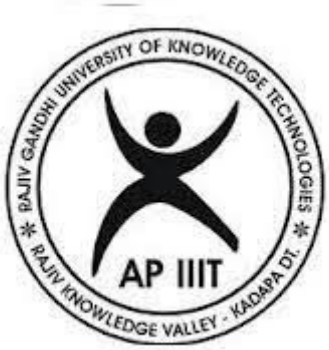
AUTOMATIC STOP GESTURE DETECTION FOR VEHICLE

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING



RGUKT
Rajiv Gandhi University of Knowledge Technologies
R.K.VALLEY

Submitted by
R161638 : M Raviteja
R161639 : M Vinay
R161641 : K Kiran Kumar

Under the Esteemed guidance of
Mrs. Ratna Kumari Ch

DECLARATION

We hereby declare that the report of the B.Tech Major Project Work entitled "**AUTOMATIC STOP GESTURE DETECTION FOR VEHICLE**" which is being submitted to Rajiv Gandhi University of Knowledge Technologies, RK Valley, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering, is a bonafide report of the work carried out by us. The material contained in this report has not been submitted to any university or institution for award of any degree.

M Raviteja : R161638
M Vinay : R161639
K Kiran Kumar : R161641
Dept. Of Computer Science and Engineering.

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

RGUKT

(A.P.Government Act 18 of 2008)
IIIT RK VALLEY, RGUKT-AP
Department of Computer Science and Engineering

CERTIFICATE FOR PROJECT COMPLETION

This is certify that the project entitled "**AUTOMATIC STOP GESTURE DETECTION FOR VEHICLE**" submitted by M.Raviteja(R161638), M.Vinay(R161639) and K.Kiran Kumar(R161641) under our guidance and supervision for the partial fulfillment for the degree Bachelor of Technology in Computer Science and Engineering during the academic year 2021-2022 at IIIT ,RK VALLEY RGUKT-AP.To the best of my knowledge, the result embodied in this dissertation work have not been submitted to any University or Institute for the award of any degree or diploma.

Project Internal Guide
Mrs.Ratna Kumari Ch
Assistant Professor
IIIT,RGUKT-AP,RK Valley

Head of the Department
Mrs.Ratna Kumari Ch
Assistant Professor
HOD of CSE
IIIT,RGUKT-AP,RK Valley

Table of Contents

ABSTRACT

1. INTRODUCTION
    - 1.1. Project Aim and Object
    - 1.2. Motivation
    - 1.3. Operational Environment

2. SYSTEM REQUIREMENTS
    - 2.1. Non Functional Requirements
    - 2.2. Functional Requirements
    - 2.3. Hardware Information
    - 2.4. Software Information

3. SYSTEM DESIGN
    - 3.1. UML DIAGRAMS
        - 3.1.1. Data Flow Diagram
        - 3.1.2. Class Diagram
        - 3.1.3. Use Case Diagram
        - 3.1.4. Activity Diagram
        - 3.1.5. State Diagram
        - 3.1.6. Object Diagram
        - 3.1.7. Sequence Diagram

4. SYSTEM IMPLEMENTATION
    - 4.1. Preliminaries
    - 4.2. Proposed System
        - 4.2.1. Video Capturing
        - 4.2.2. Analyzing each Frame
        - 4.2.3. Detecting human
        - 4.2.4. Pose estimation
        - 4.2.5. Sending Alert Message

5. CODE IMPLEMENTATION

6. CONCLUSION

7. REFERENCES

ABSTRACT

Today the technology grows very rapidly and techies find new ways to make things easier and better. One such technology is called a Human Posture estimation(HPE)[4] to help to describe the orientation or movements of a person. Generally they are used in gaming, health care, AR etc. The amount of HPE applications is exponentially increasing. Implementing relevant models based on the application becomes Interesting and an enormous challenge. Implementing HPE models have emerged to conduct effective applications which is related Vehicle stops based on the human posture.By this project we propose a quick and intuitive look to send alert messages to drivers that helps to stop the vehicle when a passenger is waiting for the vehicle.

1. INTRODUCTION

1.1. Project Aim and Objectives:

Stop Gesture Detection is a model in which we have to estimate the human model and send an alert based on the passenger posture This Stop Gesture Detection model is used to estimate the passenger posture to stop the vehicle. Human Pose Estimation is a way of extracting the pose of a human, usually in the form of a skeleton, from a given live video stream.

A Human Pose Estimation[4] is one of the top applications of Computer vision. It's easy to tell whether a person is squatting down or waving their hand. But for machines? Not so simple. When shown a photo or a video frame, a computer sees collection of pixels. Human pose estimation technology enables computers to model human bodies and recognize postures in an image or video, including real time footage. Recognizing human postures and movements has long been in focus for major industries, including sports, retail, entertainment, surveillance,and robot training etc.

The purpose of the project is to make a model that estimates human posture and help to send an alert to the driver to stop the vehicle. Model analyzes the real time capturing video from cameras, identifies a human posture, extracts the skeleton like structures and estimates whether the skeleton like structure matches with estimated structure or not. If it matches it says “YES” otherwise it says “NO”.

STOP GESTURE DETECTION provides passengers with effortless traveling via estimating the passenger posture. Computer vision tools will help to train the model to estimate the human posture in real time footage. To detect these postures we use python libraries like media pipe[3], to calculate the computational values we use numpy[2]. Cv2 to analyze the frames. By analyzing the frames and using computational values it detects the human posture.

1.2. Motivation :

Part of broader artificial intelligence and computer vision realms, Human pose Estimation(HPE)[4] technology has been gradually making its presence seen in all kinds of software apps and hardware solutions. Still, HPE seemed to be stuck at the edge, failing to cross into mainstream adoption. For vehicle drivers it's hard to focus on the road as well as the passengers besides the road who are waiting for the vehicle to stop. In most of the cases drivers can't observe the area which is outside of the roadside. Sometimes it may lead to road accidents.so we came up with solution that is “Automatic Stop Gesture Detection for Vehicle”.

The Importance of Pose Estimation

In traditional object detection , people are only perceived as a bounding box (a square). By performing pose detection and pose tracking, computers can develop an understanding of human body language. However, conventional pose tracking methods are neither fast enough nor robust enough for occlusions to be viable.

Such rapid growth in demand has marked the time for human pose estimation technology to step into widespread commercial use. Due to its wide variety of applications we thought Human pose estimation will resolve the problem that “For vehicle drivers it's hard to focus on the road as well as the passengers besides the road who are waiting for the vehicle to stop. In most of the cases drivers can't observe the area which is outside of the roadside. Sometimes it may lead to road accidents”.

1.3. Operational Environment:

PROCESSOR : AMD Processor
OPERATING SYSTEM : UBUNTU 18.04
RAM : MINIMUM 4GB
PROCESSING SPEED : 2.30GHz

2. SYSTEM REQUIREMENTS

2.1. Non Functional Requirements:

Accuracy

The model is designed with a high accuracy automated process hence it estimates the human posture with high accurate results.

Computation

The model is more reliable because it supports solving high mathematical computations to extract the model.

Speed

This model is developed in high level technologies. It will respond to the end user on the edge system with accurate results.

Performance

The model is designed to perform highly. The model is supported on a wide range of hardware and on most software platforms.

Supportability

The model supports high contrast areas. At that point it will give more accurate results to the system.

2.2. Functional Requirements:

1. Capturing video using web cam
2. Analyzing each frame
3. Deleting human
4. Extracting skeleton

2.3. Software Requirements

For developing this Automatic Stop Gesture Detection For Vehicle the following are the Software Requirements.

Operating System : Windows/Linux Distributions/Mac
Programming Language : Python3 libraries.
Storage : System storage

2.4. Hardware Requirements

For developing this system following are the Hardware Requirements.

