

HAND GESTURE RECOGNITION

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

A Mini Project Report Submitted By

MADIREDDY NARENDRA KUMAR (20011P0413)

SAI AMULYA IYYAPU (20011P0421)

THANDRA VYSHNAVI (20011P0423)

GALITHOTTI NISHANT (20011P0424)

P SAI CHAITANYA (18011M2011)

Under the esteemed guidance of

Dr. M. ASHA RANI

Professor in ECE Department



Department Of Electronics and Communication Engineering
Jawaharlal Nehru Technological University Hyderabad
College Of Engineering Hyderabad
Autonomous
(Kukatpally -Hyderabad-500085)
2023-24

Department of Electronics and Communication Engineering
Jawaharlal Nehru Technological University Hyderabad
Kukatpally – Hyderabad-500085



CERTIFICATE BY THE SUPERVISOR

**This is to certify that the mini project report entitled **HAND GESTURE
RECOGNITION** being submitted by**

MADIREDDY NARENDRA KUMAR (20011P0413)

SAI AMULYA IYYAPU (20011P0421)

THANDRA VYSHNAVI (20011P0423)

GALITHOTTI NISHANT (20011P0424)

P SAI CHAITANYA (18011M2011)

in partial fulfilment of the requirements for the award of degree in Bachelor of Technology in Electronics and Communication Engineering at the Jawaharlal Nehru Technological University during the academic year, 2023-24 is a bonafide work carried out under my guidance and supervision. The results embodied in this minor project report have been verified and found to be satisfactory.

Supervisor:

Dr. M. Asha Rani

Professor of ECE

JNTUH College of Engineering, Hyderabad.

Department Of Electronics and Communication Engineering
Jawaharlal Nehru Technological University Hyderabad -
Kukatpally -Hyderabad-500085



CERTIFICATE BY THE HEAD OF THE DEPARTMENT

This is to certify that the project report entitled **HAND GESTURE RECOGNITION**

BEING SUBMITTED BY

MADIREDDY NARENDRA KUMAR (20011P0413)

SAI AMULYA IYYAPU (20011P0421)

THANDRA VYSHNAVI (20011P0423)

GALITHOTTI NISHANT (20011P0424)

PEDAMALLA SAI CHAITHANYA (18011M2011)

in partial fulfilment of the requirements for the award of degree in Bachelor of Technology in Electronics and Communication Engineering at the Jawaharlal Nehru Technological University during the academic year, 2023-24 is a bonafide work carried out by them.

-

Head of Department
Dr.A. Rajani
Professor and Head
Department of ECE,JNTUH CEH.

Department Of Electronics and Communication Engineering
Jawaharlal Nehru Technological University Hyderabad
Kukatpally -Hyderabad-500085



DECLARATION OF THE CANDIDATES

We hereby declare The Mini Project entitled **HAND GESTURE RECOGNITION** is a bonafide record work done and submitted under the esteemed guidance of **Dr.M.Asha Rani**, Professor, Department of ECE, JNTUH CEH, in partial fulfillment of the requirements for Mini project in Electronics and Communication Engineering at the Jawaharlal Nehru Technological University during the academic year 2023-24 is a bonafide work carried out by us and the results kept in the mini project has not been reproduced. The results have not been submitted to any other institute or university for the award of a degree or diploma.

MADIREDDY NARENDRA KUMAR (20011P0413)

SAI AMULYA IYYAPU (20011P0421)

THANDRA VYSHNAVI (20011P0423)

GALITHOTTI NISHANT (20011P0424)

PEDAMALLA SAI CHAITHANYA (18011M2011)

ACKNOWLEDGEMENT

This project titled **HAND GESTURE RECOGNITION** was carried out by us. We are grateful to Prof. Dr. M. Asha Rani, and Prof. Dr. A. Rajani, Professor, and Head of the Department of Electronics and Communication Engineering, JNTU Hyderabad College of Engineering Hyderabad, for their guidance while pursuing this project.

We take this precious opportunity to acknowledge our internal project guide Prof. Dr. M. Asha Rani, Professor of Electronics and Communication Engineering, JNTUH College of Engineering, Hyderabad for her timely advice, effective guidance, and encouragement throughout the completion of our mini-project work.

We also owe a deep respect of gratitude to our parents and friends for their cheerful encouragement and valuable suggestions, without whom this work would not have been completed in this stipulated time.

We would like to articulate our heartfelt gratitude to the authorities of JNTU for their help throughout our project work. A few lines of acknowledgment do not fully express our gratitude and appreciation for all those who guided and supported us throughout this project. Lastly, we acknowledge the help received from many journals and websites. Finally, we thank one and all who helped us directly or indirectly throughout our project work.

MADIREDDY NARENDRA KUMAR (20011P0413)

SAI AMULYA IYYAPU (20011P0421)

THANDRA VYSHNAVI (20011P0423)

GALITHOTTI NISHANT (20011P0424)

PEDAMALLA SAI CHAITHANYA (18011M2011)

	INDEX	
S.No	CONTENTS	Page Number
1	LIST OF FIGURES	8
2	LIST OF TABLES	9-10
3	LIST OF ABBREVIATIONS	11
4	ABSTRACT	11
5	1.Introduction	13-17
6	1.1 Introduction to Hand Gesture Recognition	13
7	1.2 Aim	15
8	1.3 Objectives	15
9	1.4 Methodology	15-17
10	2. Literature Review	18-19
11	3. Software and Hardware Requirements	20-21
12	3.1 Software Requirement	20
13	3.2 Hardware Requirement	20-21
14	4. Overview of Hand Gesture Recognition	22-30
15	4.1 Initialization	22
16	4.2 Contour Detection and Convex Hull	23
17	4.3 Defects Analysis and Gesture Recognition	24
18	4.4 Image Processing and ROI Definition	24-26
19	4.5 Textual Feedback and Gesture Display	26-29
20	4.6 Real-Time Gesture Recognition Loop	29-30
21	4.7 Conclusions	30
22	5. Raspberry Pi	31-43
23	5.1 Speed Specifications	32-33
24	5.2 Setting up Raspberry Pi	34-35
25	5.3 Installing the Operating System	35-37
26	5.4 Why we used Raspberry Pi over other Computers	37-39

27	5.5 PuTTY Software	39-40
28	5.6 Virtual Network Computing (VNC)	41-43
29	6.1 Audio Analysis and Visualization for Hand Gesture Recognition	44-45
30	6.2 Output for Different Hand Gestures	46-47
30	7. Conclusions and Future Scope	48-51
31	7.1 Comparisons of Implementation of Hand Gesture Recognition with and without Raspberry Pi	48
32	7.2 Conclusions	49
33	7.3 Future Scope	50
34	7.4 References	51

LIST OF FIGURES

Figures No.	Caption	Page No.
Figure 1	Hand Gesture Recognition Process	14
Figure 2	Initialization	22
Figure 3	Contour Detection and Convex Hull	23
Figure 4	Function: find_defects()	24
Figure 5	Function: process_img()	25
Figure 6	The mask and green rectangle area of ROI	26
Figure 7	Function: Text-for detection(a)	27
Figure 8	Function: Text-for detection(b)	28
Figure 9	Gesture Recognition after Error Handling	29
Figure 10	Different types of Raspberry Pi boards	33
Figure 11	Advanced Menu settings	36
Figure 12	Selecting the appropriate options & writing on to the SD card	37
Figure 13	Raspberry Pi	38
Figure 14	Software configuration tool	38
Figure 15	PuTTY configuration settings	40
Figure 16	GUI after logging with Remote Desktop	40
Figure 17	VNC viewer App Interface	42
Figure 18	Authentication to VNC server	42
Figure 19	Raspberry Pi Desktop	43
Figure 20	GPIU Pinout Diagram	43
Figure 21	MATLAB code	44

Figure 22	Time & Frequency Domain Plots	45
Figure 23	Frame and mask of 3	46
Figure 24	Frame and mask of 1	46
Figure 25	Frame and mask of 5	46

LIST OF TABLES

Table No.	Title	Page No.
Table 1.1	Hand Gesture Recognition Applications	14
Table 5.1	Raspberry pi Specifications	32
Table 5.2	Difference between Raspberry Pi and other comparisons	37
Table 7.1	Output Comparisons	46

LIST OF ABBREVIATIONS

CV	Computer Vision
CNN	Convolution Neural Networks
GUI	Graphical User Interface
AI	Artificial Intelligence
SSH	Secure Shell / Secure Socket Shell
SPI	Serial Peripheral Interface
VNC	Virtual Network Computing
OS	Operating System
GPIO	General Purpose Input and Output
USB	Universal Serial Bus
IP	Internet Protocol
CPU	Central Processing Unit
I/O	Input-Output
RAM	Random Access Memory
VS Code	Visual Studio Code
IDE	Integrated Development Environment
HDMI	High Definition Multimedia Interface
ROI	Region Of Interest

TITLE: HAND GESTURE RECOGNITION

ABSTRACT

Edge Hand Gesture Recognition technology involves teaching computers to understand the different shapes and movements which our hands make. The real-world applications of this technology are far-reaching, ranging from interactive presentations to contactless control of smart devices. Furthermore, the project highlights the benefits of this approach in environments where minimizing physical contact is crucial, including medical facilities and public spaces.

Hand Gesture Recognition can be implemented using OpenCV which stands for “Open-Source Computer Vision Library.” It is an open-source software library that provides tools and algorithms for computer vision and image processing tasks. It is chosen for this project due to its ability to detect and understand hand shapes, movements and patterns which help in transforming human gestures into meaningful computer commands. This technology finds relevance in a wide range of real-world scenarios from interactive user interfaces and gaming to assistive technologies and contactless control systems.

In our project, we use the Raspberry Pi, a compact and affordable single-board computer, to create a practical hand gesture recognition system using OpenCV. The Raspberry Pi serves as the computational powerhouse and process real-time camera data to detect and interpret gestures. This technology has the potential to revolutionize user interfaces in various applications, from smart home devices to interactive displays. By using the Raspberry Pi's flexibility along with OpenCV's strong ability to understand images, we aim to bring gesture recognition to the forefront of human-computer interaction. This will lead to new and easy ways for people to use technology.

CHAPTER-1

INTRODUCTION

Introduction to Hand Gesture Recognition:

Hand gesture recognition is a fascinating field of computer vision and human-computer interaction that focuses on enabling machines to understand and interpret the movements and configurations of human hand. This technology allows users to communicate with computers and devices through gestures and movements, without the need for physical interfaces like keyboards or touchscreens. Hand gesture recognition has a wide range of applications, from gaming and virtual reality to sign language interpretation, robotics, and even healthcare. By harnessing the power of machine learning and computer vision algorithms, hand gesture recognition systems empower users to interact with technology in a more intuitive and natural way, bridging the gap between human expression and digital interfaces.

Hand gesture recognition relies on the analysis of data captured by cameras or depth sensors, which track the movement and shape of the hand in real-time. These systems can recognise a multitude of gestures, from simple commands like pointing or waving to complex sign language gestures or even fine-grained finger movements. To achieve accurate recognition, advanced machine learning techniques such as neural networks and convolutional neural networks (CNNs) are often employed to process and interpret the visual data. The growing interest in hand gesture recognition is driven by its potential to enhance human-computer interaction, making it more natural and accessible.

OpenCV, an open-source computer vision library, plays a pivotal role in hand gesture recognition. It offers a rich set of functions and tools for image processing, feature extraction and machine learning, making it well-suited for detecting and interpreting hand movements. OpenCV's versatility and compatibility with Raspberry Pi's architecture enables developers to implement complex gesture recognition algorithms efficiently. Additionally, its Python bindings make it accessible to a broad community of developers and enthusiasts.

Hand gesture recognition has the potential to make technology more accessible to individuals with disabilities, particularly those with mobility impairments. By recognizing hand gestures, people with limited mobility can control computers, navigate user interfaces, and communicate more effectively, enhancing their quality of life and independence.

