A
Mini Project Report
On

"HAND GESTURE RECOGNITION USING OPENCV AND DEEP LEARNING"

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING

(ARTIFICIAL INTELLIGENCE &

MACHINE LEARNING)

By

A.DHEERAJ REDDY	**-20R21A6602**
B.ROOPESH	**-20R21A6605**
G.ABHINAV GOUD	**-20R21A6618**
K.LAXMAN VIKAS	**-20R21A6622**
K.THARUNI DIXITHA	**-20R21A6623**

UNDER THE GUIDANCE OF

Mr. P.SRINIVASA REDDY

Department of Computer Science & Engineering-Artificial Intelligence & Machine Learning

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

Affiliated to JNTUH,Approved by AICTE

Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

2022

Department of Computer Science &Engineering-
Artificial Intelligence & Machine Learning

CERTIFICATE

This is to certify that the project entitled "**HAND GESTURE RECOGNITION USING OPENCV AND DEEP LEARNING**" has been submitted by A.DHEERAJ REDDY - (20R21A6602), B.ROOPESH - (20R21A6605) , G.ABHINAV GOUD - (20R21A6618) , K.LAXMAN VIKAS - (20R21A6622), K.THARUNI DIXITHA - (20R21A6623) in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence and Machine Learning) from MLR Institute of Technology, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Internal Guide **Head of the Department**

External Examiner

Department of Computer Science &Engineering-

Artificial Intelligence & Machine Learning

DECLARATION

We hereby declare that the project entitled "**HAND GESTURE RECOGNITION USING OPENCV AND DEEP LEARNING**"is the work done during the period from AUG 2022 to DEC 2022 and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of technology in Computer Science and Engineering (Artificial Intelligence and Machine Learning) from MLR Institute of Technology, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

A.DHEERAJ REDDY 20R21A6602
B.ROOPESH 20R21A6605
G.ABHINAV GOUD 20R21A6618
K.LAXMAN VIKAS 20R21A6622
K.THARUNI DIXITHA 20R21A6623

Department of Computer Science &Engineering-

Artificial Intelligence & Machine Learning

ACKNOWLEDGEMENT

There are many people who helped me directly and indirectly to complete my project successfully. We would like to take this opportunity to thank one and all.

First of all I would like to express my deep gratitude towards my internal guide **Mr.P. Srinivas Reddy Asst.Prof.Department** of CSM for his support in the completion of my dissertation. I wish to express my sincere thanks to **Mr. K. Sai Prasad HOD, Department of CSE** and also to principal **Dr. K. Srinivas Rao** for providing the facilities to complete the dissertation.

We would like to thank all our faculty, coordinators and friends for their help and constructive criticism during the project period. Finally, I am very much indebted to our parents for their moral support and encouragement to achieve goals.

A.DHEERAJ REDDY 20R21A6602
B.ROOPESH 20R21A6605
G.ABHINAV GOUD 20R21A6618
K.LAXMAN VIKAS 20R21A6622
K.THARUNI DIXITHA 20R21A6623

Department of Computer Science &Engineering-

Artificial Intelligence & Machine Learning

ABSTRACT

Hand gesture recognition is one of the system that can detect the gesture of hand in a real time video. The gesture of hand is classify within a certain area of interest. In this study, designing of the hand gesture recognition is one of the complicated job that involves two major problem. Firstly is the detection of hand. Another problem is to create the sign that is suitable to be used for one hand in a time. This project concentrates on how a system could detect, recognize and interpret the hand gesture recognition through computer vision with the challenging factors which variability in pose, orientation, location and scale. To perform well for developing this project, different types of gestures such as numbers and sign languages need to be created in this system. The image taken from the realtime video is analysed via Haar-cascaded Classifier to detect the gesture of hand before the image processing is done or in the other word to detect the appearance of hand in a frame. In this project, the detection of hand will be done using the theories of Region of Interest (ROI) via Python programming. The explanation of the results will be focused on the simulation part since the different for the hardware implementation is the source code to read the real-time input video. The developing of hand gesture recognition using Python and OpenCV can be implemented by applying the theories of hand segmentation and the hand detection system which use the Haar-cascade classifier.

LIST OF FIGURES & TABLES

CONTENTS

1. INTRODUCTION

1.1 OVERVIEW

In a day-to-day life, hand gesture recognition is one of the system that can detect the gesture of hand in a real time video. The gesture of hand is classify within a certain area of interest. Designing a system for hand gesture recognition is one of the goal of achieving the objectives of this project. The task of recognizing hand gestures is one of the main and important issues in computer vision. With the latest advances in information and media technology, human computer interaction (HCI) systems that involve hand processing tasks such as hand detection and hand gesture recognition. The first step in any hand processing system is to detect and locate the hand in the real-time video from the webcam. The detection of hand is challenging because of variation in pose, orientation, location and scale. Also, different intensity of light in the room adds to the variability. In the process of detection of hand. Hand gesture recognition generally involves multiple levels such as image acquisition, pre-processing, feature extraction and gesture recognition. Image acquisition involve capturing image in the video frame by frame using a webcam. The captured images go through the image pre-processing process which involves color filtering, smoothing and thresholding.

Feature extraction is a method that extracting features of the hand image such as hand contours while gesture recognition is a method to recognize hand gesture by extracting the features. In this study, designing of the hand gesture recognition is one of the complicated job that involves two major problem. Firstly is the detection of hand. User hand is detected by using webcam in real-time video. The problem would be the unstable brightness, noise, poor resolution and contrast. The detected hand in the video are recognized to identify the gestures. At this stage, the process involves are the segmentation and edge detection. According to Karishma , with various information of image like color, hand posture and shape based (shape of hand) in a real time would affect the recognition of gestures. Another problem is to create the sign that is suitable to be used for one hand in a time. The extraction of hand need to be followed to determine each number and sign used the extraction of hand involves the contour and convexity defects. Convexity defect gives an issue on how to calculate the depth of the defects. Some of the defects have far greater depth than others so to identify the depth would include some equations. The list of objectives that will need to be achieve for this project: to establish a complete system for detecting, recognizing and interpreting hand gesture recognition through computer vision using Python and OpenCV, and to create the numbers and sign languages of hand gesture shown in the system that will meets the name of the project.

This study comprises on how to implement a complete system that can detect, recognizing and interpreting the hand by using Python and OpenCV in any intensity of light, pose or orientation of hand. In order to accomplish this, a real-time gesture based system is developed. In the proposal stage, designing a system that could detect the hand through the contour of the hand. The contour of the hand refers to the curve for a two variables for the function along which that function has a contact value that is not changed. Besides, to detect the appearance of hand in a frame of the real-time video, I will be using the Haar-cascade Classifier to track the appearance of hand before the image-processing is done. The result of the appearance of hand that the system could detect will be validated for analysis. For hardware implementation section, a wired web camera is used thus the hand gesture recognition can be done and implemented.

For this project, the prototype itself has its own limitation in terms of hardware implementation. First of all, the video capturing device used for this project can only configure up to 640 by 480 pixel resolutions. This might be the disadvantage of getting the hardware to properly work for detecting the hand. To solve this, the web camera must be in a fixed position up to 30 inches (75 cm) to help capture the video more efficiently. Another limitation to this project is the variation in plane and pose. Variations can be classified as rotation, translation and scaling of camera pose . The direction of the web camera is important to obtain a good video without any shadowing effects and the intensity of light in a room is enough. According to Ray , by selecting the correct light intensity and direction, the shadows can be reduced to a minimum.

1.2 MOTIVATION

Biometric technologies make use of various physical and behavioral characteristics of human such as fingerprints, expression, face, hand gestures and movement. These features are then processed using sophisticated machines for detection and recognition and hence used for security purposes. Unlike common security measures such as passwords, security cards that can easily be lost, copied or stolen; these biometric features are unique to individuals and there is little possibility that these pictures can be replaced or altered. Among the biometric sector hand gesture recognition are gaining more and more attention because of their demand regarding security for law enforcement agency as well as in private sectors such as surveillance systems. In video conferencing system, there is a need to automatically control the camera in such a way that the current speaker always has the focus. One simple approach to this is to guide the camera based on sound or simple cues such as motion and skin color. Hand gestures are important to intelligent human and computer interaction to build fully automated systems that analyze information contained in images, fast and efficient hand gesture recognition algorithms are required.