Processor : Intel i3 or amd 64(above)
Hard Disk : 500GB
RAM : minimum 4GB or above
Processing speed : 2.4GHz
Camera : 640X480

2.5. Programming Languages and Libraries/Packages Used

1. Python3
2. Numpy
3. MediaPipe
4. cv2 Framework

3. SYSTEM DESIGN

This chapter mainly deals with the designing of the modules which are going to develop in the further chapter.It comprises DataFlow Diagrams(DFD) and Unified Modelling Language (UML)diagrams which mainly constitutes System Design[7].

3.1. UML DIAGRAMS

A UML diagram is a diagram based on the UML(Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter , maintain, or document information about the system. Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include.

3.1.1. Data Flow Diagrams:

A data-flow diagram is a way of representing a flow of data through a process or a system(usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. It is also called a bubble chart .

Level 1:

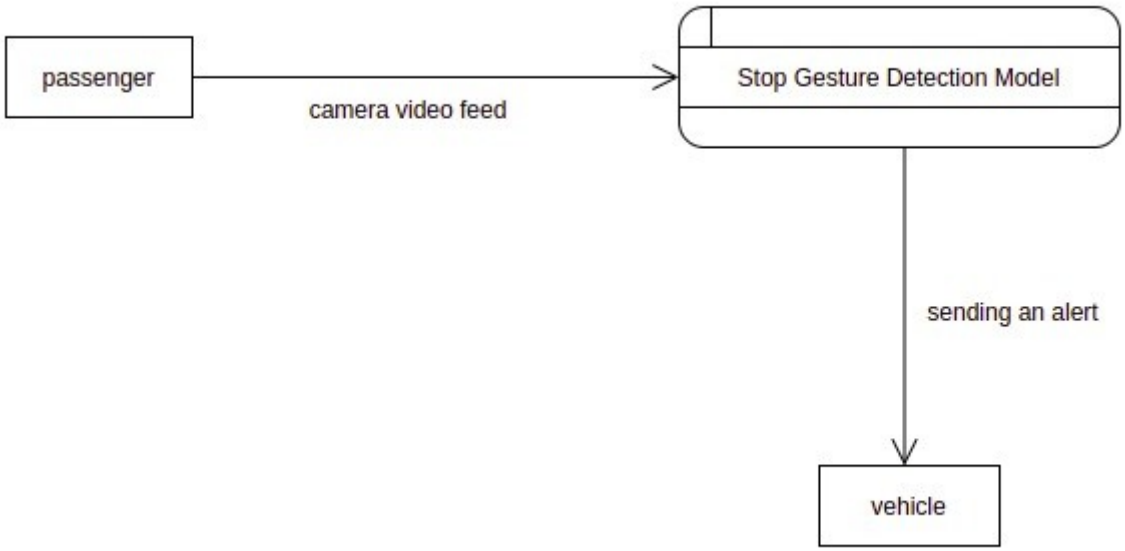


Fig: Data Flow Diagram Level-1

The above Fig explains what functionalities or modules are included in the DFD Level 1.
This abstract level design is concentrated on the main process, that is the Human pose estimation.

