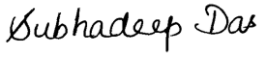
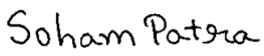
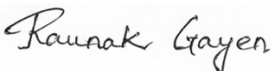
A Mini Project Report

On

ILLUMINATION CONTROL USING COMPUTER VISION

Group No.	Sl. No.	Name	Roll No.	Signature
3	1	Kaustav Saha	2020EEB047	
	2	Subhadeep Das	2020EEB051	
	3	Soham Patra	2020EEB061	
	4	Raunak Gayen	2020EEB066	

Under the Guidance of

Prof. Debjani Ganguly

INTRODUCTION

In today ‘s world, the computers have become an important aspect of life and are used in various fields however, the systems and methods that we use to interact with computers are outdated and have various issues. Hence, a very new field trying to overcome these issues has emerged namely HUMAN COMPUTER INTERACTIONS (HCI). Although, computers have made numerous advancements in both fields of Software and Hardware, still the basic way in which Humans interact with computers remains the same, using basic pointing device (mouse) and Keyboard or advanced Voice Recognition System, or maybe Natural Language processing in really advanced cases to make this communication more human and easier for us.

Our Mini-Project aims at using Hand Gestures Recognition System (mostly number of fingers raised within the region of interest) to perform various operations (Illumination Control) on a LED Lamp thereby replacing the basic devices and making the process way more user friendly than what it used to be. Our proposed system has been designed using Python programming language which in turn uses OpenCV-Python and Mediapipe Libraries to facilitate interaction within different functions of Computer through the Webcam to capture video frames.

WHAT IS COMPUTER VISION

Computer vision is a field of Artificial Intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos, and other visual inputs and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand. Computer Vision is an extra-ordinary point in its development. It indeed is a broad area of study with many specialised tasks and techniques, and also specialisations to target application domains.

WHAT IS OPENCV

OpenCV is the huge open-source library for 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 this library, one can process images and videos to identify objects, faces, or even handwriting of a human. When it is integrated with various libraries, such as NumPy,

python is capable of processing the OpenCV array structure for analysis. To identify image patterns and their various features we use vector space and perform mathematical operations on these features.

WHAT IS MEDIAPIPE

Mediapipe is a framework mainly used for building audio, video, or any time series data. With the help of the Mediapipe framework, we can build very impressive pipelines for different media processing functions. Some of the major applications of Mediapipe include Multi-hand Tracking, Face Detection, Object Detection and Tracking, etc.

INTRODUCTION TO HAND GESTURE RECOGNITION SYSTEM

A Hand Gesture Recognition System recognises the Shapes and orientation depending on implementation to task the system into performing some job. Gesture is a form of non-verbal information. As humans through vision perceive human gestures and for computer we need a camera, it is a subject of great interest for computer vision researchers such as performing an action based on gestures of the person. Some of the major applications of Hand Gesture Recognition System are -

- ## VIRTUAL PRESENCE

  Sometimes in a situation like machine, electricity failure, emergency hostiles condition or some remote areas which are inaccessible to humans, it could very dangerous for human operators to be physically appear to operate the machines or in the working conditions. So we can take help of Virtual Presence which provides the ability of physical operations. The prospects of Virtual Presence or tele-presence also include applications like space missions, underwater mission, maintenance of nuclear power reactor and anywhere the human presence is not possible or risky.

- BOMB DISPOSAL

Bomb disposal is safer when human beings are substituted by the robot arm which will work on the same concept of hand gestures recognitions. It leads to reduction in the risk of life of a human and it also encourages the efficient handling of the situation. The robot arm can be used to recognize the postures of a human made by him or her from remote place and it also performs corresponding related function.

- MOTION OF WHEEL-CHAIR

The Individuals in wheel-chair regularly confront issues with manual systems used to control the developments of seat. Motions of the hand can be embraced with the goal that each signal would be allotted to a specific development. At the point when a specific signal is given the wheelchair moves in a comparing course.

ALGORITHM OF HAND GESTURE RECOGNITION SYSTEM

- Data Obtaining

The initial move is to capture the image from camera and to define a region of Interest in the frame, it is important as the image can contain a lot of variables and these variables can result in unwanted results and the data that needs to be processed is reduced to a large extent. To capture the image a web-camera is used that continuously captures frames and is used to get the raw data for processing. The input picture we have here is uint8. The Procured image is RGB and must be processed before i.e. pre-processed before the components are separated and acknowledgement is made.

