* 1. **INTRODUCTION**

# 1. INTRODUCTION

Human computer interaction is one of the most rapidly growing technologies. Computer machines have the ability to make our daily lives much more convenient. Human Pose estimation is a computer vision task that represents the orientation of a person in a graphical format. This technique is widely applied to predict a person’s body parts or joint position. It is one of the most exciting areas of research in computer vision that has gained a lot of traction because of its abundance of applications that can benefit from such a technology.

As we all know that Natural human language communication is based on languages, poses and gestures. Keeping this in mind we have created a project which detects the pose of a human. Human Pose Estimation identifies and classifies the poses of human body parts and joints in images or videos. Essentially it is a way to capture a set of coordinates by defining the human body joints like wrist, shoulder, knees, eyes, ears, ankles, and arms, which is a key point in images and videos that can describe a pose of a person. Then, when an image or video is given to the pose estimator model as input, it identifies the coordinates of these detected body parts and joints as output and a confidence score showing precision of the estimations. We have implemented this project using computer vision and media pipe.

Human pose detection can be viewed as a way for computers to begin to understand human body language and signs, thus filling the void between computing machines and humans. The model which we have created include detecting the human body parts and tracking its landmarks which is done using media pipe. The detection includes drawing the skeletal structure of a human body. The camera will be opened using cv2 module where the human is visible and the land marks are detected. The human pose detection project contains three phases. They are:

1. Pose detection for an image which is given as input where all the 33 key points are marked and a skeletal structure is formed.
2. Pose detection for a real time video where the structure formed moves and displays the position of the human.

3. Count the number of reps performed by the human and display the count as output.

## MOTIVATION

Detection of people has long been a primary center of discussion for various applications in traditional object detection. With recent developments in machine-learning algorithms, computers can now understand human body language by performing pose detection and pose tracking. The accuracy of these detections and the hardware requirements to run them has now reached a point where it has become commercially viable.

In addition, the technology’s growth is also profoundly transformed amid the coronavirus pandemic, where high-performing real-time pose detection and tracking will bring some of the most influential trends in computer vision. For instance, it can be employed for social distancing by combining human pose estimation and distance projection heuristics. It assists people in maintaining physical distance from each other in a crowded place. Human pose estimation will significantly impact various industries, including security, business intelligence, health and safety, and entertainment. One such area where this technique has already proved its existence is autonomous driving. With the help of real-time human pose detection and tracking, computers can sense and predict pedestrian behaviour much thoroughly – allowing more consistent driving.

## OBJECTIVE OF THE PROJECT

The main objective of the proposed human pose detection project is to develop the skeletal structure of the human body in real-time and static environment and detect the body’s 33 local key points. All of these points, after detected, can be applied to any logic to execute a feature or furnish a solution to a big problem. In this project, we detect the human body pose with the key points and extract the key points that are required to calculate reps i.e., to estimate the body points of arms and shoulders, calculate the angle between them and calculate the amount of reps in real time.

## LIMITATIONS OF THE PROJECT

* Occlusion: When a person's body parts are partially or fully hidden from view, it can be difficult or impossible for a pose detection system to accurately estimate their pose. This is a common problem in crowded scenes or when people are moving quickly.
* Lighting and camera angle: Changes in lighting and camera angle can affect the accuracy of pose detection algorithms. For example, if a person is backlit or if there are shadows on their body, it may be more difficult to accurately detect their pose.
* Clothing and accessories: Clothing and accessories can also interfere with pose detection. For example, baggy or loose-fitting clothing can obscure the shape of the body, while jewelry or other accessories may be mistaken for body parts.
* Limited pose variations: Most pose detection algorithms are designed to detect a specific set of poses, such as standing, sitting, or lying down. If a person is in a pose that is not included in the algorithm's training data, it may be difficult or impossible to accurately detect their pose.
* Computational resources: Human pose detection can require significant computational resources, particularly if high-resolution images or video are being analyzed. This can limit the scalability and speed of pose detection systems.

## ORGANIZATION OF THE PROJECT

The first chapter deals with the introduction of our project, motivation for developing this project, limitations and objective of the project. The second chapter deals with the already existing systems and gives some idea on how the Proposed System must look like. The third chapter deals with the system specifications required for developing the project. It includes hardware & software specifications. The fourth chapter gives the design and implementation which includes introduction, source code, Description of key parameters & function, methods of implementation, testing and validations. Finally fifth chapter deals with the conclusion and future scope of the project.

* 1. **INTRODUCTION**

# LITERATURE SURVEY

## Research on importance of human pose detection

Human pose estimation aims at predicting the poses of human body parts in images or videos. Since pose motions are often driven by some specific human actions, knowing the body pose of a human is critical for action recognition. Localizing human body joint positions in images or videos is the task of human pose estimation. By knowing the motion of a human body in a sequence of images, we can infer what the person is doing, that is, recognizing the action. In this sense, action recognition can be viewed as a natural application of human pose estimation. With the advance of deep neural networks, the last decade has witnessed great progress on human pose estimation techniques.[1]

## Research on applications of human pose detection in various fields

It has various applications in fields such as healthcare, sports analysis, robotics, entertainment, and more. For instance, human pose estimation can be used in sports analysis to track athletes' movements and patterns, providing insights for performance improvement and injury prevention. Additionally, it can be used in rehabilitation to monitor patients undergoing physical therapy and provide feedback on their movements. Human pose estimation can also be used to analyze workplace ergonomics, enhance the virtual reality experience, enable robots to recognize human gestures and movements, improve the user experience in gaming, and much more. In healthcare, human pose estimation can be used for patient monitoring, fall detection, posture correction, and other applications. Other potential uses include fashion, dance analysis, art, education, music, and motion capture. With the advancement of technology, the applications of human pose estimation continue to expand, and it has a promising future in a wide range of industries.[2]

## Various approaches for human pose detection

Human pose detection needs to be estimated in 2D and 3D environments with various factors including posture, light, surroundings, background etc. And in these environments there are single-person and multi-person detection. Single-person detection is easy as it can be made possible using regressors. Multi-person detection includes top-down and bottom-up approaches which increase the detection efficiency to a greater extent. [3]

## Research on pose detection using Opencv

OpenCV (Open Source Computer Vision Library) is a popular computer vision library that provides tools and algorithms for image and video processing. It is widely used in various applications, including human pose estimation. OpenCV provides pre-trained classifiers for detecting human body parts such as faces, eyes, and body poses. These classifiers can be used to detect human body parts in images and videos and can be combined with human pose estimation algorithms to improve accuracy. OpenCV provides a wide range of algorithms for human pose estimation, including the OpenPose library, which is a popular library for multi-person pose estimation. OpenCV also provides tools for developing custom algorithms and models for human pose estimation. [4]

## Importance of Mediapipe for detecting poses

MediaPipe Pose is a high-fidelity body pose tracking solution that renders 33 3D landmarks and a background segmentation mask on the whole body from RGB frames (Note RGB image frame). It utilizes BlazePose topology, a superset of COCO, BlazeFace, and BlazePalm topology. It works in two stages which are detection and tracking. As As detection is not performed in each frame, MediaPipe is able to perform inference faster. There are three models in MediaPipe for pose estimation.

