

A

*Mini Project Report*

On the topic

**“Hand gesture DETECTOR”**

[PYTHON BASED]

**BACHELOR OF TECHNOLOGY**

**(COMPUTER SCIENCE AND ENGINEERING)**

***SUBMITTED BY***

***RAHUL SINGH* (ROLL NO.19243)**

***PRASHANT KUMAR*** **(ROLL NO. 19239)**

***TUSHAR SHARMA*** **(ROLL NO.19259)**

***DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING***

***KAMLA NEHRU INSTITUTE OF TECHNOLOGY (KNIT),***

***SULTANPUR, UTTAR PRADESH***

**ACKNOWLEDGEMENTS**

We extend our sincere thanks to the community of ***GOOGLE*** who continuously helped us throughout the project and without their guidance, this project would have been an uphill task.

We are also grateful to our ***colleauges*** who co-operated with us regarding some issues.

Last but not the least, smooth development of this project.

January 2022

Rahul Singh (19243)

Prashant kumar (19239)

Tushar Sharma (19259)

**Table of Contents**

**CHAPTER 1**

**INTRODUCTION**

**1. Overview**

**2. Background and Motivation**

**3. Objective**

**CHAPTER 2**

**HAND GESTURE DESCRIPTION**

## Models

### Palm Detection Model

### Hand Landmark Model

**Need of ML Pipeline**

**CHAPTER 3**

**CODE EXPLANATION**

Libraries:

Module:

**References**

**CHAPTER**

**1**

**Introduction**

Overview

*The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and hand shakes) and lack high contrast patterns.*

*MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenues.*

Background and Motivation

This project will mainly involve the basics of Python programming database management, algorithm construction and data processing. We are very excited to learn these topics.

Through the development of Hand Gesture based on Python Programming, we will get a clear understanding of overall process of the system. The core part of the music player is mainly composed of main interface, playlists, menus, play Settings, file browsing and song search.

Grasping the development of the six parts, the music player has had the preliminary scale.

Based on the function of the six categories, add some other small features.

Music player system realized the basic function of player: play, pause, and stop, up/down a, volume adjustment, lyrics display, play mode, song search, file browser, playlists query, and other functions

Objective

The final goal of the project was to build a Music Player based on python and it is user friendly with having lots of features. We will see in upcoming chapter.

There are format which are support by our Music Player

* MP3(MPEG Audio Layer-3)
* OGG(**ogging, jargon from the computer game Netrek)**
* WAV (Waveform Audio File Format )

**CHAPTER**

**2**

**Hand Gesture Description**

### *Palm Detection Model*

To detect initial hand locations, we designed a [single-shot detector](https://arxiv.org/abs/1512.02325) model optimized for mobile real-time uses in a manner similar to the face detection model in [MediaPipe Face Mesh](https://google.github.io/mediapipe/solutions/face_mesh.html). Detecting hands is a decidedly complex task: our [lite model](https://github.com/google/mediapipe/tree/master/mediapipe/modules/palm_detection/palm_detection_lite.tflite) and [full model](https://github.com/google/mediapipe/tree/master/mediapipe/modules/palm_detection/palm_detection_full.tflite) have to work across a variety of hand sizes with a large scale span (~20x) relative to the image frame and be able to detect occluded and self-occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