- ## Data Pre-Processing

Pre-Processing is carried out in 2 steps – Segmentation and Morphological filtering. Segmentation is done to change over grey-scale picture into the binary picture so we can have just two Area of Interest in picture. That is, one will be hand and another one is background.

Segmentation again has 2 processes:

- Pixel Based or Local methods like Edge Detection and Boundary Detection
- Region Based Approach which includes Region Merging, Region Splitting, and Threshold Method.

After Thresholding is done we have to make sure that no noise is present in the image, so we need to use Morphological filtering Techniques which involves 3 main parts:

- ## Contours

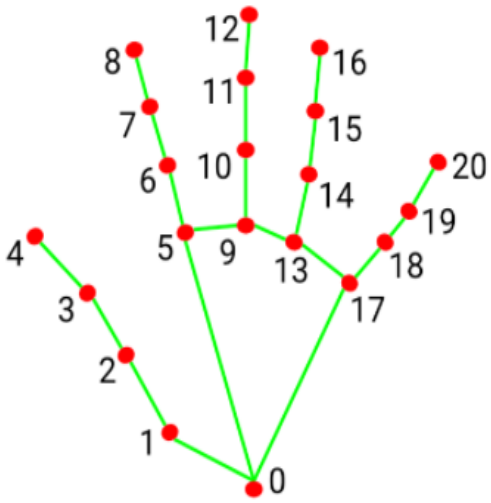
It implies the direction of hand i.e. regardless of whether the hand is on a horizontal plane or vertically set. Initially, we try to find the orientation by length to width ratio with a presumption that if the hand is vertical ten length of the box bounding them will be more than the width of the same box and, if hand is horizontally placed then width of bounding box is larger than width of the box binding the hand will be more than that of the length of the box.

- ## Finding and correcting convex hulls

A hand posture is recognized by its own orientation and by the fact that how many fingers are shown. For getting the aggregate of how many fingers are shown in hand motions that we have to process just area of the finger of the hand that we have in past advance by figuring out and analysing the centroid.

- ## Math Operations

  This can be calculated by angle = math.acos(b**2 + c**2 - a**2) * 57. This formula determines the angle between 2 fingers , helps in distinguishing between the fingers and to identify them all. We can also find out length of each raised finger's coordinate points taking centroid as the source of perspective point.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

DESCRIPTION

The purpose of this project is the use of Computer Vision and Arduino to control the Brightness of LED Lamp through Hand Gestures (Hand Gesture Recognition System). The programming languages that have been used throughout the project are Python 3.7 for OpenCV and C for Arduino. The Hardware and Software that have been implemented in this project are mentioned below.

HARDWARE COMPONENTS WE USED

- ARDUINO NANO V3

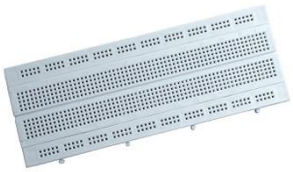
The Arduino Nano V3 is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one.

- JUMPER WIRES

Jumper wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. They come in 3 versions: male-to-male, male-to-female and female-to-female. When connecting 2 ports on a breadboard, we mostly need a male-to-male wire.

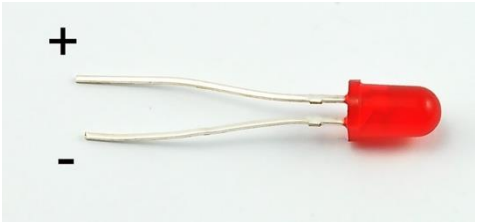
- BREADBOARD

An electronics breadboard (as opposed to the type on which sandwiches are made) actually referring to a solderless breadboard are great units for making temporary circuits and prototyping, and they require absolutely no soldering. It is a rectangular plastic board with a bunch of tiny holes in it. These holes let us easily insert electronic components to prototype (meaning to build and test an early version of) an electronic circuit, like this one with a battery, switch, resistor, and an LED (light-emitting diode). Since they need no soldering, they can be reused.



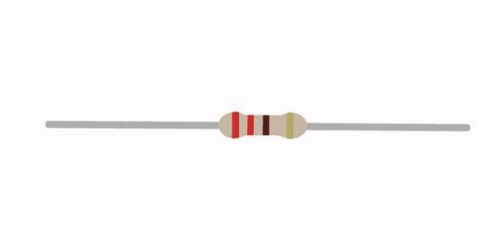
- LED

LEDs (Light Emitting Diode) are small, powerful lights that are used in many different applications. It's actually a semi-conductor light source that emits light when current flows through it. The longer leg is the positive side of the LED, called the Anode and the shorter leg is the negative side called the Cathode.