Level – 2

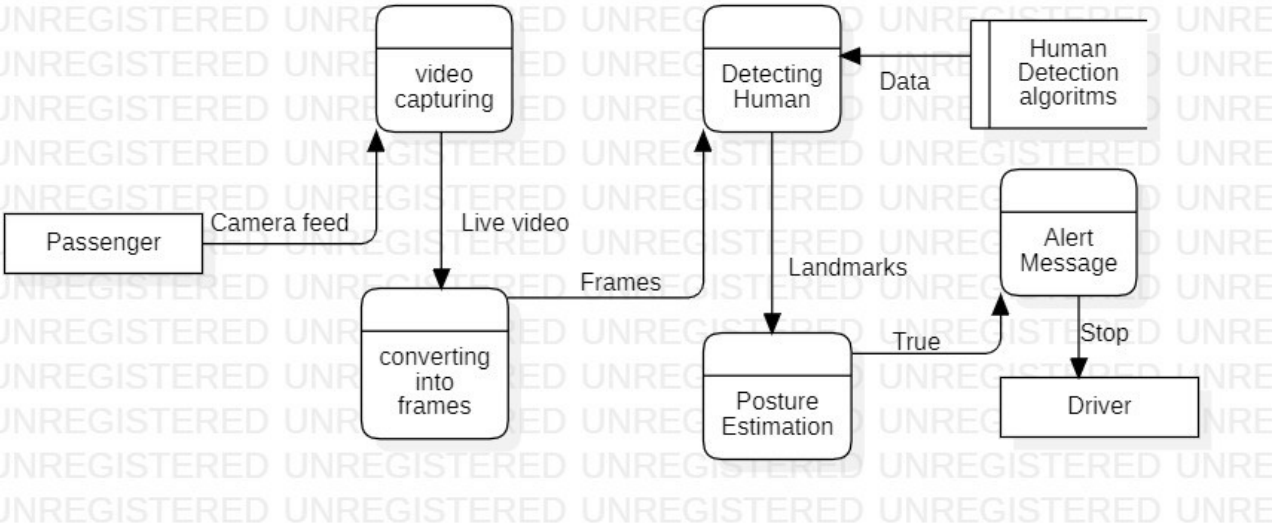


Fig: DFD level 2

3.1.2. Class Diagram:

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

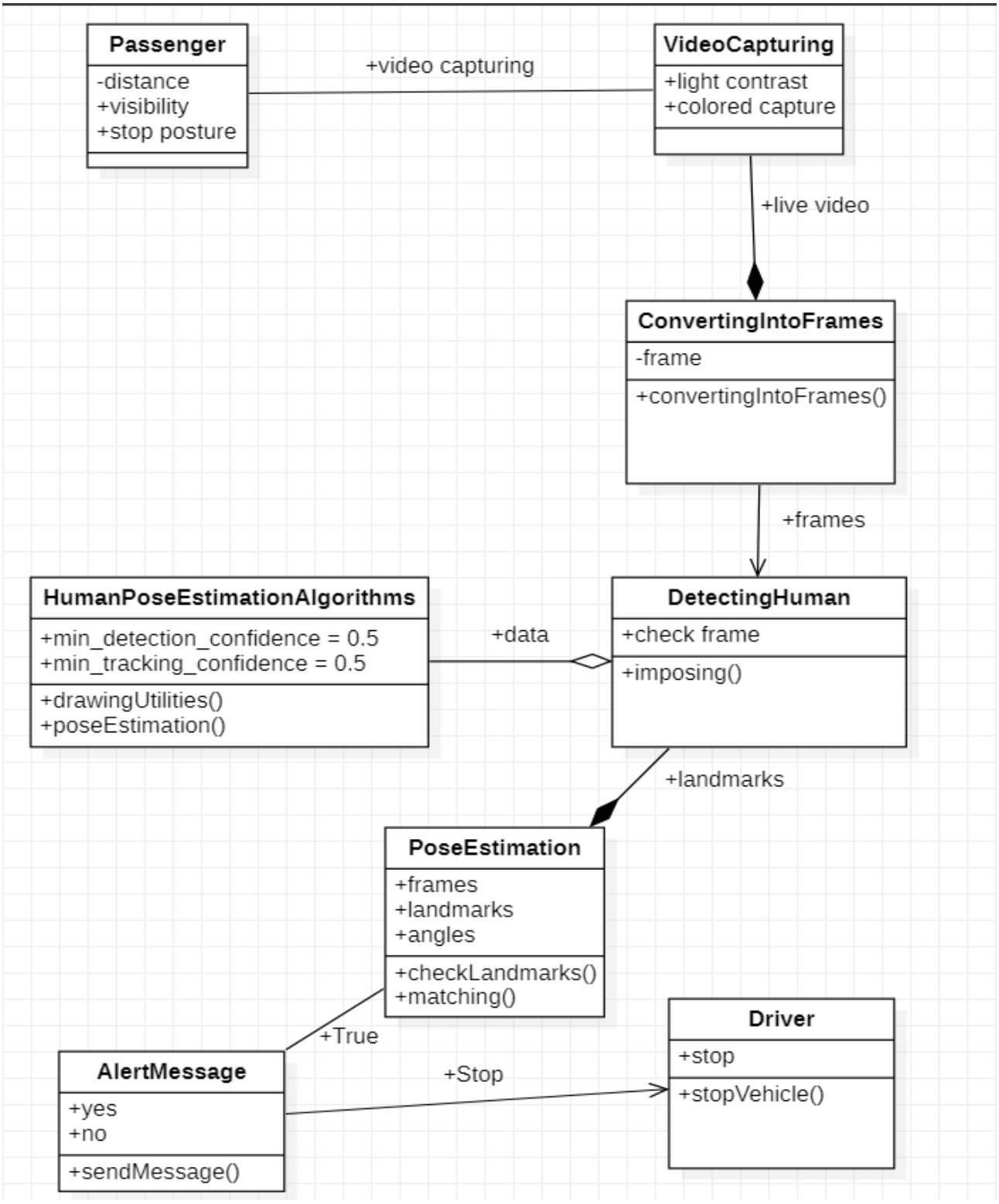


Fig: class Diagram

3.1.3. Use case Diagram:

A use case diagram at its simplest is a representation of user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other diagrams as well. The use cases are represented by either circles or ellipses.

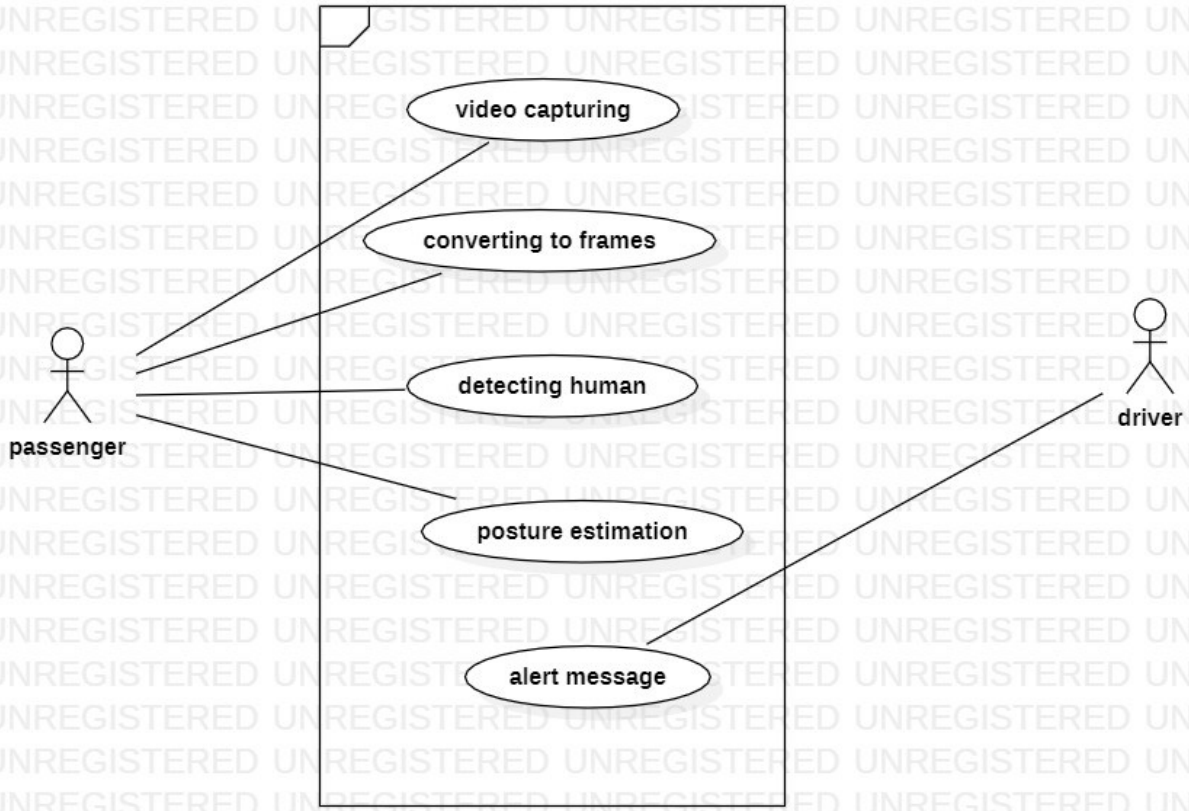


Fig Use case Diagram

3.1.4. Activity diagram:

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join etc.

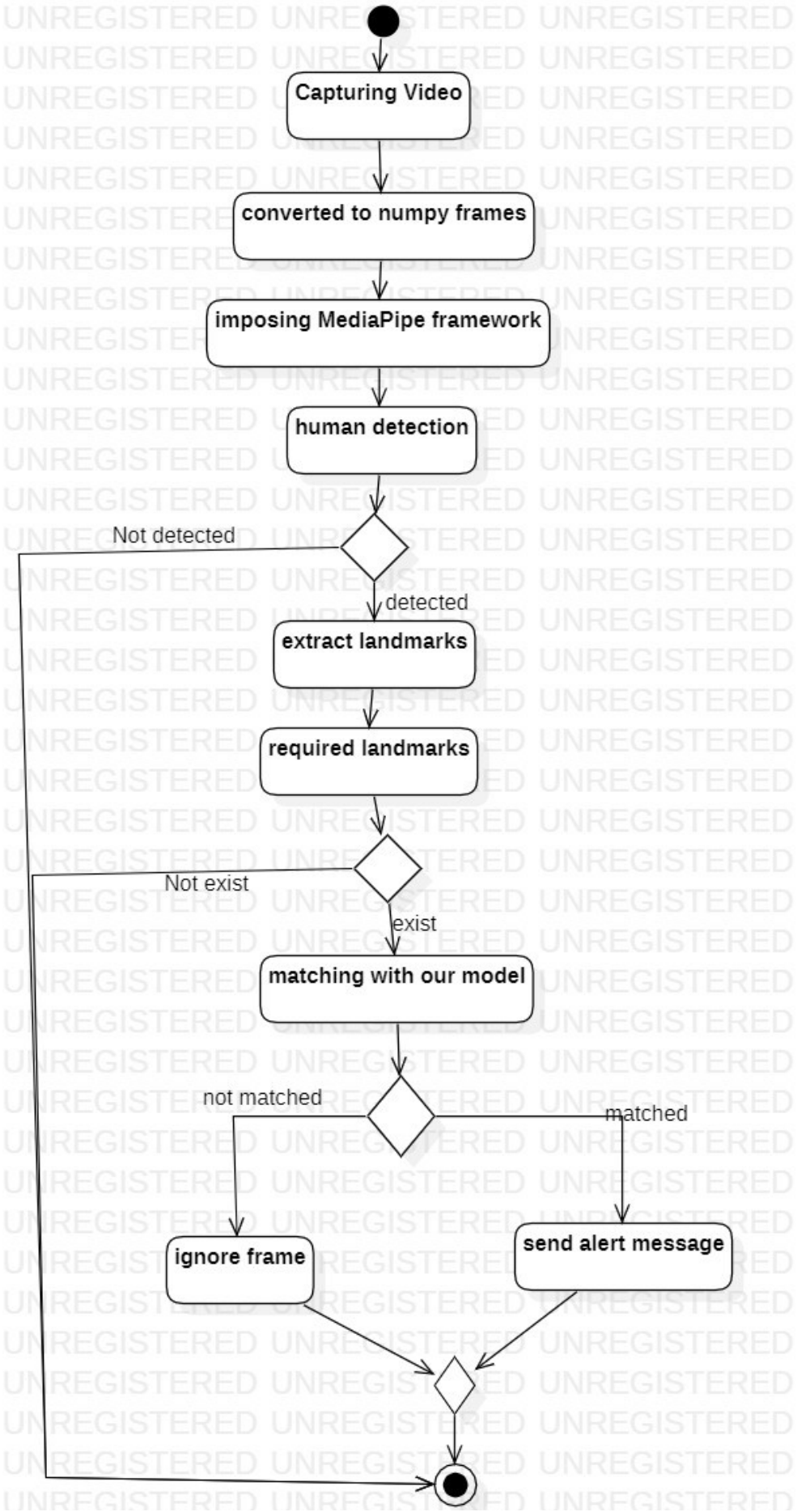


Fig: Activity diagram

3.1.5. State chart Diagram:

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of a State chart diagram is to model the lifetime of an object from creation to termination.