1.3 PURPOSE

Hand gesture recognition is of great importance for human computer interaction (HCI) because of its extensive applications in virtual reality and sign language recognition etc. Human hand is very smaller with very complex articulations comparing with the entire human body and therefore errors can be easily affected. The algorithm serves an important role in Sign Language Detection.

Gesture recognition is technology that uses sensors to read and interpret hand movements as commands. In the automotive industry, this capability allows drivers and passengers to interact with the vehicle — usually to control the infotainment system without touching any buttons or screens.

1.4 SCOPE OF STUDY

This study deals with the problem of developing a vision-based static hand gesture recognition algorithm to recognize the following six static hand gestures: Open, Close, Cut, Paste, Maximize, and Minimize. These gestures are chosen because they are commonly used to communicate and can thus be used in various applications, such as a virtual mouse that can perform six tasks (Open, Close, Cut, Paste, Maximize, Minimize) for a given application. The proposed system consists mainly of three phases: the first phase (i.e., pre-processing), the next phase (i.e., feature extraction), and the final phase (i.e., classification). The first phase includes hand segmentation that aims to isolate hand gestures from the background and remove the noises using special filters. This phase includes also edge detection to find the final shape of the hand. The next phase, which constitutes the main part of this research, is devoted to the feature extraction problem where two feature extraction methods, namely, hand contour and complex moments are employed. These two extraction methods were applied in this study because they used different approaches to extract the features, namely, a boundary-based for hand contour and a region-based for complex moments. The feature extraction algorithms deal with problems associated with hand gesture recognition such as scaling, translation, and rotation. In the classification phase where neural networks are used to recognize the gesture image based on its extracted feature, we analyze some problems related to the recognition and convergence of the neural network algorithm. As a classification method, ANN has been widely employed especially for real-world applications because of its ability to work in parallel and in online training. Thus, an ANN has been a lively field of research, In addition, a comparison between the two feature extraction algorithms is carried out in terms of accuracy and processing time (computational cost). This comparison, using these two criteria, is important to identify the strengths and weaknesses of each feature extraction method and assess the potential application of each method. Figure 1 provides an overview of the method used to develop the hand gesture recognition system.

LITERATURE SURVEY

Hasan applied multivariate Gaussian distribution to recognize hand gestures using non- geometric features. The input hand image is segmented using two different methods; skin color based segmentation by applying HSV color model and clustering based thresholding techniques . Some operations are performed to capture the shape of the hand to extract hand feature; the modified Direction Analysis Algorithm are adopted to find a relationship between statistical parameters (variance and covariance) from the data, and used to compute object (hand) slope and trend by finding the direction of the hand gesture , As shown in Figure 1.

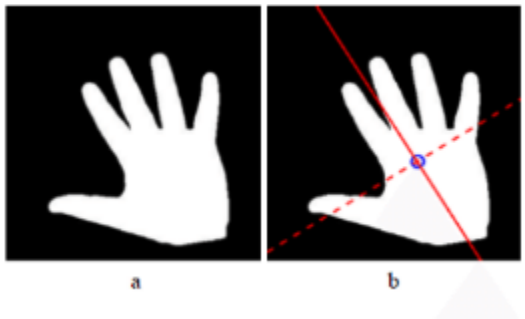

Fig 2.1

Then Gaussian distinction is applied on the segmented image, and it takes the direction of the hand as shown in figure 2.

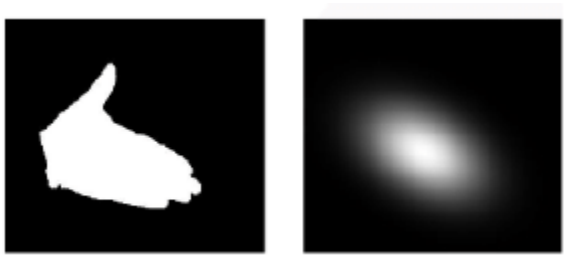

Fig 2.2

Form the resultant Gaussian function the image has been divided into circular regions in other words that regions are formed in a terrace shape so that to eliminate the rotation affect. The shape is divided into terraces with a 0.1 width for each terrace 9 terraces are resultant from the 0.1 width division which are; (1-0.9, 0.9-0.8, 0.8-0.7, 0.7-0.6, 0.6, 0.5, 0.5-0.4, 0.4-0.3, 0.3-0.2, 0.2-0.1), and one terrace for the terrace that has value smaller than 0.1 and the last one for the external area that extended out of the outer terrace . An explanation of this division is demonstrated in Figure 3.

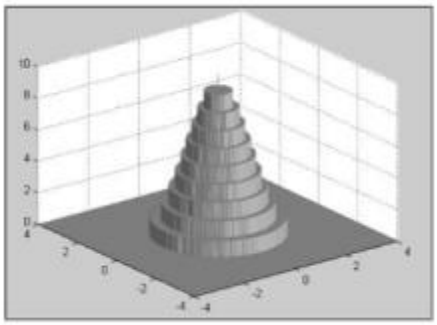

Fig 2.3

Each terrace is divided into 8 sectors which named as the feature areas, empirically discovered that number 8 is suitable for features divisions , To attain best capturing of the Gaussian to fit the segmented hand, re-estimation are performed on the shape to fit capturing the hand object, then the Gaussian shape are matched on the segmented hand to prepare the final hand shape for extracting the features, Figure 4 shown this process.

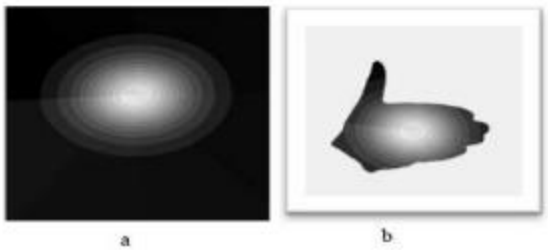

Fig 2.4

After capturing the hand shape, two types of features are extracted to form the feature vector; local feature, and global features. Local features using geometric central moments which provide two different moments μ00 , μ11 as shown by equation (1):

$$\mu_{pp} = \sum_{x}\sum_{y}(x-\mu_x)^p(y-\mu_y)^p\, f(x,y) \qquad (1)$$

$$\mu_{pp}^{(k)} = \sum_{y}\sum_{x}\left(x^{(k)}-\mu_x^{(k)}\right)^p\left(y^{(k)}-\mu_y^{(k)}\right)^p f(x^{(k)},y^{(k)}) \qquad (2)$$

$$\forall\, k \in \{1, 2, 3, \ldots, 88\} \;\&\; \forall\, p \in \{0, 1\}$$

Fig 2.5

Where μx and μy is the mean value for the input feature area , x and y are the coordinated, and for this, the input image is represented by 88*2 features, as explained in detail in equation (2). While the global features are two features the first and second moments that are the computed for the whole hand features area . These feature areas are computed by multiplying feature area intensity plus feature area's map location . In this case, any input image is represented with 178 features . The system carried out using 20 different gestures ,10 samples for each gesture, 5 samples for training and 5 for testing, with 100% recognition percentage and it decreased when the number of gestures are more than 14 gestures . In 6 gestures are recognized with 10 samples for each gesture. Euclidian distance used for theclassification of the feature .

Kulkarni recognize static posture of American Sign Language using neural networks algorithm. The input image are converted into HSV color model, resized into 80x64 and some image preprocessing operations are applied to segment the hand [31]from a uniform background, features are extracted using histogram technique and Hough algorithm. Feed forward Neural Networks with three layers are used for gesture classification. 8 samples are used for each 26 characters in sign language, for each gesture, 5 samples are used for training and 3samples for testing, the system achieved 92.78% recognition rate using MATLAB language.

