SUMMER TRAINING REPORT

Gesture Keyboard

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in

Information Technology

Guide:
Mr. D.K. Mishra
(Assistant Professor
IT Department)

Submitted by:
Gaurav Singh(00413303117)
Srishtik Batra(35413303117)
Shubham(43313303117)
Puneet(50313303117)

HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT

HAMIDPUR, DELHI 110036

Affiliated to

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

Sector - 16C Dwarka, Delhi - 110075, India

2017-21

DECLARATION

We, students of B.Tech (I.T.) hereby declare that the minor project entitled "Gesture Keyboard" which is submitted to Department of Information Technology, HMR Institute of Technology & Management, Hamidpur Delhi, affiliated to Guru Gobind Singh Indraprastha University,

Dwarka(New Delhi) in partial fulfilment of requirement for the award of the degree of Bachelor of Technology in Information Technology has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. The list of members involved in the project is listed below: -

S.No.	Student Name	Enrolment Number	Student Signature
1.	Gaurav Singh	00413303117	
2.	Srishtik Batra	35413303117	
3.	Shubham	43313303117	
4.	Puneet	50313303117	

This is to certify that the above statement made by the candidate(s) is correct to the

best of my knowledge.

New Delhi

Date: December 04,2019

Mr. D.K. Mishra

(Assistant Professor
IT Department)

Dr. Shafiq

(Head of Department)

Information Technology

HMRITM Hamidpur, New Delhi

ACKNOWLEDGEMENT

We are grateful to the authorities of IT DEPARTMENT, HMRITM for having permitted us to go ahead with the project on “Gesture Keyboard”. We are particularly thankful to Mr. D.K Mishra and Dr. Shafiq for their valuable guidance and advice during the course of this project.

We are deeply indebted to our teacher, dearest and cherished Mr. D.K. Mishra for sparing his most precious time in providing guidance to us. Without his wise counsel, inestimable encouragement, it would have been difficult for us. Gratitude is also due to them for their constant guidance and direction in writing this piece of work.

Finally, we are grateful to all who have directly or indirectly contributed to the successful completion of our project.

Thank you

ABSTRACT

Hand gesture recognition is very significant for human-computer interaction. In this work, we present a novel real-time method for hand gesture recognition. In our framework, the hand region is extracted from the background with the background subtraction method. Then, the palm and fingers are segmented so as to detect and recognize the fingers. Finally, a rule classifier is applied to predict the labels of hand gestures. The experiments on the data set of 1300 images show that our method performs well and is highly efficient. Moreover, our method shows better performance than a state-of-art method on another data set of hand gestures.

Contents

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Gesture keyboard is a method of inputting text to a device without taps or clicks on each letter. The general idea is that a user would just press the mouse button and hold it down while dragging the pointer across the lettersin the desired word before releasingthe mouse button to complete the word. This can be implemented on touch devices as well where the mouse is simply replaced by the touch tracer. The keyboard should be able to determine which word was swiped1by the path of the input and possibly other variables such as speed and change of direction.With the rapidly growing market of mobile devices and devices that relyon touch input this input method is gathering attention from all around. Forsome userssuchdevices canbenefit substantiallyfrom gestureinput. It makes one-handed use of a phone much easier and isimplemented in many modern mobile devices. Swype is a well-known company which created a third party gesture keyboard and Google introduced this type of input as an option on all android devices as late as last year. Thiswas a response to the rapidly growing interest in third party applications that offered the gesture experience.Mobile text input is intensely researched and almost every human-computer interaction conference since the 1990s contain research on the topic.(Zhai & Kristensson, 2012).

1.1 Problem Statement

Thisstudy examineshowmachine learningmethodscan be used to create a gesture keyboardwith the purpose of findingapproaches that could enhance future gesture keyboards.The goalwasto develop and implement an algorithm based on machine learningmethodsand validate if it results in a viable gesture input keyboard.In order to test whether or not this goal has been reached the algorithm was evaluated from three perspectives:

Examiningthe performance with dictionaries of different sizes.

