relief="flat", highlightbackground="#CFD8DC", highlightthickness=2)

frame.grid\_propagate(False)

frame.configure(width=300, height=320)

button = tk.Button(

frame,

text=name,

image=img,

compound="top",

font=("Segoe UI", 14, "bold"),

fg="white",

bg="#222",

activebackground="#444",

activeforeground="white",

relief="flat",

padx=20,

pady=20,

borderwidth=0,

highlightthickness=0,

command=lambda: open\_new\_page(name)

)

button.image = img

button.pack(expand=True, fill="both", padx=20, pady=20)

return frame

create\_bot\_section(container, farming\_img, "Farming Bot").grid(row=0, column=0, padx=40)

create\_bot\_section(container, surgery\_img, "Surgery Bot").grid(row=0, column=1, padx=40)

create\_bot\_section(container, industrial\_img, "Industrial Bot").grid(row=0, column=2, padx=40)

root.mainloop()

**PLANT-DISEASE-DETECTION-CODE:-**

import streamlit as st

import tensorflow as tf

import numpy as np

# Tensorflow Model Prediction

def model\_prediction(test\_image):

model = tf.keras.models.load\_model("trained\_plant\_disease\_model.keras")

image = tf.keras.preprocessing.image.load\_img(test\_image, target\_size=(128, 128))

input\_arr = tf.keras.preprocessing.image.img\_to\_array(image)

input\_arr = np.array([input\_arr]) # convert single image to batch

predictions = model.predict(input\_arr)

return np.argmax(predictions) # return index of max element

# Sidebar

st.sidebar.title("Dashboard")

app\_mode = st.sidebar.selectbox("Select Page", ["Home", "About", "Disease Recognition"])

# Main Page

if app\_mode == "Home":

# Modern CSS for Home page

st.markdown("""

<style>

@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@600;700&display=swap');

html, body, [class\*="css"] {

font-family: 'Montserrat', sans-serif;

background: linear-gradient(120deg, #f8ffae 0%, #43cea2 100%);

}

.home-card {

background: rgba(255,255,255,0.92);

border-radius: 1.5rem;

box-shadow: 0 8px 32px rgba(67,206,162,0.18);

padding: 2.5rem 2rem 2rem 2rem;

margin-top: 2.5rem;

margin-bottom: 2rem;

text-align: center;

animation: fade-in 1.2s cubic-bezier(.68,-0.55,.27,1.55);

}

@keyframes fade-in

{

0% { opacity: 0; transform: translateY(40px);}

100% { opacity: 1; transform: translateY(0);}

}

.home-title {

font-size: 2.7rem;

font-weight: 700;

color: #185a9d;

letter-spacing: 1px;

margin-bottom: 0.7rem;

}

.home-subtitle {

font-size: 1.3rem;

color: #43cea2;

margin-bottom: 1.7rem;

}

.home-list {

text-align: left;

margin: 1.5rem auto 1.5rem auto;

max-width: 520px;

}

.home-list li {

margin-bottom: 0.7rem;

font-size: 1.08rem;

}

.icon {

font-size: 2.2rem;

margin-bottom: 0.5rem;

}

.svg-divider {

margin: 2rem 0 1.5rem 0;

}

.quote {

font-size: 1.15rem;

color: #185a9d;

font-style: italic;

margin-bottom: 1.5rem;

margin-top: 1.2rem;

}

.feature-row {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 2rem;

margin: 2rem 0 1.5rem 0;

}

.feature-box {

background: #f8ffae;

border-radius: 1rem;

box-shadow: 0 2px 12px rgba(67,206,162,0.10);

padding: 1.2rem 1.5rem;

min-width: 180px;

max-width: 220px;

flex: 1 1 180px;

text-align: center;

transition: transform 0.2s;

}

.feature-box:hover {

transform: translateY(-8px) scale(1.04);

box-shadow: 0 8px 32px rgba(67,206,162,0.18);

}

.feature-icon {

font-size: 2.3rem;

margin-bottom: 0.5rem;

}

.feature-title {

font-size: 1.1rem;

font-weight: 600;

color: #185a9d;

margin-bottom: 0.3rem;

}

.feature-desc {

font-size: 0.98rem;

color: #333;