Hasan applied scaled normalization for gesture recognition based on brightness factor matching. The input image with is segmented using thresholding technique where the background is black. Any segmented image is normalized (trimmed), and the center mass of the image are determined, so that the coordinates are shifted to match the centroid of the hand object at the origin of the X and Y axis . Since this method depends on the center mass of the object, thegenerated images have different sizes see figure 9, for this reason a scaled

Normalization operation are applied to overcome this problem which maintain image dimensions and the time as well , where each block of the four blocks are scaling with a factor that is different from other block's factors. Two methods are used for extraction the features; firstly by using the edge mages,and secondly by using normalized features where only the brightness values of pixels are calculated and other black pixels are neglected to reduce the length of the feature vector . The database consists of 6 different gestures, 10 samples per gesture are used, 5 samples for training and 5 samples for testing. The recognition rate for the normalized feature problem achieved better performance than the normal feature method, 95% recognition rate for the former method and 84% for the latter one .

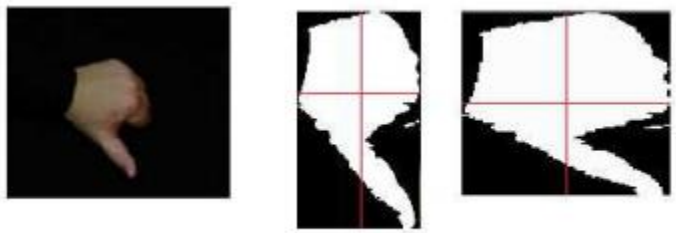

Fig 2.6

Camera used for acquire the input image, filter for skin color detection has been used followed by clustering process to find the boundary for each group in the clustered image using ordinary contour-racking algorithm. The image was divided into grids and the boundaries have been normalized. The boundary was represented as chord's size chain which has been used as histograms, by dividing the image into number of regions N in a radial form, according to specific angle. For classification process Neural Networks MLP and Dynamic Programming DP matching were used. Many experiments have implemented on different features format in addition to use different chord's size histogram, chord's size FFT. 26 static postures from American Sign Language used in the experiments. Homogeneous background was applied in the work. Stergio poulou suggested a new Self-Growing and Self-Organized Neural Gas (SGONG) network for hand gesture recognition. For hand region detection a color segmentation technique based on skin color filter in the YCbCr color space was used, an approximation of hand shape morphology has been detected using (SGONG) network; Three features were extracted using finger identification process which determines the number of the raised fingers and characteristics of hand shape, and Gaussian distribution model used for recognition.

2.1 EXISTING SYSTEM

In recent decades, due to computer software and hardware technologies of continuous innovation and breakthrough, social life and information technology have a very close relationship in the twenty-first century. In the future, especially the interfaces of consumer electronics products (e.g. smart phones, games, and infota1inment systems) will have more and more functions and be complex. How to develop a convenient human-machine interface (Human Machine Interaction/Interface, HMI) for each consumer electronics product has become an important issue. The traditional electronic input devices, such as a mouse, keyboard, and joystick are still the most common interaction way. However, it does not mean that these devices are the most convenient and natural input devices for most users. Since ancient times, gestures are a major way of communication and interaction between people. People can easily express an idea through gestures before the invention of language.

Nowadays, gestures still are naturally used by many people and especially are the most major nature interaction way for deaf people. In recent years, the gesture control technique has become a new developmental trend for many human-based electronics products, such as computers, televisions, and games. This technique let people can control these products more naturally, and intuitively in the case of the existing system. The objective of this paper is to develop a real-time hand gesture recognition system based on adaptive color HSV model and motion history image (MHI). By adaptive skin color model, the effects from lighting, environment, and camera can be greatly reduced, and the robustness of hand gesture recognition could be greatly improved.

PROPOSED SYSTEM

The proposed System is a vision-based approach in which the project will transfer the detected motion into its textual meaning, display the translated text to the user, and finally convert the text to speech. The system will identify the appropriate hand movement and search in its database for pre-defined gestures that match the action. The user is shown the class label when it has been matched. The system is built on the concept of vision. Because all of the signals are made using the hands, there is no need for any artificial gadgets for interaction.

PROPOSED SYSTEM ARCHITECTURE

The architecture system shows the architecture which is Visualized in figure below component of the system, how it Functions, and how it flows, among other things. Pre processing processes are used to increase the quality of Captured images using a web camera after that, the object and Background are removed from the images, which are then Converted to byte. Featured extraction and rearrangement help in the matching of images stored in a database, resulting in the required output in the form of text, which is then converted to Audio.

MODEL DESIGN

CNN Model

1.1st Convolution Layer: The input picture has resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights(3x3 pixels each).This will result in a
126x126 pixel image, one for each Filter-weights.

2.1st Pooling Layer: The pictures are down sampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is
down sampled to 63x63 pixels

3.2nd Convolution Layer: Now ,these 63x63 from the output of the first pooling layer is served as an input to the second convolutional layer. It is processed in the second

convolutional layer using 32 filter weights(3x3 pixels each).This will result in a 60x60 pixel image.

4.2nd Poolind Layer: The resulting images are down sampled again using max pool of 2x2 and is reduced to 30x30 resolution images.

5.1st Densely Connected Layer: Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is
reshaped to an array of 30x30x32=28800 values. The input to this layer is an array of 28800 values. The output of these layer is fed to the 2nd Densely connected layer.
we are using a dropout layer of values 0.5 to avoid overfitting.

6.2nd Densely Connected Layer: Now the output from the 1st Densely Connected Layer are used as an input to a fully connected layer with 96 neurons.

7.Final Layer: The output of the 2nd Densely Connected Layer server as an input for the final layer which will have the number of neurons as the number of classes we are
Classifying (alphabets + blank symbol).

3.2 SYSTEM REQUIREMENTS

Here are the requirements for developing and deploying the application.

3.2.1 HARDWARE REQUIREMENTS

Processor	:	i3 or above
Processor Speed	:	1.6 Ghz
Video	:	WebCam
RAM	:	8 GB

3.2.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows 10/ Linux O
Front End	:	Pycharm
Back End	:	Pycharm

3.2.3 IMPLEMENTATION TECHNOLOGIES

- Tensorflow
- Open CV
- Keras

3.2.4 FUNCTIONAL REQUIREMENTS

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being an Apache 2 licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image

database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

PYCHARM

What is PyCharm?

PyCharm is a hybrid platform developed by JetBrains as an IDE for Python. It is commonly used for Python application development. Some of the unicorn organizations such as Twitter, Facebook, Amazon, and Pinterest use PyCharm as their Python IDE!

It supports two versions: v2.x and v3.x.

We can run PyCharm on Windows, Linux, or Mac OS. Additionally, it contains modules and packages that help programmers develop software using Python in less time and with minimal effort. Further, it can also be customized according to the requirements of developers.

Features of PyCharm:

1. Intelligent Code Editor:

- It helps us write high-quality codes!
- It consists of color schemes for keywords, classes, and functions. This helps increase the readability and understanding of the code.
- It helps identify errors easily.
- It provides the autocomplete feature and instructions for the completion of the code.

2. Code Navigation:

- It helps developers in editing and enhancing the code with less effort and time.
- With code navigation, a developer can easily navigate to a function, class, or file.
- A programmer can locate an element, a symbol, or a variable in the source code within no time.
- Using the lens mode, further, a developer can thoroughly inspect and debug the entire source code.

3. Refactoring

- It has the advantage of making efficient and quick changes to both local and global variables.
- Refactoring in PyCharm enables developers to improve the internal structure without changing the external performance of the code.
- It also helps split up more extended classes and functions with the help of the extract method.

4. Assistance for Many Other Web Technologies:

- It helps developers create web applications in Python.
- It supports popular web technologies such as HTML, CSS, and JavaScript.
- Developers have the choice of live editing with this IDE. At the same time, they can preview the created/updated web page.
- The developers can follow the changes directly on a web browser.
- PyCharm also supports AnglularJS and NodeJS for developing web applications.

5. Support for Popular Python Web Frameworks

- PyCharm supports web frameworks such as Django.

- It provides the autocomplete feature and suggestions for the parameters of Django.
- It helps in debugging the codes of Django.
- It also assists web2py and Pyramid, the other popular web frameworks.

6. Assistance for Python Scientific Libraries

- PyCharm supports Python's scientific libraries such as Matplotlib, NumPy, and Anaconda.
- These scientific libraries help in building projects of Data Science and Machine Learning.
- It consists of interactive graphs that help developers understand data.
- It is capable of integrating with various tools such as IPython, Django, and Pytest. This integration helps innovate unique solutions.