Examiningthe performancewith dictionaries of differentcategoriesof words.

Examining the usabilityregarding speed and accuracy.

In order to effectively analyze the potentialof machine learning in the context of gesture keyboards it was necessary to limit the scope of the study. Different machine learning approaches will been considered but only one will be selectedfor implementation. Moreover; a limiteddictionary is chosen with words testing different aspects of input in order to analyze performance on multiple levels.

CHAPTER 2

LIBRARIES USED

2.1 Numpy

NumPy is a library for the python Programing Language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high level Mathematics to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is Open Source Software and has many contributors.

2.1.1 History

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier. There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then. In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.In 2011, PyPy started

development on an implementation of the NumPy API for PyPy.It is not yet fully compatible with NumPy.

2.1.2 Traits

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are strided views on memory.[6] In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around,

giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.

2.1.3. Limitations

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement

package called Blaze attempts to overcome this limitation.

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba. Cython is a static-compiling alternative to these.

2.2 Math

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

2.3 OpenCV

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel[]). The library is cross-platform and free for use under the open-source BSD license.

2.3.1 History

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.

Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.

Advance vision-based commercial applications by making portable,performance-optimized,code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site. On May 2016, Intel signed an agreement to acquire Itseez, the leading developer of OpenCV.

CHAPTER 3

FUNCTIONS USED

3.1 imread

Loads an image from a file.

Python: cv2.imread(filename[, flags]) → retval

Python: cv.**LoadImage**(filename, iscolor=CV_LOAD_IMAGE_COLOR) → None

Python: cv.**LoadImageM**(filename, iscolor=CV_LOAD_IMAGE_COLOR) → None

Parameters:

1. filename – Name of file to be loaded.

2. flags –

Flags specifying the color type of a loaded image:

1. CV_LOAD_IMAGE_ANYDEPTH - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.

2. CV_LOAD_IMAGE_COLOR - If set, always convert image to the color one

3. CV_LOAD_IMAGE_GRAYSCALE - If set, always convert image to the grayscale one

4. >0 Return a 3-channel color image.

Note

In the current implementation the alpha channel, if any, is stripped from the output image. Use negative value if you need the alpha channel.

1. =0 Return a grayscale image.

2. <0 Return the loaded image as is (with alpha channel).

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL). Currently, the following file formats are supported:

1.Windows bitmaps - *.bmp, *.dib (always supported)

2.JPEG files - *.jpeg, *.jpg, *.jpe (see the *Notes* section)

3. JPEG 2000 files - *.jp2 (see the *Notes* section)
4. Portable Network Graphics - *.png (see the *Notes* section)
5. Portable image format - *.pbm, *.pgm, *.ppm (always supported)
6. Sun rasters - *.sr, *.ras (always supported)
7. TIFF files - *.tiff, *.tif (see the Notes section)

Note

The function determines the type of an image by the content, not by the file extension.

On Microsoft Windows* OS and MacOSX*, the codecs shipped with an OpenCV image (libjpeg, libpng, libtiff, and libjasper) are used by default. So, OpenCV can always read JPEGs, PNGs, and TIFFs. On MacOSX, there is also an option to use native MacOSX image readers. But beware that currently these native image loaders give images with different pixel values because of the color management embedded into MacOSX.

On Linux*, BSD flavors and other Unix-like open-source operating systems, OpenCV looks for codecs supplied with an OS image. Install the relevant packages (do not forget the development files, for example, “libjpeg-dev”, in Debian* and Ubuntu*) to get the codec support or turn on the OPENCV_BUILD_3RDPARTY_LIBS flag in CMake.

Note

In the case of color images, the decoded images will have the channels stored in B G R order.

3.2 VideoCapture

class VideoCapture

Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.

VideoCapture::release

Closes video file or capturing device.