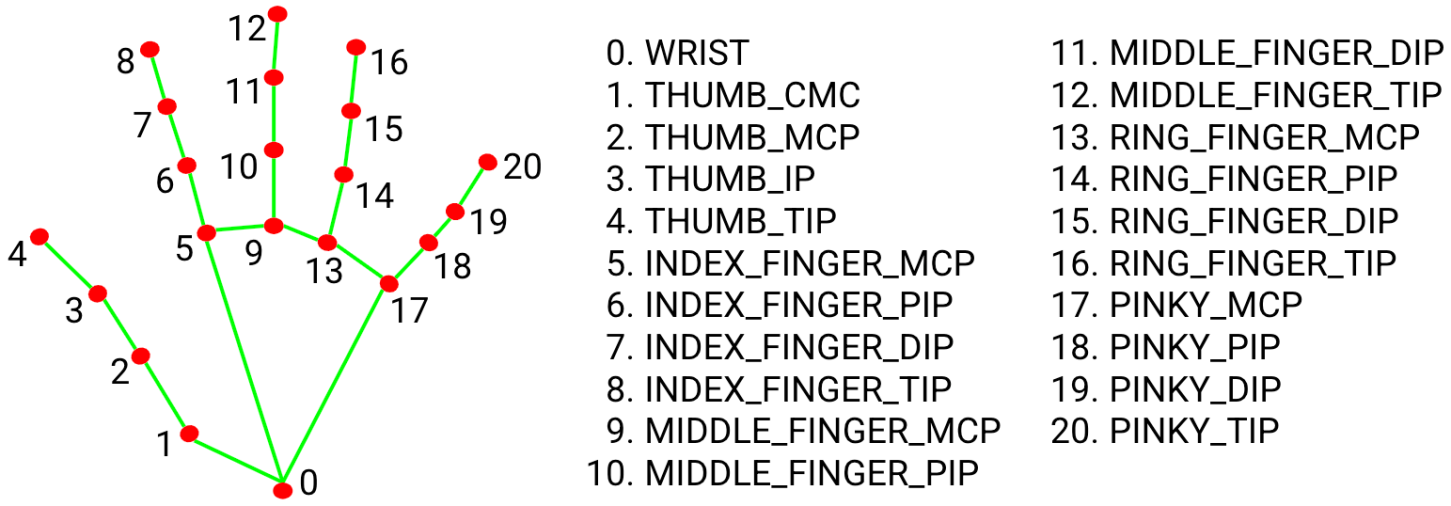
Our method addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance.

With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

### *Hand Landmark Model*

After the palm detection over the whole image our subsequent hand landmark [model](https://github.com/google/mediapipe/tree/master/mediapipe/modules/hand_landmark/hand_landmark_full.tflite) performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

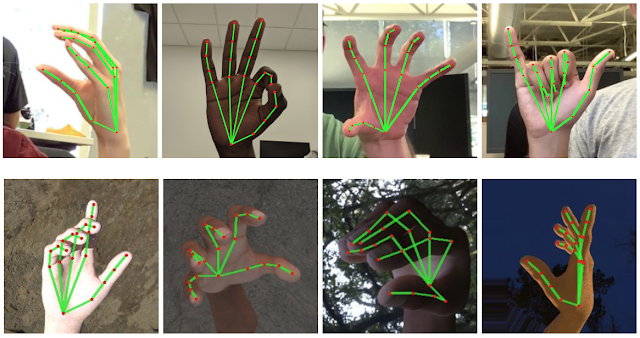


***Need of ML Pipeline***

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints. This strategy is similar to that employed in our [MediaPipe Face Mesh](https://google.github.io/mediapipe/solutions/face_mesh.html) solution, which uses a face detector together with a face landmark model.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in our pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

The pipeline is implemented as a MediaPipe [graph](https://github.com/google/mediapipe/tree/master/mediapipe/graphs/hand_tracking/hand_tracking_mobile.pbtxt) that uses a [hand landmark tracking subgraph](https://github.com/google/mediapipe/tree/master/mediapipe/modules/hand_landmark/hand_landmark_tracking_gpu.pbtxt) from the [hand landmark module](https://github.com/google/mediapipe/tree/master/mediapipe/modules/hand_landmark), and renders using a dedicated [hand renderer subgraph](https://github.com/google/mediapipe/tree/master/mediapipe/graphs/hand_tracking/subgraphs/hand_renderer_gpu.pbtxt). The [hand landmark tracking subgraph](https://github.com/google/mediapipe/tree/master/mediapipe/modules/hand_landmark/hand_landmark_tracking_gpu.pbtxt) internally uses a [hand landmark subgraph](https://github.com/google/mediapipe/tree/master/mediapipe/modules/hand_landmark/hand_landmark_gpu.pbtxt) from the same module and a [palm detection subgraph](https://github.com/google/mediapipe/tree/master/mediapipe/modules/palm_detection/palm_detection_gpu.pbtxt) from the [palm detection module](https://github.com/google/mediapipe/tree/master/mediapipe/modules/palm_detection).