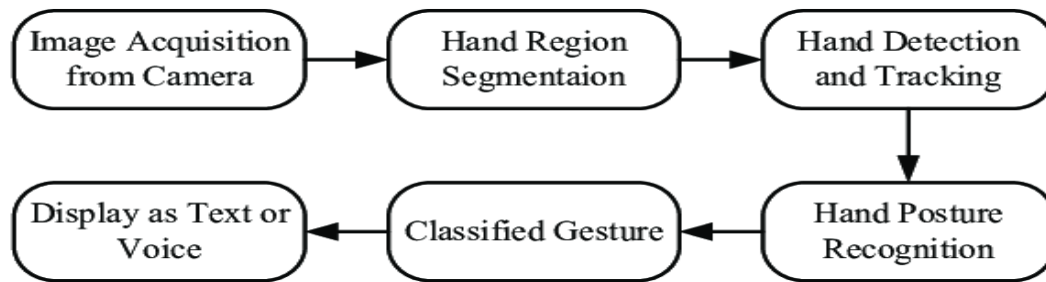


Figure 1: Hand Gesture Recognition Process

The above block diagram represents the hand gesture recognition process through a series of interconnected steps that collectively enable accurate recognition.

The hand gesture recognition process begins with image acquisition from a camera, capturing continuous frames. These frames undergo hand region segmentation to isolate the user's hand from the background, followed by hand detection and tracking to determine its position and movement. After tracking, hand posture recognition extracts key features like finger positions and shape, enabling classification into predefined gesture categories. The recognized gesture is then mapped to specific actions or commands, such as text input or voice commands, providing a user-friendly interface with both visual and auditory feedback for interaction with various applications or devices.

APPLICATIONS:

- Image Identification and Recognition
- Virtual Reality (VR) and Augmented Reality (AR)
- Sign Language Interpretation
- Human-Robot Interaction
- Smart Home control
- Medical Imaging
- Automotive Interfaces
- Gesture-controlled systems in Education and at work
- Security

Table 1.1: Hand Gesture Recognition Applications

1.2 Aim:

The aim of this project is to create an interactive system that detects and recognizes hand gestures using computer vision techniques and implement it on Raspberry Pi. The system aims to capture live video input from a webcam, process the frames and provide immediate feedback on the recognized gestures.

1.3 Objectives:

1. Develop Python code for Hand Gesture Recognition algorithm using OpenCV Library.
2. Simulation of the developed Python code.
3. Implementation of the code on Raspberry Pi.
4. Generate a plot for the audio output received in time and frequency domain using MATLAB Software.

1.4 Methodology:

Hand Gesture Recognition is a multi-step process that involves capturing, processing, and interpreting hand movements to recognize specific gestures. Below is a detailed methodology for implementing a hand gesture recognition system:

1. Image Acquisition from Camera:

- The process begins with capturing images or video frames of the user's hand using a camera. This camera may be a standard RGB camera or a depth-sensing camera, depending on the system's requirements. The camera continuously streams images, providing input data for the subsequent stages. A higher resolution and frame rate can contribute to better recognition accuracy.

2. Hand Region Segmentation:

- After acquiring the images, the system performs hand region segmentation to isolate the user's hand from the background. This step involves background subtraction and thresholding techniques to create a binary mask that highlights the hand. The segmented hand region is essential for accurate hand gesture recognition, as it reduces the computational load and focuses on the relevant area of interest.

3. Hand Detection and Tracking:

- The segmented hand region is then subjected to hand detection and tracking algorithms. These algorithms locate the hand within the segmented region and establish its position and motion in subsequent frames. Techniques like contour analysis, bounding box tracking, or optical flow can be employed for this purpose. Tracking helps ensure that the system maintains focus on the hand, even if it moves within the camera's field of view.

4. Hand Posture Recognition:

- Once the hand is accurately tracked, the system extracts relevant features, such as finger positions, hand shape, and hand orientation, from the tracked hand region. These features are used for hand posture recognition. Various machine learning or computer vision algorithms can be applied to classify the hand posture into predefined categories corresponding to specific gestures. The recognition model should be trained on a dataset of hand postures to accurately identify user gestures.

5. Classified Gesture:

- The recognized hand posture is then classified into a specific gesture category. For instance, if the user forms a fist, the system may classify it as a "closed fist" gesture. The classification result is sent to the next stage for further processing.

6. Display as Text and Voice:

- Finally, the classified gesture is converted into a meaningful action or command, such as text input or voice command. For text output, the recognized gesture can be mapped to corresponding textual characters or commands. Additionally, for voice output, a text-to-speech (TTS) system can be employed to audibly convey the recognized gesture's meaning to the user. This multi-modal output enhances the user experience by providing both visual and auditory feedback.

The procedure until the above step will be implemented on the Raspberry Pi board so that the overall output's accuracy, efficiency can be improved and the latency can be decreased. In this Deployment phase, compatibility has to be ensured with the target hardware and software platforms that is Raspberry pi board and our personal computer.

Moving forward to User interface (UI) Integration, where a user interface that displays the recognized gestures or provides feedback to the user. This interface may include visual cues, sounds or haptic feedback. In this project we would be using visual display where the allotted task for the specific hand gesture will be displayed on the screen and use audio interface where the same would be heard as audio output. The overall audio output will also be plotted using MATLAB functionalities. Implementing hand gesture recognition involves a combination of image processing and user interface design.

CHAPTER - 2

LITERATURE REVIEW

The realm of hand gesture recognition using OpenCV and Raspberry Pi has witnessed substantial advancements as researchers and developers continue to explore novel approaches and techniques. The literature review for our minor project on hand gesture recognition using OpenCV and Raspberry Pi delves into previous research in the field of computer vision and gesture recognition, highlighting key studies and contributions. This review provides essential insights to inform our project's development.

Hasan's Multivariate Gaussian Approach for Hand Gesture Recognition:

Hasan's research focused on hand gesture recognition using multivariate Gaussian distribution and non-geometric features. They employed skin-color based segmentation and clustering-based thresholding techniques to extract hand features. By dividing the hand image into 11 terraces and 8 sectors within each terrace, they aimed to capture the Gaussian function's shape effectively. Hasan's method accounted for rotation effects, resulting in promising recognition rates. Their work contributed to the field by addressing the challenges of hand gesture recognition and providing a robust approach.

Wysoski Rotation-Invariant Postures Using Boundary Histogram:

Wysoski introduced rotation-invariant postures using boundary histograms and neural networks. They utilized skin color detection and boundary normalization techniques, representing boundaries as chord size histograms. By employing Neural Networks MLP and Dynamic Programming matching, they achieved robust results in recognizing 26 static postures from American Sign Language. This approach showed the significance of boundary-based features and neural networks in sign language recognition.

Kulkarni's Neural Network Approach for Sign Language Recognition:

Kulkarni tackled static American Sign Language gesture recognition using neural networks. They adopted image preprocessing techniques, including HSV color conversion and resizing, followed by feature extraction using histograms and Hough algorithms. With a feed-forward neural network comprising three layers, they achieved a notable recognition rate of 92.78%. Kulkarni's study demonstrated the effectiveness of neural networks in recognizing sign language gestures, offering a valuable contribution to communication technology for the hearing impaired.

Stergiopoulou's Self-Growing Neural Gas Network for Hand Gesture Recognition:

Stergiopoulou proposed a novel Self-Growing and Self-Organized Neural Gas (SGONG) network for hand gesture recognition. Their approach involved skin color segmentation in the YCbCr color space, finger identification for hand shape analysis, and Gaussian distribution modelling for recognition. Stergiopoulou's work contributed to advancing gesture recognition technology by introducing an innovative neural network model capable of handling complex hand shapes.

These studies collectively demonstrate the diverse approaches and techniques employed in the field of hand gesture recognition, showcasing the continuous efforts to improve accuracy and usability in applications ranging from sign language communication to human-computer interaction.

CHAPTER - 03

Software and Hardware Requirements

3.1 Software Requirement:

Operating System (Raspberry Pi, Windows): The project supports two operating systems, Raspberry Pi and Windows. Raspberry Pi is a versatile platform for embedded projects, while Windows provides a more traditional development environment.

Language Used (Python): Python is the primary programming language for this project due to its ease of use and compatibility with Raspberry Pi. It offers a wide range of libraries for computer vision and machine learning tasks.

Libraries Used:

OpenCV: OpenCV is a critical library for computer vision applications. It provides tools and functions for image processing, video analysis, and feature extraction, making it essential for hand gesture recognition.

NumPy: NumPy is a fundamental library for scientific computing in Python. It is used for handling arrays and numerical operations, which are commonly required in image processing tasks.

Pygame: Pygame is used for creating interactive and graphical user interfaces (GUIs). It can be helpful for displaying feedback or results related to hand gesture recognition.

Imutils: Imutils is a set of convenience functions that simplify common OpenCV tasks. It can streamline the development process by providing easy-to-use functions for resizing, rotating, and displaying images.

3.2 Hardware Requirement:

Raspberry Pi 4 Model B: The Raspberry Pi serves as the core hardware for the project, providing computational power for image processing and gesture recognition.



Processor (Intel® Core™ i5-8265U): In the case of Windows development, an Intel Core i5 processor is used, providing ample computing resources for software development.



HDMI Cable and Ethernet Cable: An HDMI cable is used to connect the Raspberry Pi to a display such as Monitor, Ethernet Cable is to provide a network connection to the Raspberry Pi when required.



Input Device (Mouse or Keyboard): An input device is required for user interaction, allowing users to select options or navigate the system.

Camera Module or USB Webcam: A camera module or USB webcam is essential for capturing video input, which is then processed for hand gesture recognition.



Logitech C270

MicroSD Card and Card Holder: The microSD card is used to store the Raspberry Pi's operating system and project files. A card holder helps secure the microSD card.



These hardware and software components collectively enable the development of a hand gesture recognition system using OpenCV on Raspberry Pi or Windows, providing a versatile and interactive platform for the project.

CHAPTER-04

OVERVIEW OF HAND GESTURE RECOGNITION

Computer vision is a broad field of artificial intelligence that focuses on enabling computers to interpret and understand visual information from the world, typically through images and videos. Hand Gesture Recognition specifically deals with the task of identifying and interpreting hand movements and gestures within images or video streams. This technology enables computers to interpret and respond to hand movements and gestures, providing a more intuitive and immersive user experience.

In this project, we explore the development of a Real-time Gesture Recognition System using OpenCV, a powerful open-source computer vision library. By harnessing OpenCV's capabilities in image processing, contour detection, and convex hull analysis, coupled with audio feedback from Pygame, we have created an interactive system capable of recognizing and responding to hand gestures in real-time.

4.1 – Initialization and Setup:

```
# Initialize Pygame
pygame.mixer.init()
pygame.mixer.music.set_volume(1.0) # Adjust the volume as needed

# Define sound files for each gesture
sounds = {
    0: "tryAgainw.wav",
    1: "onew.wav",
    2: "twow.wav",
    3: "threew.wav",
    4: "fourw.wav",
    5: "hifi_soundw.wav",
    6: "tryAgainw.wav", # Adjust for the "other" gesture
}
```

Figure 2:Initialization

1. Initializing Pygame and Sound Effects:

- Function Name: `pygame.mixer.init()` :

This function initializes the Pygame library for audio playback. It sets the volume to maximum (1.0), allowing us to adjust the volume as needed for audio feedback during gesture recognition.

2. Defining Sound Files for Gestures :

- sounds = {...}

Explanation: In this dictionary, we define sound files corresponding to different hand gestures. Each gesture is associated with a sound file, which will be played when the respective gesture is recognized.

4.2 – Contour Detection and Convex Hull:

```
def contours_convex_hull(mask,roi,frame):  
    # find contours  
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  
  
    # find contour of max area(hand)  
    cnt = max(contours, key=lambda x: cv2.contourArea(x))  
  
    # approx the contour a little  
    epsilon = 0.0005 * cv2.arcLength(cnt, True)  
    approx = cv2.approxPolyDP(cnt, epsilon, True)  
  
    # make convex hull around hand  
    hull = cv2.convexHull(cnt)  
  
    # define area of hull and area of hand  
    areahull = cv2.contourArea(hull)  
    areacnt = cv2.contourArea(cnt)  
  
    # find the percentage of area not covered by hand in convex hull  
    arearatio = ((areahull - areacnt) / areacnt) * 100  
  
    # find the defects in convex hull with respect to hand  
    hull = cv2.convexHull(approx, returnPoints=False)  
    defects = cv2.convexityDefects(approx, hull)  
  
    find_defects(defects,approx,roi,areacnt, arearatio, frame) #calling other function by passing arguments
```

Figure 3:Contour Detection and Convex Hull

Contour Detection and Hand Approximation:

- Function Name: contours_convex_hull (mask, roi, frame)

Explanation: This function detects the hand's contour within the processed image (mask) and approximates the contour to simplify it. It then creates a convex hull around the hand's contour and calculates the area of the hand and the convex hull. Additionally, it identifies and analyses defects in the convex hull, which correspond to gaps between fingers.