Python: cv2.VideoCapture.**release**() → None

C: void **cvReleaseCapture**(CvCapture** **capture**)

The methods are automatically called by subsequent **VideoCapture::open()** and by VideoCapture destructor.

3.3 cvtColor

Converts an image from one color space to another.

Python: cv2.**cvtColor**(src, code[, dst[, dstCn]]) → dst

C: void **cvCvtColor**(const CvArr* **src**, CvArr* **dst**, int **code**)

Python: cv.**CvtColor**(src, dst, code) → None

Parameters:

1. src – input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.

2. dst – output image of the same size and depth as src.

3. code – color space conversion code (see the description below).

4. dstCn – number of channels in the destination image; if the parameter is 0, the number of the channels is

5. derived automatically from src and code .

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

0 to 255 for CV_8U images
0 to 65535 for CV_16U images
0 to 1 for CV_32F images

In case of linear transformations, the range does not matter. But in case of a non-linear transformation, an input RGB image should be normalized to the proper value range to get the correct results, for example, for RGB $\rightarrow$ L*u*v* transformation. For example, if you have a 32-bit floating-point image directly converted from an 8-bit image without any scaling, then it will have the 0..255 value range instead of 0..1 assumed by the function. So, before calling cvtColor , you need first to scale the image down:

```
img *= 1./255;
cvtColor(img, img, CV_BGR2Luv);
```

If you use cvtColor with 8-bit images, the conversion will have some information lost. For many applications, this will not be noticeable but it is recommended to use 32-bit images in applications that need the full range of colors or that convert an image before an operation and then convert back.

If conversion adds the alpha channel, its value will set to the maximum of corresponding channel range: 255 for CV_8U, 65535 for CV_16U, 1 for CV_32F.

The function can do the following transformations:

RGB $\leftrightarrow$ GRAY(CV_BGR2GRAY, CV_RGB2GRAY, CV_GRAY2BGR, CV_GRAY2RGB)
Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

and

$$\text{Gray to RGB[A]:} \quad R \leftarrow Y, G \leftarrow Y, B \leftarrow Y, A \leftarrow \max(\text{ChannelRange})$$

The conversion from a RGB image to gray is done with:

cvtColor(src, bwsrc, CV_RGB2GRAY);

More advanced channel reordering can also be done with **mixChannels()**.

RGB $\leftrightarrow$ CIE XYZ.Rec 709 with D65 white point
(CV_BGR2XYZ, CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB):

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} \leftarrow \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

X, Y and Z cover the whole value range (in case of floating-point images, Z may exceed 1).

RGB $\leftrightarrow$ YCrCb JPEG (or YCC)
(CV_BGR2YCrCb, CV_RGB2YCrCb, CV_YCrCb2BGR, CV_YCrCb2RGB)

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

$$Cr \leftarrow (R - Y) \cdot 0.713 + \text{delta}$$

$$Cb \leftarrow (B - Y) \cdot 0.564 + \text{delta}$$

$$R \leftarrow Y + 1.403 \cdot (Cr - \text{delta})$$

$$G \leftarrow Y - 0.714 \cdot (Cr - \text{delta}) - 0.344 \cdot (Cb - \text{delta})$$

$$B \leftarrow Y + 1.773 \cdot (Cb - \text{delta})$$

where

$$\text{delta} = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating-point images} \end{cases}$$

Y, Cr, and Cb cover the whole value range.

RGB $\leftrightarrow$ HSV
(CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB)

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - \min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V - \min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - \min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - \min(R, G, B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

8-bit images

$$V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2 \text{(to fit to 0 to 255)}$$

16-bit images (currently not supported)

$$V < -65535V, S < -65535S, H < -H$$

32-bit images

H, S, and V are left as is

RGB $\leftrightarrow$ HLS (CV_BGR2HLS, CV_RGB2HLS, CV_HLS2BGR, CV_HLS2RGB).

