A

Mini Project Report on

“IMAGE PROCESSING ASSISTANT”

MCA III Semester (Session: 2022-23)

Submitted to:

Mr. Sanjaya Tripathi

Submitted By:

Anurag Singh (2100970140014)
Dhruv Singh (2100970140020)
Abhilakh Singh (2100970140001)

Dept of Computer Application
Galgotias College Of Engg. & Technology
Greater Noida (U.P.) -201306

Department of Computer Applications

GALGOTIAS COLLEGE OF ENGINEERING AND TECHNOLOGY

1, Knowledge Park-II, Greater Noida-201306
(Affiliated to AKTU, Lucknow)

Certificate

Date: ………………

This is to certify that the dissertation entitled **"Image Processing Assistant"** by **Mr. Anurag Singh,** student of **MASTER OF COMPUTER APPLICATIONS** , [2022-2023], of **GALGOTIAS COLLEGE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA,** affiliated to **AKTU, LUCKNOW ,** is hereby accepted and approved as a credible work. It is further certified that this work has not been submitted for similar purpose anywhere else. His work has been found satisfactory for the partial fulfillment of the award of the degree of MCA.

Internal Examiner **External Examiner**

Head of Department

ABSTRACT

The objective of this project is to build Image Processing Assistant based on Python to Perform various image task at one place and equipped with a virtual assistant that will provide voice enable support.

This project provide security to your Images and data rather than going to malware infected insecure fraud website to process your particular function on your private data.

With the help of this project, You can interact it with Modern UI and also Voice Assistant You can perform various task like image to text , image to sketch, text to handwritten notes.

Acknowledgement

It is high privilege for me to express my deep sense of gratitude to all those faculty members who helped me in the completion of the mini project, especially my internal guide, **Mr. Sanjaya Tripathi** who was always there at hour of need.

My special thanks to **Professor (Dr. Kanchan Hans)** HOD- Department of Computer Applications, **Galgotias College of Engineering and Technology**, for helping me in the completion of mini project work and its report submission.

Anurag Singh
2100970140014

TABLE OF CONTENT

LIST OF FIGURES

Introduction

Image Processing Assistant is a python based GUI project and provides lots of features that we can do it with just one software containing many tools and providing an AI virtual assistant to guide them and initialize the software.
Lots of libraries and APIs are used to made this project happen. PyQt is one of the main library used to build Python GUI interface.
In this modern world we do a lot of things with image and sometimes out confidential data get leaked on various fraud sites containing malwares in it.

- Images define the world, each image has its own story, it contains a lot of crucial information that can be useful in many ways. This information can be obtained with the help of the technique known as Image Processing.

- It is the core part of computer vision which plays a crucial role in many real-world examples like robotics, self-driving cars, and object detection. Image processing allows us to transform and manipulate thousands of images at a time and extract useful insights from them. It has a wide range of applications in almost every field.

- Python is one of the widely used programming languages for this purpose. Its amazing libraries and tools help in achieving the task of image processing very efficiently.

- Today’s world is full of data, and images make up a significant portion of this data. However, to be put to any use, these images need to be processed. Image processing is how we analyze and manipulate a digital image to improve its quality or extract information from it.

- Typical tasks in image processing include displaying images, basic manipulations like cropping, flipping, rotating, etc., image segmentation, classification and feature extractions, image restoration, and image recognition. Due to Python’s growing popularity as a scientific programming language and the free availability of many state-of-art image processing tools in its ecosystem, it’s an apt choice for these image processing tasks.

1. System Analysis

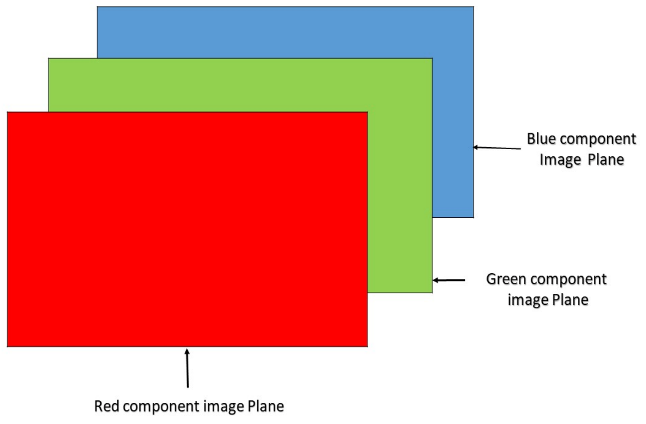
What is image processing?

As the name says, image processing means processing the image and this may include many different techniques until we reach our goal.

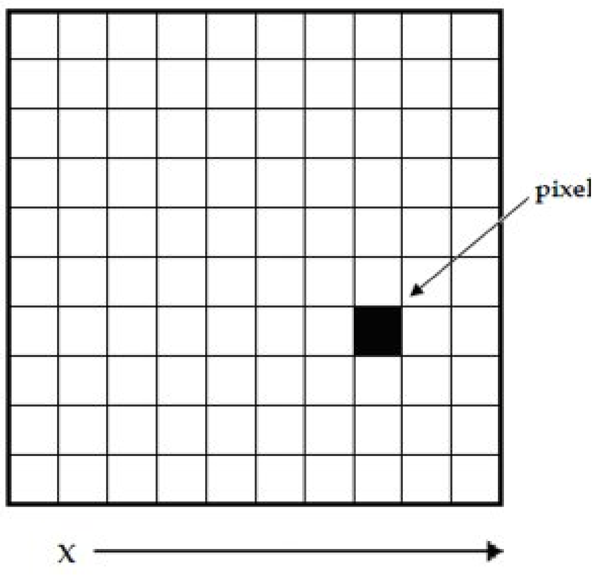
The final output can be either in the form of an image or a corresponding feature of that image. This can be used for further analysis and decision making.

But what is an image?

An image can be represented as a 2D function F(x,y) where x and y are spatial coordinates. The amplitude of F at a particular value of x,y is known as the intensity of an image at that point. If x,y, and the amplitude value is finite then we call it a digital image. It is an array of pixels arranged in columns and rows. Pixels are the elements of an image that contain information about intensity and color. An image can also be represented in 3D where x,y, and z become spatial coordinates. Pixels are arranged in the form of a matrix. This is known as an RGB image.


Blue component Image Plane
Green component image Plane
Red component image Plane

[RGB image is a three layered image]



RGB image: It contains three layers of 2D image, these layers are Red, Green, and Blue channels.

Grayscale image: These images contain shades of black and white and contain only a single channel.

Part 2 Data Design

Readers interested in how SplitPay organizes and handles data should consult this section, which covers data structures and flow patterns utilized by the system
.
•Part 3 (Architectural and Component-Level Design) oThis section describes the SplitPay system class by class, including interface details, class hierarchies, performance/design constraints, process details, and algorithmic models.

•Part 4 (User Interface Design) oThis section covers all of the details related to the structure of the graphical user interface (GUI), including some preliminary mockups of the SplitPay Android application. Readers can view this section for a tentative glimpse of what the final product will look like.

•Part 5 (Restrictions, Limitations, and Constraints) oThis section discusses the general constraints imposed upon the project

•Part 6 (Testing Issues) oReaders interested in the software testing process should consult this section, which offers a list of test cases, expected responses, and other pertinent information.

2. Objective of Project

Purpose of Image Processing Assistant is to provide lots of features that we can do it with just one software containing many tools and providing an AI virtual assistant to guide them and initialize the software.

Image Processing Assistant is a python based GUI project and provides lots of features that we can do it with just one software containing many tools and providing an AI virtual assistant to guide them and initialize the software.

Lots of libraries and APIs are used to made this project happen. PyQt is one of the main library used to build Python GUI interface.

In this modern world we do a lot of things with image and sometimes out confidential data get leaked on various fraud sites containing malwares in it.

A design viewpoint in which the design target is a personal object (a consumer product), such as a device or software app, that a user buys for private use. This project is simple to understand, and the source code has been presented in an understandable manner. To make software fast in processing with the good user interface so that user can manage and change it, this should be used for long time without error and maintenance

3. About Image Processing

Image to Sketch

Gaussian Image Processing

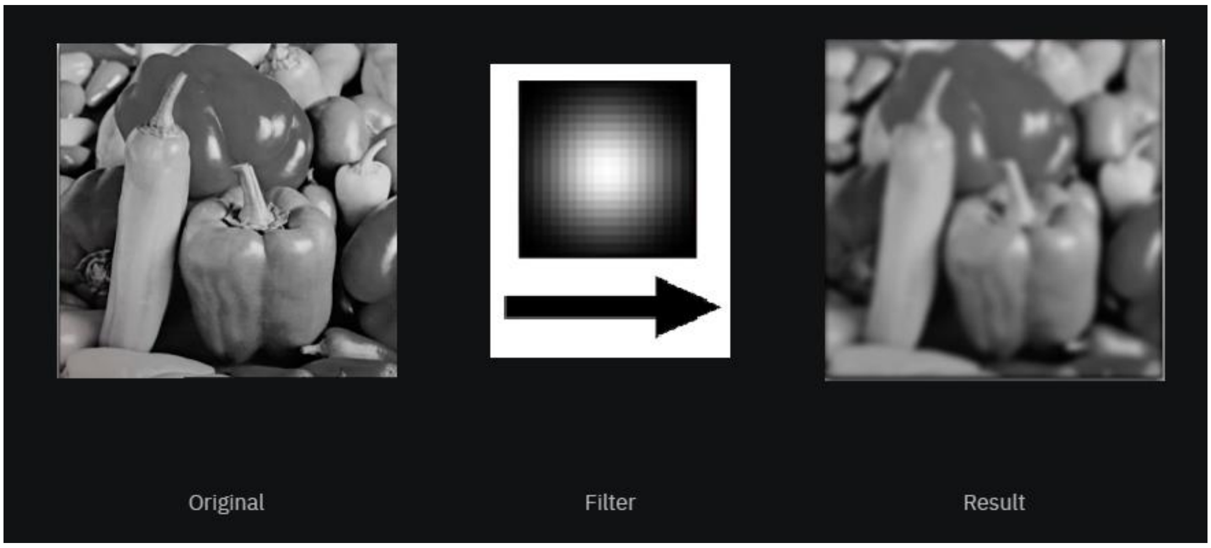
Gaussian blur which is also known as gaussian smoothing, is the result of blurring an image by a Gaussian function.