JUPYTER NOTEBOOK

Project Jupyter is a suite of software products used in interactive computing. IPython was originally developed by Fernando Perez in 2001 as an enhanced Python interpreter. A web based interface to IPython terminal in the form of IPython notebook was introduced in 2011. In 2014, Project Jupyter started as a spin-off project from IPython.

Packages under Jupyter project include −

Jupyter notebook − A web based interface to programming environments of Python, Julia, R and many others

QtConsole − Qt based terminal for Jupyter kernels similar to IPython

nbviewer − Facility to share Jupyter notebooks

JupyterLab − Modern web based integrated interface for all products.

Standard distribution of Python comes with a **REPL (Read-Evaluate-Print Loop)** environment in the form of Python shell with >>> prompt. IPython (stands for Interactive Python) is an enhanced interactive environment for Python with many functionalities compared to the standard Python shell.

Features of IPython

IPython offers more features compared to the standard Python. They are as follows − Offers a powerful interactive Python shell.

Acts as a main kernel for Jupyter notebook and other front end tools of Project Jupyter.

Possesses object introspection ability. Introspection is the ability to check properties of an object during runtime.

- Syntax highlighting.
- Stores the history of interactions.
- Tab completion of keywords, variables and function names.
- Magic command system useful for controlling Python environment and performing OS tasks.
- Ability to be embedded in other Python programs.

SYSTEM DESIGN

4.1 System Architecture

The proposed architecture consists of THREE stages:

(1) Hand detection

(2) Hand gesture recognition

(3) Finger detection

Now we introduce our part-based hand gesture recognition system. Which illustrates the framework, which consists of two major modules: hand detection and hand gesture recognition

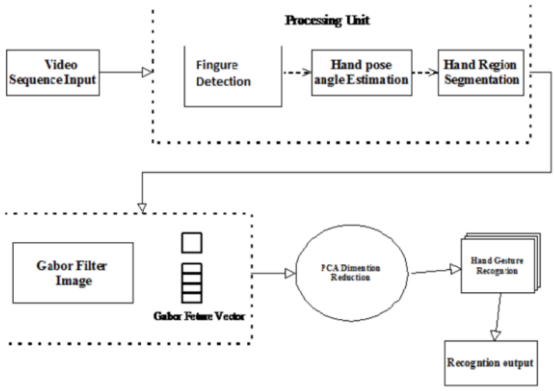

Fig 4.1.1 process flow

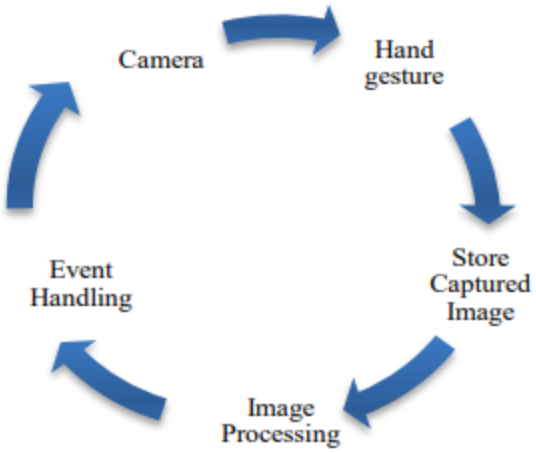

Fig 4.1.2 Working System

4.2 Algorithm:

Point Pattern Matching

For finding the validity ratio Point Pattern Matching algorithm is used.

C - Denotes the center points

D - Denotes the distance mask

T - Denotes the No. of test image to match

M - Denotes the No. of Matched Points 1, 2, 3 are the key points.

The procedure to find the Validity ratio of One Database Image versus Test Input Image.

The working of point pattern matching algorithm is as follows:

1. Take a test image
2. Pre process the test image.
3. Initialize the distRatio = 0.65 and threshold= 0.035
4. Run the SHIFT match algorithm
5. Key point matching starts its execution by running the threshold. It gets the key point matched between test and all 36 trained images. We get the validity ratio.
6. Check that we got more than one result or not.
7. If we get more than1 result then increment the SHIFT distRatio by 0.05 and threshold by 0.005 and repeat the steps from 4 to 7.
8. If we get only one result then display the result.

IMPLEMENTATION

DATA SET GENERATION

We researched for pre-made datasets for the project but couldn't find any that met our needs in the form of raw photos. We were only able to discover datasets in the form of RGB values. As a result, we decided to develop our own data set. The following are the steps used to construct our data set. To create our dataset, we utilized the Open Computer Vision (OpenCV) package. To begin, we took around 400 photographs of each ASI. Symbol for training reasons and approximately 150 images for testing purposes. We begin by capturing each frame displayed by our machine's camera. We define a Region of Interest (ROI) in each frame, which is represented by a blue bordered square

FEATURE EXTRACTION

The representation of an image a 3D matrix with dimensional equal to the image's height and breadth, and depth equal to the value of each pixel (1 in Grayscale and 3 in RGB). These pixel values are also used by CNN to extract valuable features.

We acquire the required portion of the hand, and convert it into black and white and detect the values represent the features using thresholding and Gaussian Blur.

CLASSIFICATION METHOD

CNN

To detect objects, identify faces, and so on, CNNs use image recognition and classification . They're built up of neurons with weight matrix that can be learned. Each neuron gets several inputs and then computers a weighted toral, which it then runs through an activation function before responding with an output. CNNs are typically used to categorize pictures and cluster them based on similarities, and subsequently recognize objects. Many CNN-based algorithms can recognize people, street signs, animals, and other objects.

Our approach for this project was to predict the user's final symbol using two levels of algorithms.

1.ALGORITHM LAYER 1:

. After feature extraction, apply a guassian blur filter and a threshold to the frame captured using OpenCV to obtain the processed image.

. The processed image is sent into the CNN model for predictions, and if a letter is recognized for more than 15 frames, the letter is printed and considered for word formation.

. The blank symbol is used to represent the space between the words

2.ALGORITHM LAYER 2:

. We identify several groups of symbol that have comparable results when identified

. We then use classifiers designed specifically for those sets to differentiate between them.

. AUTOCORRECT FEATURE

We introduce the Hunspell propose Python package, which suggests appropriate replacements for each (incorrect) input text, and we present a list of words that match the currebt term, fromwhich the user may choose one to append to the current text. This helps in the prediction of complex words and helps to reduce spelling errors.

TRAINING

The user will start training the Model by showing the signs and then passes the corresponding key in his keyboard to classify and store it to the appropriate Class. The user can specify as many number of samples to be stored in the database. To improve accuracy, the number of samples should be more than 5. To begin the database construction process, click start video to open the web camera. The Stored Images are then Preprocessed to obtain feature rich and accurate and then trained using the TensorFlow and Jupyter NoteBook.

6.1 Source code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import csv
import copy
import argparse
import itertools
from collections import Counter
from collections import deque

import cv2 as cv
import numpy as np
import mediapipe as mp

from utils import CvFpsCalc
from model import KeyPointClassifier
from model import PointHistoryClassifier


def get_args():
    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, default=0)
    parser.add_argument("--width", help='cap width', type=int, default=960)
    parser.add_argument("--height", help='cap height', type=int, default=540)

    parser.add_argument('--use_static_image_mode', action='store_true')
    parser.add_argument("--min_detection_confidence",
                        help='min_detection_confidence',
                        type=float,
                        default=0.7)
    parser.add_argument("--min_tracking_confidence",
                        help='min_tracking_confidence',
                        type=int,
```

```
                    default=0.5)

    args = parser.parse_args()

    return args


def main():
    # Argument parsing #################################################################
    args = get_args()

    cap_device = args.device
    cap_width = args.width
    cap_height = args.height

    use_static_image_mode = args.use_static_image_mode
    min_detection_confidence = args.min_detection_confidence
    min_tracking_confidence = args.min_tracking_confidence

    use_brect = True

    # Camera preparation ###############################################################
    cap = cv.VideoCapture(cap_device)
    cap.set(cv.CAP_PROP_FRAME_WIDTH, cap_width)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)