* + - BlazePose GHUM Heavy
    - BlazePose GHUM Full
    - BlazePose GHUM Lite [5]

## Research on why Mediapose is used rather than YOLOv7 Pose

YOLOv7 Pose was introduced in the YOLOv7 repository a few days after the initial release in July ‘22. It is a single-stage, multi-person pose estimation model. YOLOv7 pose is unique, as it deviates from the conventional 2-stage pose estimation algorithms. With the reduced complexity in single-stage models, we can expect them to be faster and more efficient. It has 17 COCO Keypoints which are lesser than 33 COCO Keypoints of Mediapipe. Also, it needs GPU but Mediapipe completes the task without using GPU. Mediapipe is lightweight and can run at more Frames Per Second rather than YOLOv7. [5]

## Top-Down approach for detection

In human pose estimation, the top-down approach is a technique that involves first detecting the entire human body or its parts in an image or video, and then estimating the pose of each detected body part. The top-down approach starts with a detection step, which uses object detection or segmentation techniques to identify the presence and location of the human body or its parts in the image. This step can use pre-trained models or classifiers to detect different parts of the body, such as the head, torso, arms, and legs. Once the body or its parts are detected, the pose estimation step uses machine learning or computer vision algorithms to estimate the pose of each part. This step involves analyzing the detected body part and estimating its orientation, position, and other relevant characteristics.

However, the top-down approach may be limited in situations where the images or videos have complex backgrounds, occlusions, or partial views of the human body. In such cases, the bottom-up approach, which involves first detecting individual body parts and then assembling them into a full pose, may be more effective. [6]

## Bottom-Up approach for detection

In human pose estimation, the bottom-up approach is a technique that involves first detecting individual body parts in an image or video, and then grouping them together to form a full pose. The bottom-up approach starts with a part detection step, which uses computer vision techniques to identify the presence and location of different body parts, such as the head, torso, arms, and legs. This step can use pre-trained models or classifiers to detect different parts of the body, and may involve techniques such as keypoint detection, segmentation, or template matching. Once the body parts are detected, the pose estimation step involves grouping them together to form a full pose. This step can use techniques such as clustering, graph-based methods, or optimization algorithms to match the detected parts with their corresponding body joints and estimate the pose.

However, the bottom-up approach may be limited in situations where the body parts are not detected accurately or where there is significant variability in the appearance or pose of the human body. In such cases, the top-down approach, which involves first detecting the entire human body or its parts and then estimating the pose of each detected part, may be more effective. [6]

## EXISTING SYSTEM AND DISADVANTAGES OF EXISTING SYSTEM

* + - Existing systems for human pose detection typically rely on traditional computer vision techniques that involve detecting key features on the body, such as the hands or face. These systems often require complex image processing algorithms and may have limitations in their ability to detect movements accurately and in real-time.
    - One disadvantage of these traditional systems is that they may not be able to accurately detect movements in complex environments with varying lighting conditions and background clutter. Additionally, these systems may struggle to detect movements that are occluded, such as when the user's hand is behind their body.
    - Another disadvantage of traditional pose detection systems is that they may only detect a few key points on the body, which can lead to inaccurate tracking of movements. This can limit their effectiveness in applications such as sports analysis, where accurate tracking of movements is essential.
    - Furthermore, traditional pose detection systems may be difficult to customize for specific applications and may require significant development time and resources to implement.
    - In contrast, the human pose detection project using Media Pipe addresses some of the limitations of traditional systems by using a deep neural network to detect 33 body landmarks with increased accuracy and real-time performance. The system is also highly customizable and accessible, making it suitable for a wide range of applications.

## PROPOSED SYSTEM

* + - The proposed system for human pose detection uses the Media Pipe framework, which is a cross-platform solution for building real-time machine learning pipelines. The system aims to detect and track 33 body landmarks, including the nose, eyes, ears, shoulders, elbows, wrists, hips, knees, and ankles, in real-time.
    - The system leverages the power of deep neural networks to achieve high accuracy in pose detection. It is based on a pre-trained pose detection model that has been trained on a large dataset of labeled human poses. The model is optimized for real-time performance on mobile and desktop devices, making it suitable for a wide range of applications.
    - Python programming language is used for developing this project.
    - To use the system, a video or live camera feed is input into the system. The system then analyzes the frames in real-time and detects the 33 body landmarks using the pre-trained model. The detected landmarks are then used to track the movements of the user, providing real-time feedback and monitoring.
    - The model makes use of media pipe package for tracking different body parts. The media pipe takes human body image as input and produces landmarks as output.

# SYSTEM SPECIFICATIONS

## HARDWARE REQUIREMENTS

* + - System : i3
    - Hard disk : 40 GB
    - RAM : 4 GB
    - Webcam

## SOFTWARE REQUIREMENTS

* + - Operating system : Windows 8
    - Coding language : Python 3.7
    - IDE : PyCharm

1. **DESIGN AND IMPLEMENTATION**
   1. **INTRODUCTION**

Human pose estimation is a computer vision task that involves detecting and estimating the position and orientation of the human body. Our goal is to detect the human pose and make the user able to count the number of reps that he /she is doing while present in frame with the webcam. To achieve this task , we are taking Mediapipe as the important tool to detect the images using 33 COCO points .

**Video Capturing Modules**

With OpenCV we can capture a video from the camera. It lets us create a video capture object which is helpful to capture videos through webcam and then we can perform desired operations an that video. In this pictures can be conveyed in concealing layered with three channels red, green, blue. A video capture id is given as the id for the cam. Hand gesture based keyboard framework utilizes the instructive algorithmic rule and it changes over the coordinates of tip from camera screen to pc window full-screen.[9]

Steps to capture a video

* + - * Use cv2.VideoCapture() to get a video capture object for the camera.
      * Set up an infinite while loop and use the read() method to read the frames using the above created object.
      * Use cv2.imshow() method to show the frames in the video.
      * Breaks the loop when the user clicks a specific key.

## Palm Detection Module

It is used to detect initial hand locations. Detecting hands is a complex task. Our model have to work across variety of hand sizes with a large scale span relative to the image frame and be able to detect occluded and self-occluded hands. Here we first train a palm detector and next an encoder and decoder feature extract is used for bigger scene content awareness even for small objects. At last we minimize the focal loss during training to support large anchors resulting from high scale variance. Using this technique we can achieve an average precision of 95.7% in palm detection.[10]

## Hand Tracking Module

After running palm detection over the whole image, our subsequent hand landmark model performs precise landmark localization of 21 2.5D coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The model has three outputs

1. 21 hand landmarks consisting of x, y, and relative depth.
2. A hand flag indicating the probability of hand presence in the input image.
3. A binary classification of handedness, e.g. left or right hand.