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V_{max} \leftarrow \max(R, G, B)$$

$$V_{min} \leftarrow \min(R, G, B)$$

$$L \leftarrow \frac{V_{max} + V_{min}}{2}$$

--

$$S \leftarrow \begin{cases} \frac{V_{max}-V_{min}}{V_{max}+V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max}-V_{min}}{2-(V_{max}+V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/S & \text{if } V_{max} = R \\ 120+60(B-R)/S & \text{if } V_{max} = G \\ 240+60(R-G)/S & \text{if } V_{max} = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq L \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

8-bit images

$$V \leftarrow 255 \cdot V, S \leftarrow 255 \cdot S, H \leftarrow H/2 \text{ (to fit to 0 to 255)}$$

16-bit images (currently not supported)

$$V < -65535 \cdot V, S < -65535 \cdot S, H < -H$$

32-bit images

H, S, V are left as is

RGB $\leftrightarrow$ CIE L*a*b* (CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB).

In case of 8-bit and 16-bitimages, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X \leftarrow X/X_n, \text{where} X_n = 0.950456$$

$$Z \leftarrow Z/Z_n, \text{where} Z_n = 1.088754$$

$$L \leftarrow \begin{cases} 116 * Y^{1/3} - 16 & \text{for } Y > 0.008856 \\ 903.3 * Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$a \leftarrow 500(f(X) - f(Y)) + delta$$

$$b \leftarrow 200(f(Y) - f(Z)) + delta$$

where

$$f(t) = \begin{cases} t^{1/3} & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{for } t \leq 0.008856 \end{cases}$$

and

$$\text{delta} = \begin{cases} 128 & \text{for 8-bit images} \\ 0 & \text{for floating-point images} \end{cases}$$

This outputs $0 \leq L \leq 100$, $-127 \leq a \leq 127$, $-127 \leq b \leq 127$. The values are then converted to the destination data type:

8-bit images

$$L \leftarrow L * 255/100, \quad a \leftarrow a + 128, \quad b \leftarrow b + 128$$

- 16-bit images (currently not supported)
- 32-bit images

L, a, and b are left as is

RGB $\leftrightarrow$ CIE L*u*v* (CV_BGR2Luv, CV_RGB2Luv, CV_Luv2BGR, CV_Luv2RGB).

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit 0 to 1 range.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$L \leftarrow \begin{cases} 116Y^{1/3} & \text{for } Y > 0.008856 \\ 903.3Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$u' \leftarrow 4 * X/(X + 15 * Y + 3Z)$$

$$v' \leftarrow 9 * Y/(X + 15 * Y + 3Z)$$

$$u \leftarrow 13 * L * (u' - u_n) \quad \text{where} \quad u_n = 0.19793943$$

$$v \leftarrow 13 * L * (v' - v_n) \quad \text{where} \quad v_n = 0.46831096$$

This output $0 \leq L \leq 100$, $-134 \leq u \leq 220$, $-140 \leq v \leq 122$.

The values are then converted to the destination data type:

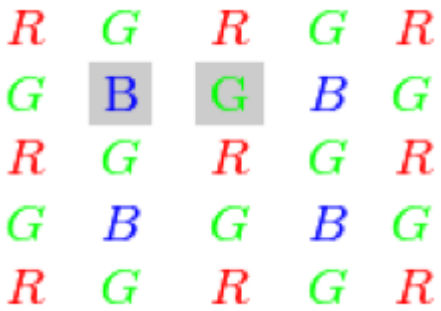
- 8-bit images

$$L \leftarrow 255/100L,\ u \leftarrow 255/354(u + 134),\ v \leftarrow 255/262(v + 140)$$