It is used to reduce image noise and reduce details. The visual effect of this blurring technique is similar to looking at an image through the translucent screen. It is sometimes used in computer vision for image enhancement at different scales or as a data augmentation technique in deep learning.

The basic gaussian function looks like:

In practice, it is best to take advantage of the Gaussian blur's separable property by dividing the process into two passes. In the first pass, a one-dimensional kernel is used to blur the image in only the horizontal or vertical direction. In the second pass, the same one-dimensional kernel is used to blur in the remaining direction. The resulting effect is the same as convolving with a two-dimensional kernel in a single pass. Let's see an example to understand what gaussian filters do to an image.

If we have a filter which is normally distributed, and when its applied to an image, the results look like this:



You can see that some of the edges have little less detail. The filter is giving more weight to the pixels at the center than the pixels away from the center. Gaussian filters are low-pass filters i.e. weakens the high frequencies. It is commonly used in edge detection.

4. System Features

It Contains various features GUI and Voice Input.
It has various Navigation button Like HOME EXIT AND RESET on every window
But main Window has-

1. *Image To Text*
2. *Image To Sketch*
3. *Text To Handwriting*
4. *Gesture Mouse*
5. *Gesture Control*
6. *QR Reader*

Image To Text

It will convert the text in image to actual Text into an .txt file

4.1.1 Description and Priority

This Function will convert the content in image into text format so user can copy and pasty and use anywhere he/she wants these days we do a lot of work related to image so this function might be very useful out of our other function.

4.1.3 Functional Requirements

User should have an image and know how to copy path of an image.

Image To Sketch

It will convert image to sketch

4.1.1 Description and Priority

Sometimes we have to instantly convert the image to an sketch so we can actually copy and create of our own so without going to million of fake sites to convert the image this tool can be useful.

4.1.3 Functional Requirements

User should have an image and know how to copy path of an image.

Text To Handwriting

It will convert the entered text into Handwritten notes in the form of images and PDFs.

4.1.1 Description and Priority

We stuck in a lot of situations where we have to write something huge and printout physically but this tool can be useful try it

4.1.3 Functional Requirements

User have to select 3rd button and enter some text paste it or anything then hit convert it will convert and save image format in default folder and will open pdf of those images so you can printout.

Mouse Gesture

Control mouse pointer with your finger

4.1.1 Description and Priority

This function is still under working but user can be able to control mouse pointer using their finger and hand gesture.

4.1.3 Functional Requirements

User should have proper fingers

Gesture Control

Using this function user can control volume using their fingers.

4.1.1 Description and Priority

function user can control volume using their fingers.

4.1.3 Functional Requirements

User should know how gesture works

QR Reader

QR reader will scan and read the QR code in the camera

4.1.1 Description and Priority

This function works on OpenCV so camera will open and user have to show any bar code / QR Code it will show the data in it.

4.1.3 Functional Requirements

Working Camera

Voice Assistant

This image process software is equipped with Voice assistant that will make things more easier to access.
As we know Python is a suitable language for scriptwriters and developers. Let's write a script for Voice Assistant using Python. The query for the assistant can be manipulated as per the user's need.
Speech recognition is the process of converting audio into text. This is commonly used in voice assistants like Alexa, Siri, etc. Python provides an API called SpeechRecognition to allow us to convert audio into text for further processing

- **Speech Recognition:-** Since we're building an Application of voice assistant, one of the most important things in this is that your assistant recognizes your voice (means what you want to say/ ask). To install this module type the below command in the terminal.

Voice based personal assistants have gained a lot of popularity in this era of smart homes and smart devices. These personal assistants can be easily configured to perform many of your regular tasks by simply giving voice commands. Google has popularized voice-based search that is a boon for many like senior citizens who are not comfortable using the keypad/keyboard.

- The **SpeechRecognition library** allows Python to access audio from your system's microphone, transcribe the audio, and save it.
- Google's **text-to-speech package, gTTS** converts your audio questions to text. The response from the look-up function that you write for fetching answer to the question is converted to an audio phrase by gTTS. This package interfaces with Google Translate's API.
- **Web browser package** provides a high-level interface that allows displaying Web-based pages to users. Selenium is another option for displaying web pages. However, for using this you need to install and provide the browser-specific web driver.

***Wikipedia** is used to fetch a variety of information from the Wikipedia website*

6 . Tools

Image Process-

1. OpenCV

OpenCV (Open Source Computer Vision Library) is one of the most widely used libraries for computer vision applications. OpenCV-Python is the Python API for OpenCV. OpenCV-Python is not only fast since the background consists of code written in C/C++ but is also easy to code and deploy (due to the Python wrapper in the foreground). This makes it a great choice to perform computationally intensive computer vision programs
There are several ways you can use opencv in image processing, a few are listed below:

- Converting images from one color space to another i.e. like between BGR and HSV, BGR and gray etc.
- Performing thresholding on images, like, simple thresholding, adaptive thresholding etc.
- Smoothing of images, like, applying custom filters to images and blurring of images.
- Performing morphological operations on images.
- Building image pyramids.
- Extracting foreground from images using GrabCut algorithm.
- Image segmentation using watershed algorithm.

2. Scikit-image

It is an open-source library used for image preprocessing. It makes use of machine learning with built-in functions and can perform complex operations on images with just a few functions.
It works with numpy arrays and is a fairly simple library even for those who are new to python. Some operations that can be done using scikit image are :

- To implement thresholding operations use **try_all_threshold()** method on the image. It will use seven global thresholding algorithms. This is in the **filters** module.
- To implement edge detection use **sobel()** method in the **filters** module. This method requires a 2D grayscale image as an input, so we need to convert the image to grayscale.
- To implement gaussian smoothing use **gaussian()** method in the **filters** module.
- To apply histogram equalization, use **exposure** module, to apply normal histogram equalization to the original image, use **equalize_hist()** method and to apply adaptive equalization, use **equalize_adapthist()** method.
- To rotate the image use **rotate()** function under the **transform** module.
- To rescale the image use **rescale()** function from the **transform** module.

- To apply morphological operations use **binary_erosion()** and **binary_dilation()** function under the **morphology** module.

3. **PIL/pillow**

PIL stands for Python Image Library and **Pillow** is the friendly PIL fork by Alex Clark and Contributors. It’s one of the powerful libraries. It supports a wide range of image formats like PPM, JPEG, TIFF, GIF, PNG, and BMP.
It can help you perform several operations on images like rotating, resizing, cropping, grayscaling etc. Let’s go through some of those operations
To carry out manipulation operations there is a module in this library called **Image.**

- To load an image use the **open()** method.
- To display an image use **show()** method.
- To know the file format use **format** attribute
- To know the size of the image use **size** attribute
- To know about the pixel format use **mode** attribute.
- To save the image file after desired processing, use **save()** method. Pillow saves the image file in *png* format.
- To resize the image use **resize()** method that takes two arguments as width and height.
- To crop the image, use **crop()** method that takes one argument as a box tuple that defines position and size of the cropped region.
- To rotate the image use **rotate()** method that takes one argument as an integer or float number representing the degree of rotation.
- To flip the image use **transform()** method that take one argument among the following: Image.FLIP_LEFT_RIGHT, Image.FLIP_TOP_BOTTOM, Image.ROTATE_90, Image.ROTATE_180, Image.ROTATE_270.

4. **NumPy**

With this library you can also perform simple image techniques, such as flipping images, extracting features, and analyzing them.
Images can be represented by numpy multi-dimensional arrays and so their type is **NdArrays**. A color image is a numpy array with 3 dimensions. By slicing the multi-dimensional array the RGB channels can be separated.
Below are some of the operations that can be performed using NumPy on the image (image is loaded in a variable named **test_img** using imread).

- To flip the image in a vertical direction, use **np.flipud(test_img).**
- To flip the image in a horizontal direction, use **np.fliplr(test_img).**
- To reverse the image, use **test_img[::-1]** (the image after storing it as the numpy array is named as <img_name>).
- To add filter to the image you can do this:

Example: **np.where(test_img > 150, 255, 0)**, this says that in this picture if you find anything with 150, then replace it with 255, else 0.

- You can also display the RGB channels separately. It can be done using this code snippet:

To obtain a red channel, do **test_img[:,:,0]**, to obtain a green channel, do **test_img[:,:,1]** and to obtain a blue channel, do **test_img[:,:,2].**

5. PyQt

PyQt:
PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British firm Riverbank Computing.
PyQt connects the Qt C++ cross-platform framework with the Python language, it is a GUI module.
Qt is more than a GUI toolkit, which is why it features abstractions of network sockets or threads, along with Unicode, SQL, databases, SVG, OpenGL, XML, an operational we browser, a service system and a vast array of GUI widgets.
The principle on which a Qt class functions is related to a slot mechanism responsible for offering communication between items with the purpose of designing re-usable software components with ease.