4.3 – Defect Analysis and Gesture Recognition:

- Function Name: find_defects (defects, approx, roi, areacnt, arearatio, frame)

Explanation: This function analyses the defects found in the convex hull to determine the number of fingers extended in the gesture. It also considers the area of the hand and the ratio of the area not covered by the hand in the convex hull. Based on these parameters, it recognizes and classifies the hand gesture.

```
def find_defects(defects, approx, roi, areacnt, arearatio, frame):
    l=0 # defect count
    # code for finding no. of defects due to fingers
    for i in range(defects.shape[0]):
        s, e, f, d = defects[i, 0]
        start = tuple(approx[s][0])
        end = tuple(approx[e][0])
        far = tuple(approx[f][0])
        pt = (100, 180)
        # find length of all sides of triangle
        a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
        b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
        c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
        s = (a + b + c) / 2
        ar = math.sqrt(s * (s - a) * (s - b) * (s - c))
        # distance between point and convex hull
        d = (2 * ar) / a
        # apply cosine rule here
        angle = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 57
        # ignore angles > 90 and ignore points very close to convex hull(they generally come due to noise)
        if angle <= 90 and d > 30:
            l += 1
            cv2.circle(roi, far, 3, [255, 0, 0], -1)
        # draw lines around hand
        cv2.line(roi, start, end, [0, 0, 255], 2)
    text_for_detection(l+1, areacnt, arearatio, frame) # Note that we need to call with l+1
```

Figure 4: Function: find_defects()

4.4 – Image Processing and ROI Definition:

The fundamental idea driving this algorithm is to make use of a live camera feed and process each frame in real-time. The first crucial step involves converting the region of interest (ROI) extracted from the frame from BGR into the HSV (Hue, Saturation, Value) color space. A green rectangle is drawn around the ROI to define the gesture-detection area.

- The command **roi = frame[100:500, 100:500]** defines the region of interest. The ROI is specified by specifying the position [100:500, 100:500] with 100 to 499 rows and 100 to 499 columns on the stored image, frame.


```

def process_image(img):
    if img is not None:
        # define region of interest
        roi = frame[100:500, 100:500]

        cv2.rectangle(frame, (100, 100), (500, 500), (0, 255, 0), 2) #draw green rectangle to detect gestures inside

        hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # define range of skin color in HSV
        lower_skin = np.array([0, 20, 70], dtype=np.uint8)
        upper_skin = np.array([20, 255, 255], dtype=np.uint8)

        # extract skin color image
        mask = cv2.inRange(hsv, lower_skin, upper_skin)

        # extrapolate the hand to fill dark spots within
        mask = cv2.dilate(mask, kernel, iterations=4)

        # blur the image
        mask = cv2.GaussianBlur(mask, (5, 5), 100)

    return mask

```

Figure 5:Function- process_img()

- The next command **cv2.rectangle(frame, (100, 100), (500,500), (0, 255, 0), 2)** is used to draw a green rectangle on the frame using the command **cv2.rectangle**. This rectangle represents the region of interest, within which the hand contour has to be detected and recognized. (100, 100) specifies the top-left corner and (500, 500) represents the bottom-right corner of the rectangle. (0, 255, 0) represents the green color and 2 specifies that the thickness of the rectangle's border is of 2 pixels thickness.
- Moving on to **hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)**, here the transformation of the BGR color space within the roi to HSV (Hue, Saturation and Value) is done. The function **cv2.COLOR_BGR2HSV** is used for this purpose. HSV is a common technique for skin color detection because it's easier to define a range of skin colors in HSV space.
- Followed by this is defining two numpy arrays that represent the lower and upper bounds for a specific skin color range in the HSV. The lower bound is represented by **lower_skin = np.array([0, 20, 70], dtype = np.uint8)** and the upper bound is represented by **upper_skin = np.array([20, 255, 255], dtype = np.uint8)**. Here [0, 20, 70] corresponds to low hue (reddish), moderate saturation and moderate brightness, and [20, 255, 255] represents a higher hue (yellowish- green), full saturation and maximum brightness.
- The command **mask = cv2.inRange(hsv, lower_skin, upper_skin)** provides a binary mask using the command **cv2.inRange** function. This mask identifies the pixels within the HSV image that match the defined skin color range. Pixels falling within the specified range that is those representing the hand contour are displayed in white color in the mask and those outside the range that is those not representing the hand contour are rendered as black.

- The next command **mask = cv2.dilate(mask, kernel, iterations = 4)** applies morphological dilation on the mask using the kernel and the dilation operation is iterated four times.
- This line **mask = cv2.GaussianBlur(mask, (5, 5), 100)** applies a Gaussian blur filter to binary mask using the cv2.GaussianBlur function with a 5x5 pixel kernel and a strong blur effect with standard deviation of 100. This filter reduces noise and smoothens the image.
- The prepared mask is returned using the command **return mask**.

To distinguish the human skin color within the ROI, we specify a range of HSV values that correspond to the skin color, effectively creating a mask. This mask takes on a binary format, displaying regions as either zero or one. To enhance its accuracy, we proceed to dilate the mask which fills in gaps, irregularities in the hand contour and smoothens edges.

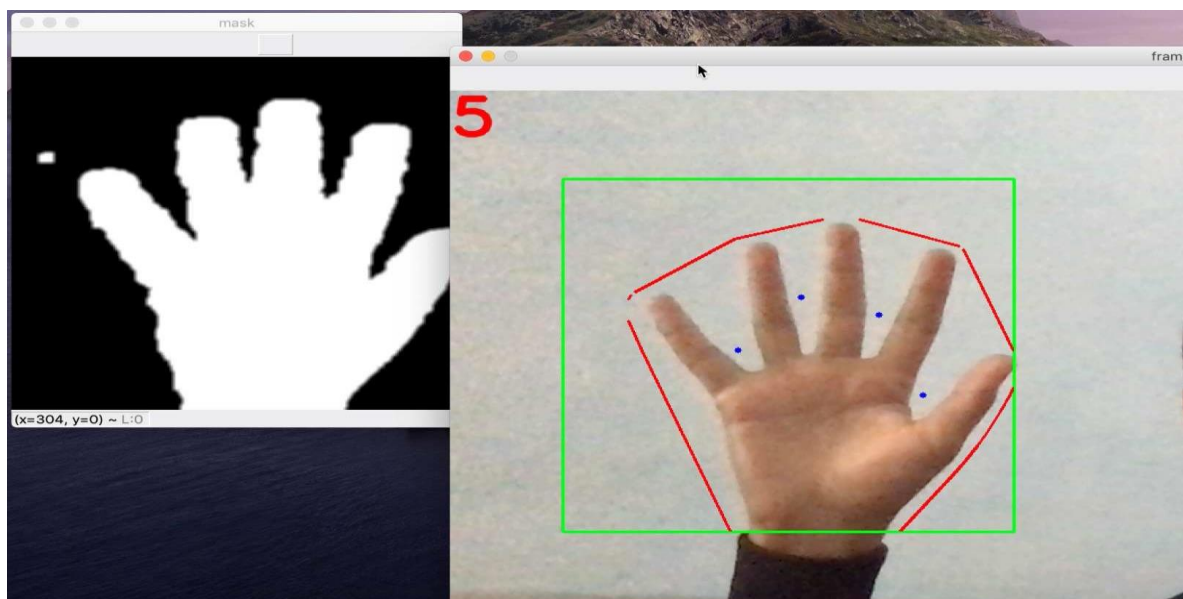


Figure 6: The mask and green rectangle area of ROI

4.5 – Textual Feedback and Gesture Display:

- Function Name: text_for_detection (l, areacnt, arearatio, frame)
- Explanation: This function provides textual feedback on the frame based on the detected gesture. It displays messages such as "Reposition," "One," "Three," and "High Five" on the frame to visually indicate the recognized gesture. It also handles cases where the gesture is not recognized.

```

def text_for_detection(l,areacnt, arearatio,frame):
    '''
    hand gestures:
    0=empty
    1=one
    2=two
    3=three
    4=four
    5=high five
    6=other/reposition
    '''
    texts = ['Reposition!',
             'Put your hand',
             'Like',
             'Three',
             'High Five!']
    if l == 1:
        if areacnt < 2000:
            cv2.putText(frame, texts[1], (0, 50), font, 2, (255, 0, 0), 3, cv2.LINE_AA)
        else:
            if arearatio < 12:
                cv2.putText(frame, '0', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
            elif arearatio < 17.5:
                cv2.putText(frame, texts[2], (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
            else:
                cv2.putText(frame, '1', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

```

Figure 7:Function:Text_for detection(a)

- This function takes four parameters **l**, **areacnt**, **arearatio** and **frame** which represent the defects count, area of the contour, percentage of area not covered by the hand and the processed-image respectively. The function **cv2.putText** is used to draw the text on the frame. Also the **cv2.LINE_AA** is an anti-aliasing flag which is used to enable or disable the anti-aliasing when rendering the texts, lines, shapes of an image. When enabled it can be used to smoothen the edges of the text making the less jagged or pixelated.
- A list consisting of all the texts that will be displayed for the various hand gestures detected, as its elements is considered.
- The actual segregating of the contours and assigning the appropriate texts takes place with the help of if & else loops. The outer loop, looping parameter is dependent on the value of l, and the inner loop is dependent on the value of areacnt and arearatio. If l is equal to 1 and if areacnt is less than 2000 then the element at 1st index in the texts list which is 'Put your hand' is drawn at the position (0, 50) on the image, frame, which is at the top left corner, with the font Hershey Duplex that is initialized at the beginning at the global variables. The 2 in the command represents the font scale factor which here means that the text will be drawn twice its normal size and the tuple (255, 0, 0) represents that the text will be drawn with full intensity blue color using BGR color code used in OpenCv library applications, and 3 shows the thickness of the text character's lines. This command is shown when no hand contour is detected i.e., no hand is placed at the webcam.

- If the areacnt is not less than 2000, it means that the hand is placed at the webcam for gesture recognition then the else statements are executed. If the arearatio of the frame is less than 12 then '0' is drawn on the frame with full intensity red color due to the tuple (0, 0, 255) keeping rest of the font parameters same as earlier. Next, if the arearatio is in between 12 and 17.5 then the element at 2nd index in the texts list which is 'Like' is drawn on the frame. If neither of these conditions are satisfied, that is if the arearatio is greater than 17.5 with l equal to 1, then '1' is drawn on the frame.

```

elif l == 2:
    cv2.putText(frame, '2', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

elif l == 3:
    if arearatio < 27:
        cv2.putText(frame, '3', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)
    else:
        cv2.putText(frame, texts[3], (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

elif l == 4:
    cv2.putText(frame, '4', (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

elif l == 5:
    cv2.putText(frame, texts[4], (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

elif l == 6:
    cv2.putText(frame, texts[0], (0, 50), font, 2, (255, 0, 0), 3, cv2.LINE_AA)
else:
    cv2.putText(frame, texts[0], (10, 50), font, 2, (255, 0, 0), 3, cv2.LINE_AA)

#show the resulting frame
cv2.imshow('frame', frame)

```

Figure 8:Function:Text_for detection(b)

- If l is equal to 2 then '2' is drawn on the frame in full intensity red color keeping all the other font parameters same as earlier. This shows that two fingers are raised before the webcam.
- If l is equal to 3, it depicts that 3 fingers are raised in front of the webcam, then the inner loop conditions are checked with the arearatio parameter. If arearatio is less than 27 and greater than 17.5, then '3' is drawn on the frame using the same font. But if this condition is not satisfied, that is if the arearatio is greater than 27 then the element having 3rd index in the texts list, which is 'Three' is drawn on the frame.
- If l is equal to 4, it shows that four fingers are raised, then '4' is drawn on the frame with full intensity red with rest font parameters same as in earlier cases.