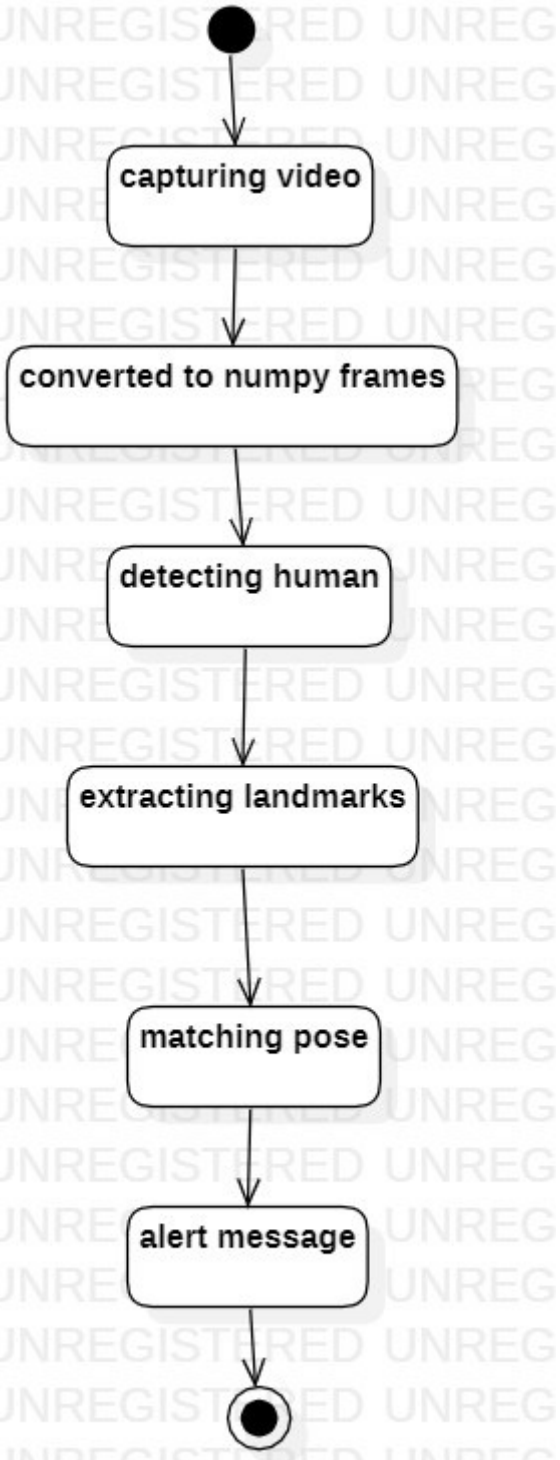


Fig: State Chart Diagram

3.1.6. Object diagram:

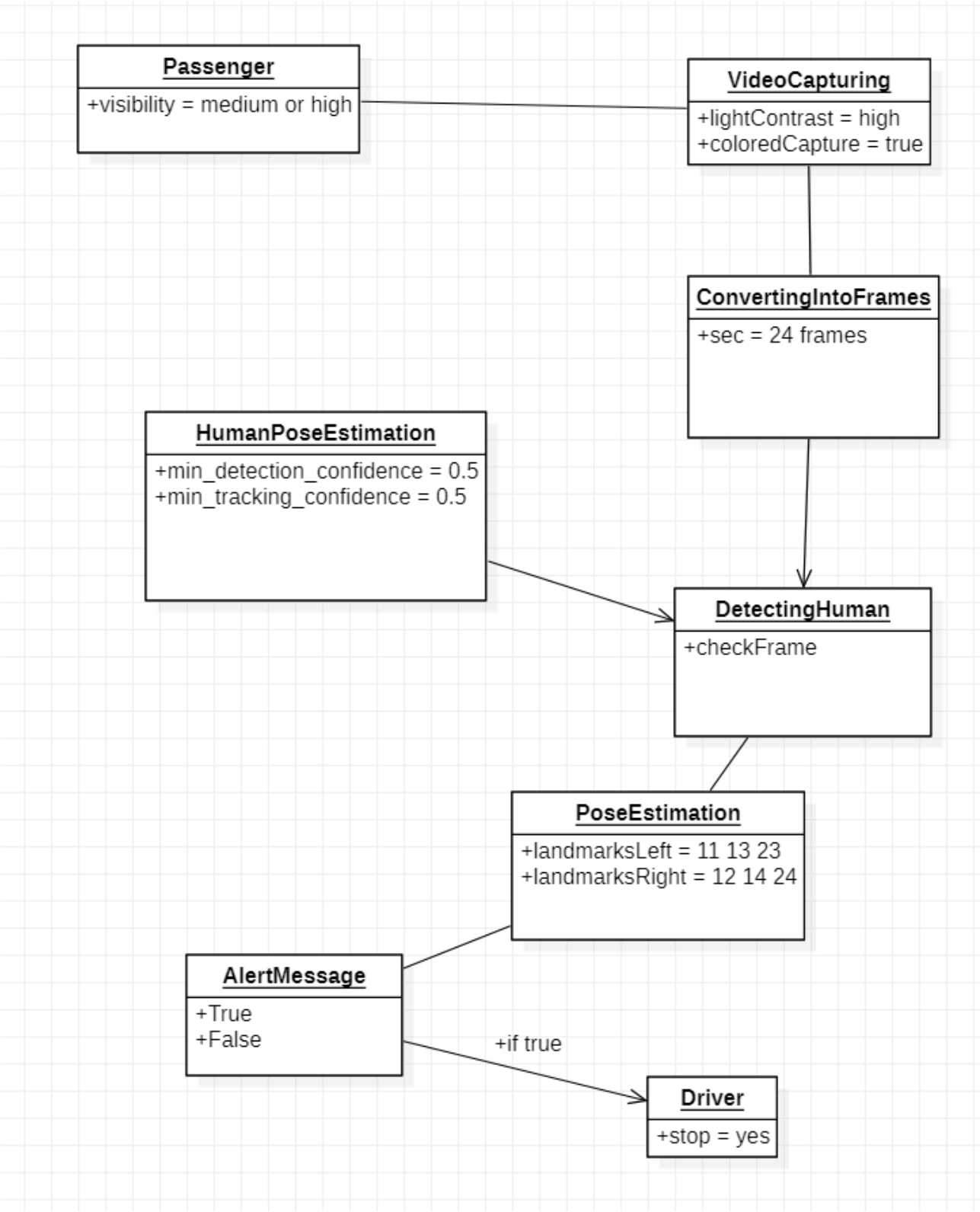


Fig : object diagram

3.1.7. Sequence diagram:

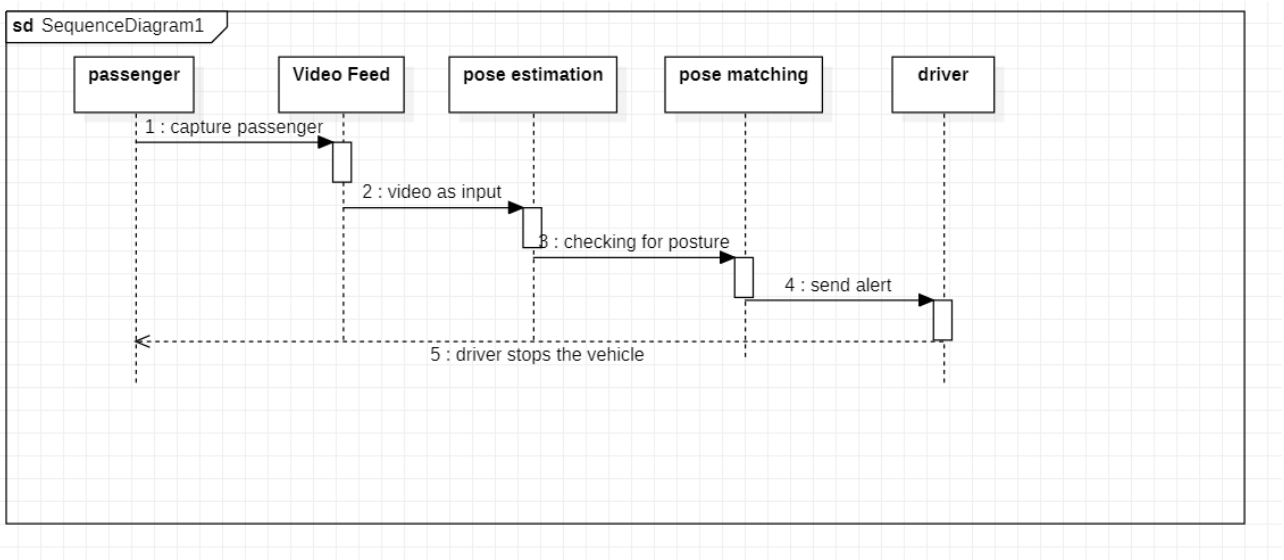


fig: Sequence Diagram

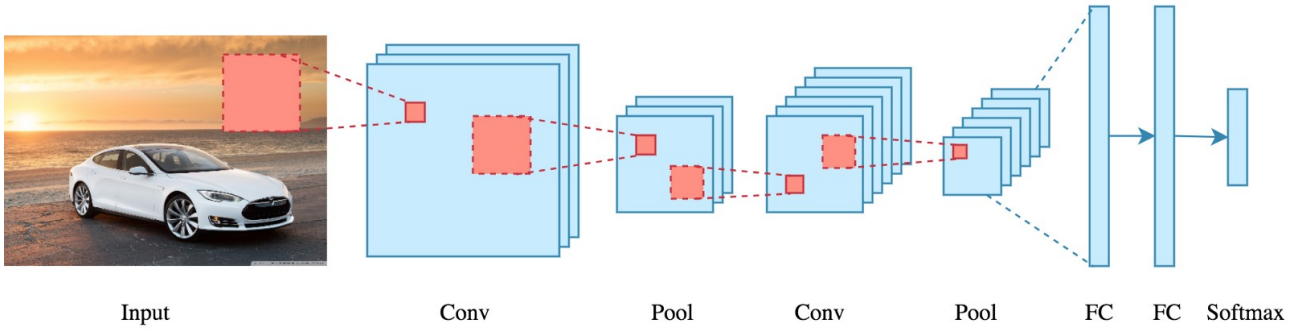
4. SYSTEM IMPLEMENTATION

We had developed this project in python libraries such as numpy, media pipe[3], cv2. We have been using video capturing as the input**.**

4.1. Preliminaries :

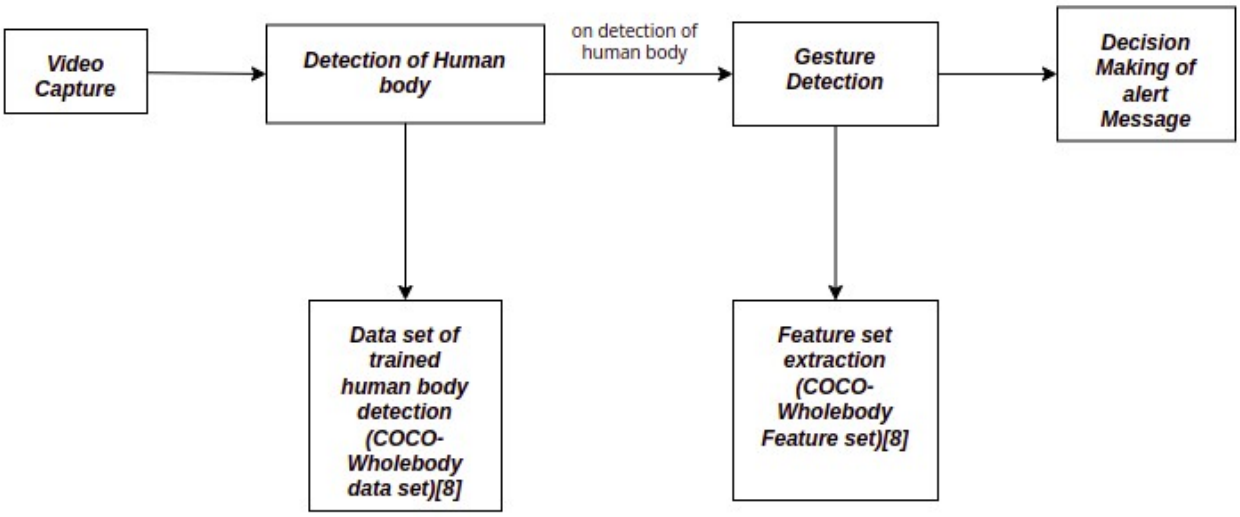
CNN Algorithm:

A **convolutional neural network**[9] is most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics **convolution** is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.



4.2. Proposed System :

In our Human pose estimation model we focus on the accuracy and speed. This model helps to estimate human posture and send an alert to the driver to stop the vehicle. Model analyzes the real time capturing video from cameras, identifies a human posture. Extracts the skeleton like structures and estimates whether the skeleton like structure matches with estimated structure or not. If it matches it says “YES” otherwise it says “NO”.



Stop Gesture Detection Model

Steps Involved

- Video capturing
- Detecting human
- Pose estimation
- Extracting landmarks
- Matching with required pose
- Display alert message

4.2.1. Video Capturing

OpenCV library cv2 provides a built-in solution to engage a streaming device, capture a video stream, and provide video frames. We can use this by calling the cv2 VideoCapture library.

This library can read frames of video and display them in a window.To capture the video from camera we need to have opencv libraries which facilitates to capture the real time video
Import cv2
cap=cv2.VideoCapture(0)