5.SYSTEM DESIGN

Hardware Interfaces

- *Keyboard & Mouse*
- *Working Camera*
- *CPU x86 architecture*
- *CPU Dual Core or more*
- *Ram: 4GB minimum*
- *Storage > 200 GB*
- *Internet Connection Needed*

Software Interfaces

- Windows Operating System
- Latest Windows 10/11
- Python, Java, Sql should be installed
- All mentioned library

Languages Used: Python, SQL, XML

Libraries: PyQt (QtCore, uic, QtWidgets, QtGui), subprocess, os, pytesseract, Pywhatkit, PIL, pyttsx3, Python-CV, mediapipe, pyautogui, numpy,etc
APIs: Google speech_recognition, python weather,etc

6.1 SOURCE CODE:

main.py

```
import sys
from PyQt5.QtCore import Qt, QPoint
from PyQt5.uic import loadUi
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QDialog, QApplication, QWidget
from PyQt5.QtGui import QPixmap

import subprocess
import os
import cmd



class WelcomeHome(QDialog):
    def __init__(self):
        super(WelcomeHome, self).__init__()
        loadUi("welcome.ui",self)
        self.setWindowFlag(Qt.FramelessWindowHint)
        self.start.clicked.connect(self.WelcomeScreen)


    def WelcomeScreen(self):
        welcomess=WelcomeScreen()
        widget.addWidget(welcomess)
        widget.setCurrentIndex(widget.currentIndex()+1)


class WelcomeScreen(QDialog):
    def __init__(self):
        super(WelcomeScreen, self).__init__()
        loadUi("mainhome.ui",self)
        self.pushButton_3.clicked.connect(self.gotoimgsketch)
        self.handwritingbtn.clicked.connect(self.gototexthandwriting)
        self.pushButton_2.clicked.connect(self.gotoimgtext)
        self.pushButton_4.clicked.connect(self.gotohandgesture)
        self.pushButton_7.clicked.connect(self.gotoqrreader)

        self.home.clicked.connect(self.__init__)

        #self.exitbtn.clicked.connect(self.close)


    def gotoimgsketch(self):
        imgsketch=imgsketchscreen()
        widget.addWidget(imgsketch)
        widget.setCurrentIndex(widget.currentIndex()+1)

    def gototexthandwriting(self):
        texthandwriting=handwritingscreen()
        widget.addWidget(texthandwriting)
        widget.setCurrentIndex(widget.currentIndex()+1)

    def gotoimgtext(self):
        imgtxt=imgtxtscreen()
        widget.addWidget(imgtxt)
        widget.setCurrentIndex(widget.currentIndex()+1)

    def gotohandgesture(self):
        command="python handgesture.py"
        subprocess.Popen(command)

    def gotoqrreader(self):
        command="python QR-Code-master/custom_qr.py"
        subprocess.Popen(command)



class imgtxtscreen(QDialog):
    def __init__(self):
        super(imgtxtscreen, self).__init__()
        loadUi("img-txt.ui",self)
        self.home.clicked.connect(self.gotohome)

    def gotohome(self):
        welcomescreen=WelcomeScreen()
        widget.addWidget(welcomescreen)
        widget.setCurrentIndex(widget.currentIndex()+1)



class imgsketchscreen(QDialog):
    def __init__(self):
        super(imgsketchscreen, self).__init__()
        loadUi("img-sketch.ui",self)
        self.home.clicked.connect(self.gotohome)
        #self.sketchsubmit.clicked.connect(self.haha)

        # file = open('geek.txt','w')
        # file.write(" ", sketchpath)
        # print('sketchpath')
        #self.exit.clicked.connect(self.close)
    def haha(self):
        mytext = self.textEdit.toPlainText()
        self.textEdit.setPlainText(mytext)
        with open('file/somefile.txt', 'w+') as f:
            f.write(mytext)
            f.close

       # os.system("sketch.py 1")
        # command="python sketch.py"
        # subprocess.Popen(command)


    def save_text(self):
        text, ok = self.le.text()
        filename=self.textEdit.toPlainText(self, 'Save File', '.')
       # filename = QtWidget.QFileDialog.getSaveFileName(self, 'Save File', '.')
        fname = open(filename, 'w')
        fname.write(self.le.setText(str(text)))
        fname.close()

    def gotohome(self):
        welcomescreen=WelcomeScreen()
        widget.addWidget(welcomescreen)
        widget.setCurrentIndex(widget.currentIndex()+1)

class handwritingscreen(QDialog):
    def __init__(self):
        super(handwritingscreen, self).__init__()
        loadUi("text-handwriting.ui",self)
        self.home.clicked.connect(self.gotohome)
        self.convert.clicked.connect(self.goconvert)


    def goconvert(self):
        mytext = self.textEdit_2.toPlainText()
        with open('textinput.txt', 'w') as filef:
            filef.write(str(mytext))



        os.system('python txttohandwriting.py')
        os.system('image_handwritten.pdf')

    def gotohome(self):
        welcomescreen=WelcomeScreen()
        widget.addWidget(welcomescreen)
        widget.setCurrentIndex(widget.currentIndex()+1)



def pushButton_3_clicked():
    print("Opening Image to Sketch")
    command="python sketch.py"
    subprocess.Popen(command)


#main
app = QApplication(sys.argv)
welcome = WelcomeHome()
widget = QtWidgets.QStackedWidget()
widget.addWidget(welcome)
widget.setFixedHeight(800)
widget.setFixedWidth(1080)
widget.show()
try:
    sys.exit(app.exec_())
except:
    print("Exiting")
```

imgtotxt.py

```
from PIL import Image
from pytesseract import pytesseract

#Define path to tessaract.exe
path_to_tesseract = r"Tesseract-OCR\tesseract.exe"

#Define path to image
path_to_image = 'text.png'

#Point tessaract_cmd to tessaract.exe
pytesseract.tesseract_cmd = path_to_tesseract

#Open image with PIL
img = Image.open(path_to_image)

#Extract text from image
text = pytesseract.image_to_string(img)

print(text)
```

sketch.py

```
import cv2
print(cv2.__version__)
imagepath=input("Drag and drop the Image ->")
# imagepath = open("somefile.txt", mode='r', encoding='utf-8')
# path=imagepath.read()
# imagepath.close()
# print(path)
image = cv2.imread(imagepath)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
invert=cv2.bitwise_not(gray_image)
blur=cv2.GaussianBlur(invert, (21, 21), 0)
invertedblur = cv2.bitwise_not(blur)

pencil_sketch = cv2.divide(gray_image, invertedblur,scale=225.0)
cv2.imwrite("output.png",pencil_sketch)
cv2.imshow("original image", image)
#cv2.waitKey()
cv2.imshow("pencil sketch", pencil_sketch)
cv2.waitKey()
```

handgesture.py

```
import cv2
import mediapipe as mp
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np
import cv2
import mediapipe as mp
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np

cap = cv2.VideoCapture(0) #Checks for camera

mpHands = mp.solutions.hands #detects hand/finger
hands = mpHands.Hands()   #complete the initialization configuration of hands
mpDraw = mp.solutions.drawing_utils

#To access speaker through the library pycaw
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volbar=400
volper=0

volMin,volMax = volume.GetVolumeRange()[:2]

while True:
    success,img = cap.read() #If camera works capture an image
    imgRGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Convert to rgb

    #Collection of gesture information
    results = hands.process(imgRGB) #completes the image processing.

    lmList = [] #empty list
    if results.multi_hand_landmarks: #list of all hands detected.
        #By accessing the list, we can get the information of each hand's corresponding flag bit
        for handlandmark in results.multi_hand_landmarks:
            for id,lm in enumerate(handlandmark.landmark): #adding counter and returning it
                # Get finger joint points
                h,w,_ = img.shape
                cx,cy = int(lm.x*w),int(lm.y*h)
                lmList.append([id,cx,cy]) #adding to the empty list 'lmList'
            mpDraw.draw_landmarks(img,handlandmark,mpHands.HAND_CONNECTIONS)

    if lmList != []:
        #getting the value at a point
                        #x      #y
        x1,y1 = lmList[4][1],lmList[4][2]  #thumb
        x2,y2 = lmList[8][1],lmList[8][2]  #index finger
        #creating circle at the tips of thumb and index finger
        cv2.circle(img,(x1,y1),13,(255,0,0),cv2.FILLED) #image #fingers #radius #rgb
        cv2.circle(img,(x2,y2),13,(255,0,0),cv2.FILLED) #image #fingers #radius #rgb
        cv2.line(img,(x1,y1),(x2,y2),(255,0,0),3)  #create a line b/w tips of index finger and thumb

        length = hypot(x2-x1,y2-y1) #distance b/w tips using hypotenuse
 # from numpy we find our length,by converting hand range in terms of volume range ie b/w -63.5 to 0
        vol = np.interp(length,[30,350],[volMin,volMax])
        volbar=np.interp(length,[30,350],[400,150])
        volper=np.interp(length,[30,350],[0,100])


        print(vol,int(length))
        volume.SetMasterVolumeLevel(vol, None)

        # Hand range 30 - 350
        # Volume range -63.5 - 0.0
        #creating volume bar for volume level
        cv2.rectangle(img,(50,150),(85,400),(0,0,255),4) # vid ,initial position ,ending position ,rgb
,thickness
        cv2.rectangle(img,(50,int(volbar)),(85,400),(0,0,255),cv2.FILLED)
        cv2.putText(img,f"{int(volper)}%",(10,40),cv2.FONT_ITALIC,1,(0, 255, 98),3)
        #tell the volume percentage ,location,font of text,length,rgb color,thickness
    cv2.imshow('Image',img) #Show the video
    if cv2.waitKey(1) & 0xff==ord(' '): #By using spacebar delay will stop
        break

cap.release()     #stop cam
cv2.destroyAllWindows() #close window
```