- If *l* is equal to 5, it represents that the entire palm or five fingers are raised before the webcam, then the element having 4th index in the texts list, which is 'High Five!' is drawn on the frame with full intensity blue with the rest font parameters same as earlier.
- If *l* is equal to 6, it shows that none of the above conditions are satisfied and user has to show the gesture again, then the element with 0th index which is 'Reposition!' is drawn on the frame with full intensity blue as per the BGR color code mentioned in the command with rest parameters same as earlier.
- To obtain the audio output we use the pygame library. A sounds dictionary is already defined at the start of the code, this dictionary is now used to obtain the audio output. For this, **pygame.mixer.music.load()** functionality of pygame is used, which selects the corresponding sounds given as parameters to the function. To play the selected sound the **pygame.mixer.music.play()** function is used. Since, the hand gesture recognition code developed is real time and the hand gestures are detected continuously, to obtain correct audio output with certain break for understanding the output, a delay of one second is given using **time.sleep(1.0)** function.

4.6 – Real-time Gesture Recognition Loop:

This code section contains the main loop of the program, where frames are continuously captured from the webcam using OpenCV.

```
cap = cv2.VideoCapture(0)
while True:
    try: # an error comes if it does not find anything in window as it cannot find contour of max area
        # therefore this try error statement
        ret, frame = cap.read()
        frame = cv2.flip(frame, 1)
        h,w,d = frame.shape
        roi = frame[100:500, 100:500] #defining ROI to detect gestures
        mask = process_image(frame)
        contours_convex_hull(mask,roi,frame)
        # show the masking window
        cv2.imshow('mask', mask)

    except:
        pass
    k = cv2.waitKey(1) & 0xFF
    if k == 27 or k == ord('q'):
        break
# After the loop release the cap object
cap.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

Figure 9:Gesture Recognition Loop and Error Handling

- A variable *cap* is initialized with `cv2.VideoCapture(0)`, to capture video frames from the default camera (index 0). It opens the camera for video input.

- A while loop is entered which runs indefinitely until the user presses the Esc key, that is key 27 or the 'q' key in the keyboard. This entire while loop is written within a try and except block such that if an error occurs when no gesture is detected, then exception handling can be done and it can be running without showing any errors. Within the while loop, an attempt to read a frame from the camera is done using `cap.read()`. The variable `ret` stores a Boolean value indicating whether the frame was successfully read or not and the frame contains the captured image frame.
- The captured frame is flipped horizontally using `cv2.flip(frame, 1)`. This is done to correct the orientation of the camera feed. The dimensions of the frame are measured using the method `frame.shape`, which calculates the height(h), width(w) and depth(d) of the frame. Next an roi is defined which is the region of interest to detect the gestures using `roi = frame[100:500, 100:500]`.
- Up next is processing of this frame to create a binary mask using the function `process_image(frame)`. Next to obtain the contours and subsequently the convex hull of the frame, the `contours_convex_hull` function, which is explained above is called by passing the binary mask, roi and frame as the arguments. This function detects and analyzes the hand gestures within the roi. In order to show the binary mask the function `cv2.imshow('mask', mask)` is used. The binary mask is displayed to visualize the processed image.
- The execution waits for a key press using `cv2.waitKey(1) & 0xFF`. If the Esc key or the q key are pressed then the loop exits and the program exits.
- After the loop, it releases the video capture object with `cap.release()` to free up the camera resources and finally closes all OpenCV windows with `cv2.destroyAllWindows()` function.

4.7 – Conclusion:

Each function serves a specific purpose in the overall process of capturing, processing and recognizing hand gestures in real-time. We highlight the seamless integration of OpenCV for image processing and Pygame for audio feedback, resulting in an engaging and interactive system capable of recognizing and responding to hand gestures in real-time.

CHAPTER-05

RASPBERRY PI

The Raspberry Pi, born out of a collaborative effort between the Raspberry Pi Foundation and Broadcom, stands as a pioneering series of diminutive single-board computers. Since its inception, it has captured the imagination of tech enthusiasts and educators alike, serving as a versatile platform that encourages experimentation and learning. Its inviting green circuit board beckons users to delve into programming, software creation, and a multitude of creative applications. In this abstract, we delve into the Raspberry Pi's evolution, its widespread adoption, and the pivotal role it plays in fostering computer science education and innovation.