Here “0” indicates the input from the device camera.

4.2.2 Analyze Each Frame

In this step we divide each second into 24 equal frames. In each frame it detects whether the frame has human posture or not. If the frame has human posture it extracts the skeleton like structure.

ret,frame=cap.read()

If returns the boolean and frame respectively. Ret contains True or False. If the frame exists it contains “True” or else “False”.

4.2.3. Detecting Human

For detecting humans and pose we used the Mediapipe framework. Mediapipe is a framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform framework works in Desktop/server, Android, ios, and embedded devices like Raspberry pi and Jestson nano.

The frames extracted from OpenCV are BGR format. So, we first convert it to RGB format.Once we have our video frames in RGB, we can apply MediaPipe’s Pose on video frames to track body posture during a workout.

Mediapipe provides many customizable machine learning pretrained models those are:

- Mediapipe Face detection
- Mediapipe Face Mesh
- Mediapipe Hands
- Mediapipe Holistic
- Mediapipe Objectron
- Mediapipe Pose
- Mediapipe selfie segmentation

4.2.4. Pose Estimation

MediaPipe[3] Pose is an ML solution that tracks body pose using 33 landmarks. It utilizes BlazePose research by removing background from a complete RGB frame. Pose-generated pipelines are lightweight opposing conventional ML solutions.

4.2.5. Sending Alert Message

After extracting the skeleton from human posture it checks whether the extracted posturematches with the estimated posture or not. The decision of sending the alert message depends on the output given from the pose estimation. If the pose estimated returns “True” then extracted posture matches with estimated posture, the system will say “YES” otherwise it says “NO”.