- 220Ω RESISTOR

Resistors act to reduce current flow, and, at the same time, act to lower voltage levels within circuits. In electronic circuits, resistors are used to limit current flow, to adjust signal levels, bias active elements, and terminate transmission lines among other uses. Here we have used a 220 Ohm ¼ W Resistor with 5% tolerance limit.



- CABLE FOR ARDUINO NANO

The Cable for Arduino Nano (USB 2.0 A to USB 2.0 Mini B) 30cm connects speed-critical devices, such as smartphones and peripherals that require a Mini-B connection to our computer. It has up to 480 Mbps data transfer rate.

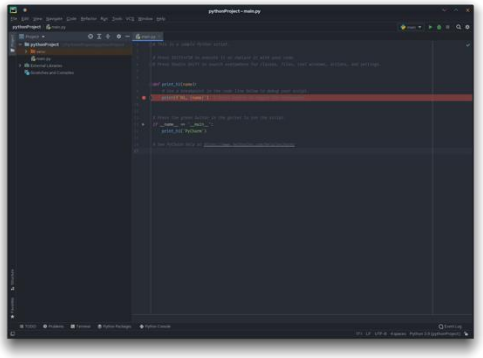
SOFTWARE WE USED

- ARDUINO IDE

The Arduino Integrated Development Environment is a cross-platform open-source application that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of third-party cores, other vendor development boards. The Version we used is 1.8.16.

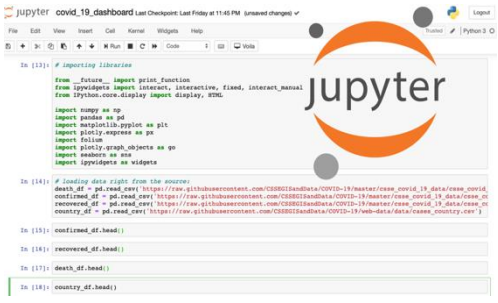
- PYCHARM

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains (formerly known as IntelliJ). It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda. PyCharm is cross-platform with Windows, MacOS and Linux.



- JUPYTER NOTEBOOK

The Jupyter Notebook is a web application for creating and sharing documents that contain code, visualizations, and text. It can be used for data science, statistical modelling, machine learning, and much more. It also can be executed on a local desktop requiring no internet access or can be installed on a remote server and accessed through the internet. The



app has a Dashboard, a Control Panel showing local files and allowing to open notebook documents or shutting down their kernels.

PYTHON LIBRARIES WE USED

- OPENCV-PYTHON
- MEDIAPIPE
- PYSERIAL
- NUMPY
- PYTHON-MATH

PYTHON CODE

- HandTrackingModule.py

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


class handDetector():

    def _init_(self, mode=False, maxHands=1, modelComplexity=1, detectionCon=0.5,
trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.modelComplex = modelComplexity
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
self.modelComplex, self.detectionCon,
```

```
                                        self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils  # object for Drawing
        self.tipIds = [4, 8, 12, 16, 20]

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # converting to RGB bcoz
hand recognition works only on RGB image
        self.results = self.hands.process(imgRGB)  # processing the RGB image
        if self.results.multi_hand_landmarks:  # gives x,y,z of every landmark or
if no hand than NONE
            for handLms in self.results.multi_hand_landmarks:  # each hand
landmarks in results
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                               self.mpHands.HAND_CONNECTIONS)  #
joining points on our hand

        return img

    def findPosition(self, img, handNo=0, draw=True):
        xList = []
        yList = []
        bbox = []
        self.lmlist = []
        if self.results.multi_hand_landmarks:  # gives x,y,z of every landmark
            myHand = self.results.multi_hand_landmarks[handNo]  # Gives result for
particular hand
            for id, lm in enumerate(myHand.landmark):  # gives id and lm(x,y,z)
                h, w, c = img.shape  # getting h,w for converting decimals x,y into
pixels
                cx, cy = int(lm.x * w), int(lm.y * h)  # pixels coordinates for
landmarks
                # print(id, cx, cy)
                xList.append(cx)
                yList.append(cy)
                self.lmlist.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            bbox = xmin, ymin, xmax, ymax

            if draw:
                cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20), (bbox[2] + 20,
bbox[3] + 20), (0, 255, 0), 2)

        return self.lmlist, bbox

    def fingersUp(self):  # checking which finger is open
        fingers = []  # storing final result
        # Thumb < sign only when  we use flip function to avoid mirror inversion
else > sign
        if self.lmlist[self.tipIds[0]][1] > self.lmlist[self.tipIds[0] - 1][
            1]:  # checking x position of 4 is in right to x position of 3
```