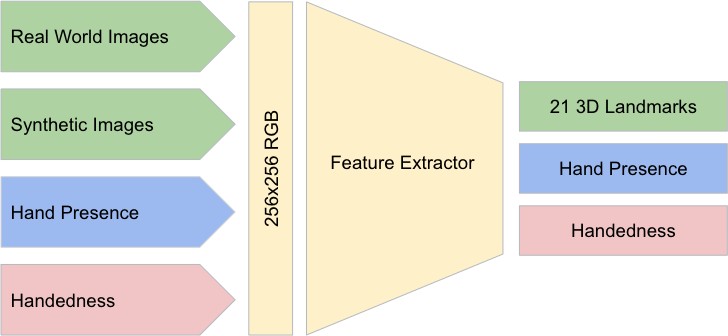
The 2D coordinates are learned from both real-world images as well as synthetic datasets as discussed below, with the relative depth with respect to the wrist point being learned only from synthetic images. To recover from tracking failure, we developed another output of the model for producing the probability of the event that a reasonably aligned hand is indeed present in the provided crop. If the score is lower than a threshold then the detector is triggered to reset tracking. Handedness is another important attribute for effective interaction using hands in AR/VR. This is especially useful for some applications where each hand is associated with a unique functionality. Thus, we developed a binary classification head to predict whether the input hand is the left or right hand. Our setup targets real-time mobile GPU inference, but we have also designed lighter and heavier versions of the model to address CPU inference on the mobile devices lacking proper GPU support and higher accuracy requirements of accuracy to run on desktop, respectively.[10][11]

Fig 6.2.3.1 Architecture Of Hand Landmark Model

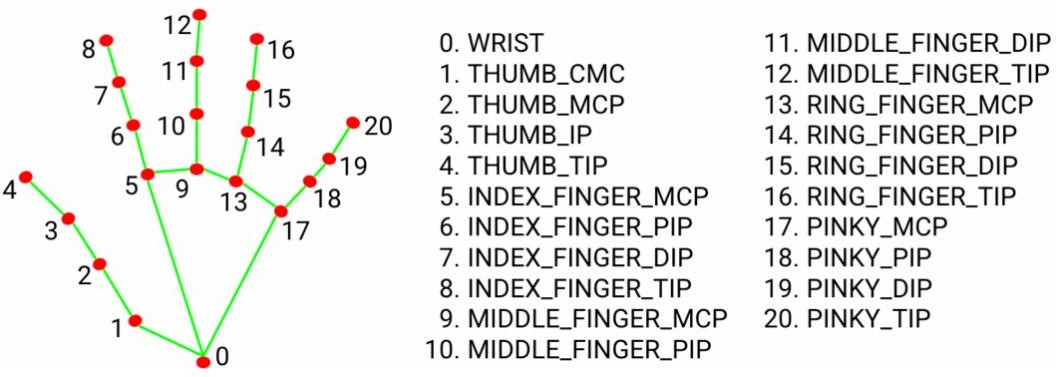


Fig 6.2.3.2 Hand Landmark Model

## REQUIREMENTS FOR THE PROJECT

* + 1. **Anaconda Navigator**

Write Anaconda matter here

## Python Programming Language

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

## Python Programming Characteristics

* + - * It provides rich data types and easier to read syntax than any other programming languages.
      * It is a platform independent scripted language with full access to operating system API's.
      * Compared to other programming languages, it allows more run-time flexibility.
      * It includes the basic text manipulation facilities of Perl and Awk.
      * A module in Python may have one or more classes and free functions.
      * Libraries in Pythons are cross-platform compatible with Linux, Macintosh, and Windows.
      * For building large applications, Python can be compiled to byte-code.
      * Python supports functional and structured programming as well as OOP.
      * It supports interactive mode that allows interacting Testing and debugging of snippets of code.
      * In Python, since there is no compilation step, editing, debugging and testing is fast.

## Applications of Python Programming Web Applications

You can create scalable Web Apps using frameworks and CMS (Content Management

System) that are built on Python. Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS. Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

## Scientific and Numeric Computing

There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: SciPy and NumPy that are used in general purpose computing. And, there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on. Also, the language is heavily used in machine learning, data mining and deep learning.

## Creating software Prototypes

Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must. However, Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

## Good Language to Teach Programming

Python is used by many companies to teach programming to kids and newbies. It is a good language with a lot of features and capabilities. Yet, it's one of the easiest language to learn because of its simple easy-to-use syntax.

## OpenCV

OpenCV is an open-source library for computer vision, machine learning and image processing. It supports a wide variety of programming languages like Python, C++, Java etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in our Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV. It contains imageprocessing algorithms for object detection and real-time computer vision applications can be developed by using the computer vision library. The OpenCV library is used in image and video processing and also analysis such as face detection and object detection. The OpenCV is used to capture the images through webcam in this project and for which we will be importing cv2.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

The term Computer Vision (CV) is used and heard very often in artificial intelligence (AI) and deep learning (DL) applications. The term essentially means giving a computer the ability to see the world as we humans do.

Computer Vision is a field of study which enables computers to replicate the human visual system. As already mentioned above, It’s a subset of artificial intelligence which collects information from digital images or videos and processes them to define the attributes. The entire process involves image acquiring, screening, analysing, identifying and extracting information. This extensive processing helps computers to understand any visual content and act on it accordingly.

Computer vision projects translate digital visual content into explicit descriptions to gather multi-dimensional data. This data is then turned into a computer-readable language to aid the decision-making process. The main objective of this branch of artificial intelligence is to teach machines to collect information from pixels.

A digital image is an image composed of picture elements, also known as pixels, each with finite, discrete quantities of numeric representation for its intensity or grey level. So the computer sees an image as numerical values of these pixels and in order to recognise a certain image, it has to recognise the patterns and regularities in this numerical data.

Here is a hypothetical example of how pixels form an image. The darker pixels are represented by a number closer to the zero and lighter pixels are represented by numbers approaching one. All other colours are represented by the numbers between 0 and 1.

But usually, you will find that for any colour image, there are 3 primary channels – Red, green and blue and the value of each channel varies from 0-255. In more simpler terms we can say that a digital image is actually formed by the combination of three basic colour channels Red, green, and blue whereas for a grayscale image we have only one channel whose values also vary from 0-255.

Computer Vision overlaps significantly with the following fields

* Image Processing − It focuses on image manipulation.
* Pattern Recognition − It explains various techniques to classify patterns.
* Photogrammetry − It is concerned with obtaining accurate measurements from images.

Image processing deals with image-to-image transformation. The input and output of image processing are both images. Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Features of OpenCV Library

* Read and write images
* Capture and save videos
* Process images (filter, transform)
* Perform feature detection
* Detect specific objects such as faces, eyes, cars, in the videos or images.
* Analyse the video, i.e., estimate the motion in it, subtract the background, and track objects in it.[14]

## Mediapipe

Change Mediapipe matter

MediaPipe is a Framework for building machine learning pipelines for processing timeseries data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano.