txttohandwriting.py

```
from PIL import Image
BG = Image.open("myfont/bg.png")
sizeOfSheet =BG.width
gap, _  = 0,0
allowedChars = 'qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM,.-?!()
1234567890'
def writee(char):
    global gap, _
    if char == '\n':
        pass
    else:
        char.lower()
        cases = Image.open("myfont/%s.png"%char)
        BG.paste(cases, (gap, _))
        size = cases.width
        gap += size
        del cases

def letterwrite(word):
    global gap, _
    if gap > sizeOfSheet - 95*(len(word)):
        gap = 0
        _ += 200
    for letter in word:
        if letter in allowedChars:
            if letter.islower():
                pass
            elif letter.isupper():
                letter = letter.lower()
                letter += 'upper'
            elif letter == '.':
                letter = "fullstop"
            elif letter == '!':
                letter = 'exclamation'
            elif letter == '?':
                letter = 'question'
            elif letter == ',':
                letter = 'comma'
            elif letter == '(':
                letter = 'braketop'
            elif letter == ')':
                letter = 'braketcl'
            elif letter == '-':
                letter = 'hiphen'
            writee(letter)
def worddd(Input):
    wordlist=Input.split(' ')
    for i in wordlist:
        letterwrite(i)
        writee('space')
if __name__ == '__main__':
    try:
        with open('textinput.txt', 'r') as file:
            data = file.read().replace('\n', '')
        l=len(data)
        nn=len(data)//600
        chunks, chunk_size = len(data), len(data)//(nn+1)
        p=[ data[i:i+chunk_size] for i in range(0, chunks, chunk_size) ]

        for i in range(0,len(p)):
            worddd(p[i])
            writee('\n')
            BG.save('%dhandwritten.png'%i)
            BG1= Image.open("myfont/bg.png")
            BG=BG1
            gap = 0
            _ =0
    except ValueError as E:
        print("{}\nTry again".format(E))

from fpdf import FPDF
from PIL import Image

imagelist=[]
for i in range(0,len(p)):
    imagelist.append('%dhandwritten.png'%i)

#Converting images to pdf
#Source:https://datatofish.com/images-to-pdf-python/


def pdf_creation(PNG_FILE,flag=False):
    rgba = Image.open(PNG_FILE)
    rgb = Image.new('RGB', rgba.size, (255, 255, 255))  # white background
    rgb.paste(rgba, mask=rgba.split()[3])               # paste using alpha channel as mask
    rgb.save('image_handwritten.pdf', append=flag)  #Now save multiple images in same pdf file

#First create a pdf file if not created
pdf_creation(imagelist.pop(0))

#Now I am opening each images and converting them to pdf
#Appending them to pdfs
for PNG_FILE in imagelist:
    pdf_creation(PNG_FILE,flag=True)
```

AIVirtualMouse.py

```
import cv2
import numpy as np
import HandTrackingModule as htm
import time
import autopy

######################
wCam, hCam = 640, 480
frameR = 100     #Frame Reduction
smoothening = 7  #random value
######################

pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()

# print(wScr, hScr)

while True:
    # Step1: Find the landmarks
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)

    # Step2: Get the tip of the index and middle finger
    if len(lmList) != 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]

        # Step3: Check which fingers are up
        fingers = detector.fingersUp()
        cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR),
                      (255, 0, 255), 2)

        # Step4: Only Index Finger: Moving Mode
        if fingers[1] == 1 and fingers[2] == 0:

            # Step5: Convert the coordinates
            x3 = np.interp(x1, (frameR, wCam-frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam-frameR), (0, hScr))

            # Step6: Smooth Values
            clocX = plocX + (x3 - plocX) / smoothening
            clocY = plocY + (y3 - plocY) / smoothening

            # Step7: Move Mouse
            autopy.mouse.move(wScr - clocX, clocY)
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            plocX, plocY = clocX, clocY

        # Step8: Both Index and middle are up: Clicking Mode
        if fingers[1] == 1 and fingers[2] == 1:
```

```
        # Step9: Find distance between fingers
        length, img, lineInfo = detector.findDistance(8, 12, img)

        # Step10: Click mouse if distance short
        if length < 40:
            cv2.circle(img, (lineInfo[4], lineInfo[5]), 15, (0, 255, 0), cv2.FILLED)
            autopy.mouse.click()

    # Step11: Frame rate
    cTime = time.time()
    fps = 1/(cTime-pTime)
    pTime = cTime
    cv2.putText(img, str(int(fps)), (28, 58), cv2.FONT_HERSHEY_PLAIN, 3, (255, 8, 8), 3)

    # Step12: Display
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

QR-Code-master/custom_qr.py

```
# import the necessary packages
from imutils.video import VideoStream
from pyzbar import pyzbar
import argparse
import datetime
import imutils
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", type=str, default="QR-Code-master/barcodes.csv",
	help="path to output CSV file containing barcodes")
args = vars(ap.parse_args())

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
# vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# open the output CSV file for writing and initialize the set of
# barcodes found thus far
csv = open(args["output"], "w")
found = set()


# loop over the frames from the video stream
while True:
	# grab the frame from the threaded video stream and resize it to
	# have a maximum width of 400 pixels
	frame = vs.read()
	frame = imutils.resize(frame, width=400)

	# find the barcodes in the frame and decode each of the barcodes
	barcodes = pyzbar.decode(frame)

	# loop over the detected barcodes
	for barcode in barcodes:
		# extract the bounding box location of the barcode and draw
		# the bounding box surrounding the barcode on the image
		(x, y, w, h) = barcode.rect
		cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

		# the barcode data is a bytes object so if we want to draw it
		# on our output image we need to convert it to a string first
		barcodeData = barcode.data.decode("utf-8")
		barcodeType = barcode.type

		# draw the barcode data and barcode type on the image
		text = "{} ({})".format(barcodeData, barcodeType)
		cv2.putText(frame, text, (x, y - 10),
			cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

		# if the barcode text is currently not in our CSV file, write
		# the timestamp + barcode to disk and update the set
		if barcodeData not in found:
```

```
                    csv.write("{},{}\n".format(datetime.datetime.now(),
                        barcodeData))
                    csv.flush()
                    found.add(barcodeData)

        # show the output frame
        cv2.imshow("Image Process Assistant- QR Scanner", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

# close the output CSV file do a bit of cleanup
print("[INFO] cleaning up...")
csv.close()
cv2.destroyAllWindows()
vs.stop()
```

6.2 Graphical UI Code

welcome.ui

```
# -*- coding: utf-8 -*-

################################################################################
#######
## Form generated from reading UI file 'welcomeJxwJSd.ui'
##
## Created by: Qt User Interface Compiler version 5.15.2
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################
#######

from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *


class Ui_Dialog(object):
    def setupUi(self, Dialog):
        if not Dialog.objectName():
            Dialog.setObjectName(u"Dialog")
        Dialog.resize(1080, 800)
        Dialog.setMinimumSize(QSize(1080, 800))
        self.bgwidget = QWidget(Dialog)
        self.bgwidget.setObjectName(u"bgwidget")
        self.bgwidget.setGeometry(QRect(0, 0, 1080, 800))
        self.bgwidget.setMinimumSize(QSize(1080, 800))
        self.bgwidget.setMaximumSize(QSize(770, 550))
        self.bgwidget.setStyleSheet(u"QWidget#bgwidget{\n"
"background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1, stop:0 rgba(137, 89,
218, 255), stop:1 rgba(255, 255, 255, 255));\n"
"}")
        self.label_3 = QLabel(self.bgwidget)
        self.label_3.setObjectName(u"label_3")
        self.label_3.setGeometry(QRect(370, -20, 491, 191))
        self.label_3.setStyleSheet(u" color:rgb(255, 255, 255);\n"
"font: 75 italic 50pt \"Comic Sans MS\";")
        self.label_4 = QLabel(self.bgwidget)
        self.label_4.setObjectName(u"label_4")
        self.label_4.setGeometry(QRect(330, 180, 641, 121))
        self.label_4.setStyleSheet(u"border-radius:50px;\n"
"font: 30pt \"Cooper Black\";\n"
"\n"
"\n"
"")
        self.start = QPushButton(self.bgwidget)
        self.start.setObjectName(u"start")
        self.start.setGeometry(QRect(460, 390, 181, 71))
        self.start.setStyleSheet(u"border-radius:20px;\n"
"background-color: rgb(28, 115, 255);\n"
"font: 32pt \"MS Shell Dlg 2\";")
        self.progressBar = QProgressBar(self.bgwidget)
        self.progressBar.setObjectName(u"progressBar")
        self.progressBar.setGeometry(QRect(320, 620, 441, 41))
        self.progressBar.setStyleSheet(u"QProgressBar{\n"
"border-radius:20px;\n"
"background-color: #ffe7fa;\n"
"}\n"
"QProgressBar::chunk{\n"
"border-radius:20px;\n"
"background-color:#a20cff\n"
"}")
        self.progressBar.setValue(24)
        self.pushButton = QPushButton(self.bgwidget)
        self.pushButton.setObjectName(u"pushButton")
        self.pushButton.setGeometry(QRect(1050, 10, 31, 23))
        self.pushButton.setStyleSheet(u"QPushButton{\n"
"           font: 63 16pt \"Bahnschrift SemiBold\";\n"
"           color: rgb(255, 255, 255);\n"
"border-radius:15px;\n"
"\n"
"}\n"
"")

        self.retranslateUi(Dialog)

        QMetaObject.connectSlotsByName(Dialog)
    # setupUi

    def retranslateUi(self, Dialog):
        Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Dialog", None))
        self.label_3.setText(QCoreApplication.translate("Dialog", u"WELCOME", None))
        self.label_4.setText(QCoreApplication.translate("Dialog", u"IMAGE PROCESSING\n"
"          ASSISTANT", None))
        self.start.setText(QCoreApplication.translate("Dialog", u"Start", None))
        self.pushButton.setText(QCoreApplication.translate("Dialog", u"X", None))
    # retranslateUi
```

mainwindow.ui