```
            fingers.append(1)
        else:
            fingers.append(0)

        # Fingers
        for id in range(1, 5):  # checking tip point is below tippoint-2 (only in Y
direction)
            if self.lmlist[self.tipIds[id]][2] < self.lmlist[self.tipIds[id] -
2][2]:
                fingers.append(1)
            else:
                fingers.append(0)

            # totalFingers = fingers.count(1)

        return fingers

    def findDistance(self, p1, p2, img, draw=True, r=15, t=3):  # finding distance
between two points p1 & p2
        x1, y1 = self.lmlist[4][1], self.lmlist[4][2]  # getting x,y of p1
        x2, y2 = self.lmlist[8][1], self.lmlist[8][2]  # getting x,y of p2
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2  # getting centre point

        if draw:  # drawing line and circles on the points
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
            cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)

        length = math.hypot(x2 - x1, y2 - y1)

        return length, img, [x1, y1, x2, y2, cx, cy]


def main():
    PTime = 0  # previous time
    CTime = 0  # current time
    cap = cv2.VideoCapture(0)
    detector = handDetector()

    while True:
        success, img = cap.read()  # T or F,frame
        img = detector.findHands(img)
        lmlist, bbox = detector.findPosition(img)
        if len(lmlist) != 0:
            print(lmlist[4])

        CTime = time.time()  # current time
        fps = 1 / (CTime - PTime)  # FPS
        PTime = CTime  # previous time is replaced by current time

        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_COMPLEX, 3,
(255, 0, 255),
                    3)  # showing Fps on screen
```

```
        cv2.imshow("Image", img)  # showing img not imgRGB
        cv2.waitKey(1)


if _name_ == "_main_":
    main()
```

- BrightnessControl.py

```
import serial
import cvzone
import math
import cv2
import numpy as np
import screen_brightness_control as sbc
import HandTrackingModule as htm

bbar = 0
minBRIGHTNESS = 0
maxBRIGHTNESS = 100
serialcomm = serial.Serial('COM3', 9600)
serialcomm.timeout = 1
cap = cv2.VideoCapture(0)
detector = htm.handDetector(detectionCon=0.8, maxHands=1)
l = []
while True:
    success, img = cap.read()
    img = cv2.resize(img, (500, 500))
    img = detector.findHands(img)
    l, box = detector.findPosition(img, draw=False)
    if l:
        # f=detector.fingersUp()
        x1 = l[4][1]
        y1 = l[4][2]
        x2 = l[8][1]
        y2 = l[8][2]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        cv2.circle(img, (cx, cy), 7, (255, 0, 0), cv2.FILLED)
        cv2.circle(img, (x1, y1), 7, (0, 255, 255), 1)
        cv2.circle(img, (x2, y2), 7, (0, 255, 255), 1)
        cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        d = int(math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2) * 1.0))
        brightness = np.interp(d, [20, 100], [minBRIGHTNESS, maxBRIGHTNESS])
        bbar = np.interp(d, [50, 300], [400, 150])
        cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 255), 3)
        cv2.rectangle(img, (50, int(bbar)), (85, 400), (255, 0, 255), cv2.FILLED)
        cv2.putText(img, f'{int(brightness)}%', (85, 450),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 4, (255, 0, 255), 3)
        sbc.set_brightness(int(brightness), None)
```

```
        d = int((d / 110) * 255)
        e = '\n'
        if 0 < d < 256:
            cv2.putText(img, str(d), (20, 30), cv2.FONT_HERSHEY_COMPLEX, .7, (255,
255, 255), 1)
            serialcomm.write(str(d).encode())
            serialcomm.write(e.encode())

    cv2.imshow('Image', img)
    if cv2.waitKey(20) & 0xFF == 27:
        break
cv2.destroyAllWindows()
```

WORKING OF THE PYTHON CODE

First we have to install the required modules via the following commands –

```
pythonProject pip install opencv-python
pythonProject pip install mediapipe
```

After that is done, let's now check the working of the webcam.