MediaPipe powers revolutionary products and services we use daily. Unlike powerhungry machine learning Frameworks, MediaPipe requires minimal resources. It is so tiny and efficient that even embedded IoT devices can run it. In 2019, MediaPipe opened up a new world of opportunity for researchers and developers following its public release.

The MediaPipe perception pipeline is called a Graph. Let us take the example of the first solution, Hands. We feed a stream of images as input which comes out with hand landmarks rendered on the images.

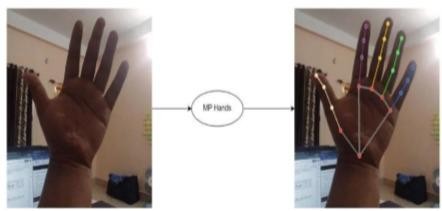


Fig 6.3.4.1 Mediapipe Hands  
add our images

The flow chart below represents the MP (Abbr. MediaPipe) hand solution graph.

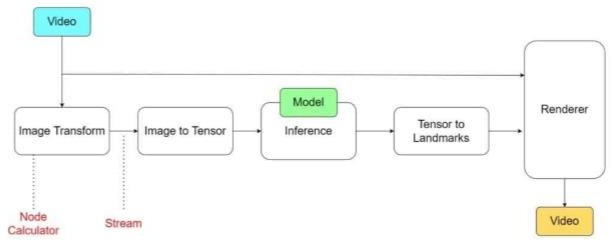
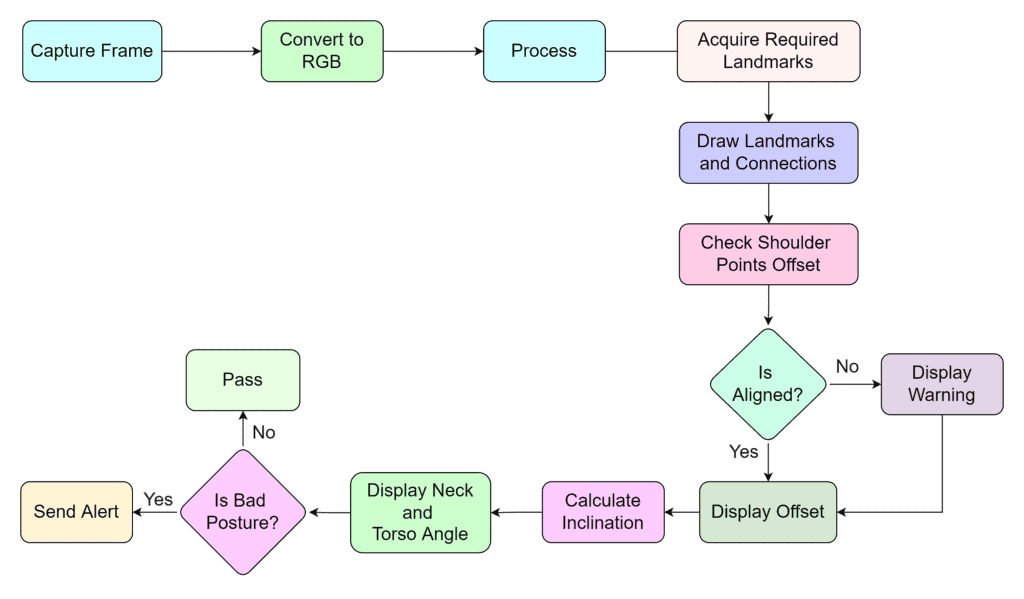


Fig 6.3.4.2 Hand Solution Graph

Replace the above image with this



In computer science jargon, a graph consists of Nodes connected by Edges. Inside the MediaPipe Graph, the nodes are called Calculators, and the edges are called Streams. Every stream carries a sequence of Packets that have ascending time stamps. In the image above, we have represented Calculators with rectangular blocks and Streams using arrows.

Solutions are open-source pre-built examples based on a specific pre-trained TensorFlow or TFLite model. You can check Solution specific models here. MediaPipe Solutions are built on top of the Framework.

The solutions are available in C++, Python, JavaScript, Android, iOS, and Coral. As of now, the majority of the solutions are available only in C++ (except KNIFT and IMT) followed by Android, with Python not too far behind.

The other wrapper languages, too, are growing fast with a very active development state. As you can see, even though MediaPipe Framework is cross-platform, that does not imply the same for the solutions. MediaPipe is currently at alpha version 0.7. We can expect the solutions to get more support with the beta releases.

MediaPipe supports multimodal graphs. To speed up the processing, different calculators run in separate threads. For performance optimization, many built-in calculators come with options for GPU acceleration. Working with time series data must be in proper synchronization; otherwise, the system will break. The graph ensures this so that flow is handled correctly according to the timestamps of packets. The Framework handles synchronization, context sharing, and inter-operations with CPU calculators.

MediaPipe depends on OpenCV for video and FFMPEG for audio data handling. It also has other dependencies like OpenGL/Metal, Tensorflow, Eigen [1], etc.

MediaPipe solutions are straightforward, and you can cover them in a day or two.On the other hand, the learning curve can be pretty steep for the C++ MediaPipe Framework. Don’t worry; we will get there by taking baby steps.

Overall, it is a beautiful, fast-growing library that delivers promising results. Implementing MediaPipe in projects nullifies most of the hassles we usually face while working on an ML project. No need to worry about synchronization and cumbersome setups. It allows you to focus on the actual development part.

In the upcoming posts, we will show how to build interesting Augmented Reality filters using the MediaPipe Face solution. Later in this series, we will cover customizing calculators of pre-built MediaPipe solutions and building custom graphs.

## Features

* **Input image processing** - Processing includes image rotation, resizing, normalization, and color space conversion.
* **Score threshold** - Filter results based on prediction scores.

|  |  |
| --- | --- |
| **Task inputs** | **Task outputs** |
| The Hand Landmarker accepts an input of one of the following data types:   * Still images. * Decoded video frames. * Live video feed. | The Hand Landmarker outputs the following results:   * Handedness of detected hands. * Landmarks of detected hands in image coordinates. * Landmarks of detected hands in world coordinates. |

Table 6.3.4.1 Features Of Mediapipe

## Solution API’s in mediapipe static\_image\_mode

If it is set to false the solution treats the input images as video stream. At first it tries to detect hands and upon a successful detection further localizes the hand landmarks. In subsequent images, once all max\_num\_hands are detected and the corresponding hand landmarks are localized. It simply tracks those landmarks without invoking another detection until it looses track of any of the hands. This reduces latency and is ideal for processing video frames. If set to true, hand detection runs on every input image, ideal for processing a batch of static. It is by default set to false.

## max\_num\_hands

It specifies the maximum number of hands to detect at a time. By default set to 2.

## model\_complexity

It specifies the complexity of the hand landmark model. It specifies the value as either 0 or 1. Landmark accuracy is directly proportional to model\_complexity. Default value is 1.