    # Model load #######################################################################
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands(
        static_image_mode=use_static_image_mode,
        max_num_hands=2,
        min_detection_confidence=min_detection_confidence,
        min_tracking_confidence=min_tracking_confidence,
    )

    keypoint_classifier = KeyPointClassifier()

    point_history_classifier = PointHistoryClassifier()

    # Read labels ######################################################################
    with open('model/keypoint_classifier/keypoint_classifier_label.csv',
              encoding='utf-8-sig') as f:
        keypoint_classifier_labels = csv.reader(f)
        keypoint_classifier_labels = [
            row[0] for row in keypoint_classifier_labels
        ]
    with open(
            'model/point_history_classifier/point_history_classifier_label.csv',
            encoding='utf-8-sig') as f:
        point_history_classifier_labels = csv.reader(f)
        point_history_classifier_labels = [
            row[0] for row in point_history_classifier_labels
```

```
    ]

# FPS Measurement ########################################################
cvFpsCalc = CvFpsCalc(buffer_len=10)

# Coordinate history #################################################################
history_length = 16
point_history = deque(maxlen=history_length)

# Finger gesture history ################################################
finger_gesture_history = deque(maxlen=history_length)

#  ########################################################################
mode = 0

while True:
    fps = cvFpsCalc.get()

    # Process Key (ESC: end) #################################################
    key = cv.waitKey(10)
    if key == 27:  # ESC
        break
    number, mode = select_mode(key, mode)

    # Camera capture #####################################################
    ret, image = cap.read()
    if not ret:
        break
    image = cv.flip(image, 1)  # Mirror display
    debug_image = copy.deepcopy(image)

    # Detection implementation
#############################################################
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

    image.flags.writeable = False
    results = hands.process(image)
    image.flags.writeable = True

    #  ####################################################################
    if results.multi_hand_landmarks is not None:
        for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
                                              results.multi_handedness):
            # Bounding box calculation
            brect = calc_bounding_rect(debug_image, hand_landmarks)
            # Landmark calculation
            landmark_list = calc_landmark_list(debug_image, hand_landmarks)

            # Conversion to relative coordinates / normalized coordinates
            pre_processed_landmark_list = pre_process_landmark(
                landmark_list)
            pre_processed_point_history_list = pre_process_point_history(
```

```
                debug_image, point_history)
            # Write to the dataset file
            logging_csv(number, mode, pre_processed_landmark_list,
                        pre_processed_point_history_list)

            # Hand sign classification
            hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
            if hand_sign_id == 2:  # Point gesture
                point_history.append(landmark_list[8])
            else:
                point_history.append([0, 0])

            # Finger gesture classification
            finger_gesture_id = 0
            point_history_len = len(pre_processed_point_history_list)
            if point_history_len == (history_length * 2):
                finger_gesture_id = point_history_classifier(
                    pre_processed_point_history_list)

            # Calculates the gesture IDs in the latest detection
            finger_gesture_history.append(finger_gesture_id)
            most_common_fg_id = Counter(
                finger_gesture_history).most_common()

            # Drawing part
            debug_image = draw_bounding_rect(use_brect, debug_image, brect)
            debug_image = draw_landmarks(debug_image, landmark_list)
            debug_image = draw_info_text(
                debug_image,
                brect,
                handedness,
                keypoint_classifier_labels[hand_sign_id],
                point_history_classifier_labels[most_common_fg_id[0][0]],
            )
        else:
            point_history.append([0, 0])

        debug_image = draw_point_history(debug_image, point_history)
        debug_image = draw_info(debug_image, fps, mode, number)

        # Screen reflection #############################################################
        cv.imshow('Hand Gesture Recognition', debug_image)

    cap.release()
    cv.destroyAllWindows()


def select_mode(key, mode):
    number = -1
    if 48 <= key <= 57:  # 0 ~ 9
        number = key - 48
    if key == 110:  # n
```

```
        mode = 0
    if key == 107:  # k
        mode = 1
    if key == 104:  # h
        mode = 2
    return number, mode



def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

    landmark_point = []

    # Keypoint
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)
        # landmark_z = landmark.z

        landmark_point.append([landmark_x, landmark_y])

    return landmark_point


def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, landmark_point in enumerate(temp_landmark_list):
        if index == 0:
            base_x, base_y = landmark_point[0], landmark_point[1]

        temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
        temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
    temp_landmark_list = list(
        itertools.chain.from_iterable(temp_landmark_list))

    # Normalization
    max_value = max(list(map(abs, temp_landmark_list)))

    def normalize_(n):
        return n / max_value

    temp_landmark_list = list(map(normalize_, temp_landmark_list))

    return temp_landmark_list
```

```
def pre_process_point_history(image, point_history):
    image_width, image_height = image.shape[1], image.shape[0]

    temp_point_history = copy.deepcopy(point_history)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, point in enumerate(temp_point_history):
        if index == 0:
            base_x, base_y = point[0], point[1]

        temp_point_history[index][0] = (temp_point_history[index][0] -
                                        base_x) / image_width
        temp_point_history[index][1] = (temp_point_history[index][1] -
                                        base_y) / image_height

    # Convert to a one-dimensional list
    temp_point_history = list(
        itertools.chain.from_iterable(temp_point_history))

    return temp_point_history


def logging_csv(number, mode, landmark_list, point_history_list):
    if mode == 0:
        pass
    if mode == 1 and (0 <= number <= 9):
        csv_path = 'model/keypoint_classifier/keypoint.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_list])
    if mode == 2 and (0 <= number <= 9):
        csv_path = 'model/point_history_classifier/point_history.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *point_history_list])
    return


def draw_landmarks(image, landmark_point):
    if len(landmark_point) > 0:
        # Thumb
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (255, 255, 255), 2)
```

```
# Index finger
cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
        (255, 255, 255), 2)

# Middle finger
cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
        (255, 255, 255), 2)

# Ring finger
cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
        (255, 255, 255), 2)

# Little finger
cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
```

```
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
            (255, 255, 255), 2)

        # Palm
        cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
            (255, 255, 255), 2)

    # Key Points
    for index, landmark in enumerate(landmark_point):
        if index == 0:  # 手首1
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 1:  # 手首2
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 2:  # 親指：付け根
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 3:  # 親指：第1関節
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
```

```
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 4:  # 親指：指先
            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
        if index == 5:  # 人差指：付け根
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 6:  # 人差指：第2関節
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 7:  # 人差指：第1関節
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 8:  # 人差指：指先
            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
        if index == 9:  # 中指：付け根
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

    return image


def draw_bounding_rect(use_brect, image, brect):
    if use_brect:
        # Outer rectangle
        cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[3]),
                     (0, 0, 0), 1)

    return image


def draw_info_text(image, brect, handedness, hand_sign_text,
                   finger_gesture_text):
    cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22),
                 (0, 0, 0), -1)

    info_text = handedness.classification[0].label[0:]
    if hand_sign_text != "":
        info_text = info_text + ':' + hand_sign_text
    cv.putText(image, info_text, (brect[0] + 5, brect[1] - 4),
               cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1, cv.LINE_AA)

    if finger_gesture_text != "":
        cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
```

```
                cv.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 0), 4, cv.LINE_AA)
        cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
                cv.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2,
                cv.LINE_AA)

    return image


def draw_point_history(image, point_history):
    for index, point in enumerate(point_history):
        if point[0] != 0 and point[1] != 0:
            cv.circle(image, (point[0], point[1]), 1 + int(index / 2),
                    (152, 251, 152), 2)

    return image


def draw_info(image, fps, mode, number):
    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
            1.0, (0, 0, 0), 4, cv.LINE_AA)
    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
            1.0, (255, 255, 255), 2, cv.LINE_AA)

    mode_string = ['Logging Key Point', 'Logging Point History']
    if 1 <= mode <= 2:
        cv.putText(image, "MODE:" + mode_string[mode - 1], (10, 90),
                cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
                cv.LINE_AA)
        if 0 <= number <= 9:
            cv.putText(image, "NUM:" + str(number), (10, 110),
                    cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
                    cv.LINE_AA)
    return image


if __name__ == '__main__':
    main()
```

TRAINING MODEL

6.2 Jupyter Classification

Specify each path

In [39]:
```
dataset = 'model/keypoint_classifier/keypoint.csv'
model_save_path = 'model/keypoint_classifier/keypoint_classifier.hdf5'
tflite_save_path = 'model/keypoint_classifier/keypoint_classifier.tflite'
```

Set number of classes

In [40]:
```
NUM_CLASSES = 5
```

Dataset reading

In [41]:
```
X_dataset = np.loadtxt(dataset, delimiter=',', dtype='float32', usecols=list(range(1, (21 * 2) + 1)))
```

In [42]:
```
y_dataset = np.loadtxt(dataset, delimiter=',', dtype='int32', usecols=(0))
```

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X_dataset, y_dataset, train_size=0.75, random_state=RANDOM_SEED)
```