```
import cv2
import time
cap = cv2.VideoCapture(0)
pTime = 0
cTime =0
while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, f'FPS:{int(fps)}', (20, 70), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow("Test", img)
    cv2.waitKey(1)
```

Our PC is working fine as we can see and it also shows Frames per Second (FPS) on the top left corner of the output window.

In the **HandTrackingModule.py** file as we have stated above, our handDetector Class has been implemented to detect hand. It exports the landmarks in pixel format. Some extra functionalities have also been added like finding how many fingers are up or distance between two fingers and it also provides bounding box info of the hand format.

The function **findHands()** accepts an RGB image and as the name suggests searches for hands in it. It takes the image as one of its arguments, and takes a flag argument to draw output on the image. The function returns an Image with or without drawings.

The function **fingersUp()** is used to find how many fingers are open and returns answer in a list. It considers left and right hands separately. It returns a list of which fingers are up.

The function **findDistance()** finds the distance between two landmarks based on their index numbers. It takes two points p1 and p2 as arguments

and an image to draw on and returns the image with distance between those 2 points p1 and p2 drawn on it.

The **findPosition()** function gives the position of the hand along with the id.

Then comes the **main()** function where we initialize our module and also write a while loop to run the model.

On successful running of HandTrackingModule.py we are getting this as Output –

Now we have imported this Module (HandTrackingModule.py) in our main file that is **BrightnessControl.py**

In BrightnessControl.py –

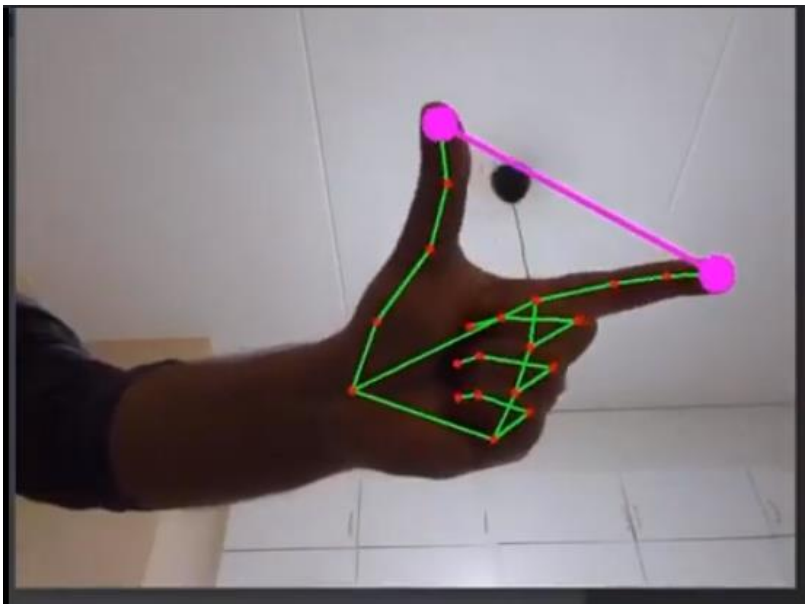
```
x1 = l[4][1]
y1 = l[4][2]
x2 = l[8][1]
y2 = l[8][2]
```

these are the coordinates of fingers which we are using to control the LED. Here we have mainy used Index and Thumb fingers and these 4 coordinates as we can see above represent indexes of tips of Thumb and Index.

```
cv2.circle(img, (cx, cy), 7, (255, 0, 0), cv2.FILLED)
```

```
        cv2.circle(img, (x1, y1), 7, (0, 255, 255), 1)
        cv2.circle(img, (x2, y2), 7, (0, 255, 255), 1)
        cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        d = int(math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2) * 1.0))
        brightness = np.interp(d, [20, 100], [minBRIGHTNESS, maxBRIGHTNESS])
        bbar = np.interp(d, [50, 300], [400, 150])
        cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 255), 3)
        cv2.rectangle(img, (50, int(bbar)), (85, 400), (255, 0, 255), cv2.FILLED)
        cv2.putText(img, f'{int(brightness)}%', (85, 450),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 4, (255, 0, 255), 3)
        sbc.set_brightness(int(brightness), None)
        d = int((d / 110) * 255)
        e = '\n'
        if 0 < d < 256:
            cv2.putText(img, str(d), (20, 30), cv2.FONT_HERSHEY_COMPLEX, .7, (255,
255, 255), 1)
            serialcomm.write(str(d).encode())
            serialcomm.write(e.encode())
```

The above piece of code is responsible for the circles and line which we have drawn on the hand to show the control bar connecting the tip of the index finger and thumb.



Distance is also calculated between index and thumb which is converted to a range of 0 – 255. This range data is then sent to the Arduino Nano V3 with the help of the Serial Library as we can see in the code below –