## minimum\_detection\_confidence

The minimum confidence score for the hand detection to be considered successful in palm detection model. It ranges between 0.0-1.0. By default the value is 0.5.

## minimum\_tracking\_confidence

The minimum confidence score for the hand tracking to be considered successful. This is the bounding box IoU threshold between hands in the current frame and the last frame. In Video mode and Stream mode of Hand Landmarker, if the tracking fails, Hand Landmarker triggers hand detection. Otherwise, it skips the hand detection. It ranges between 0.0-1.0. By default the value is 0.5.

## Output functions

**multi\_hand\_landmarks**

It is collection of detected or tracked hands, where each hand is represented as a list of 21 hand landmarks. Each landmark is composed of x, y, z. here x and y are normalized to [0.0- 0.1] by the image width and height respectively. z represents the landmark depth with the depth at the wrist begin the origin. The smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

## multi\_hand\_world\_landmarks

It is collection of detected or tracked hands, where each hand is represented as a list of 21 hand landmarks in world coordinates. Each landmark is composed of x, y, z in real world 3D coordinates in metres with the origin at the hands approximate geometric centre.

## multi\_handedness

It is collection of handedness of detected or tracked hands. Each hand is composed of label and score. Label is a string of value either left or right. Score is the estimated probability of the predicted handedness and is always greater than or equal to 0.5, handedness is detected by assuming the input image is mirrored i.e taken with a front facing camera with images flipped horizontally.[10]

## Time

The Python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code. This article will walk you through the most commonly used functions and objects in time.

First, time.time() returns the number of seconds that have passed since the epoch. The return value is a floating point number to account for fractional seconds:

>>> from time import time

>>> time() 1551143536.9323719

The number you get on your machine may be very different because the reference point considered to be the epoch may be very different.

Measuring time in seconds is useful for a number of reasons:You can use a float to calculate the difference between two points in time.A float is easily serializable, meaning that it can be stored for data transfer and come out intact on the other side.Sometimes, however, you may want to see the current time represented as a string. To do so, you can pass the number of seconds you get from time() into time.ctime().

As you saw before, you may want to convert the Python time, represented as the number of elapsed seconds since the epoch, to a string. You can do so using ctime():

>>> from time import time, ctime

>>> t = time()

>>> ctime(t)

'Tue Apr 19 19:11:56 2023'

Here, you’ve recorded the current time in seconds into the variable t, then passed t as an argument to ctime(), which returns a string representation of that same time. The representation of time dependent on your physical location is called local time and makes use of a concept called time zones.[15]

## Math

The methods in this module accepts int, float, and complex numbers. It even accepts Python objects that has a \_complex() or float\_() method. The methods in this module almost always return a complex number. If the return value can be expressed as a real number, the return value has an imaginary part of 0.

The cmath module has a set of methods and constants. This module provides access to the mathematical functions defined by the C standard. These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place. The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

## 6.3.6 Numpy

Numpy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Numpy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behaviour is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

## Pynput

The pynput library allows us to control and monitor input devices. It contains subpackages for each type of input device supported:

* + - * pynput.mouse**-**Contains classes for controlling and monitoring a mouse or trackpad.
      * pynput.keyboard**-**Contains classes for controlling and monitoring the keyboard.

The package pynput.keyboard contains classes for controlling and monitoring the keyboard.[16]

A keyboard listener is a threading. Thread, and all callbacks will be invoked from the thread. Call pynput.keyboard.Listener.stop from anywhere, raise StopException or return False from a callback to stop the listener. The key parameter passed to callbacks is a pynput.keyboard.Key, for special keys, a pynput.keyboard. KeyCode for normal alphanumeric keys, or just None for unknown keys. When using the non-blocking version above, the current thread will continue executing. This might be necessary when integrating with other GUI frameworks that incorporate a main-loop, but when run from a script, this will cause the program to terminate immediately.

The keyboard listener thread

The listener callbacks are invoked directly from an operating thread on some platforms, notably Windows. This means that long running procedures and blocking operations should not be invoked from the callback, as this risks freezing input for all processes. A possible workaround is to just dispatch incoming messages to a queue, and let a separate thread handle them.

Handling keyboard listener errors

If a callback handler raises an exception, the listener will be stopped. Since callbacks run in a dedicated thread, the exceptions will not automatically be reraised. To be notified about callback errors, call Thread.join on the listener instance.

Toggling event listening for the keyboard listener

Once pynput.keyboard.Listener.stop has been called, the listener cannot be restarted, since listeners are instances of threading.Thread. If your application requires toggling listening events, you must either add an internal flag to ignore events when not required, or create a new listener when resuming listening.

Synchronous event listening for the keyboard listener

To simplify scripting, synchronous event listening is supported through the utility class pynput.keyboard. Events. This class supports reading single events in a non-blocking fashion, as well as iterating over all events.

Keyboard controller class functions

class pynput.keyboard.Controller A controller for sending virtual keyboard events to the system.

* exception InvalidCharacterException - The exception raised when an invalid character is encountered in the string passed to Controller.type(). Its first argument is the index of the character in the string, and the second the character.
* exception InvalidKeyException - The exception raised when an invalid key parameter is passed to either Controller.press() or Controller.release(). Its first argument is the key parameter.
* alt\_gr\_pressed - Whether altgr is pressed.
* alt\_pressed - Whether any alt key is pressed.
* ctrl\_pressed - Whether any ctrl key is pressed.
* modifiers - The currently pressed modifier keys. Only the generic modifiers will be set; when pressing either Key.shift\_l, Key.shift\_r or Key. shift, only Key.shift will be present.
* press(key) - Presses a key. A key may be either a string of length 1, one of the Key members or a KeyCode. Strings will be transformed to KeyCode using KeyCode.char(). Members of Key will be translated to their value().

Parameters key – The key to press. Raises

* + InvalidKeyException – if the key is invalid
  + ValueError – if key is a string, but its length is not 1
* pressed(\*args) - Executes a block with some keys pressed. Parameters keys – The keys to keep pressed.
* release(key) - Releases a key. A key may be either a string of length 1, one of the Key members or a KeyCode. Strings will be transformed to KeyCode using KeyCode.char(). Members of Key will be translated to their value().

Parameters key – The key to release. If this is a string, it is passed to touches() and the returned releases are used.

Raises

* + InvalidKeyException – if the key is invalid.
  + ValueError – if key is a string, but its length is not 1.
* shift\_pressed - Whether any shift key is pressed, or caps lock is toggled.
* tap(key) - Presses and releases a key. Parameters key – The key to press. Raises
  + InvalidKeyException – if the key is invalid.
  + ValueError – if key is a string, but its length is not 1.
* touch(key, is\_press) - Calls either press() or release() depending on the value of is\_press.

Parameters

* + key – The key to press or release.
  + is\_press (bool) – Whether to press the key.

Raises InvalidKeyException – if the key is invalid.

* type(string) - Types a string. This method will send all key presses and releases necessary to type all characters in the string.

Parameters string (str) – The string to type.