```
# -*- coding: utf-8 -*-

################################################################################
## Form generated from reading UI file 'mainhomeAkXPsK.ui'
##
## Created by: Qt User Interface Compiler version 5.15.2
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################

from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *


class Ui_Dialog(object):
    def setupUi(self, Dialog):
        if not Dialog.objectName():
            Dialog.setObjectName(u"Dialog")
        Dialog.resize(1080, 800)
        self.bgwidget = QWidget(Dialog)
        self.bgwidget.setObjectName(u"bgwidget")
        self.bgwidget.setGeometry(QRect(0, 0, 1080, 800))
        self.bgwidget.setStyleSheet(u"background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1,
y2:1, stop:0 rgba(137, 89, 218, 255), stop:1 rgba(255, 255, 255, 255));")
        self.home = QPushButton(self.bgwidget)
        self.home.setObjectName(u"home")
        self.home.setGeometry(QRect(310, 640, 131, 41))
        self.home.setStyleSheet(u"border-style: outset;\n"
"border-width: 2px;\n"
"border-radius: 2px;\n"
"border-color: #393939;\n"
"font: 75 8pt \"MS Shell Dlg 2\";")
        self.reset = QPushButton(self.bgwidget)
        self.reset.setObjectName(u"reset")
        self.reset.setGeometry(QRect(490, 640, 131, 41))
        self.reset.setStyleSheet(u"border-style: outset;\n"
"border-width: 2px;\n"
"border-radius: 2px;\n"
"border-color: #393939;")
        self.exit = QPushButton(self.bgwidget)
        self.exit.setObjectName(u"exit")
        self.exit.setGeometry(QRect(660, 640, 131, 41))
        self.exit.setStyleSheet(u"border-style: outset;\n"
"border-width: 2px;\n"
"border-radius: 2px;\n"
"border-color: #393939;")
        self.label = QLabel(Dialog)
        self.label.setObjectName(u"label")
        self.label.setGeometry(QRect(300, 110, 511, 61))
        font = QFont()
        font.setPointSize(18)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setMouseTracking(False)
        self.label.setLayoutDirection(Qt.LeftToRight)
        self.label.setAutoFillBackground(False)
        self.label.setStyleSheet(u"border-radius:50px;")
        self.pushButton_6 = QPushButton(Dialog)
        self.pushButton_6.setObjectName(u"pushButton_6")
        self.pushButton_6.setGeometry(QRect(100, 440, 180, 80))
        font1 = QFont()
        font1.setFamily(u"Eras Demi ITC")
        font1.setBold(True)
        font1.setItalic(False)
        font1.setWeight(75)
        font1.setStyleStrategy(QFont.PreferDefault)
        self.pushButton_6.setFont(font1)
        self.pushButton_6.setCursor(QCursor(Qt.OpenHandCursor))
        self.pushButton_6.setAutoFillBackground(False)
        self.pushButton_6.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")
        self.pushButton_4 = QPushButton(Dialog)
        self.pushButton_4.setObjectName(u"pushButton_4")
        self.pushButton_4.setGeometry(QRect(470, 440, 180, 80))
        self.pushButton_4.setFont(font1)
        self.pushButton_4.setCursor(QCursor(Qt.OpenHandCursor))
        self.pushButton_4.setAutoFillBackground(False)
        self.pushButton_4.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")
        self.pushButton_2 = QPushButton(Dialog)
        self.pushButton_2.setObjectName(u"pushButton_2")
        self.pushButton_2.setGeometry(QRect(100, 220, 180, 80))
        self.pushButton_2.setFont(font1)
        self.pushButton_2.setCursor(QCursor(Qt.OpenHandCursor))
        self.pushButton_2.setAutoFillBackground(False)
        self.pushButton_2.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")
        self.pushButton_3 = QPushButton(Dialog)
        self.pushButton_3.setObjectName(u"pushButton_3")
        self.pushButton_3.setGeometry(QRect(470, 220, 180, 80))
        self.pushButton_3.setFont(font1)
        self.pushButton_3.setCursor(QCursor(Qt.OpenHandCursor))
        self.pushButton_3.setAutoFillBackground(False)
        self.pushButton_3.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")
        self.mic = QPushButton(Dialog)
        self.mic.setObjectName(u"mic")
        self.mic.setGeometry(QRect(500, 310, 111, 121))
        font2 = QFont()
        font2.setFamily(u"Eras Demi ITC")
        font2.setPointSize(12)
        font2.setBold(False)
        font2.setItalic(False)
        font2.setWeight(50)
        font2.setStyleStrategy(QFont.NoAntialias)
        self.mic.setFont(font2)
        self.mic.setCursor(QCursor(Qt.OpenHandCursor))
        self.mic.setAutoFillBackground(False)
        self.mic.setStyleSheet(u"font: 12pt \"Eras Demi ITC\";\n"
";\n"
"border-color: rgb(0, 0, 0);\n"
"\n"
"border-radius:55px;\n"
"border-width: 6px;")
        icon = QIcon()
        icon.addFile(u"mic.png", QSize(), QIcon.Normal, QIcon.Off)
        self.mic.setIcon(icon)
        self.mic.setIconSize(QSize(50, 50))
        self.mic.setCheckable(True)
        self.mic.setAutoRepeatInterval(107)
        self.label_2 = QLabel(Dialog)
        self.label_2.setObjectName(u"label_2")
        self.label_2.setGeometry(QRect(80, 0, 1050, 131))
        font3 = QFont()
        font3.setFamily(u"Cooper Black")
        font3.setPointSize(40)
        font3.setBold(False)
        font3.setItalic(False)
        font3.setWeight(50)
        self.label_2.setFont(font3)
        self.label_2.setMouseTracking(False)
        self.label_2.setLayoutDirection(Qt.LeftToRight)
        self.label_2.setAutoFillBackground(False)
        self.label_2.setStyleSheet(u"border-radius:50px;\n"
"font: 40pt \"Cooper Black\";\n"
"\n"
"\n"
"")
        self.handwritingbtn = QPushButton(Dialog)
        self.handwritingbtn.setObjectName(u"handwritingbtn")
        self.handwritingbtn.setGeometry(QRect(820, 220, 180, 80))
        self.handwritingbtn.setFont(font1)
        self.handwritingbtn.setCursor(QCursor(Qt.OpenHandCursor))
        self.handwritingbtn.setAutoFillBackground(False)
        self.handwritingbtn.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")
        self.pushButton_7 = QPushButton(Dialog)
        self.pushButton_7.setObjectName(u"pushButton_7")
        self.pushButton_7.setGeometry(QRect(820, 440, 180, 80))
        self.pushButton_7.setFont(font1)
        self.pushButton_7.setCursor(QCursor(Qt.OpenHandCursor))
        self.pushButton_7.setAutoFillBackground(False)
        self.pushButton_7.setStyleSheet(u"QPushButton{\n"
"    background-color: #e6e6e6;\n"
"    border-style: outset;\n"
"    border-width: 6px;\n"
"    border-radius: 20px;\n"
"    border-color: black;\n"
"    font: bold 14px;\n"
"    \n"
"}")

        self.retranslateUi(Dialog)

        self.mic.setDefault(False)