```
serialcomm = serial.Serial('COM3', 9600)
serialcomm.timeout = 1
d = int(math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2) * 1.0))
if 0 < d < 256:
            cv2.putText(img, str(d), (20, 30), cv2.FONT_HERSHEY_COMPLEX, .7, (255,
255, 255), 1)
            serialcomm.write(str(d).encode())
            serialcomm.write(e.encode())
```

This is the way how our HandTrackingModule.py and BrightnessControl.py are working behind the scenes just to capture our Hand Gestures and simultaneously send those signals to the Arduino Nano V3 which in turn is connected to the LEDs.

ARDUINO CODE

```
const int redPin = 3;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(redPin,OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  while(Serial.available() > 0) {
    int red = Serial.parseInt();
    if(Serial.read() == '\n') {
      red = constrain(red,0,255);
      analogWrite(redPin,red);
      Serial.print(red);
    }
  }
}
```

WORKING OF THE ARDUINO CODE

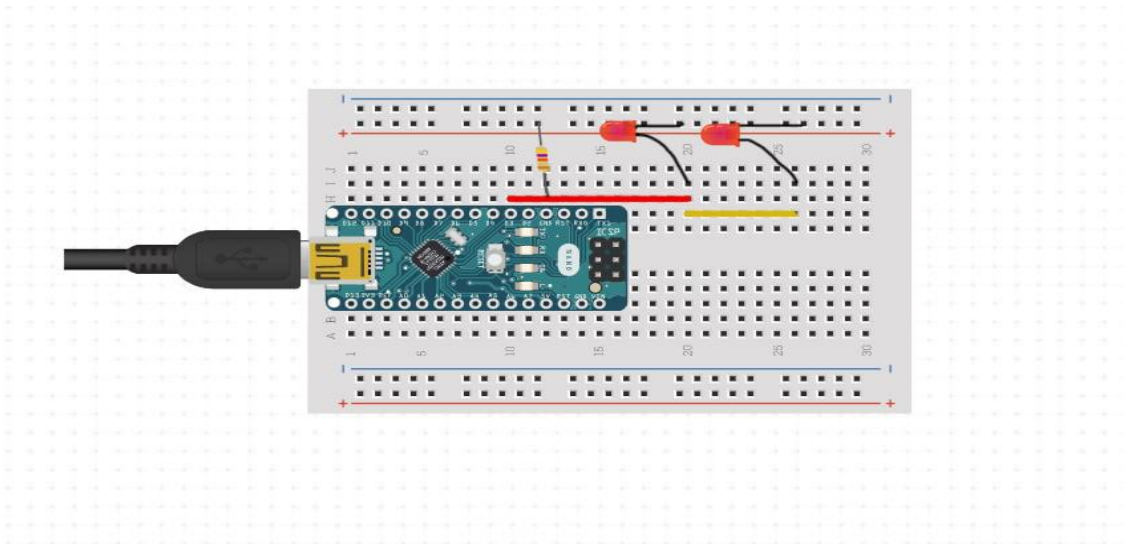
In order to get started with the Arduino Code that has been written in the Arduino IDE, we first need to make sure that our Arduino Board, Processor and Port have been properly configured in the Arduino IDE to avoid any connection error.
In the Menu bar we have to select **Tools > Board > Arduino Nano.** Then we have to choose **Tools > Processor > ATmega328P (Old Bootloader).** Now we can write our code that's majorly in C Programming Language.

The Arduino Code that we have stated above is receiving data from our Laptop / PC through Serial Communication which is UART. Now let's understand how code is working –

```
int red = Serial.parseInt();
if(Serial.read() == '\n') {
```

This line of code is responsible for receiving the data serially which is being transmitted by the Python Code which has been written above under PYTHON CODE section. Here '\n' refers to escape sequence. Data is received when it comes across this character which is stored in the variable 'red' of type integer.
Now the block of code given below sets the data according to the received value which is in the range of 0-255 as Arduino is a 8 bit microcontroller. After the value is received, it's then set and finally sent through serial communication which in turn triggers the LEDs connected to the circuit.

The Circuit is connected as given in the Circuit Diagrams below.



FINAL OUTPUT

Link for the above Video - https://youtu.be/nhm_bjlnSAA