Raises InvalidCharacterException – if an untypable character is encountered.[17]

## FLOWCHART

Hand tracking model

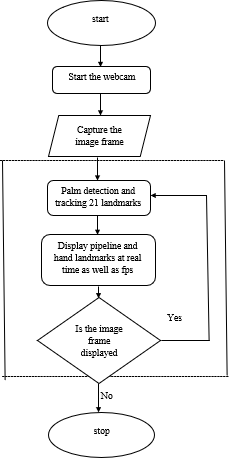


Fig 6.4.1 Hand Tracking Model

* + - In order to start the process we first need to take hand input
    - For capturing of the hand we need to import cvzone. With the help of videocapture we will get access to the camera.
    - After placing the hand in front of camera at a certain distance with the help of mediapipe it detects a maximum of 2 hands from the video, after that 21 landmark projections are displayed on each hand
    - Pipelines are used to join 21 landmarks.
    - If no hand is present in-front of the camera the landmarks won’t be detected.

Main model

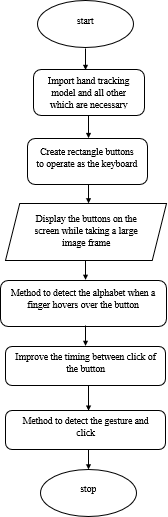


Fig 6.4.2 Main Model

* + - After creating hand landmarks next we need to design our keyboard.
    - We have created rectangle boxes by specifying the height, width, color for our keyboard.
    - After that we have to arrange the alphabets, special characters and remaining keys in the rectangle boxes which we have created earlier.
    - Next we have to import handtracking model. After importing the handtracking model when we place hands in-front of the screen the landmarks will be displayed
    - Whenever we place our finger on a particular key the respective key box colour changes and the size of the key increases
    - When the tip of index finger and middle finger are combined then the particular key will be entered at that time the colour of the particular key box changes to green.

## SOURCE CODE

* + 1. **Handtracking Module**

import cv2

import mediapipe as mp import math

class HandDetector:

def init (self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5): self.mode = mode

self.maxHands = maxHands self.detectionCon = detectionCon self.minTrackCon = minTrackCon self.mpHands = mp.solutions.hands

self.hands = self.mpHands.Hands(self.mode, self.maxHands,

self.detectionCon, self.minTrackCon) self.mpDraw = mp.solutions.drawing\_utils

self.tipIds = [4, 8, 12, 16, 20] self.fingers = []

self.lmList = []

def findHands(self, img, draw=True):

imgRGB = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) self.results = self.hands.process(imgRGB)

if self.results.multi\_hand\_landmarks:

for handLms in self.results.multi\_hand\_landmarks: if draw:

self.mpDraw.draw\_landmarks(img, handLms,

self.mpHands.HAND\_CONNECTIONS)

return img

def findPosition(self, img, handNo=0, draw=True): xList = []

yList = [] bbox = [] bboxInfo =[]

self.lmList = []

if self.results.multi\_hand\_landmarks:

myHand = self.results.multi\_hand\_landmarks[handNo] for id, lm in enumerate(myHand.landmark):

h, w, c = img.shape

px, py = int(lm.x \* w), int(lm.y \* h) xList.append(px)

yList.append(py) self.lmList.append([px, py]) if draw:

cv2.circle(img, (px, py), 5, (255, 0, 255), cv2.FILLED) xmin, xmax = min(xList), max(xList)

ymin, ymax = min(yList), max(yList) boxW, boxH = xmax - xmin, ymax - ymin bbox = xmin, ymin, boxW, boxH

cx, cy = bbox[0] + (bbox[2] // 2), \ bbox[1] + (bbox[3] // 2)

bboxInfo = {"id": id, "bbox": bbox,"center": (cx, cy)} if draw:

cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),

(bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20), (0, 255, 0), 2)

return self.lmList, bboxInfo

def fingersUp(self):

if self.results.multi\_hand\_landmarks: myHandType = self.handType() fingers = []

# Thumb

if myHandType == "Right":

if self.lmList[self.tipIds[0]][0] > self.lmList[self.tipIds[0] - 1][0]: fingers.append(1)

else:

fingers.append(0)

else:

if self.lmList[self.tipIds[0]][0] < self.lmList[self.tipIds[0] - 1][0]: fingers.append(1)

else:

fingers.append(0)

for id in range(1, 5):

if self.lmList[self.tipIds[id]][1] < self.lmList[self.tipIds[id] - 2][1]: fingers.append(1)

else:

fingers.append(0) return fingers

def findDistance(self, p1, p2, img, draw=True): if self.results.multi\_hand\_landmarks:

x1, y1 = self.lmList[p1][0], self.lmList[p1][1] x2, y2 = self.lmList[p2][0], self.lmList[p2][1] cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

if draw:

cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)

cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)

cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)

cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

length = math.hypot(x2 - x1, y2 - y1) return length, img, [x1, y1, x2, y2, cx, cy]

def handType(self):

if self.results.multi\_hand\_landmarks:

if self.lmList[17][0] < self.lmList[5][0]: return "Right"

else:

return "Left"

def main():

cap = cv2.VideoCapture(0)

detector = HandDetector(detectionCon=0.8, maxHands=1) while True:

# Get image frame success, img = cap.read()

# Find the hand and its landmarks img = detector.findHands(img)

lmList, bboxInfo = detector.findPosition(img) print(detector.handType()) cv2.imshow("Image", img)

cv2.waitKey(1)

if name == " main ": main()

## Main Module

import cv2

from cvzone.HandTrackingModule import HandDetector from time import sleep

import numpy as np import cvzone

from pynput.keyboard import Controller cap = cv2.VideoCapture(0)

cap.set(3, 1280)

cap.set(4, 720)

detector = HandDetector(detectionCon=0.5)

keys = [["1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "\_", "="],

["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "[", "]"],

["A", "S", "D", "F", "G", "H", "J", "K", "L", ";", "'", ":"],

["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/", "\"", "|"],

["!", "@", "#", "$", "%", "^", "&", "\*", "(", ")", "\\", "`"],

["~", "<", ">", "?", "-", "+", "\*", "{", "}", "?", " "]]

finalText = ""

keyboard = Controller()

def drawAll(img, buttonList): for button in buttonList:

x, y = button.pos

w, h = button.size

cvzone.cornerRect(img, (button.pos[0], button.pos[1], button.size[0], button.size[1]), 20,

rt=0)

cv2.rectangle(img, button.pos, (x + w, y + h), (255, 0, 255), cv2.FILLED)

cv2.putText(img, button.text, (x + 10, y + 25), cv2.FONT\_HERSHEY\_PLAIN, 2, (255,

255, 255), 2)

return img class Button():

def init (self, pos, text, size=[40, 40]): self.pos = pos

self.size = size self.text = text

buttonList = []

for i in range(len(keys)):

for j, key in enumerate(keys[i]): buttonList.append(Button([55 \* j , 60 \* i + 20 ], key))

while True:

success, img = cap.read()