}

</style>

""", unsafe\_allow\_html=True)

st.markdown('<div class="home-card">', unsafe\_allow\_html=True)

st.markdown('<div class="icon">🌱🔍</div>', unsafe\_allow\_html=True)

st.markdown('<div class="home-title">PLANT DISEASE RECOGNITION SYSTEM</div>', unsafe\_allow\_html=True)

st.markdown('<div class="home-subtitle">Empowering Farmers with AI for a Healthier Harvest</div>', unsafe\_allow\_html=True)

st.image("D:\\Robo Arm project\\gesture\_MeArm-main\\images\\home\_page.jpeg", use\_column\_width=True, caption="Smart Plant Disease Detection"

# Decorative SVG divider

st.markdown("""

<div class="svg-divider">

<svg height="30" width="100%" viewBox="0 0 100 10" preserveAspectRatio="none">

<path d="M0,10 Q50,0 100,10" fill="none" stroke="#43cea2" stroke-width="2"/>

</svg>

</div>

""", unsafe\_allow\_html=True)

# Motivational quote

st.markdown("""

<div class="quote">

“Healthy plants, healthy planet.<br>

Let AI help you grow better, greener, and smarter.”

</div>

""", unsafe\_allow\_html=True)

# --- Modern Feature Row ---

st.markdown("""

<div class="feature-row">

<div class="feature-box">

<div class="feature-icon">🤳</div>

<div class="feature-title">Easy Upload</div>

<div class="feature-desc">Just snap or upload a leaf photo to get started.</div>

</div>

<div class="feature-box">

<div class="feature-icon">🧠</div>

<div class="feature-title">AI Powered</div>

<div class="feature-desc">Advanced deep learning detects diseases with high accuracy.</div>

</div>

<div class="feature-box">

<div class="feature-icon">💡</div>

<div class="feature-title">Instant Advice</div>

<div class="feature-desc">Get actionable suggestions for plant health in seconds.</div>

</div>

<div class="feature-box">

<div class="feature-icon">📈</div>

<div class="feature-title">Track & Improve</div>

<div class="feature-desc">Monitor your plants and improve yield over

time.</div>

</div>

</div>

""", unsafe\_allow\_html=True)

st.markdown("""

<ul class="home-list">

<li>🚀 <b>Upload Image:</b> Go to <b>Disease Recognition</b> and upload a plant image.</li>

<li>🤖 <b>AI Analysis:</b> Our system uses advanced deep learning to detect diseases.</li>

<li>📊 <b>Instant Results:</b> Get fast, accurate predictions and actionable advice.</li>

</ul>

""", unsafe\_allow\_html=True)

st.markdown("""

<div style="margin-top:1.5rem;">

<b>Why Choose Us?</b>

<ul class="home-list">

<li>✅ <b>High Accuracy</b> with state-of-the-art AI</li>

<li>✅ <b>User-Friendly</b> and intuitive interface</li>

<li>✅ <b>Fast & Efficient</b> results for quick action</li>

</ul>

</div>

""", unsafe\_allow\_html=True)

st.markdown("""

<div style="margin-top:2rem;">

<a href="https://github.com/" target="\_blank" style="color:#185a9d;text-decoration:none;font-weight:bold;">

🌐 Learn more about our project & team

</a>

</div>

""", unsafe\_allow\_html=True)

st.markdown('</div>', unsafe\_allow\_html=True)

# About Project

elif app\_mode == "About":

st.header("About")

st.markdown("""

#### About Dataset

This dataset is recreated using offline augmentation from the original dataset.The original dataset can be found on this github repo.

This dataset consists of about 87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes.The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure.

A new directory containing 33 test images is created later for prediction purpose.

#### Content

1. train (70295 images)

2. test (33 images)

3. validation (17572 images)

""")

# Prediction Page

elif app\_mode == "Disease Recognition":

st.header("🌱 Disease Recognition")

st.markdown(

"""

<style>

.uploadbox {

background: linear-gradient(90deg, #f8ffae 0%, #43cea2 100%);

border-radius: 1.2rem;

padding: 2rem 1rem 1rem 1rem;

box-shadow: 0 4px 24px rgba(0,0,0,0.10);

margin-bottom: 2rem;

}

.predict-btn {

background: linear-gradient(90deg, #43cea2 0%, #185a9d 100%);

color: #fff !important;

border: none;

border-radius: 2rem;

padding: 0.7rem 2.5rem;

font-size: 1.2rem;

font-weight: bold;

margin-top: 1rem;

transition: box-shadow 0.2s;

box-shadow: 0 2px 8px rgba(67,206,162,0.2);