Model building

In [44]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input((21 * 2, )),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

Fig 6.1.1

In [45]:
```
model.summary()  # tf.keras.utils.plot_model(model, show_shapes=True)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout_4 (Dropout)         (None, 42)                0

 dense_6 (Dense)             (None, 20)                860

 dropout_5 (Dropout)         (None, 20)                0

 dense_7 (Dense)             (None, 10)                210

 dense_8 (Dense)             (None, 5)                 55

=================================================================
Total params: 1,125
Trainable params: 1,125
Non-trainable params: 0
_________________________________________________________________
```

In [46]:
```
# Model checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    model_save_path, verbose=1, save_weights_only=False)
# Callback for early stopping
es_callback = tf.keras.callbacks.EarlyStopping(patience=20, verbose=1)
```

In [47]:
```
# Model compilation
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Fig 6.1.2

Model training

In [48]:
```
model.fit(
    X_train,
    y_train,
    epochs=1000,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[cp_callback, es_callback]
)
```

```
31/32 [============================>.] - ETA: 0s - loss: 0.5486 - accuracy: 0.7956
Epoch 188: saving model to model/keypoint_classifier\keypoint_classifier.hdf5
32/32 [==============================] - 0s 4ms/step - loss: 0.5489 - accuracy: 0.7956 - val_loss: 0.2512 - val_accuracy: 0.9
555
Epoch 189/1000
28/32 [=========================>....] - ETA: 0s - loss: 0.5649 - accuracy: 0.7938
Epoch 189: saving model to model/keypoint_classifier\keypoint_classifier.hdf5
32/32 [==============================] - 0s 4ms/step - loss: 0.5538 - accuracy: 0.7984 - val_loss: 0.2481 - val_accuracy: 0.9
555

Epoch 190/1000
26/32 [=======================>......] - ETA: 0s - loss: 0.5693 - accuracy: 0.7785
Epoch 190: saving model to model/keypoint_classifier\keypoint_classifier.hdf5
32/32 [==============================] - 0s 4ms/step - loss: 0.5748 - accuracy: 0.7790 - val_loss: 0.2495 - val_accuracy: 0.9
532
Epoch 191/1000
27/32 [========================>.....] - ETA: 0s - loss: 0.5353 - accuracy: 0.8073
Epoch 191: saving model to model/keypoint_classifier\keypoint_classifier.hdf5
32/32 [==============================] - 0s 5ms/step - loss: 0.5448 - accuracy: 0.8042 - val_loss: 0.2463 - val_accuracy: 0.9
608
```

In [49]:
```
# Model evaluation
val_loss, val_acc = model.evaluate(X_test, y_test, batch_size=128)
```

```
11/11 [==============================] - 0s 2ms/step - loss: 0.2531 - accuracy: 0.9563
```

Fig 6.1.3 Training Model-1

In [50]:
```
# Loading the saved model
model = tf.keras.models.load_model(model_save_path)
```

In [51]:
```
# Inference test
predict_result = model.predict(np.array([X_test[0]]))
print(np.squeeze(predict_result))
print(np.argmax(np.squeeze(predict_result)))
```

```
1/1 [==============================] - 0s 73ms/step
[9.2435974e-01 7.5487599e-02 7.9600781e-05 1.3809054e-05 5.9255006e-05]
0
```

Fig 6.1.4 Training model-1

Confusion matrix

In [52]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

def print_confusion_matrix(y_true, y_pred, report=True):
    labels = sorted(list(set(y_true)))
    cmx_data = confusion_matrix(y_true, y_pred, labels=labels)

    df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)

    fig, ax = plt.subplots(figsize=(7, 6))
    sns.heatmap(df_cmx, annot=True, fmt='g' ,square=False)
    ax.set_ylim(len(set(y_true)), 0)
    plt.show()

    if report:
        print('Classification Report')
        print(classification_report(y_test, y_pred))

Y_pred = model.predict(X_test)
y_pred = np.argmax(Y_pred, axis=1)

print_confusion_matrix(y_test, y_pred)
```

Fig 6.1.5 C.M

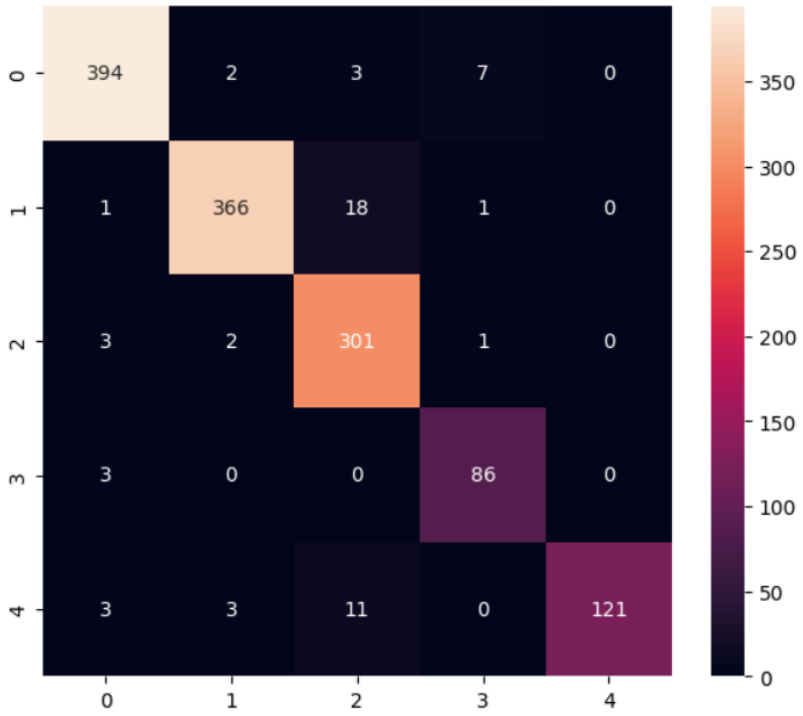

Fig 6.1.6 Confusion Matrix

Classification Report

	precision	recall	f1-score	support
0	0.98	0.97	0.97	406
1	0.98	0.95	0.96	386
2	0.90	0.98	0.94	307
3	0.91	0.97	0.93	89
4	1.00	0.88	0.93	138
accuracy			0.96	1326
macro avg	0.95	0.95	0.95	1326
weighted avg	0.96	0.96	0.96	1326

Fig 6.1.7 Classification Report

Convert to model for Tensorflow-Lite

In [53]:
```
# Save as a model dedicated to inference
model.save(model_save_path, include_optimizer=False)
```

In [54]:
```
# Transform model (quantization)

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()

open(tflite_save_path, 'wb').write(tflite_quantized_model)
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be direc
tly callable after loading.

INFO:tensorflow:Assets written to: C:\Users\91799\AppData\Local\Temp\tmpf8l43e3w\assets

INFO:tensorflow:Assets written to: C:\Users\91799\AppData\Local\Temp\tmpf8l43e3w\assets
```

Out[54]: 6616

Fig 6.1.8 Model- tensor flow

Inference test

In [55]:
```
interpreter = tf.lite.Interpreter(model_path=tflite_save_path)
interpreter.allocate_tensors()
```

In [56]:
```
# Get I / O tensor
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

In [57]:
```
interpreter.set_tensor(input_details[0]['index'], np.array([X_test[0]]))
```

In [58]:
```
%%time
# Inference implementation
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

```
CPU times: total: 0 ns
Wall time: 989 µs
```

In [59]:
```
print(np.squeeze(tflite_results))
print(np.argmax(np.squeeze(tflite_results)))
```

```
[9.24359858e-01 7.54875690e-02 7.96006279e-05 1.38090545e-05
 5.92549513e-05]
0
```