This iconic computer, originally designed to facilitate basic computer science education in schools, soon transcended its initial purpose. Its affordability and open design made it a compelling choice for diverse projects, ranging from gaming devices to weather stations and beyond. Moreover, the Raspberry Pi has become a gateway for individuals of all ages to embark on their journey into the realms of computer science, democratizing access to technology and fostering a new generation of innovators.

Generations and Models:

In 2012, the company launched the Raspberry Pi and the current generations of regular Raspberry Pi boards are **Zero, 1, 2, 3, and 4**.

Generation 1 Raspberry Pi had the following four options:

- Model A
- Model A +
- Model B
- Model B +

Among these models, the **Raspberry Pi B models** are the original credit-card sized format.

On the other hand, the **Raspberry Pi A models** have a smaller and more compact footprint and hence, these models have the reduced connectivity options.

Raspberry Pi Zero models, which come with or without GPIO (general-purpose input output) headers installed, are the most compact of all the Raspberry Pi boards types.

5.1 Speed Specifications:

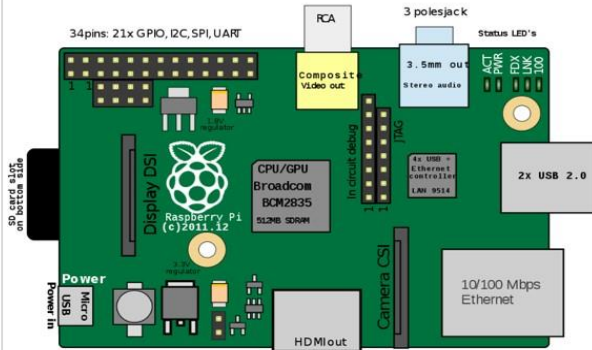
The table below gives the speed specifications of various Raspberry Pi models and generations.

Raspberry Pi Version	Weight (in grams)	GPIO	CPU Speed	Cores	RAM
Raspberry Pi 5(Oct-2023)	50	40 Pin	2.4 GHz	Quad	4GB/8GB
Raspberry Pi 4 Model B (2018-20)	46	40 Pin	1.5 GHz	Quad	1,2,4,8 GB
1Raspberry Pi 3 Model B+ (2018)	50	40 Pin	1.4 GHz	Quad	1 GB
Raspberry Pi 3 Model B+ (2016)	40	40 Pin	1.2 GHz	Quad	1 GB
Raspberry Pi 3 Model A+ (2018)	28	40 Pin	1.4 GHz	Quad	512 MB
Raspberry Pi Zero Wireless with headers (2017)	10	40 Pin	1 GHz	Single	512 MB
Raspberry Pi Zero Wireless (2016)	10	40 Pin Unpopulated	1 GHz	Single	512 MB
Raspberry Pi Zero (2015)	8	40 Pin Unpopulated	1 GHz	Quad	512 MB
Raspberry Pi 2 Model B (2015)	42	40 Pin	1.2 GHz	Single	1 GB
Raspberry Pi 1 Model B+ (2014)	42	40 Pin	700 MHz	Single	512 MB
Raspberry Pi 1 Model B (2012)	38	21 Pin (26 Pin Header)	700 MHz	Single	512 MB
Raspberry Pi 1 Model A+ (2014)	23	40 Pin	700 MHz	Single	512 MB
Raspberry Pi 1 Model A (2013)	30	21 Pin (26 Pin Header)	700 MHz	Single	256 MB

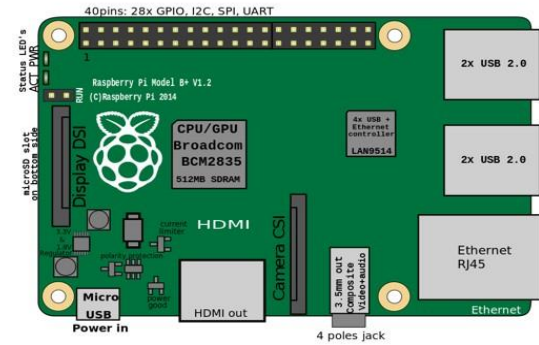
Table 3.1: Raspberry Pi Specifications

Raspberry Pi Model B

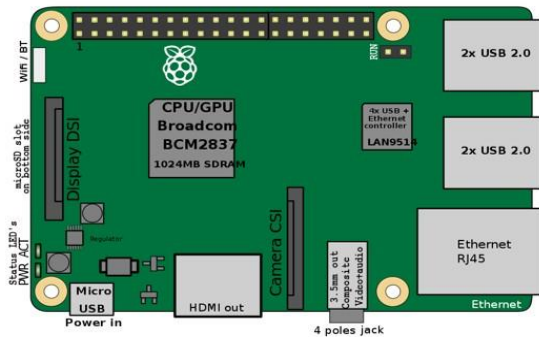
Raspberry Pi 1 Model B



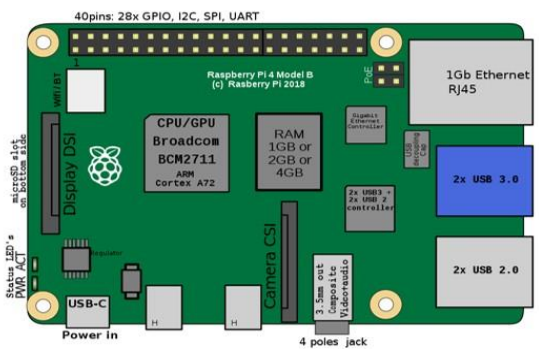
Raspberry Pi 1 Model B+



Raspberry Pi 3 Model B

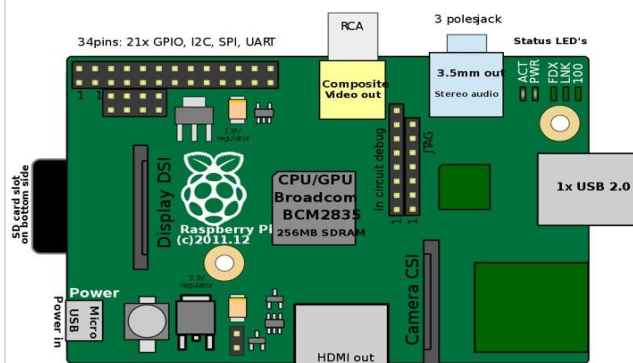


Raspberry Pi 4 Model B



Raspberry Pi Model A

Raspberry Pi 1 Model A



Raspberry Pi 1 Model A+

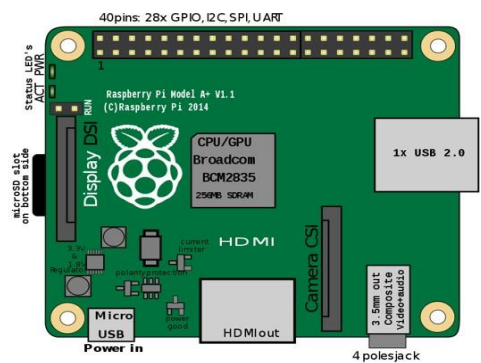


Fig: 10: Different types of Raspberry Pi Boards

5.2 Setting up Raspberry Pi :

To start using Raspberry Pi computer, We require the following things:

Monitor or TV:

We need a screen to see what is going on with your Raspberry Pi. Most computer monitors or even regular TVs will work just fine, but it is best if they have an HDMI input like modern TVs. We also need a cable to link your monitor or TV to your Raspberry Pi.

Keyboard and Mouse:

For control purposes, a standard keyboard and a mouse are employed with the Raspberry Pi. Wireless options are acceptable, provided they are already paired.

Power Supply:

To provide power to our Raspberry Pi, we need a dependable power supply, similar to a phone charger. We recommend using the official Raspberry Pi Power Supply because it's designed to deliver steady power, even when there are changes in power needs. This is especially important when we connect different devices to our Raspberry Pi. The official power supply includes a dedicated cable, making sure we don't accidentally use a lower-quality cable that might cause problems.

SD Card:

The SD card functions as the "brain" of our Raspberry Pi, holding the computer's operating system. We need a small memory card called an SD card, usually with a capacity of at least 8GB. To move the computer's "brain" to the SD card, we use a tool called the "Raspberry Pi Imager." It is crucial to mention that we mostly use micro SD cards for most Raspberry Pi models, although some older models use larger SD cards.

Network Cable:

To link our Raspberry Pi to our home network and the internet, we employ a network cable, much like the one used to connect our computer for internet access. This connection is crucial if we aim to allow our Raspberry Pi to reach online resources.

Speakers:

The Raspberry Pi has a standard audio out socket. This socket is compatible with headphones and speakers that use a 3.5mm audio jack. We can plug headphones directly to it.

Cables:

Following are some of the cables, which you need for the connections to the Raspberry Pi computer:

- HDMI cable
- HDMI-to-DVI adapter, if you are using a Digital Visual Interface (DVI) monitor.
- Audio cable
- Ethernet cable

These are the essential components we need to set up and operate our Raspberry Pi for different projects and tasks. Once we have gathered all these items, we can then proceed to configure and use our Raspberry Pi effectively.

5.3 Installing the Operating System:

To install the operating system on our Raspberry Pi, we suggest using the Raspberry Pi Imager. Here's how we can go about it:

1. Download Raspberry Pi Imager: Let's begin by downloading the latest version of Raspberry Pi Imager and installing it. If you're using a second Raspberry Pi, we can install it from a terminal by using the command “`sudo apt install rpi-imager`”.

2. Connect SD Card Reader: Insert the SD card into an SD card reader and connect it to our computer.

3. Open Raspberry Pi Imager: We can now launch the Raspberry Pi Imager software. From the list provided, we should select the operating system we want to install.

4. Choose SD Card: We should pick the SD card where we want to write the operating system image.

5. Write the Image: Once we've reviewed our selections and are ready, clicking the "Write" button will start writing the data to the SD card.

6. Insert SD Card into Raspberry Pi: Next, we can insert the SD card into our Raspberry Pi and power it up. During the first boot, a configuration wizard will run, allowing us to set up our Raspberry Pi.

It's important to note that Raspberry Pi OS no longer uses the default username "pi" and password "raspberrypi." However, in older versions of the operating system, or if we're working with an existing installation, this default user might still be present. To ensure the security of our Raspberry Pi, it's advisable to change the default password immediately.

When using Raspberry Pi Imager, after selecting the operating system, we may see an "Advanced Options" menu if it's supported by the operating system. This menu allows us to perform tasks such as enabling SSH, setting our Raspberry Pi's hostname, and configuring the default user before the first boot. Using the Advanced Options menu in Imager can skip the configuration wizard that typically runs on first boot.

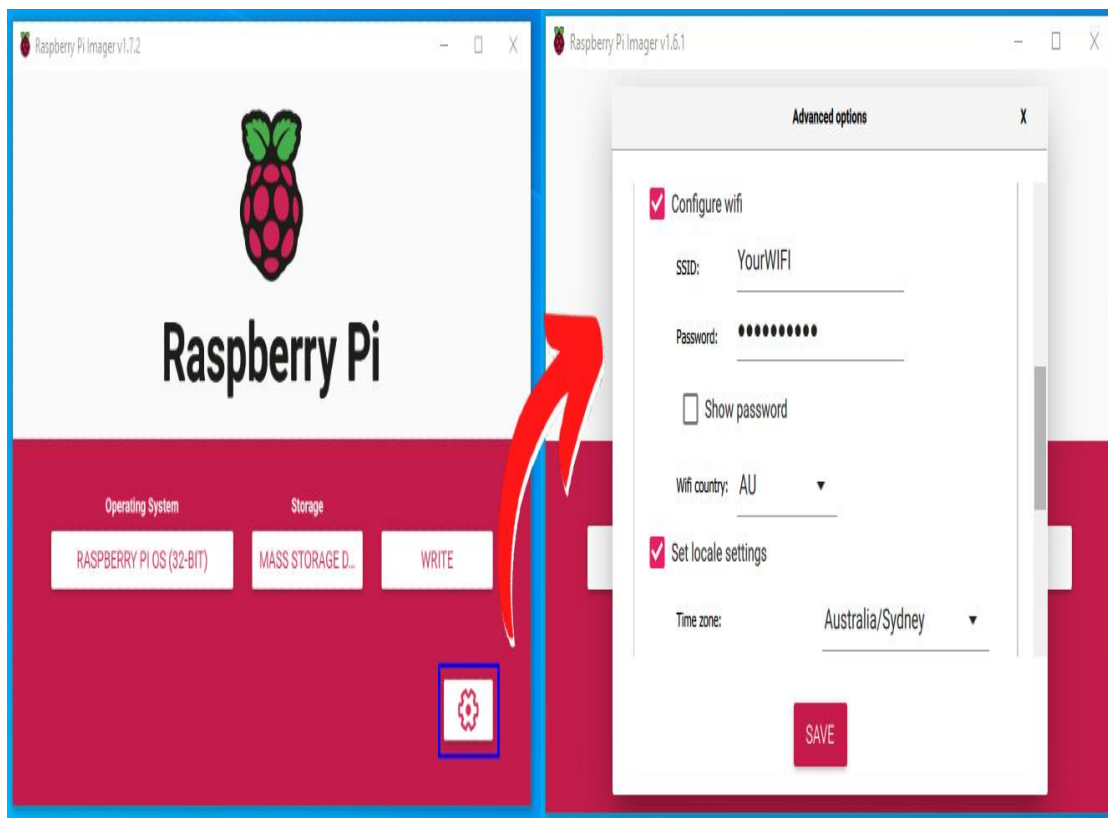


Figure 11: Advanced Menu Settings

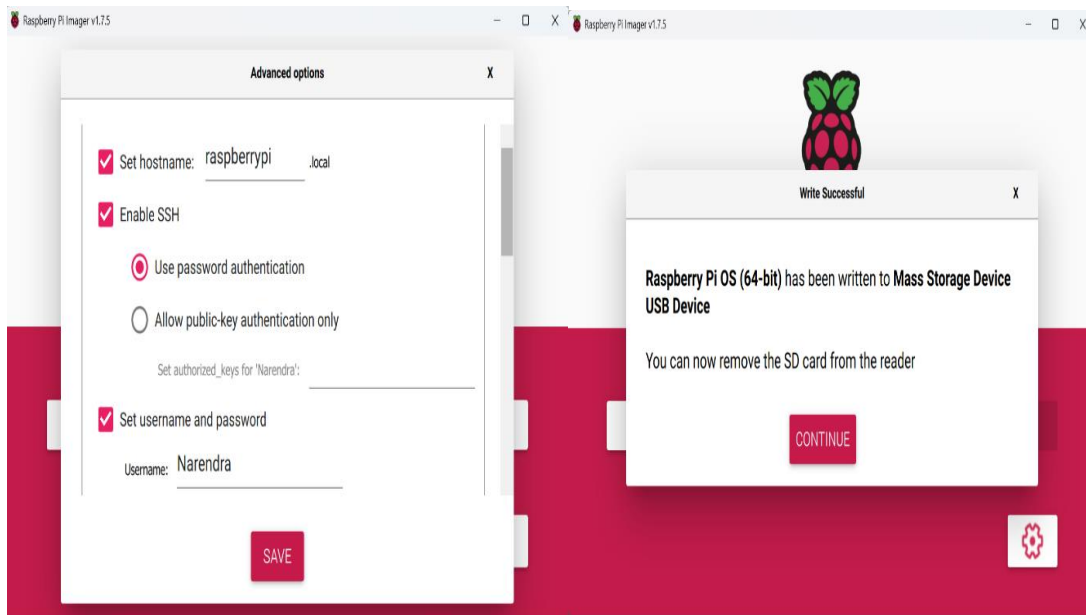


Figure 102: Selecting the Appropriate options and Writing on to the SD Card

5.4 Why we use Raspberry Pi over other Computers:

Aspect	Raspberry Pi	Other Computers
Affordability	Inexpensive	Varies, often more costly
Compact Size	Small form factor	Varies, can be large
Low Power Consumption	Energy-efficient	Varies, can be power-hungry
GPIO Pins	Available for hardware interfacing	May require additional hardware
Community Support	Large active community	Depends on the platform
Open Source	Based on open-source software	Varies, can be proprietary
Performance	Moderate for many tasks	High performance available
Customization	Flexible configuration	Limited by pre-built models
Educational Value	Widely used in education	Varies, depends on use case
Diverse Applications	Suitable for various projects	Depends on hardware and OS

Table 3.2 :Differences between Raspberry Pi and other Computers.

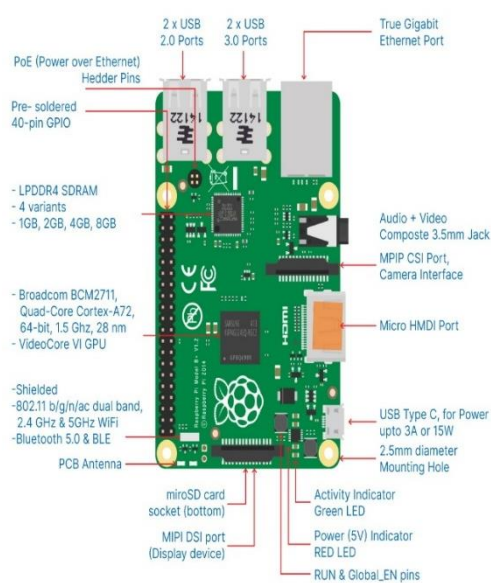


Figure 13: Raspberry Pi

Development and Optimization: The Raspberry Pi OS is continuously under development, with a strong emphasis on improving the stability and performance of various Debian packages specifically tailored for the Raspberry Pi platform. This commitment guarantees a dependable and high-performing environment for our projects.

Configuration Tool: To set up our Raspberry Pi, we can utilize the "raspi-config" tool, which was initially developed by Alex Bradbury. Accessing this tool is straightforward: just open a terminal and type the command "sudo raspi-config".

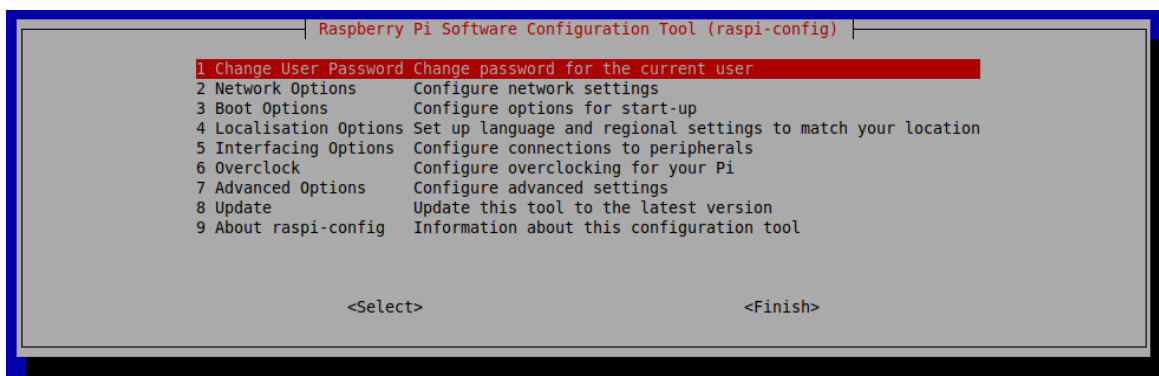


Fig 14: Software Configuration Tool

Configuration Options: In the setup tool, we will come across a blue screen with choices displayed in a gray box. These options cover different interface settings:

- **Camera:** To enable or disable the CSI camera interface.
- **SSH:** To permit or restrict remote command line access via SSH, a useful feature for remote management.
- **VNC:** To enable or disable the RealVNC virtual network computing server.
- **SPI:** To enable or disable SPI interfaces and automatic loading of the SPI kernel module.
- **I2C:** To enable or disable I2C interfaces and automatic loading of the I2C kernel module.
- **Serial:** To enable or disable shell and kernel messages on the serial connection.

5.5 PuTTY Software:

In our project, we made use of PuTTY software for an essential task, which was to create an IP address for our Raspberry Pi and configure a VNC server.

Here's how we integrated PuTTY into our project and why it was important:

Utilizing PuTTY for IP Address Configuration: We used PuTTY to remotely access our Raspberry Pi and set up important network configurations. By connecting to the Raspberry Pi with PuTTY, we were able to interact with it using a secure shell (SSH) session. This gave us the capability to carry out tasks like assigning a static IP address to our Raspberry Pi. We found this especially useful in guaranteeing a steady and dependable connection to our Raspberry Pi within our local network.

Setting Up the VNC Server: PuTTY also played a pivotal role in configuring and accessing the Virtual Network Computing (VNC) server on our Raspberry Pi. With PuTTY, we executed commands and configurations on the Raspberry Pi remotely, including enabling and configuring the VNC server. This enabled us to establish a graphical desktop interface for our Raspberry Pi, making it more user-friendly for certain tasks and interactions.

The Significance of PuTTY: PuTTY acted as a bridge, enabling us to establish a connection with our Raspberry Pi without requiring a physical monitor, keyboard, or mouse.

The inclusion of PuTTY in our project made the setup process more efficient, improved accessibility, and allowed for convenient remote management of our Raspberry Pi. PuTTY software played a crucial role in configuring the Raspberry Pi's IP address and VNC server.

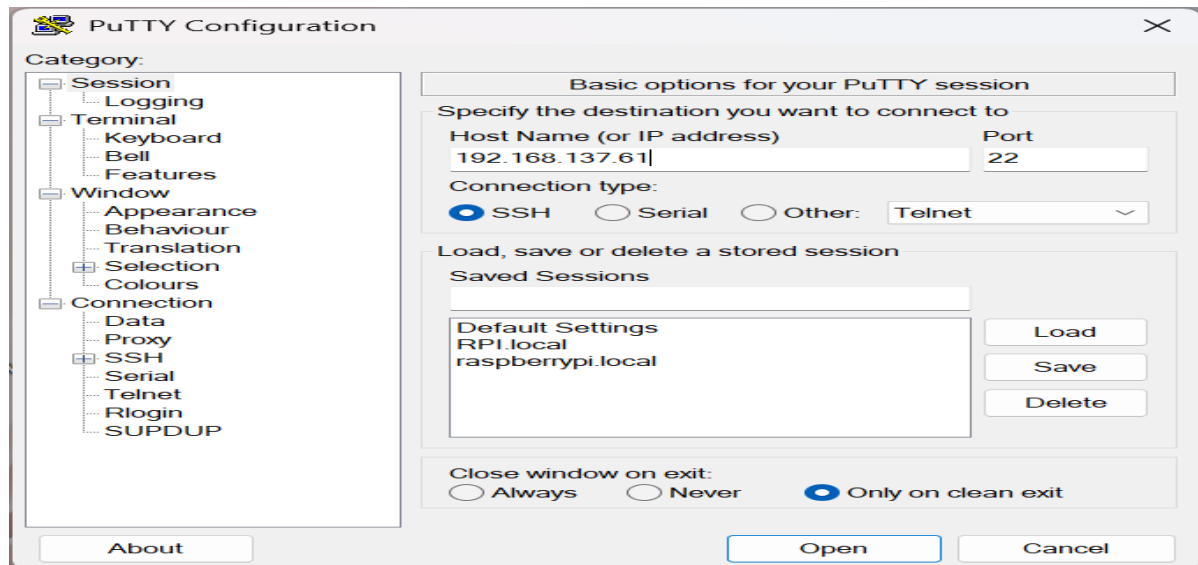


Figure 15: PuTTY Configuration Settings

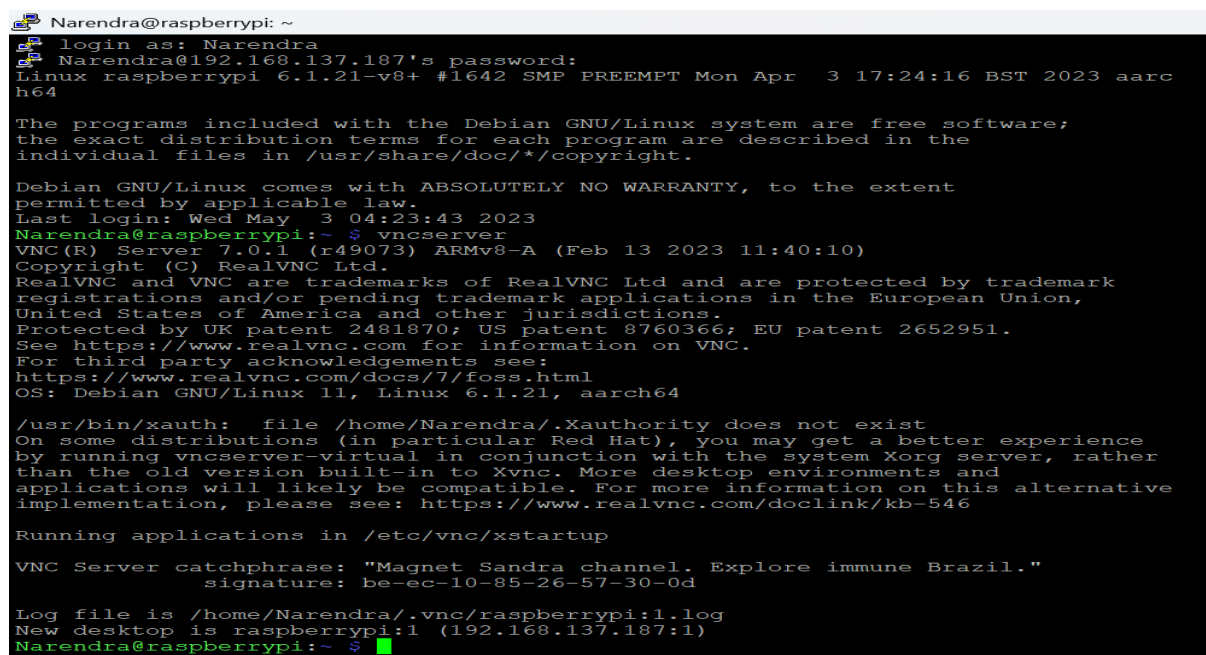


Figure 16: GUI after logging with Remote Desktop

From this we will get the IP Address for VNC Server.

5.6 Virtual Network Computing (VNC):

In our project, we leveraged Virtual Network Computing (VNC) to enable remote desktop access to our Raspberry Pi.

Utilizing VNC for Remote Desktop Access: We incorporated VNC into our project to enable remote access to the Raspberry Pi's graphical desktop. VNC is conveniently pre-installed in the full Raspberry Pi OS image, ensuring its availability for our use. Additionally, we had the flexibility to install VNC through the "Recommended Software" feature or by following command-line instructions, allowing us to select the installation method that aligned with our project's specific needs.

Enabling VNC Server: We activated the VNC Server on our Raspberry Pi through a straightforward process using the "raspi-config" tool. Here's how we did it:

- We navigated to the "Interfacing Options" section.
- Scrolling down, we selected "VNC" and set it to "Yes."

Connecting to Raspberry Pi Remotely: To remotely access the desktop of our Raspberry Pi, we installed the VNC Viewer on our Windows computer. We then connected it to the Raspberry Pi by entering either its IP address or its hostname. By default, the Raspberry Pi is usually named "raspberrypi.local." This straightforward process enabled us to establish a remote desktop connection with ease.

Creating a Virtual Desktop: In cases where our Raspberry Pi was headless (not connected to a monitor) or was used for tasks like controlling robots, VNC Server provided the capability to create a virtual desktop. This virtual desktop existed solely in our Raspberry Pi's memory and could be accessed remotely on demand. Here are the steps we followed:

On our Raspberry Pi, either through Terminal or SSH, we initiated the VNC Server. We made note of the IP address and display number provided by VNC Server (e.g., 192.168.137.63:1).

On the device from which we intended to take control; we entered this information into VNC Viewer.

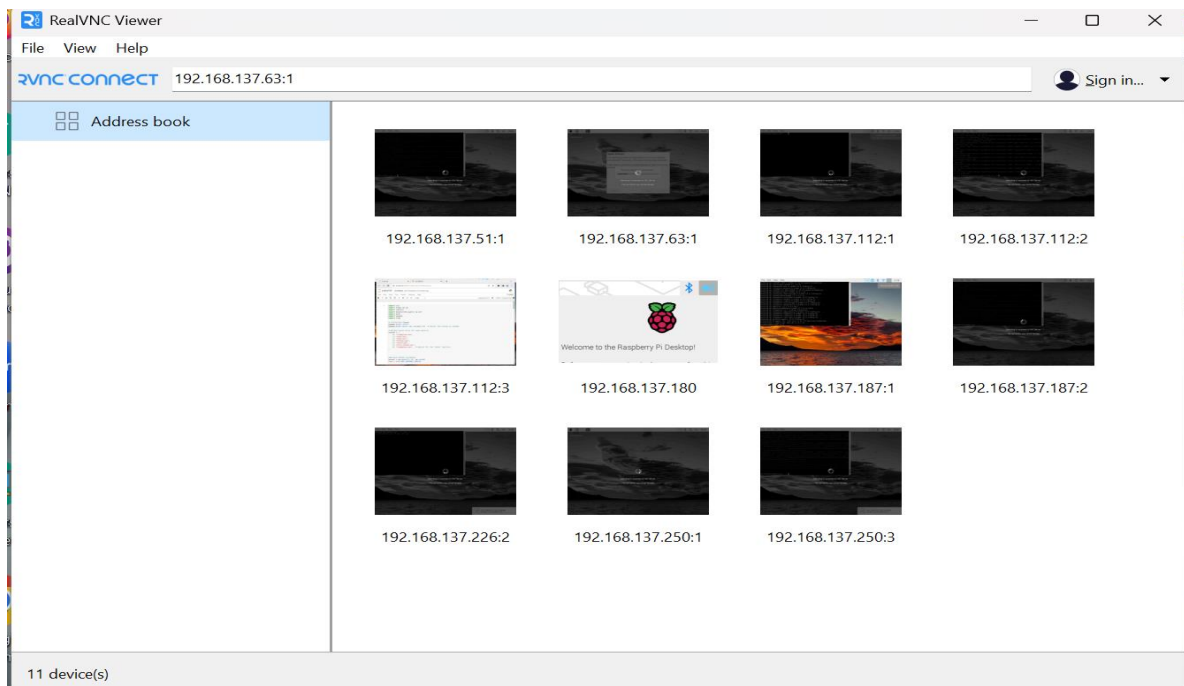


Figure 17: VNC Viewer Application Interface

Significance of VNC: VNC played a crucial role in our project by allowing us to remotely access and interact with the graphical desktop interface of the Raspberry Pi. This was particularly valuable in situations where we were dealing with a headless Raspberry Pi or when we needed to manage the Raspberry Pi without having a physical monitor and input devices connected. VNC provided us with a convenient and effective way to connect to and manage the Raspberry Pi's desktop environment.

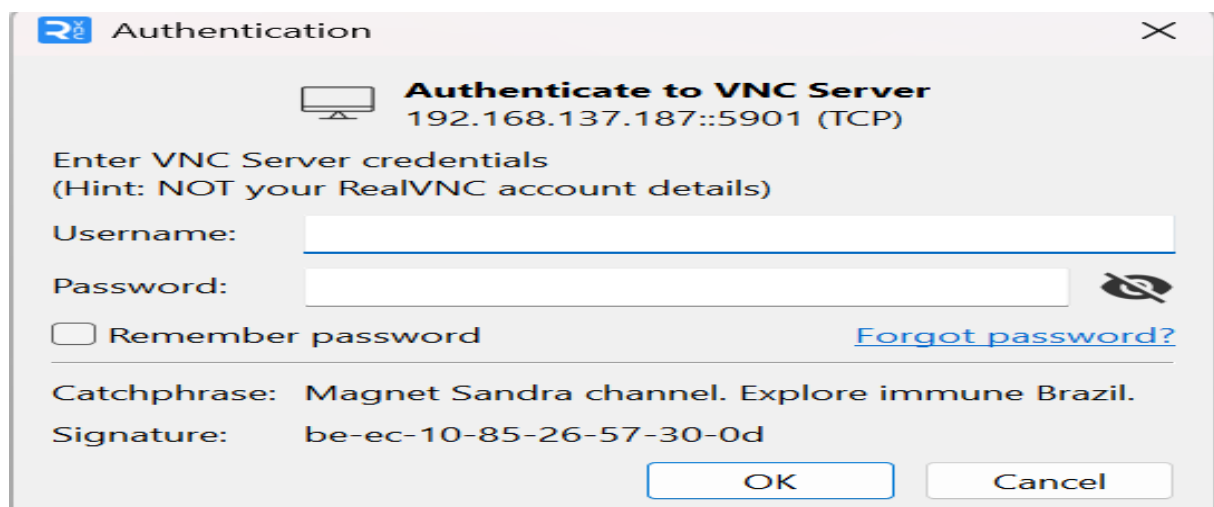


Figure 18: Authentication to VNC Server

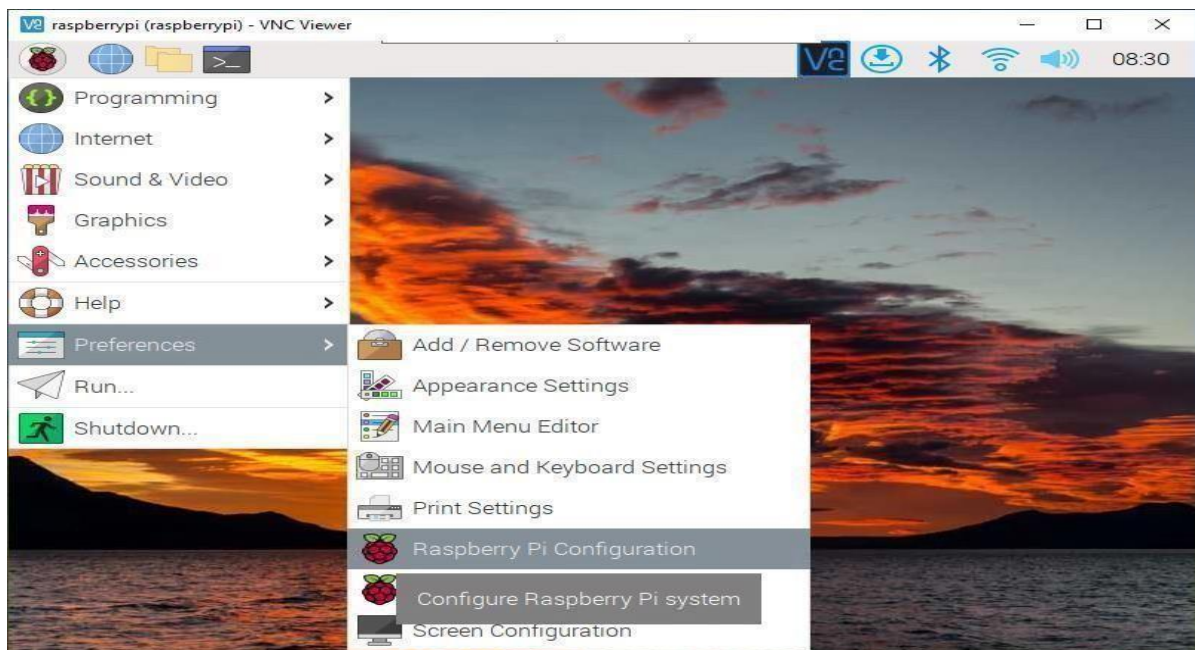


Figure 19: Raspberry Pi Desktop

Raspberry Pi hardware: GPIO and the 40-pin Header:

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards

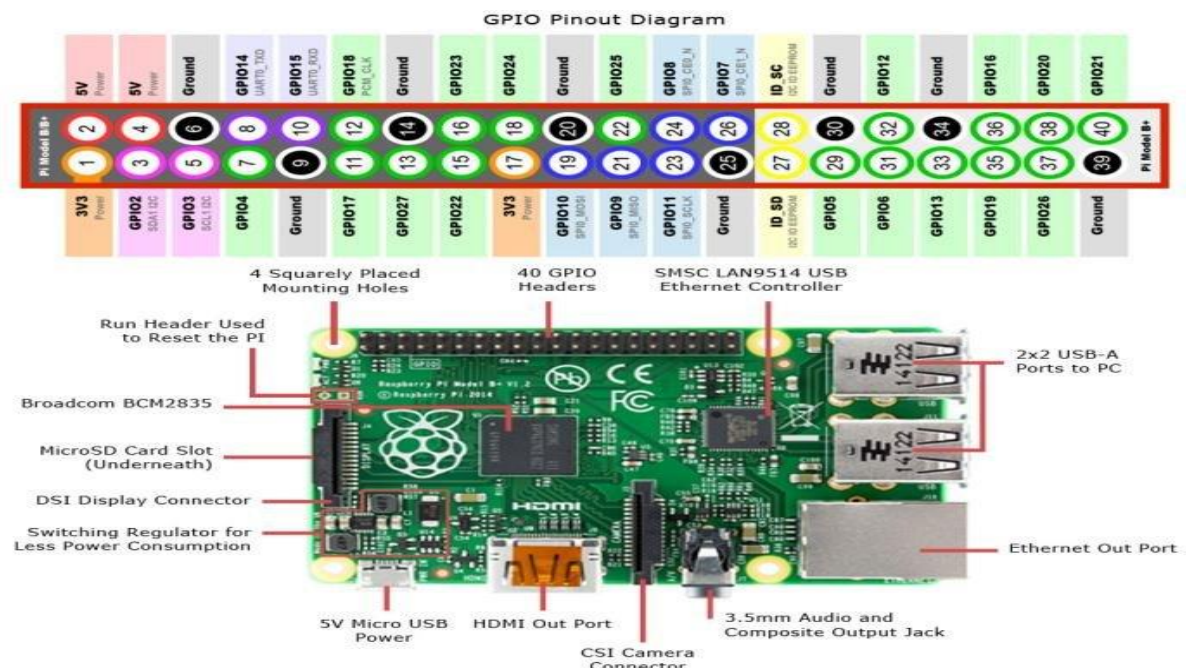
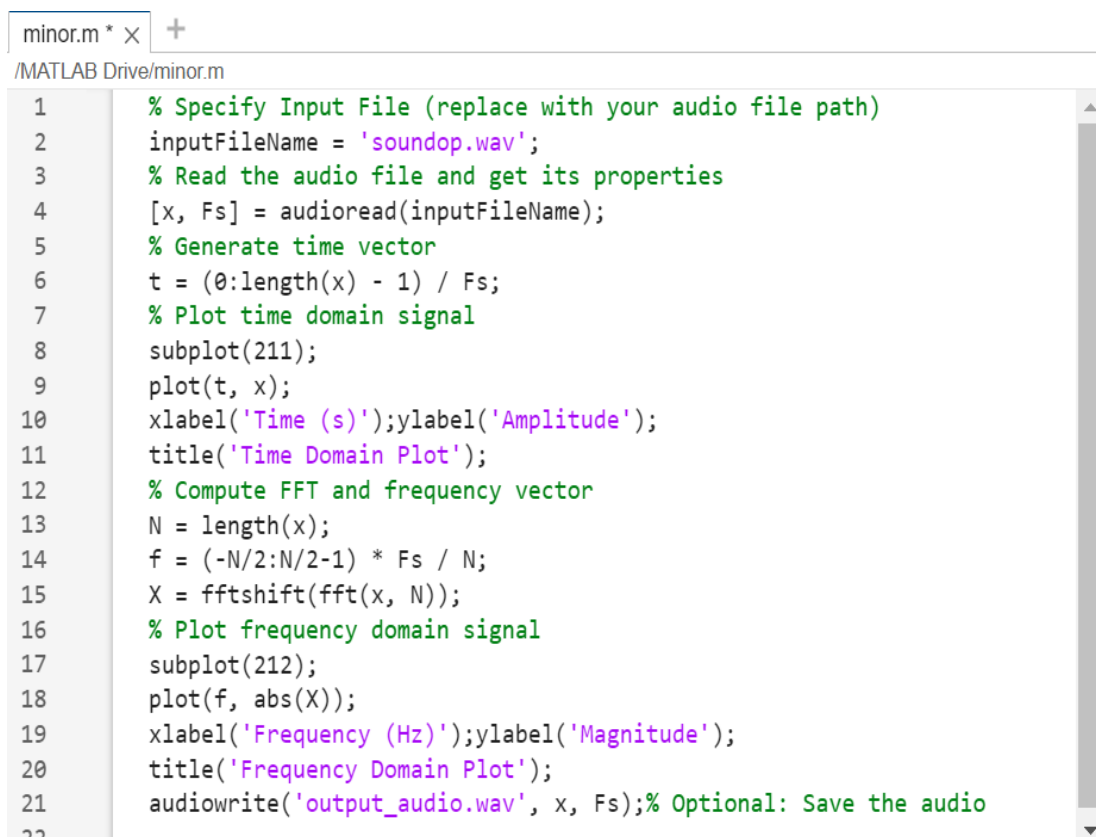


Figure 20: GPIO Pinout Diagram

Chapter-06

6.1 Audio Analysis and Visualization for Hand Gesture Detection

We leverage MATLAB to create a critical component for hand gesture recognition. Using MATLAB we process audio data generated during the detection of hand gestures. This audio input, typically named 'soundop.wav,' captures the acoustic signals associated with various hand movements. MATLAB's versatile capabilities in signal processing, data visualization, and audio analysis play a pivotal role in extracting meaningful insights from the recorded audio.