- 16-bit images (currently not supported)
- 32-bit images

  L, u, and v are left as is

Bayer $\rightarrow$ RGB
(CV_BayerBG2BGR, CV_BayerGB2BGR, CV_BayerRG2BGR, CV_BayerGR2BGR, CV_BayerBG2RGB, CV_BayerGB2RGB, CV_BayerRG2RGB,CV_BayerGR2RGB). The Bayer pattern is widely used in CCD and CMOS cameras. It enables you to get color pictures from a single plane where R,G, and B pixels (sensors of a particular component) are interleaved as follows:



The output RGB components of a pixel are interpolated from 1, 2, or 4 neighbors of the pixel having the same color. There are several modifications of the above pattern that can be achieved by shifting the pattern one pixel left and/or one pixel up. The two letters C_1 and C_2 in the conversion constants CV_Bayer C_1C_2 2BGR and CV_Bayer C_1C_2 2RGB indicate the particular pattern type. These are components from the second row, second and third columns, respectively. For example, the above pattern has a very popular “BG” type.

3.4 inRange

Checks if array elements lie between the elements of two other arrays.

Python: cv2.**inRange**(src, lowerb, upperb[, dst]) → dst

Python: cv.**InRange**(src, lower, upper, dst) → None

Python: cv.**InRangeS**(src, lower, upper, dst) → None

Parameters:

1. src – first input array.

2. lowerb – inclusive lower boundary array or a scalar.

3. upperb – inclusive upper boundary array or a scalar.

4. dst – output array of the same size as src and CV_8U type.

The function checks the range as follows:

For every element of a single-channel input array:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0$$

For two-channel arrays:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0 \wedge \texttt{lowerb}(I)_1 \leq \texttt{src}(I)_1 \leq \texttt{upperb}(I)_1$$

and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.

3.5 dilate

Dilates an image by using a specific structuring element.

Python: cv.Dilate(src, dst, element=None, iterations=1) → None

Parameters:

1. src – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or``CV_64F.

2. dst – output image of the same size and type as src.

3. element – structuring element used for dilation; if element=Mat() , a 3 x 3 rectangular structuring element is used.

4. anchor – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.

5. iterations – number of times dilation is applied.

6. borderType – pixel extrapolation method (see borderInterpolate() for details).

7. borderValue – border value in case of a constant border (see createMorphologyFilter() for details).

The function dilates the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum is taken:

$$\texttt{dst}(x,y) = \max_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x', y+y')$$

The function supports the in-place mode. Dilation can be applied several (iterations) times. In case of multi-channel images, each channel is processed independently.

3.6 GaussianBlur

Blurs an image using a Gaussian filter.

Python: cv2.**GaussianBlur**(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst

Parameters:

1. src – input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.

2. dst – output image of the same size and type as src.

3. ksize – Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma* .

4. sigmaX – Gaussian kernel standard deviation in X direction.

5. sigmaY – Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and

ksize.height , respectively (see getGaussianKernel() for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX,

and sigmaY.

6. borderType – pixel extrapolation method (see borderInterpolate() for details).

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

3.7 Convex Hull

Convex Hull will look similar to contour approximation, but it is not (Both may provide same results in some cases). Here, **cv2.convexHull()** function checks a curve for convexity defects and corrects it. Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects. For example, check the below image of hand. Red line shows the convex hull of hand. The double-sided arrow marks shows the convexity defects, which are the local maximum deviations of hull from contours.

image

There is a little bit things to discuss about it its syntax:

```
1 hull = cv2.convexHull(points[, hull[, clockwise[, returnPoints]]
```

Arguments details:

points are the contours we pass into.

hull is the output, normally we avoid it.

clockwise : Orientation flag. If it is True, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise.

returnPoints : By default, True. Then it returns the coordinates of the hull points. If False, it returns the indices of contour points corresponding to the hull points.

So to get a convex hull as in above image, following is sufficient:

```
hull = cv2.convexHull(cnt)
```

But if you want to find convexity defects, you need to pass returnPoints = False. To understand it, we will take the rectangle image above. First I found its contour as cnt. Now I found its convex hull with returnPoints = True, I got following values: [[[234 202]], [[51 202]], [[51 79]], [[234 79]]] which are the four corner points of rectangle. Now if do the same with returnPoints = False, I get following result: [[129],[67],[0],[142]]. These are the indices of corresponding points in contours. For eg, check the first value: cnt[129] = [[234, 202]] which is same as first result (and so on for others).