}

.predict-btn:hover {

box-shadow: 0 4px 24px rgba(24,90,157,0.2);

}

/\* Subtle shimmer animation for loader \*/

@keyframes shimmer {

0% { background-position: -400px 0; }

100% { background-position: 400px 0; }

}

.shimmer-loader

{

height: 40px;

width: 220px;

border-radius: 2rem;

margin: 0 auto 1.2rem auto;

background: #e0f7fa;

background-image: linear-gradient(

120deg,

#e0f7fa 30%,

#b2ebf2 50%,

#e0f7fa 70%

);

background-size: 400px 100%;

animation: shimmer 1.5s infinite linear;

}

</style>

""",

unsafe\_allow\_html=True

)

st.markdown('<div class="uploadbox">', unsafe\_allow\_html=True)

test\_image = st.file\_uploader("📷 \*Choose an Image:\*")

st.markdown('</div>', unsafe\_allow\_html=True)

show\_img = st.button("👁 Show Image", key="showimg")

predict\_btn = st.button("🔮 Predict", key="predictbtn")

if show\_img and test\_image is not None:

st.image(test\_image, use\_column\_width=True, caption="Uploaded Image")

if predict\_btn and test\_image is not None:

# Subtle shimmer loader animation

loader\_placeholder = st.empty()

loader\_placeholder.markdown(

"""

<div style="text-align:center;margin-top:2rem;">

<div class="shimmer-loader"></div>

<span style="font-size:2.5rem;">⏳</span>

<p style="font-size:1.2rem;color:#185a9d;">Analyzing image, please wait...</p>

</div>

""",

unsafe\_allow\_html=True

)

result\_index = model\_prediction(test\_image)

loader\_placeholder.empty() # Remove loader and analyzing text after prediction

class\_name = [

'Apple\_\_Apple\_scab', 'Apple\_Black\_rot', 'Apple\_Cedar\_apple\_rust', 'Apple\_\_healthy',

'Blueberry\_\_healthy', 'Cherry(including\_sour)\_Powdery\_mildew',

'Cherry\_(including\_sour)healthy', 'Corn(maize)\_Cercospora\_leaf\_spot Gray\_leaf\_spot',

'Corn\_(maize)Common\_rust', 'Corn\_(maize)Northern\_Leaf\_Blight', 'Corn(maize)\_healthy',

'Grape\_\_Black\_rot', 'Grape\_Esca(Black\_Measles)', 'Grape\_\_Leaf\_blight(Isariopsis\_Leaf\_Spot)',

'Grape\_\_healthy', 'Orange\_Haunglongbing(Citrus\_greening)', 'Peach\_\_\_Bacterial\_spot',

'Peach\_\_healthy', 'Pepper,\_bell\_Bacterial\_spot', 'Pepper,\_bell\_\_healthy',

'Potato\_\_Early\_blight', 'Potato\_Late\_blight', 'Potato\_\_healthy',

'Raspberry\_\_healthy', 'Soybean\_healthy', 'Squash\_\_Powdery\_mildew',

'Strawberry\_\_Leaf\_scorch', 'Strawberry\_healthy', 'Tomato\_\_Bacterial\_spot',

'Tomato\_\_Early\_blight', 'Tomato\_Late\_blight', 'Tomato\_\_Leaf\_Mold',

'Tomato\_\_Septoria\_leaf\_spot', 'Tomato\_\_Spider\_mites Two-spotted\_spider\_mite',

'Tomato\_\_Target\_Spot', 'Tomato\_Tomato\_Yellow\_Leaf\_Curl\_Virus', 'Tomato\_\_Tomato\_mosaic\_virus',

'Tomato\_\_\_healthy'

]

st.markdown(

f"""

<div style="

background: linear-gradient(90deg, #43cea2 0%, #185a9d 100%);

padding: 2.5rem 1rem 2rem 1rem;

border-radius: 1.5rem;

box-shadow: 0 6px 32px rgba(24,90,157,0.18);

margin-top: 2.5rem;

text-align: center;

animation: pop-in 1s cubic-bezier(.68,-0.55,.27,1.55);