Vimg = detector.findHands(img)

lmList, bboxInfo = detector.findPosition(img) img = drawAll(img, buttonList)

if lmList:

for button in buttonList: x, y = button.pos

w, h = button.size

if x < lmList[8][0] < x + w and y < lmList[8][1] < y + h:

cv2.rectangle(img, (x - 2, y - 2), (x + w + 2, y + h + 2), (175, 0, 175), cv2.FILLED) cv2.putText(img, button.text, (x+10 , y+30 ), cv2.FONT\_HERSHEY\_PLAIN, 3,

(255, 255, 255), 3)

l, \_, \_ = detector.findDistance(8, 12, img, draw=False) print(l)

## when clicked if l < 30:

keyboard.press(button.text)

cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 0), cv2.FILLED) cv2.putText(img, button.text, (x+10 , y+30 ), cv2.FONT\_HERSHEY\_PLAIN, 3,

(255, 255, 255), 3)

finalText += button.text sleep(0.2)

cv2.rectangle(img, (50, 400), (700, 550), (175, 0, 175), cv2.FILLED)

cv2.putText(img, finalText, (60, 450), cv2.FONT\_HERSHEY\_PLAIN, 3, (255, 255, 255),

3)

cv2.imshow("Image", img) cv2.waitKey(1)