        QMetaObject.connectSlotsByName(Dialog)
    # setupUi

    def retranslateUi(self, Dialog):
        Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Dialog", None))
        self.home.setText(QCoreApplication.translate("Dialog", u"HOME", None))
        self.reset.setText(QCoreApplication.translate("Dialog", u"RESET", None))
        self.exit.setText(QCoreApplication.translate("Dialog", u"EXIT", None))
        self.label.setText(QCoreApplication.translate("Dialog", u"CHOOSE YOUR IMAGE
PROCESSING TOOL", None))
        self.pushButton_6.setText(QCoreApplication.translate("Dialog", u"Gesture Mouse", None))
        self.pushButton_4.setText(QCoreApplication.translate("Dialog", u"Gesture Control", None))
        self.pushButton_2.setText(QCoreApplication.translate("Dialog", u"Image to Text", None))
        self.pushButton_3.setText(QCoreApplication.translate("Dialog", u"Image to Sketch", None))
        self.mic.setText("")
        self.label_2.setText(QCoreApplication.translate("Dialog", u"IMAGE PROCESSING ASSISTANT",
None))
        self.handwritingbtn.setText(QCoreApplication.translate("Dialog", u"Text to\n"
" Handwriting", None))
        self.pushButton_7.setText(QCoreApplication.translate("Dialog", u"QR READER", None))
    # retranslateUi