5. CODE IMPLEMENTATION

Lets see the code implementation of this project where we extract the human posture. We have used libraries like mediapipe, python 3, numpy, and cv2 to implement this model.

```
import mediapipe as mp
import cv2
import numpy as np
```

Then We need to find the angle between shoulder, elbow, shoulder to hip. To find the angles between three points we implemented a function[6]

```
def cangle(a, b, c):  # (11,13,15)
    a = np.array(a)  # first
    b = np.array(b)  # mid
    c = np.array(c)  # end
    radians = np.arctan2(c[1] - b[1], c[0] - b[0]) - np.arctan2(a[1] -
b[1], a[0] - b[0])  # ((cy-by)-(cx-bx))
    angle = np.abs(radians * 180.0 / np.pi)  # converting into degrees from
radians
    if angle > 180.0:
        angle = 360 - angle
    return angle
```

We need to import drawing utilities to draw skeleton like structure of the body and pose estimation modules to estimate the pose for this

```
mp_drawing = mp.solutions.drawing_utils  # it gives all drawing utilities
visualizing poses
mp_pose = mp.solutions.pose  # importing pose estimation models
```

Then we gave live video as the input. To feed the video we used cv2 libraries here 0 is the camera attribute to capture the colored video.

```
# video feed
cap = cv2.VideoCapture(0)
print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

Now we are going to setup media pipe instance

```
with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()
```

In the above steps we captured the video using cv2. In opencv by default the image is in BGR feed But in media pipe it is in RGB feed. So, to estimate the human posture using media pipe librarie we need to convert the image feed from BGR to RGB.

```
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # make detection
        results = pose.process(image)

        # reclor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # extract landmarks
```

After feeding the video we need to extract the landmarks. Sometimes we aren't able to extract the landmarks due to low contrast. For this we are implementing this step in try block.

```
        try:
            landmarks = results.pose_landmarks.landmark
```

We need to extract structure, we need to estimate coordinates for both left and right sides.To get coordinates from left side

```
            left_shoulder =
[landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,

landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]  # 11
            left_elbow =
[landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,
                    landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]  #
13
            left_wrist =
[landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,
                    landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]  #
15
            left_hip = [landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x,
                   landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y]  # 23
```

To get coordinates from right side

```
            right_shoulder =
[landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x,

landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y]  #
            right_elbow =
[landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y]
#
            right_wrist =
[landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].y]
#
            right_hip = [landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x,
                   landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].y]  #
```

Now we need to calculate the angles

```
            left_elbow_angle = cangle(left_shoulder, left_elbow,
left_wrist)
            left_shoulder_angle = cangle(left_hip,left_shoulder,
left_elbow)
            #calculate angle for right elbow and right angle
            right_elbow_angle = cangle(right_shoulder, right_elbow,
right_wrist)
            right_shoulder_angle = cangle(right_hip, right_shoulder,
right_elbow)
            # visualize angle for left side
            cv2.putText(image, str(int(left_shoulder_angle)),
tuple(np.multiply(left_shoulder, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
            cv2.putText(image, str(int(left_elbow_angle)),
tuple(np.multiply(left_elbow, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
```

Now visualizing the angles from the left and right sides.

```
            # visualize angle for left side
            cv2.putText(image, str(int(left_shoulder_angle)),
tuple(np.multiply(left_shoulder, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
            cv2.putText(image, str(int(left_elbow_angle)),
tuple(np.multiply(left_elbow, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
            #visualize angle for right side
            cv2.putText(image, str(int(right_shoulder_angle)),
tuple(np.multiply(right_shoulder, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
            cv2.putText(image, str(int(right_elbow_angle)),
```

```
tuple(np.multiply(right_elbow, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),
                        2, cv2.LINE_AA)
```

In this below step we are going to implement curl counter logic. Curl counter logic is used to count the angle when angle is changed when the angle is changed by hand when it comes from down position.

```
            if left_shoulder_angle<=50 or left_shoulder_angle>110:
                left_stage="up"
                left_count = 0
            if left_shoulder_angle>75 and  left_shoulder_angle <110 and
left_elbow_angle>160 and left_stage=="up":  # to count only when the angle
```

```
            if left_shoulder_angle>100:
                left_count=0
            if right_shoulder_angle<=50 or right_shoulder_angle>110:
                    right_stage="up"
                    right_count = 0
            if right_shoulder_angle >75 and right_shoulder_angle <110 and
right_elbow_angle>160 and right_stage=="up":  # to count only when the
```

If the input human posture mathes with the trained model

```
                left_stage="down"
                left_count = 1
                print("1STOP THE BUS")
```