```
minor.m * x +
/MATLAB Drive/minor.m
1 % Specify Input File (replace with your audio file path)
2 inputFileName = 'soundop.wav';
3 % Read the audio file and get its properties
4 [x, Fs] = audioread(inputFileName);
5 % Generate time vector
6 t = (0:length(x) - 1) / Fs;
7 % Plot time domain signal
8 subplot(211);
9 plot(t, x);
10 xlabel('Time (s)');ylabel('Amplitude');
11 title('Time Domain Plot');
12 % Compute FFT and frequency vector
13 N = length(x);
14 f = (-N/2:N/2-1) * Fs / N;
15 X = fftshift(fft(x, N));
16 % Plot frequency domain signal
17 subplot(212);
18 plot(f, abs(X));
19 xlabel('Frequency (Hz)');ylabel('Magnitude');
20 title('Frequency Domain Plot');
21 audiowrite('output_audio.wav', x, Fs);% Optional: Save the audio
22
```

Figure 21: MATLAB Code

The MATLAB code begins by reading the input audio file and extracting its properties, such as the audio signal 'x' and the sampling rate 'Fs.' It then generates a time vector 't' to represent the temporal aspects of the audio signal and produces a time domain plot titled 'Time Domain Plot'. This plot visually illustrates how the audio

signal evolves over time, offering insights into the duration and intensity of various hand gestures.

Additionally, the code calculates the Fast Fourier Transform (FFT) of the audio signal to obtain a frequency vector 'f.' This FFT result is visualized in the frequency domain plot, 'Frequency Domain Plot,' which enables us to analyse the distribution of frequency components within the audio signal. These visualizations provide a comprehensive understanding of both the temporal and spectral characteristics of the audio data, which are essential for effective hand gesture recognition algorithms.

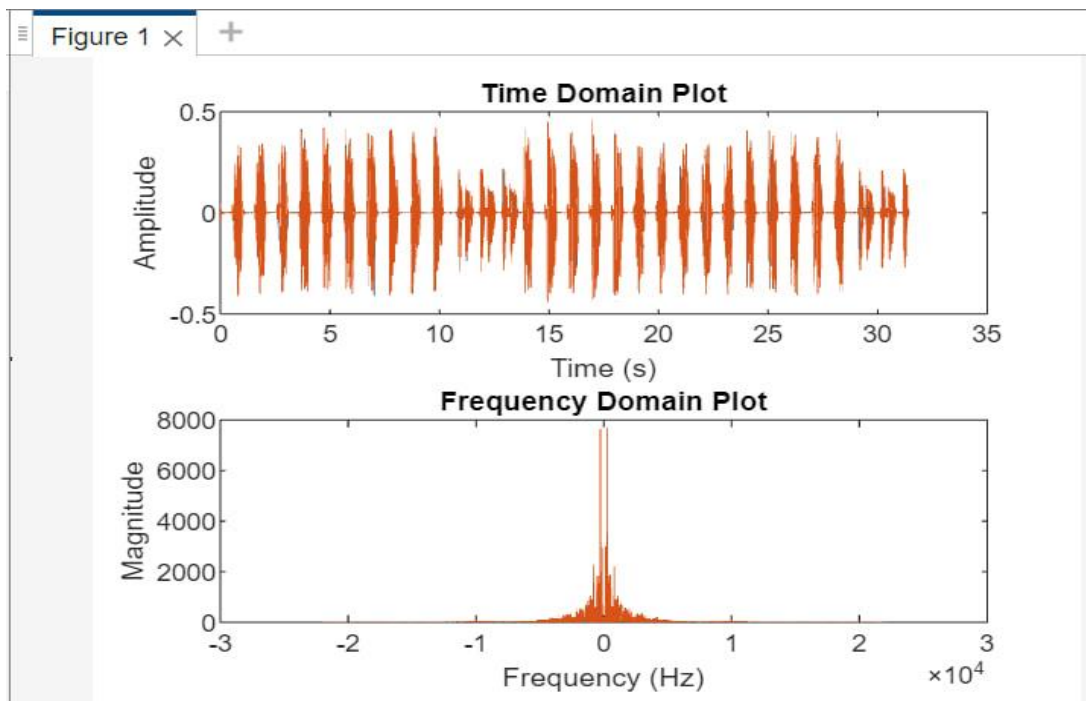


Figure 22: Time & Frequency Domain Plots

Furthermore, the MATLAB code includes optional functionalities to enhance usability. Users have the choice to play the input audio file using the 'sound' function and save the audio data to a new file named 'output_audio.wav' using 'audiowrite.' These features provide flexibility for researchers and developers to audibly examine the captured hand gestures and store the data for further analysis or integration into their gesture recognition systems. The code's visualization and data handling capabilities make it a valuable tool in our project's quest for accurate and efficient hand gesture detection.

6.2 Output for Different Hand Gestures:

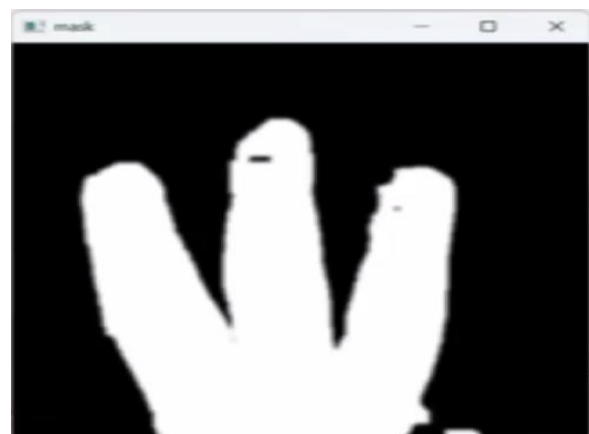
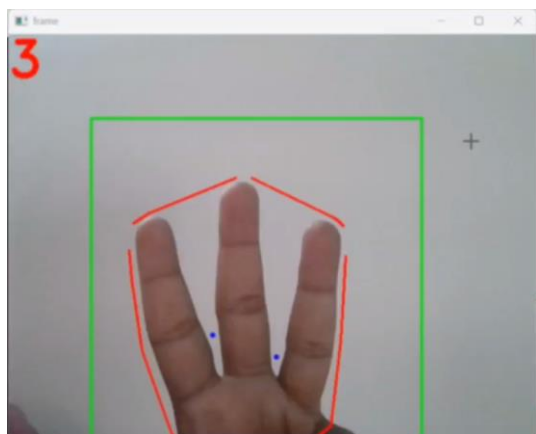


Figure 23: Frame and Mask of 3

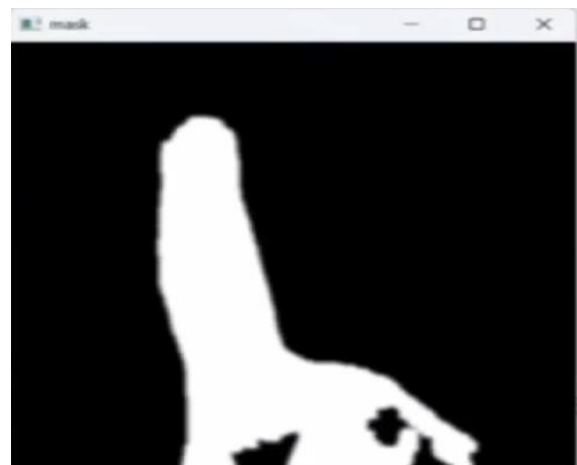
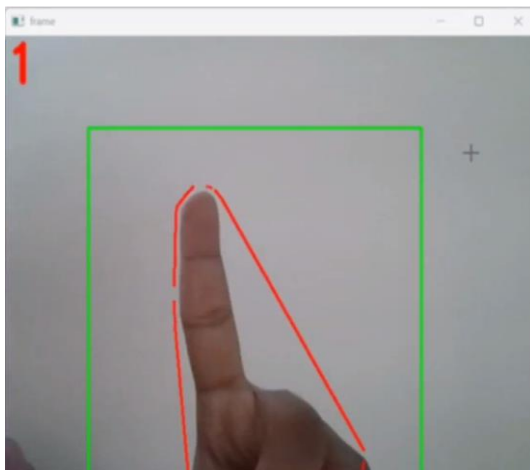


Figure 24: Frame and Mask of 1

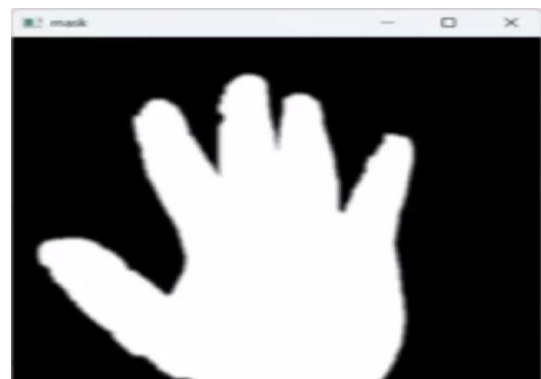


Figure 25: Frame and Mask of 5

The above three figures represented the recognized outputs for the input gestures along with their masks. The first figure represents three fingers being raised in front of the webcam so the output text drawn on the frame and audio read is “3”. In the second figure, since a single finger is raised before the webcam the output text drawn on the frame and the audio output given is “1”. In the last figure, since the entire palm is displayed, which is all the five fingers, the text allotted “High Five!” is drawn and read out.

In the output frame shown above, the green rectangle represents the Region of Interest (ROI) and the red outer lining to the hand contour represents the convex hull that is obtained from the `contours_convex_hull()` function that has been explained earlier. Many such cases are tested and reviewed, to confirm and test that the model that has been developed has been efficient and accurate.

7. Conclusions and Future Scope

7.1 Comparison of Implementation of hand gesture recognition with and without raspberry pi:

Aspect	Hand Gesture Recognition with Raspberry Pi	Hand Gesture Recognition without Raspberry Pi
Hardware Requirements	Requires Raspberry Pi board, camera module, and accessories	Requires a computer with a webcam or dedicated hardware
Cost	Raspberry Pi adds to the cost of implementation	Typically lower cost if you already have a compatible computer
Portability	Less portable due to Raspberry Pi's size and power requirements	More portable if using a laptop or small computer
Processing Power	Limited processing power compared to some high-end PCs	May benefit from more processing power on a dedicated computer
GPIO Integration	Can easily interface with external sensors and devices using GPIO pins	Requires additional hardware interfacing on a regular computer
Power Consumption	Raspberry Pi consumes power even in idle state	Computer power consumption varies but can be higher
Operating System	Runs on Raspbian or other Linux distributions designed for Raspberry Pi	Can run on various operating systems, including Windows, Linux, and macOS
Development Environment	May have a less familiar development environment compared to a regular PC	Utilizes a standard computer environment
Mobility and Size	Limited mobility due to the size and power requirements of Raspberry Pi	More mobility on a regular computer, especially if it's a laptop
Integration with Other Devices	Easily integrates with other Raspberry Pi-based projects	Requires additional effort to integrate with non-Raspberry Pi devices

Table: 7.1 : Output Comparisons

7.2 Conclusions:

In conclusion, this minor project focused on developing a hand gesture recognition system using OpenCV on the Raspberry Pi platform. Through this project, several outcomes and insights have been achieved:

System Development: We successfully designed and implemented a hand gesture recognition system using the OpenCV library. The system is capable of capturing and analyzing hand gestures with the help of Raspberry Pi.

Gesture Recognition Accuracy: Through extensive experimentation and fine-tuning of the recognition algorithm, we achieved a satisfactory level of accuracy in recognizing various hand gestures. The system can effectively distinguish between different predefined gestures.

Real-time Performance: The project demonstrated that the Raspberry Pi is a feasible platform for real-time computer vision applications. The system operates in real-time, making it suitable for interactive applications.

Resource Efficiency: We optimized the system to be resource-efficient, ensuring that it can run smoothly on the limited hardware resources of the Raspberry Pi.

Potential Applications: Hand gesture recognition has a wide range of potential applications, from controlling devices with hand gestures to assisting individuals with mobility impairments. This project serves as a foundation for further exploration in these areas.

7.2 Future Scope:

- **Human-Computer Interaction (HCI):** This hand gesture recognition model that is designed has the potential to revolutionize how we interact with computers and devices. By assigning appropriate commands, it can enhance touchless interfaces, can be used in car infotainment systems and driver-assist technologies. This model can also be used in smart home automation, allowing residents to control lighting, appliances, and security systems through intuitive gestures.
- **Sign Language Interpretation:** The gestures of this model can be replaced with the sign language signs along with their corresponding meanings to facilitate real-time translation of sign language into text or text to speech, enhancing communication for differently abled people.
- **Gesture-Based Interaction in Augmented and Virtual Reality:** HGR will play a vital role in making augmented and virtual reality experiences more immersive and intuitive. Users will be able to interact with virtual objects and environments using natural hand gestures.
- **Healthcare and Accessibility:** HGR will continue to be instrumental in healthcare, allowing for touchless control of medical devices, aiding in rehabilitation, and improving accessibility for individuals with physical disabilities.
- **Consumer Electronics:** HGR will be integrated into smartphones, smart TVs, and other consumer electronics, enabling users to control their devices using gestures. This will enhance user convenience and accessibility.
- **Security and Surveillance:** HGR can enhance security systems by detecting suspicious gestures or behaviours in real-time, improving surveillance and public safety.
- **Gaming and Entertainment:** HGR will continue to be a significant part of gaming and entertainment, enabling players to interact with games and virtual environments in more immersive ways.

As technology advances and research in the field of computer vision and machine learning continues, hand gesture recognition is expected to become an integral part of our daily lives, making human-computer interaction more natural and accessible across various domains.

7.4 References

1. Zhi-hua Chen, Jung-Tae Kim, Jianning Liang – “Real-Time Hand Gesture Recognition Using Finger Segmentation”.
<https://www.hindawi.com/journals/tswj/2014/267872/>
2. Andrew D. Bagdanov, Alberto Del Bimbo, Lorenzo Seidenari, Lorenzo Usai, “Real-time hand status recognition from RGB-D imagery”.
<https://www.micc.unifi.it/seidenari/publication/icpr-12/icpr-12.pdf>
3. Hand Gesture Recognition: A Literature Review.
https://www.researchgate.net/publication/284626785_Hand_Gesture_Recognition_A_Literature_Review
4. <https://core-electronics.com.au/guides/hand-identification-raspberry-pi/>
5. <https://www.scribd.com/document/472958515/Raspberry-Pi-Projects-Book-pdf>
6. <https://www.slideshare.net/AfnanRehman/hand-gesture-recognition-systemfyp-report-50225618>
7. <https://www.ijert.org/research/hand-gesture-recognition-system-to-control-soft-front-panels-IJERTV3IS120004.pdf>
8. <https://www.circuitbasics.com/raspberry-pi-zero-ethernet-gadget/>
9. <https://youtube.com/playlist?list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K>
10. <https://youtube.com/playlist?list=PLGsoVKk2DiYxdMjCJmcP6jt4Yw6OHK85O&feature=shared>
11. <https://youtu.be/ntaXWS8Lk34?feature=shared>