Fig 6.1.9 Inference Test

RESULTS

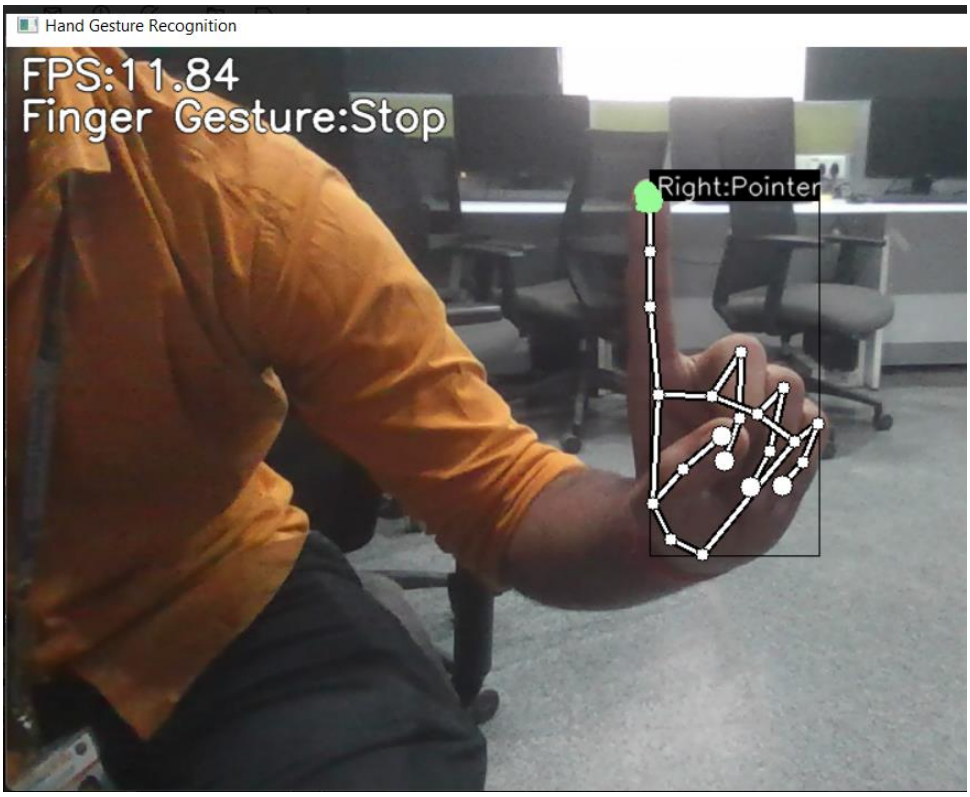

Fig 6.2.1 Gesture-Pointer

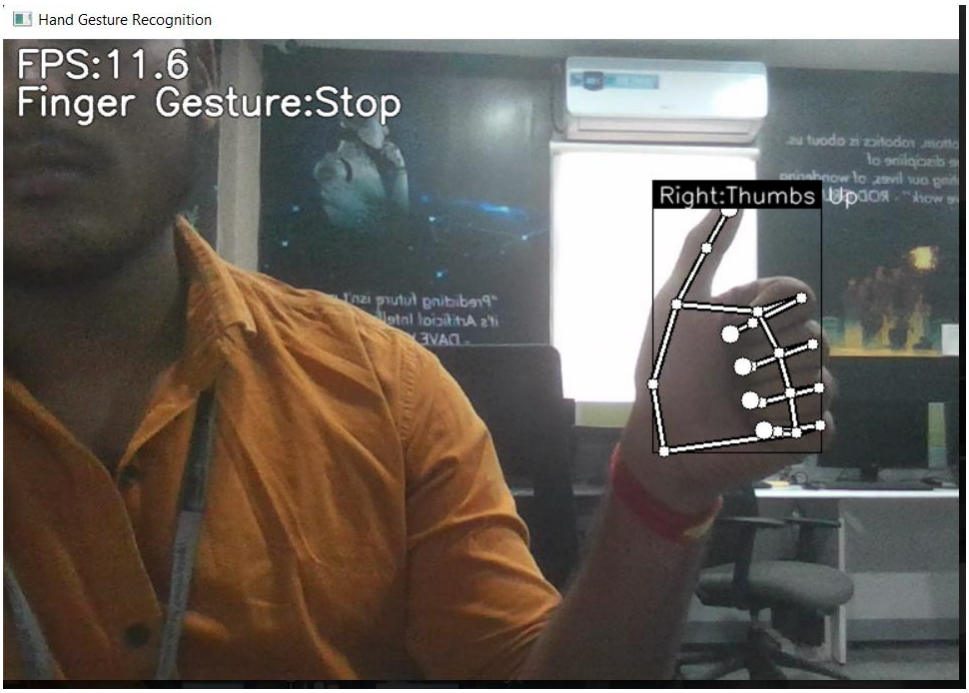

Fig 6.2.2 Gesture-Thumbs Up

Fig 6.2.3 Gesture- OK

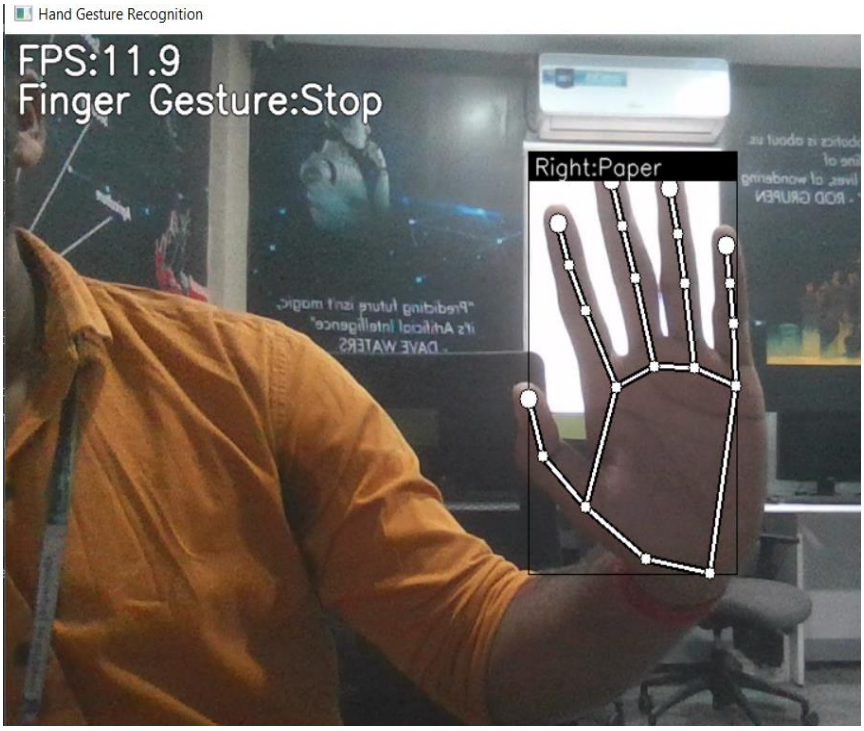

Fig 6.2.4 Gesture- Paper

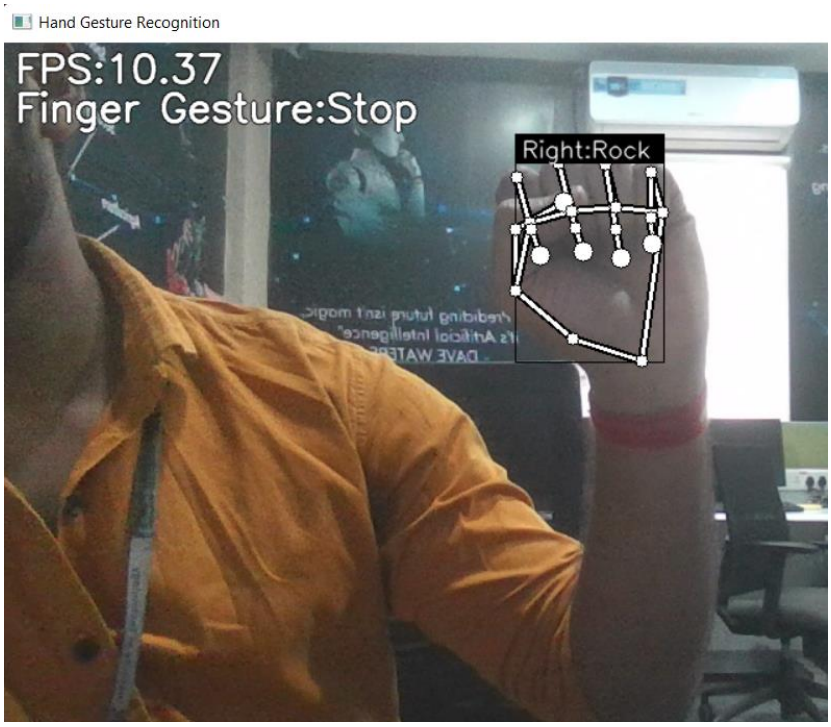

Fig 6.2.5 Rock

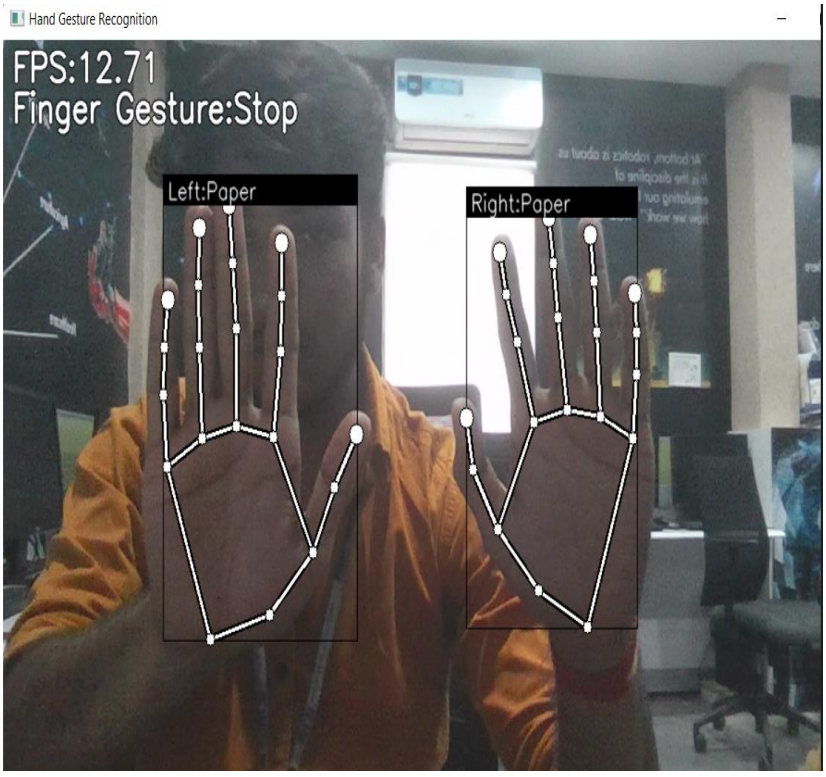

Fig 6.2.6 Gesture- Both Hands

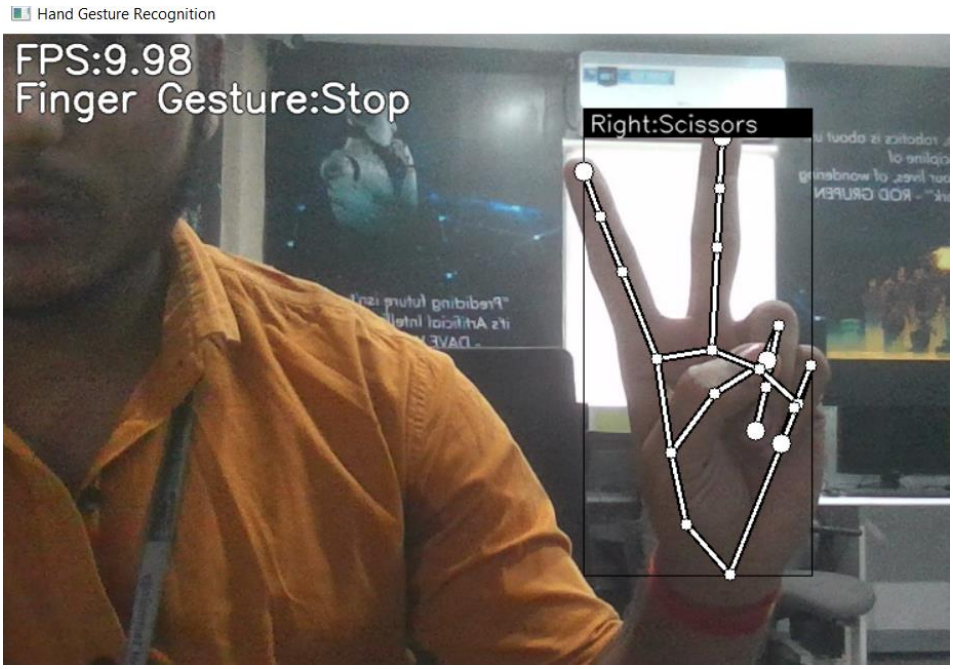

Fig 6.2.7 Gesture- Scissors

CONCLUSION AND FUTURE SCOPE

As a conclusion based on the result of the project, it can be seen that developing the hand gesture recognition using Python and OpenCV can be implemented by applying the theories of hand segmentation and the hand detection system which use the Haar-cascade classifier. To summarize it, this system has accomplished several objective in this project: (1) manage to establish a complete system for detecting, recognizing and interpreting hand gesture recognition through computer vision using Python and OpenCV, and (2) able to create the numbers and sign languages of hand gesture shown in the system that will meets the name of the project.

FUTURE SCOPE

The system could also be made smart to be trained for only one or two gestures rather than all and then made ready for testing. This will require only a few changes in the current interface code, which were not performed due to the shortage of time.

One time training constraint for real time system can be removed if the algorithm is made efficient to work with all skin types and light conditions which seems impossible by now altogether. Framing with COG (Centre of gravity) to control orientation factor could make this system more perfect for real application.