```
        cv2.rectangle(image, (0, 0), (130, 50), (200, 100, 16), -1)
        #put data
        cv2.putText(image, 'STOP:', (12, 25), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 0), 1, cv2.LINE_AA)
        if left_count==1 or right_count==1:
            cv2.putText(image, 'YES', (63, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.8, (255, 255, 255), 2, cv2.LINE_AA)
        else:
            cv2.putText(image, 'NO', (63, 25), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 1, cv2.LINE_AA)
```

In this step we are going to render detection the last arguments for color change of dots and connections respectively.

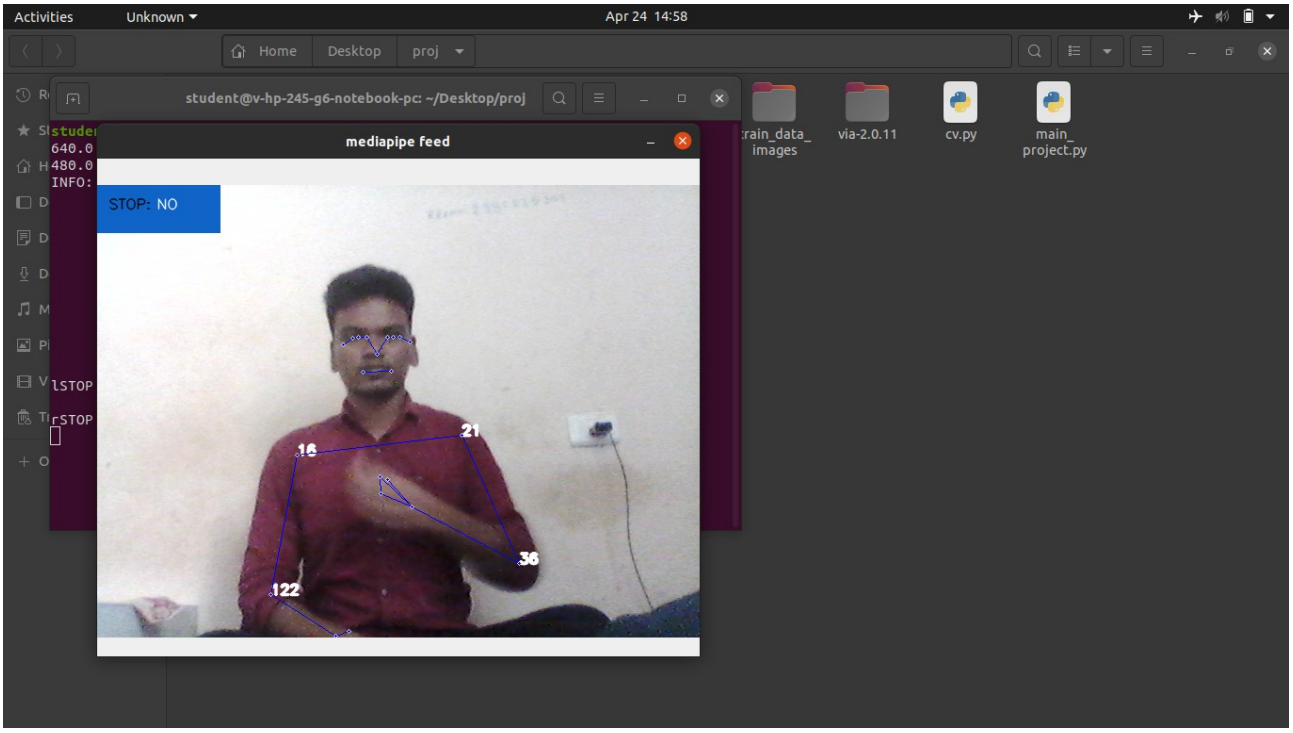
```
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(255, 0, 0),
thickness=1, circle_radius=1),
                                  # color in BGR format here
                                  mp_drawing.DrawingSpec(color=(0, 255,
255), thickness=1, circle_radius=1))
```

Finally we are going to print the results, the result will give the all pose landmarks by x,y,z and visibility. And it will show the connections of landmarks.

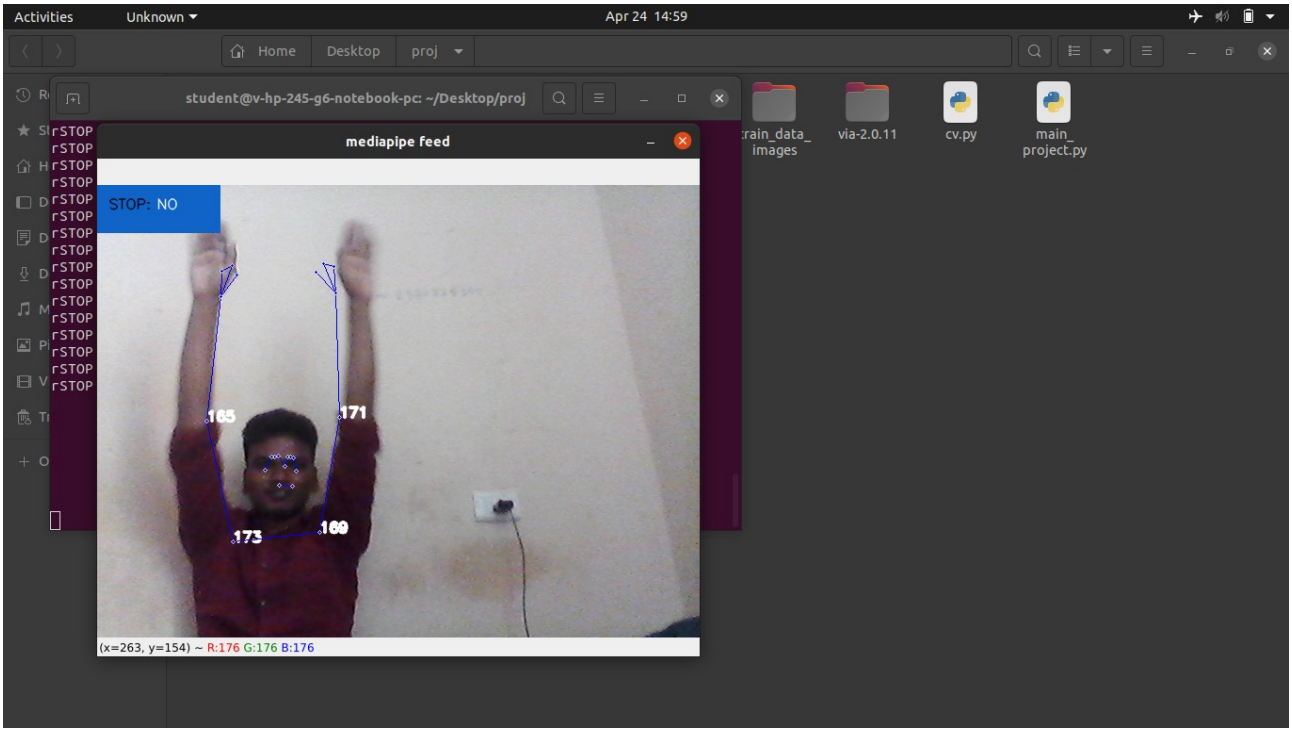
```
            break
    cap.release()
    cv2.destroyAllWindows()
```