**CHAPTER**

**3**

**Code Explanation**

Libraries:

In our daily life, we see every person has a hobby and that is listening to music. So in order to listen to music, they all need a Music player(hardware or software) where they can play their favorite songs. And we have to install this music player on our computer, based the Operating system i.e Windows, Macintosh, Android, Linux, etc. Then we can listen to our favorite songs.In our daily life, we see every person has a hobby and that is listening to music. So in order to listen to music, they all need a Music player(hardware or software) where they can play their favorite songs. And we have to install this music player on our computer, based the Operating system i.e Windows, Macintosh, Android, Linux, etc. Then we can listen to our favorite songs.

**Libraries used for Music Player Application:**

Now we will tell you about the Libraries we will use in our code:

**1. OpenCv**

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

import cv2 \*

### ****2. Mediapipe module****

MediaPipe is a an open-source framework from Google for building multimodal (eg. video, audio, any time series data), cross platform (i.e Android, iOS, web, edge devices) applied ML pipelines. It is performance optimized with end-to-end ondevice inference in mind. Mediapipe is currently under active development and includes multiple demos, that can be run out-of-the box after installing Mediapipe on reTerminal.

pip install Mediapipe

To use this module into your code you need to write this:

import mediapipe

**3. Pyautogui module**

PyAutoGUI lets your Python scripts control the mouse and keyboard to automate interactions with other applications. The API is designed to be simple. PyAutoGUI works on Windows, macOS, and Linux, and runs on Python 2 and 3.

To use this module in your code you need to import it’s and command for the same is as follows:

import pyautogui

**1. The MouseMovement() Function**

This function take the value of finger tips ,and according to this finger tip it set the value of mouse pointer Now we will define the Mouse Movement Function and the code is:

def mouseMovement(a=0,b=0):  
 global sizex,sizey  
 try:  
 x, y = pyautogui.position()  
 if a>0:  
 newx = x+10  
 elif a<0:  
 newx = x-10  
 if b>0:  
 newy = y-10  
 elif b<0:  
 newy = y+10  
  
 if(newx>sizex):  
 newx=sizex  
 if(newy>sizey):  
 newy=sizey  
  
 pyautogui.moveTo(newx, newy)

**4. Code for using imported file easily in our system**

cap=cv2.VideoCapture(0)  
mpHands=mp.solutions.hands  
hands=mpHands.Hands()  
mpDraw=mp.solutions.drawing\_utils

1. **Time**

Python has a module named time to handle time-related tasks. Here time function use to show movement and capturing capacity of hand. To use functions defined in the module, we need to import the module first. Here's how:

import time

Def\_Btn = tk.Button**(**app,text='Default Button'**)**

Def\_Btn.pack**()**

Themed\_Btn = ttk.Button**(**app,text='Themed button'**)**

Themed\_Btn.pack**()**

# Scrollbar Widgets

Def\_Scrollbar = tk.Scrollbar**(**app**)**

Def\_Scrollbar.pack**(**side='right',fill='y'**)**

Themed\_Scrollbar = ttk.Scrollbar**(**app,orient='horizontal'**)**

Themed\_Scrollbar.pack**(**side='top',fill='x'**)**

# Entry Widgets

Def\_Entry = tk.Entry**(**app**)**

Def\_Entry.pack**()**

Themed\_Entry = ttk.Entry**(**app**)**

Themed\_Entry.pack**()**

app.mainloop**()**

We can apply different themes whatever we want or whatever we like to our Music Player GUI interface.

***References***

1. <https://pyautogui.readthedocs.io/en/latest/mouse.html>

2. <https://google.github.io/mediapipe/solutions/hands>

3. <https://www.geeksforgeeks.org/opencv-python-tutorial/>

4. <https://www.youtube.com/c/MurtazasWorkshopRoboticsandAI>

**THE END**