The system’s speed for preprocessing could be improved if the code is developed in VC/VC.Net

REFERENCES

[1] J.Jenkinwinston (96207106036), M.Maria Gnanam (96207106056), R.Ramasamy (96207106306), Anna University of Technology, Tirunelveli: Hand Gesture Recognitionsystem Using Haar Wavelet.

[2] Laura Dipietro, Angelo M. Sabatini, Senior Member, IEEE, and Paolo Dario, Fellow, IEEE, A Survey of Glove-Based Systems and Their Applications.

[3] Kay M. Stanney HANDBOOK OF VIRTUAL ENVIRONMENTS Design, Implementation, and Applications, Gesture Recognition Chapter #10 by Matthew Turk

[4] Daniel Thalman, Gesture Recognition Motion Capture, Motion Retargeting, and Action Recognition

[5] Hatice Gunes, Massimo Piccardi, Tony Ja, 2007, Research School of Information Sciences and Engineering Australian National University Face and Body Gesture Recognition for a Vision-Based Multimodal Analyzer

[6] J. Heinzm ann and A. Zelinsky Robust Real - Time Face Tracking and Gesture Recognition

[7] Vladimir Vezhnevets, Vassili Sazonov, Alla Andreeva, Moscow State University A Survey on Pixel-Based Skin Color Detection Techniques,

[8] TEY YI CHI, Universiti Teknologi Malaysi, FUZZY SKIN DETECTION

[9] Robyn Ball and Philippe Tissot, Texas A&M University, Demonstration of Artificial Neural Network in Matlab

[10] Howard Demuth, Mark Beale, Neural Network Toolbox

[11] Ari Y. Benbasat and Joseph A. Paradiso in MIT Media Laboratory, Cambridge An Inertial Measurement Framework for Gesture Recognition and Applications

[12] Peggy Gerardin, Color Imaging Course Winter Semester 2002-2003, Color Image Segmentation

[13] Michael Nielsen, Moritz Störring, Thomas B. Moeslund, and Erik Granum, March 2003, A procedure for developing intuitive and ergonomic gesture interfaces for manmachine interaction