## OUTPUT SCREENS

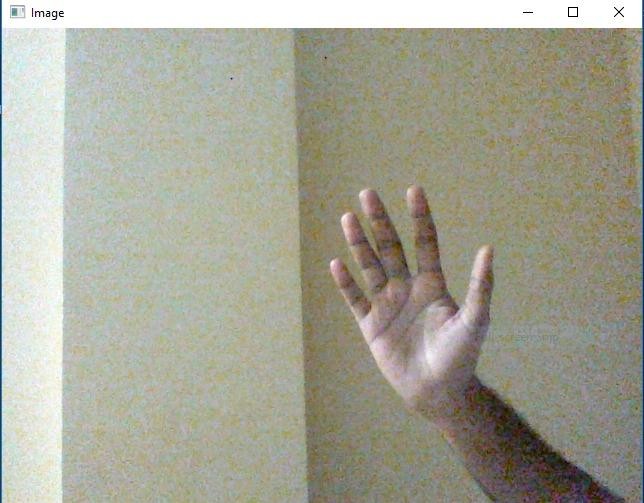
****

Fig 6.6.1 Hand with no landmarks

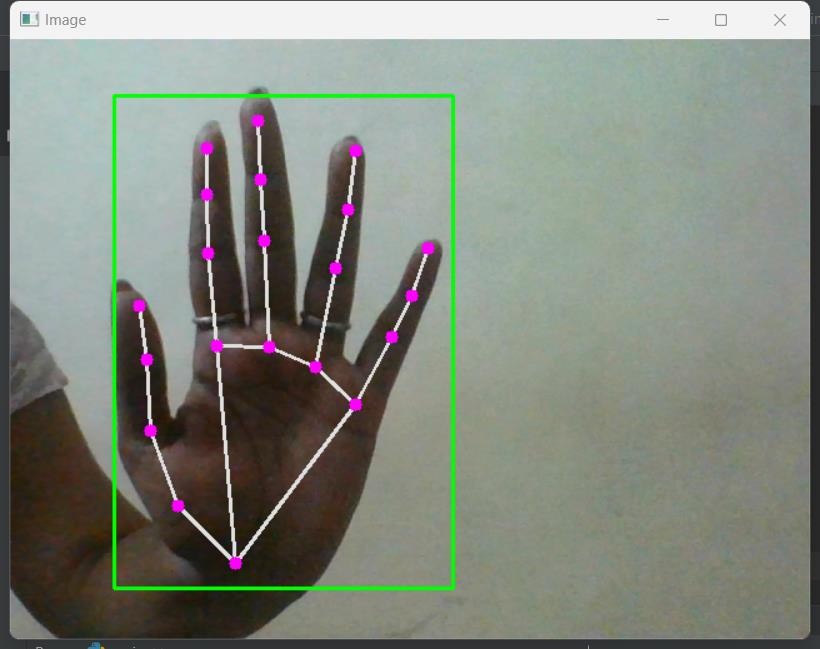


Fig 6.6.2 Hand with landmarks

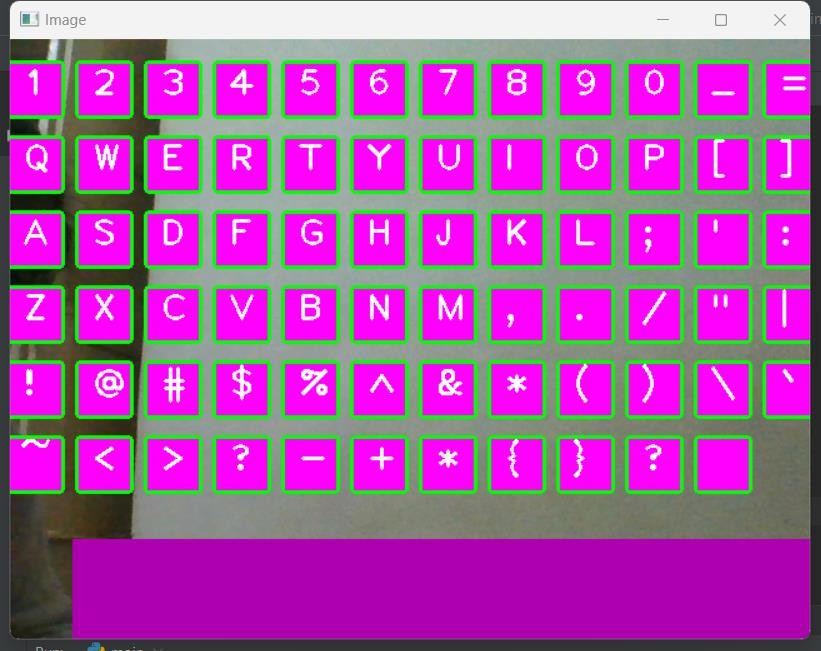


Fig 6.6.3 Keyboard

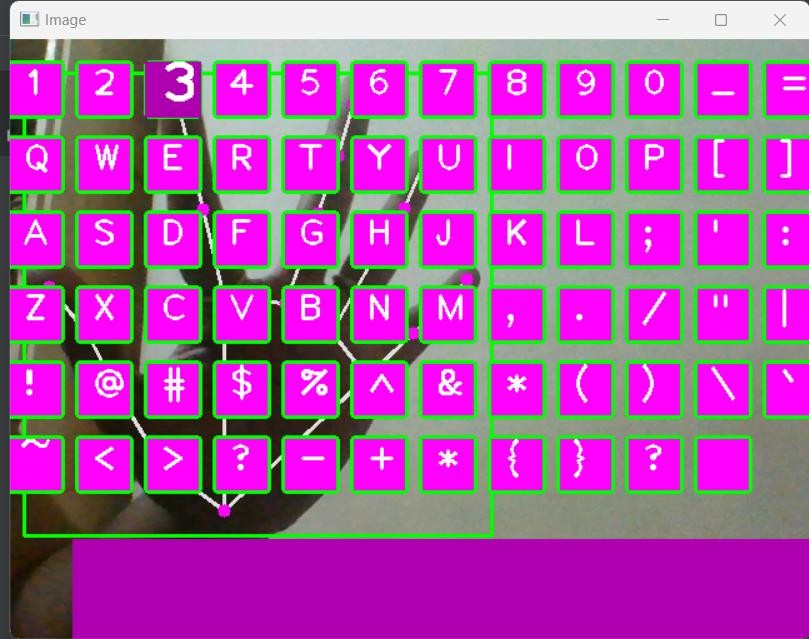


Fig 6.6.4 Key hovering when finger placed on it

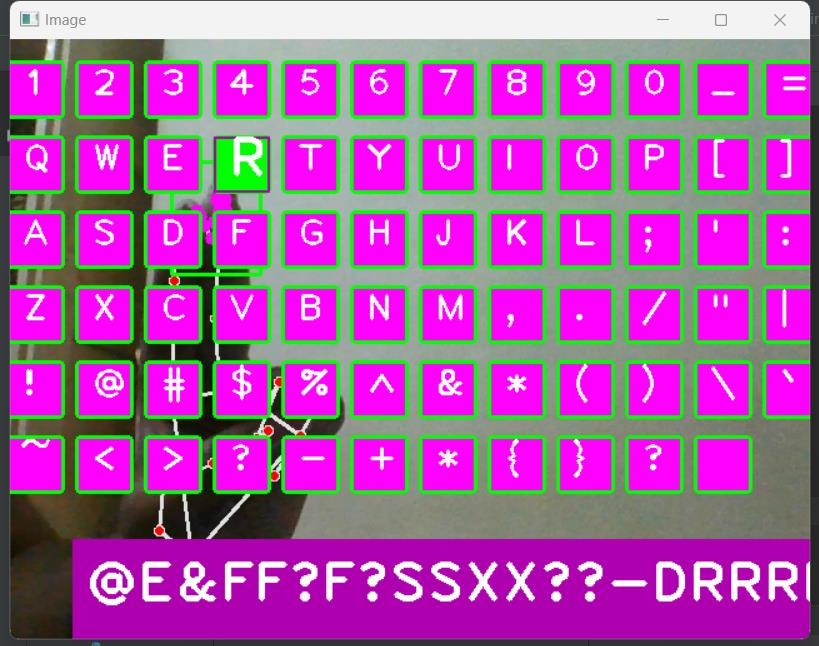


Fig 6.6.5 Keyboard when key pressed

## SOFTWARE TESTING

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented to 'detection'.

## Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. This testing mode is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Unit tests are written from a programmer's perspective. They ensure that a particular method of a class successfully performs a set of specific tasks. Each test confirms that a method produces the expected output when given a known input.

## Performance Testing

Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as Reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing.

Performance testing can verify that a system meets the specifications claimed by its manufacturer or vendor. The process can compare two or more devices or programs in terms of parameters such as speed, data transfer rate, bandwidth, throughput, efficiency or reliability.

Performance testing can also be used as a diagnostic aid in locating communications bottlenecks. Often a system will work much better if a problem is resolved at a single point or in a single component.

## Integration Testing

Integration testing, also known as integration and testing (I&T), is a software development process which program units are combined and tested as groups in multiple ways. In this context, a unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution. Integration testing is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

## Test Cases

Test case 1

|  |  |  |
| --- | --- | --- |
| Test no | Tester | Mismatched keys |
| 1 | A | Z |
| 2 | B | G |
| 3 | D | T |
| 4 | K | A |
| 5 | S | W |
| 6 | V | Y |

Table 6.7.4.1 Test Case 1

Test case 2

|  |  |  |
| --- | --- | --- |
| Test no | Tester | Mismatched keys |
| 1 | A | N/A |
| 2 | B | N/A |
| 3 | D | N/A |
| 4 | 6 | N/A |
| 5 | $ | N/A |
| 6 | \* | N/A |

Table 6.7.4.2 Test Case 2

* 1. **CONCLUSION**

# CONCLUSION

The hand gesture based keyboard was developed in the python language, using the OpenCV library. The system was able to control the keyboard by tracking the user hands with the help of real time camera. The keyboard is simulated as an actual keyboard as well as the alphabets are well aligned. The result includes key press and key functionalities. The system designed can be worked in all conditions just requirement is a bright light and hand as well as a webcam for fast detection and better result. The system has the potential of being a viable replacement for the computer keyboard. The system efficiency will help us in saving space at work, reduce plastic waste by eliminating the keyboard.

## FUTURE SCOPE

System is quite autonomous. It’s one of the best features is that it is easy to use. Future work will be focused on algorithm improvement by merging the models created and making an entire virtual system handler such that it has functionalities of virtual mouse and keyboard as well as it can control the volume, brightness and other functionalities. It also includes improvement in keyboard. We plan to add a caps-lock button. Also, a button that switches the keyboard characters to special characters so that they can be used when they are needed. It can even be implemented in gaming industry.

# REFERENCES

1. Liangchen Songa , Gang Yub , Junsong Yuana , Zicheng Liuc “Human Pose Estimation and Its Application to Action Recognition: A Survey”, (ResearchGate) 2021.
2. https://www.v7labs.com/blog/human-pose-estimation-guide#h4

# Shradha Dubey and Manish Dixit “A comprehensive survey on human pose estimation approaches” (Springer Link) 2022.

1. https://learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/
2. https://learnopencv.com/yolov7-pose-vs-mediapipe-in-human-pose-estimation
3. https://www.v7labs.com/blog/human-pose-estimation-guide
4. Ashok, M. A., Revathi, M. P., & Nirmal, B. “FINGER RECONGITION AND GESTURE BASED VIRTUAL KEYBOARD.” Turkish Journal of Computer and Mathematics Education (TURCOMAT), 11(3), 2237-2245, in 2020
5. Rishikesh Kumar, Poonam Chaudhary,” User defined custom virtual keyboard”, in IEEE 2017
6. Pravin Kshirsagar, “ Gesture based motion tracking keyboard using machine learning”, in 2020
7. <https://www.geeksforgeeks.org/python-play-a-video-using-opencv/>
8. https://developers.google.com/mediapipe/solutions/vision/hand\_landmarker
9. <https://www.arxiv-vanity.com/papers/2006.10214/>
10. <https://intellipaat.com/blog/what-is-pycharm/>
11. <https://www.jetbrains.com/pycharm/>
12. Open CV [Online]. Available: <https://opencv.org/>
13. <https://realpython.com/python-time>module
14. <https://pynput.readthedocs.io/en/latest/>
15. <https://buildmedia.readthedocs.org/media/pdf/pynput/latest/pynput.pdf>