```

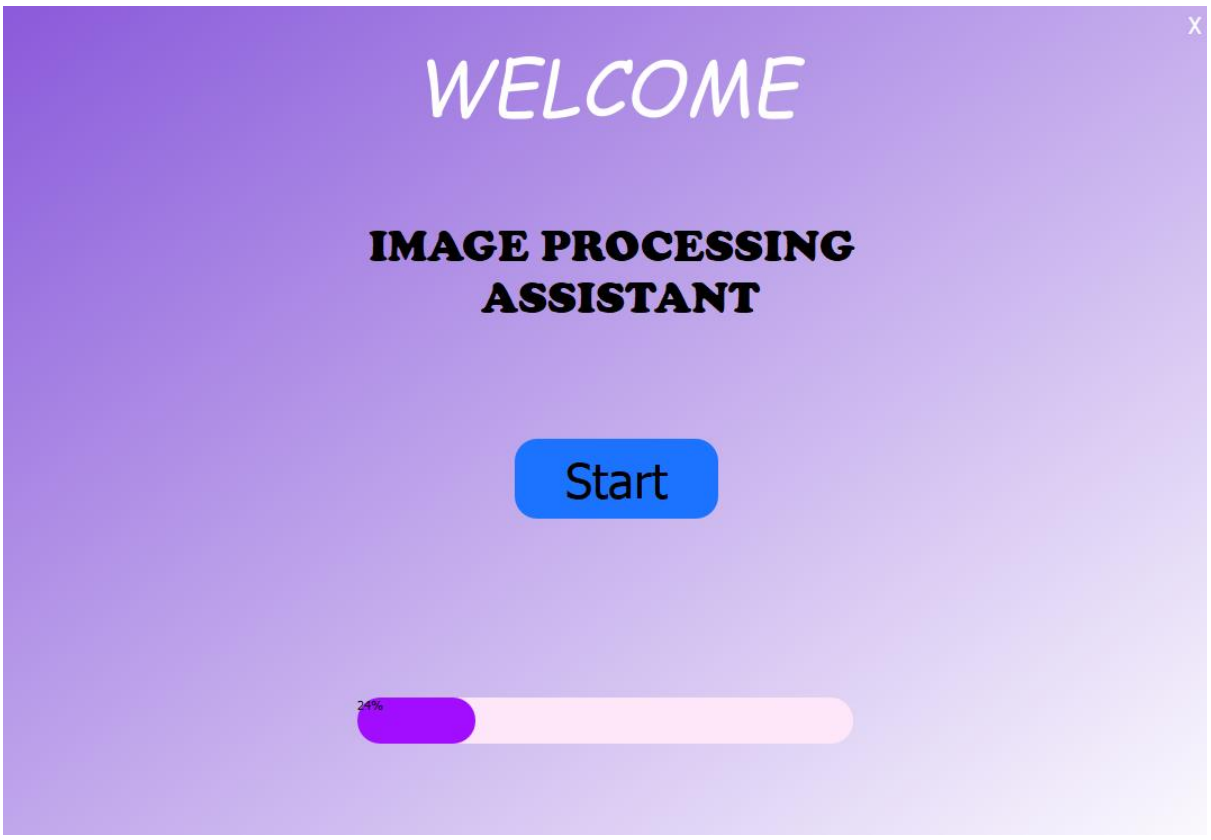
text-handwriting.ui

img-text.ui

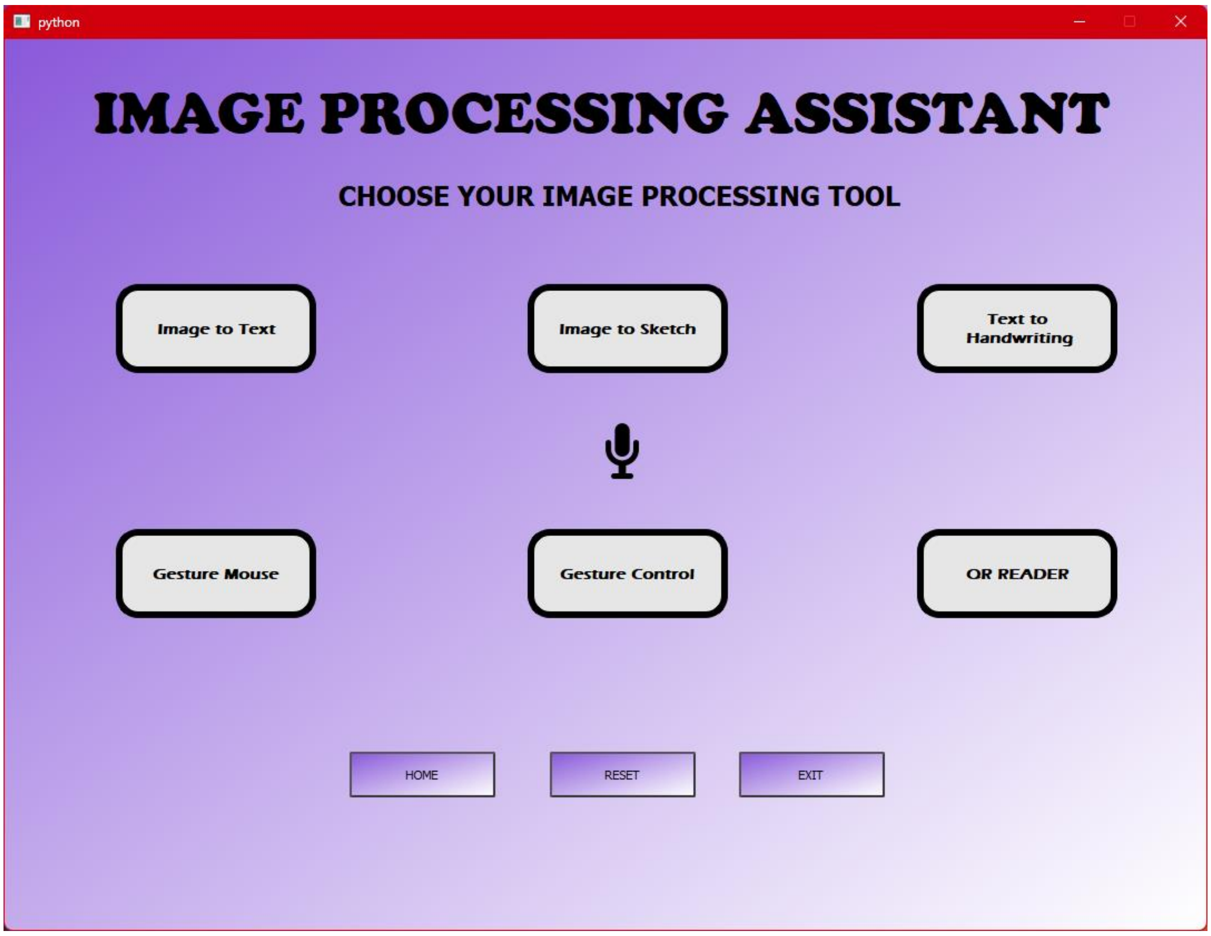
img-sketch.ui

9. OUTPUT

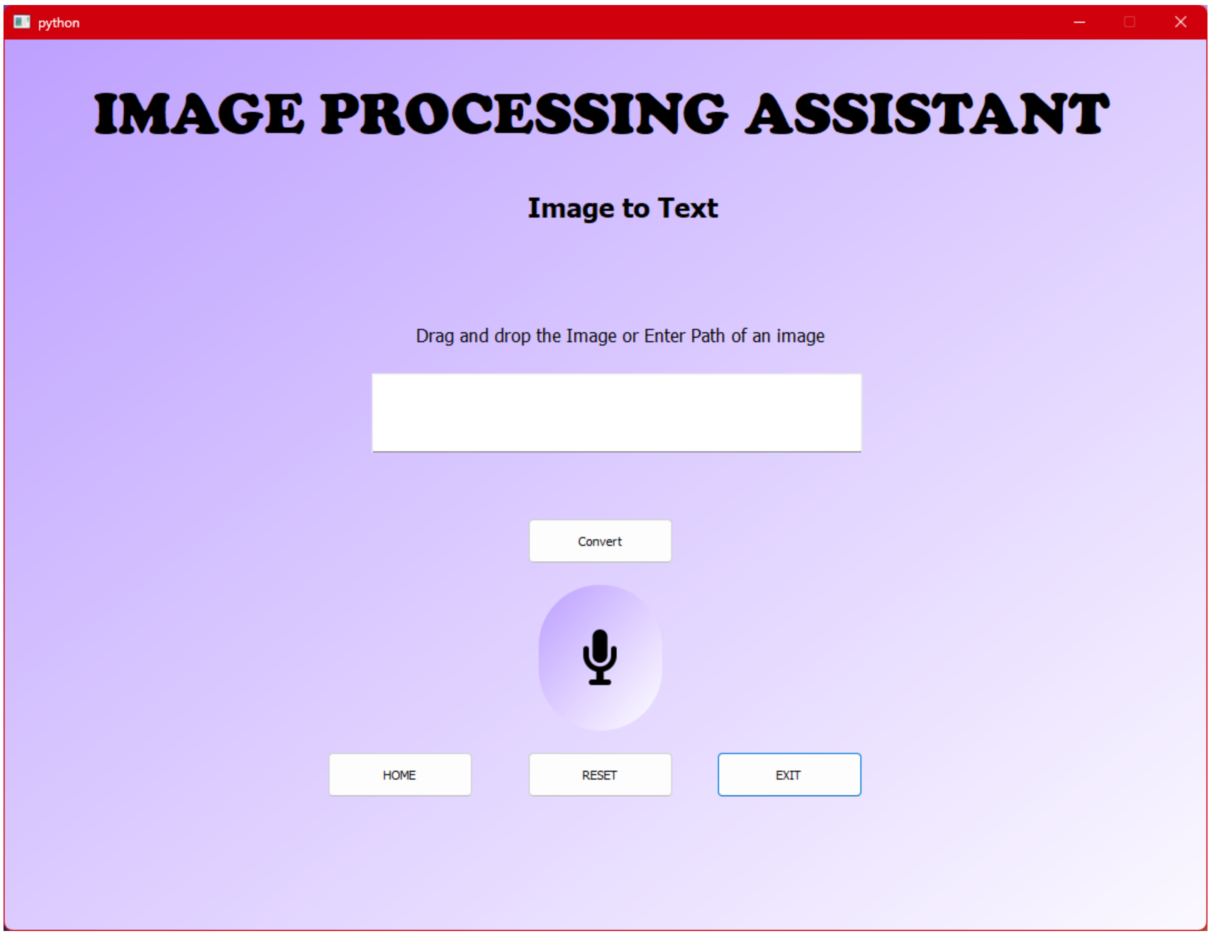
Welcome Screen



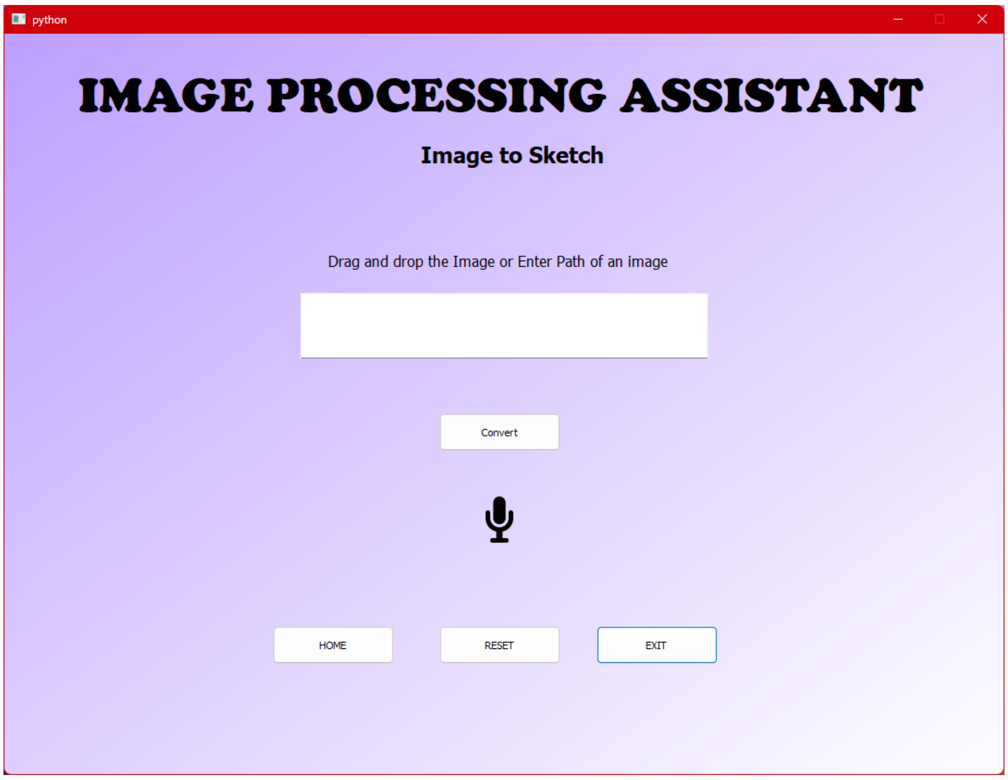
Main Window


python
IMAGE PROCESSING ASSISTANT
CHOOSE YOUR IMAGE PROCESSING TOOL
Image to Text
Image to Sketch
Text to Handwriting
Gesture Mouse
Gesture Control
QR READER
HOME
RESET
EXIT

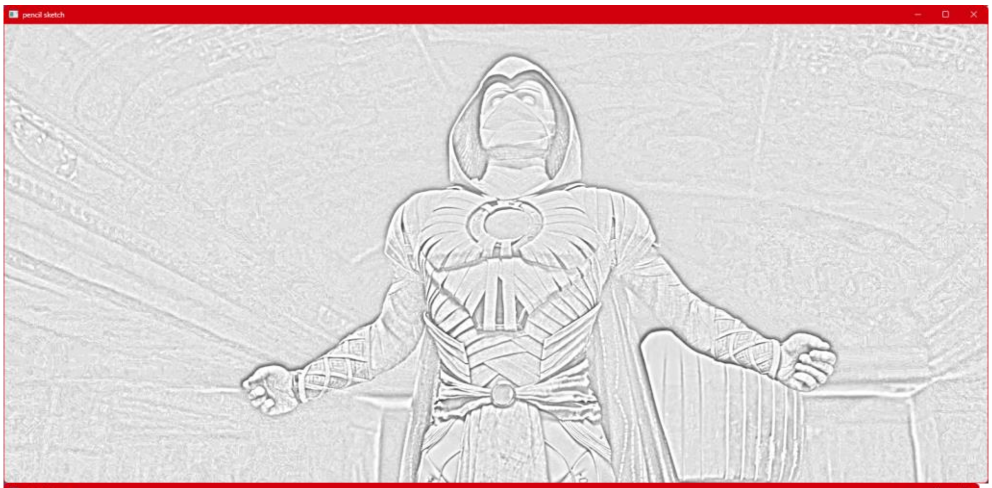
Button 1: Image to Text

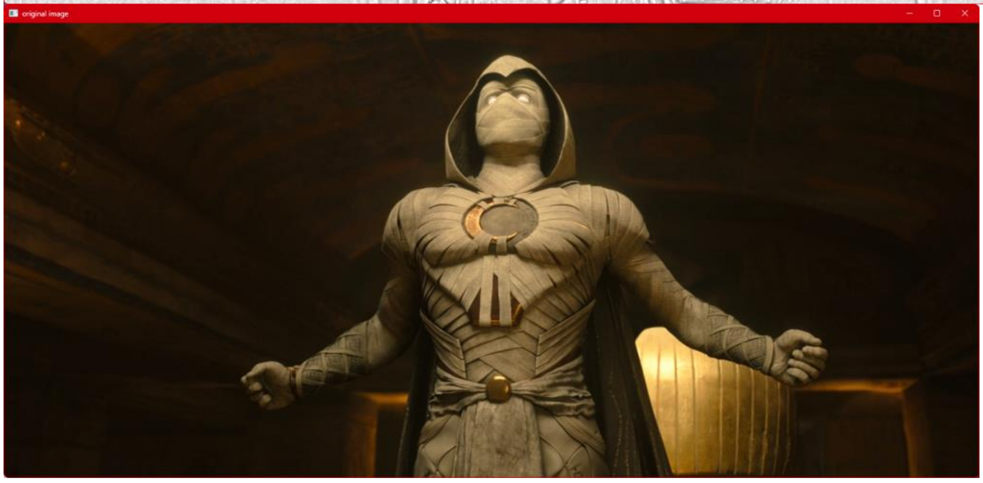

python
IMAGE PROCESSING ASSISTANT
Image to Text
Drag and drop the Image or Enter Path of an image
Convert
HOME
RESET
EXIT

Button 2: Image to Sketch

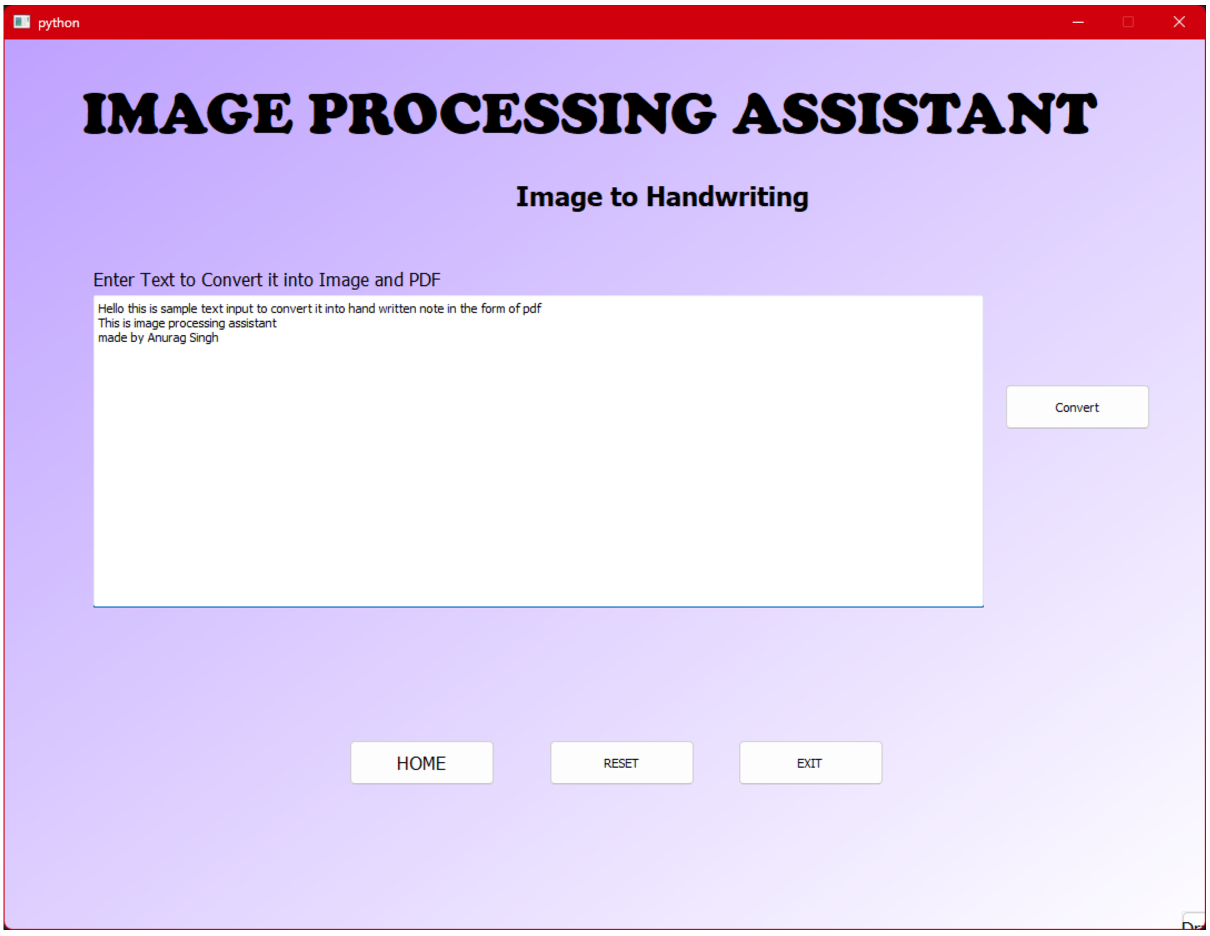

python
IMAGE PROCESSING ASSISTANT
Image to Sketch
Drag and drop the Image or Enter Path of an image
Convert
HOME
RESET
EXIT

OUTPUT :

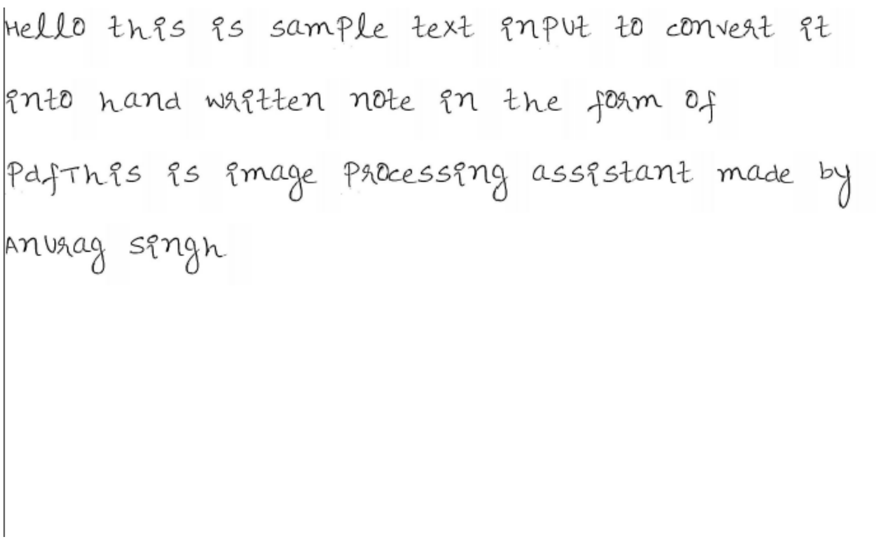




Button 3: Image to Handwriting


python
IMAGE PROCESSING ASSISTANT
Image to Handwriting
Enter Text to Convert it into Image and PDF
Convert
HOME
RESET
EXIT

Hello this is sample text input to convert it into hand written note in the form of pdf
This is image processing assistant
made by Anurag Singh