```
        cv2.imshow("mediapipe feed", image)
        if cv2.waitKey(10) & 0xFF == ord('q'):
```

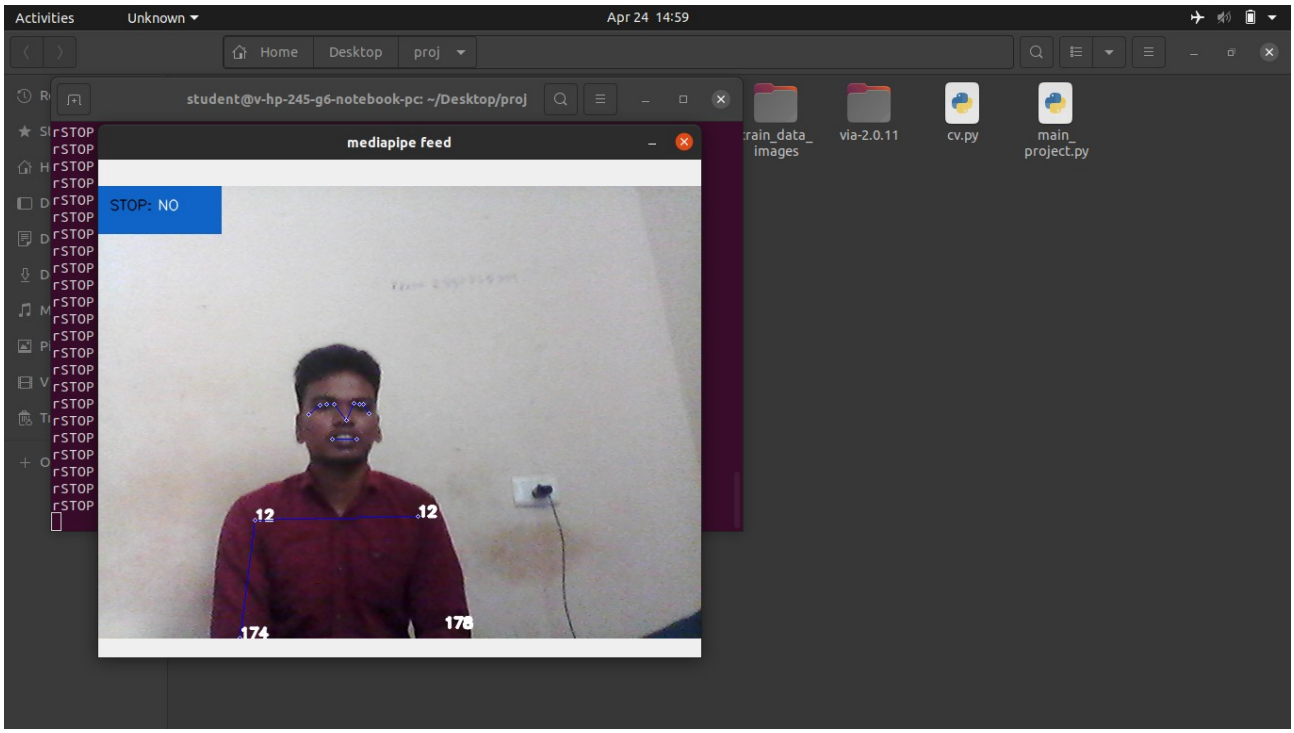
RESULT


Activities
Unknown
Apr 24 14:58
Home
Desktop
proj
student@v-hp-245-g6-notebook-pc: ~/Desktop/proj
mediapipe feed
STOP: NO
16
21
36
122
train_data_images
via-2.0.11
cv.py
main_project.py

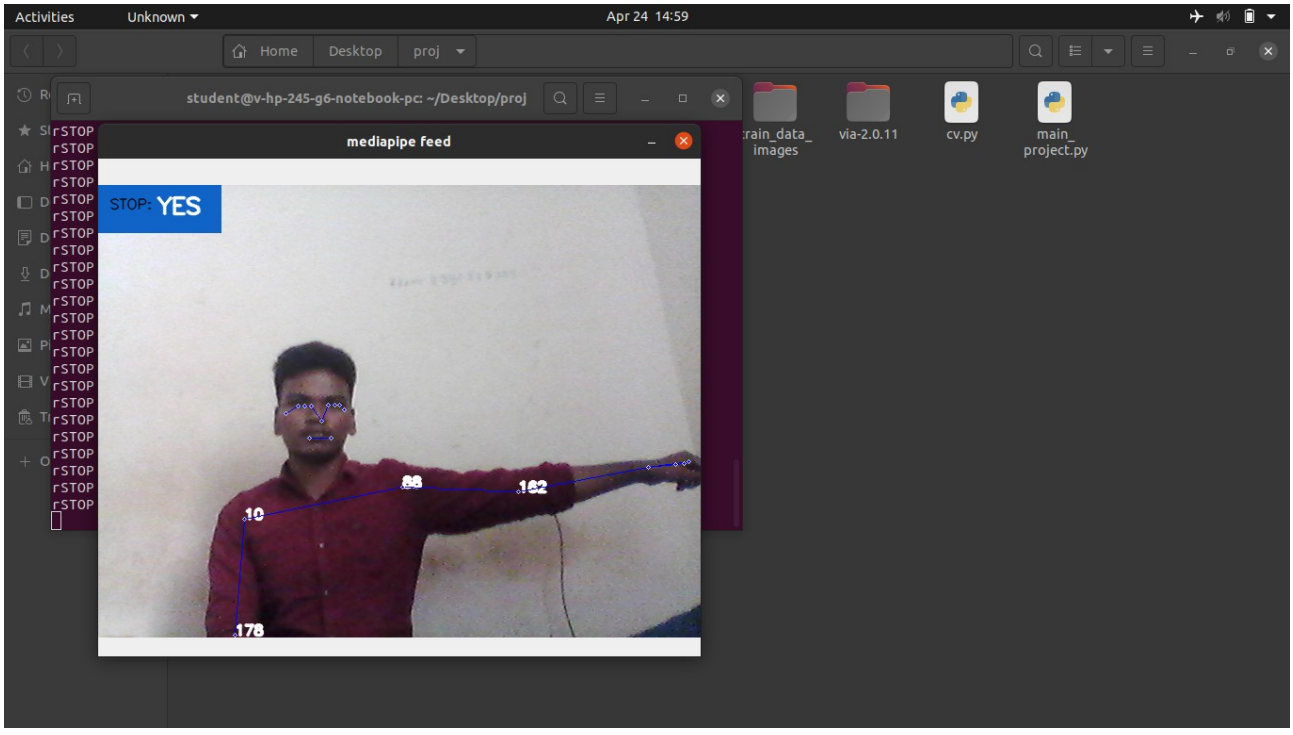
(i)


Activities
Unknown
Apr 24 14:59
Home
Desktop
proj
student@v-hp-245-g6-notebook-pc: ~/Desktop/proj
mediapipe feed
STOP: NO
165
171
173
169
(x=263, y=154) ~ R:176 G:176 B:176
train_data_images
via-2.0.11
cv.py
main_project.py

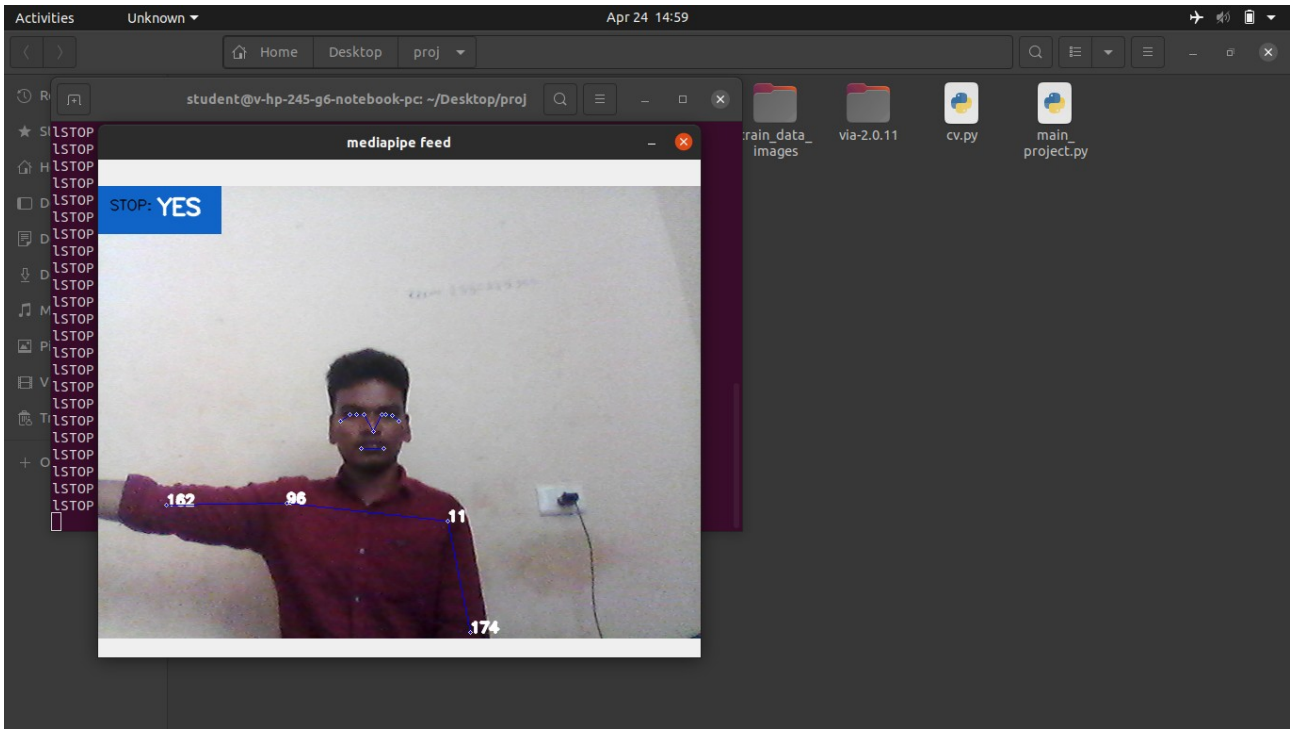
(ii)



(iii)



(iv) when passenger trying to stop the vehicle with left hand



(v) when passenger wants to stop the vehicle right hand

Alert Message

Displaying an alert message or not is based on the flag which is output from the module “matching with estimated model”.

- If the module gives “true” as output then it displays alert message by showing “YES” on the top left corner of the frame.
- If the module gives “False” as output then id displays the “NO” on the top left corner of the frame.

To know whether the message is displaying correctly or not we give positive inputs as well as negative inputs to our model. It gives expected outputs as per the flags.

If the extracted skeleton-like structure mathes with the pose which we have defined then the result shows “YES” (from image (iv) and (v)), Otherwise it shows “NO”(from image (i),(ii),(iii)).

6. CONCLUSION

Automatic Stop Gesture Detection For Vehicle is a program that estimates human posture. It is used to send an alert to the vehicle when a passenger is waiting. From the passenger standpoint it's hard to catch a vehicle.
When we implement this model on vehicles like buses it's easy to recognize passengers who are waiting for the bus. This model provides a better experience to passengers.

7. REFERENCES

1. https://ieeexplore.ieee.org/document/8701267
2. https://www.w3schools.com/python/numpy/numpy_getting_started.asp
3. https://google.github.io/mediapipe
4. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42237.pdf
5. https://en.wikipedia.org/wiki/Systems_design#:~:text=Systems%20design%20is%20the%20process,systems%20theory%20to%20product%20development.
6. https://www.codegrepper.com/code-examples/c/calculate+angle+between+two+points
7. https://deseng.ryerson.ca/dokuwiki/design:system_diagram#:~:text=A%20system%20diagram%20is%20a,all%20fall%20under%20this%20rubric.
8. https://github.com/jin-s13/COCO-WholeBody
9. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/