">

<style>

@keyframes pop-in {{

0% {{ transform: scale(0.7); opacity: 0; }}

80% {{ transform: scale(1.05); opacity: 1; }}

100% {{ transform: scale(1); }}

}}

</style>

<h2 style="color: #fff; margin-bottom: 1rem; letter-spacing:1px;">

<span style="font-size:3rem;">🌿</span><br>

<span style="font-size:2rem;">Prediction Result</span>

</h2>

<p style="font-size: 1.7rem; color: #fff; font-weight: bold; margin-bottom:0.5rem;">

{class\_name[result\_index].replace('\_', ' ')}

</p>

<span style="font-size:1.1rem;color:#e0f7fa;">Thank you for using our system!</span>

</div>

""",

unsafe\_allow\_html=True

)

elif predict\_btn and test\_image is None:

st.warning("Please upload an image before predicting.")

-:**FUTURE SCOPE**: -

The proposed system offers a wide range of opportunities for future development and real-world deployment. In industrial environments, the use of advanced AI models can lead to smarter automation through predictive analysis and adaptive control. The surgery bot can be improved with haptic feedback and real-time data streaming, allowing remote medical operations or realistic training simulations. In the agricultural sector, the AI model can be further trained to detect a broader range of plant diseases, pests, and even soil conditions. Integrating environmental sensors like temperature, humidity, and soil moisture can convert the farming bot into a comprehensive smart farming assistant. Deploying the AI models on edge devices such as Raspberry Pi or ESP32 with AI capability will reduce reliance on cloud processing and improve speed and reliability in rural or low-connectivity regions. Additionally, the inclusion of cloud storage, voice-enabled commands, and multilingual user interfaces can enhance user interaction, data accessibility, and global scalability. With continued innovation, this gesture-controlled AI-integrated system has the potential to make significant contributions to industrial automation, healthcare, and sustainable agriculture.

The system can be enhanced with predictive “**AI models**”for smarter automation in industrial applications.

“**Haptic feedback**” and **remote control features** “can be added to the surgery bot for real-time surgical assistance and simulation.

In farming, the AI model can be expanded to identify **more crop diseases**, pest infestations, and soil deficiencies.

Integration with **environmental sensors** can turn it into a complete smart agriculture solution.

Deploying the model on **edge devices** will enable real-time processing without dependency on internet connectivity.

**Cloud synchronization** can allow remote data storage, analytics, and performance tracking.

Adding **voice command** and **multilingual support** will make the system more user-friendly and accessible globally.

The entire platform can be made **modular and scalable** allowing it to adapt to other sectors like disaster response and space robotics.

**-:CONCLUSION: -**

In this project, we successfully developed an AI-powered system that integrates gesture-based hand tracking, ESP32-CAM live video streaming, and a plant disease detection model to serve three distinct sectors: industrial automation, surgical assistance, and smart farming. The system utilizes real-time dual-hand gesture recognition through MediaPipe to control a robotic arm, allowing intuitive and contactless operation. Live camera streaming from the ESP32 module provides continuous monitoring, enhancing both remote visibility and safety. For the farming sector, the system captures plant images and processes them using a trained AI model to detect diseases accurately, enabling timely intervention and promoting healthier crop yields. This fusion of computer vision, AI, IoT, and robotics demonstrates a powerful, flexible platform capable of addressing critical needs across multiple domains. The project not only showcases innovative cross-domain integration but also sets the groundwork for future advancements in autonomous, intelligent, and adaptive control systems.

**-:BIBLIOGRAPHY:-**

 **Python 3 Documentation**  
<https://docs.python.org/3/>

 **NumPy Documentation**  
<https://numpy.org/doc/>

 **TensorFlow Documentation**  
<https://www.tensorflow.org/api_docs>

 **MediaPipe Documentation by Google**  
<https://developers.google.com/mediapipe>

 **Google AI Studio**  
<https://makerstudio.withgoogle.com/>

 **Arduino Uno Documentation**  
<https://docs.arduino.cc/hardware/uno-rev3>

 **ESP32-CAM Board Documentation**  
<https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-assistant/>

 **D1 Controller (NodeMCU / ESP8266) Documentation**  
<https://docs.wemos.cc/en/latest/d1/d1_mini.html>

 **Servo Motor Controller (PCA9685) Documentation**  
<https://learn.adafruit.com/16-channel-pwm-servo-driver>

 **Motor Driver Module (L298N) Documentation**  
<https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/>