Hello this is sample text input to convert it into hand written note in the form of PdfThis is image processing assistant made by Anurag singh

Button 5: Gesture Control

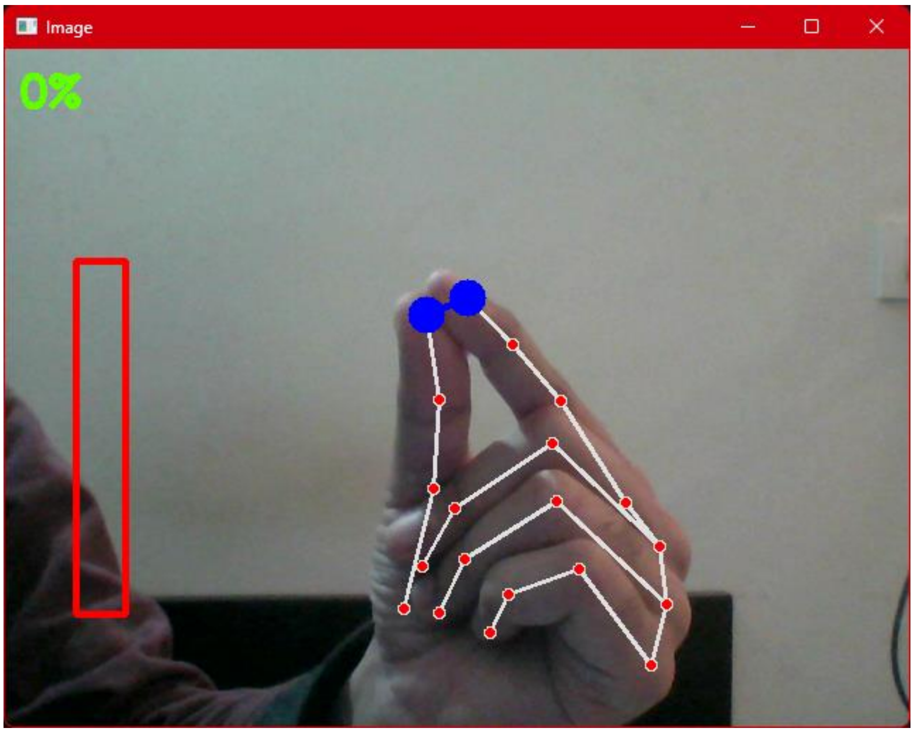


Image
0%

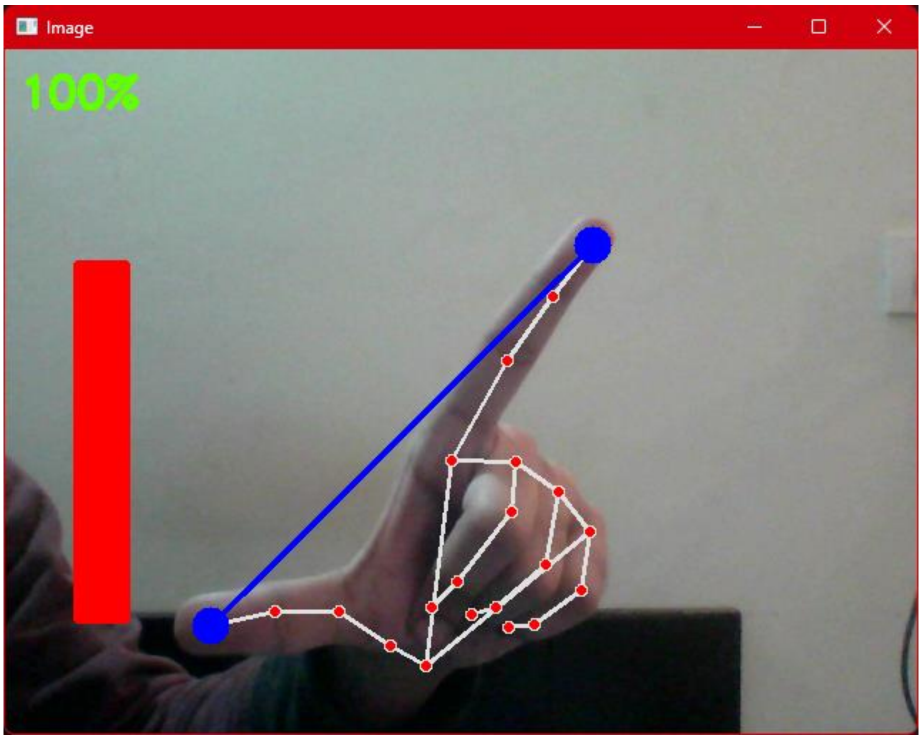


Image
100%

Button 6: QR Reader

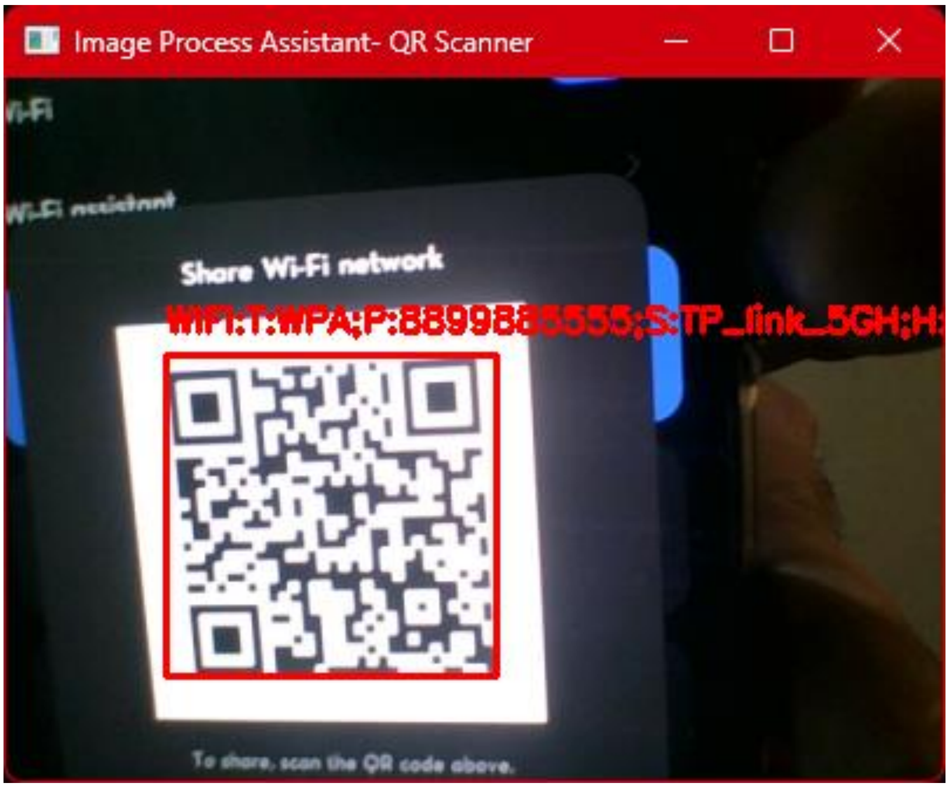


Image Process Assistant- QR Scanner
Wi-Fi
Wi-Fi assistant
Share Wi-Fi network
To share, scan the QR code above.

WIFI:T:WPA;P:8899885555;S:TP_link_5GH;H:

Conclusion

With the advent of fast and cheap machines, digital image processing has become a very highly demanded field of study and practice. It provides solutions to various real-life applications in an economical way. Various techniques have been developed to build intelligent systems; many of them are in progress at various facilities internationally. This chapter has provided some introductory notes on image processing, its brief history, methodologies, tasks, software, and applications. It will help to kick start the community interested to have some knowhow on the image processing subject. The future of digital image processing has a high probability to contribute toward the build of smart and intelligent world in terms of health, education, defense, traffic, homes, offices, cities, etc.