3.8 Contour Area

Contour area is given by the function **cv2.contourArea()** or from moments, **M['m00']**.

```
area = cv2.contourArea(cnt)
```

3.9 Convexity Defects

We saw what is convex hull in second chapter about contours. Any deviation of the object from this hull can be considered as convexity defect.

OpenCV comes with a ready-made function to find this, **cv2.convexityDefects()**. A basic function call would look like below:

```
hull = cv2.convexHull(cnt,returnPoints = False)
defects = cv2.convexityDefects(cnt,hull)
```

Note:

Remember we have to pass returnPoints = False while finding convex hull, in order to find convexity defects.

It returns an array where each row contains these values - **[start point, end point, farthest point, approximate distance to farthest point]**. We can visualize it using an image. We draw a line joining start point and end point, then draw a circle at the farthest point. Remember first three values returned are indices of cnt. So we have to bring those values from cnt.

3.10 waitKey

Waits for a pressed key.

Python: cv2.waitKey([delay]) → retval

Python: cv.WaitKey(delay=0) → int

Parameters: delay – Delay in milliseconds. 0 is the special value that means “forever”.

The function waitKey waits for a key event infinitely (when $\texttt{delay} \leq 0$) or for delay milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is running on your computer at that time. It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

CHAPTER 4
CODE DESCRIPTION

1.Importing library files

```
#%%
import cv2
import numpy as np
import math
```

2.Capturing frames and converting it to HSV

```
cap = cv2.VideoCapture(0)

while(1):

    try:  #an error comes if it does not find anything in window as it cannot find contour of max area
          #therefore this try error statement

        ret, frame = cap.read()
        frame=cv2.flip(frame,1)
        kernel = np.ones((3,3),np.uint8)

        #define region of interest
        roi=frame[100:300, 100:300]


        cv2.rectangle(frame,(100,100),(300,300),(0,255,0),0)
        hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)



    # define range of skin color in HSV
        lower_skin = np.array([0,20,70], dtype=np.uint8)
        upper_skin = np.array([20,255,255], dtype=np.uint8)

     #extract skin colur imagw
        mask = cv2.inRange(hsv, lower_skin, upper_skin)
```

3.Finding contours and creating covexHull

```
#extrapolate the hand to fill dark spots within
    mask = cv2.dilate(mask,kernel,iterations = 4)

#blur the image
    mask = cv2.GaussianBlur(mask,(5,5),100)



#find contours
    _,contours,hierarchy= cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

#find contour of max area(hand)
    cnt = max(contours, key = lambda x: cv2.contourArea(x))

#approx the contour a little
    epsilon = 0.0005*cv2.arcLength(cnt,True)
    approx= cv2.approxPolyDP(cnt,epsilon,True)


#make convex hull around hand
    hull = cv2.convexHull(cnt)

 #define area of hull and area of hand
    areahull = cv2.contourArea(hull)
    areacnt = cv2.contourArea(cnt)

#find the percentage of area not covered by hand in convex hull
    arearatio=((areahull-areacnt)/areacnt)*100

 #find the defects in convex hull with respect to hand
    hull = cv2.convexHull(approx, returnPoints=False)
    defects = cv2.convexityDefects(approx, hull)
```

4.Finding angles between fingers

```
#code for finding no. of defects due to fingers
    for i in range(defects.shape[0]):
        s,e,f,d = defects[i,0]
        start = tuple(approx[s][0])
        end = tuple(approx[e][0])
        far = tuple(approx[f][0])
        pt= (100,180)


        # find length of all sides of triangle
        a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
        b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
        c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
        s = (a+b+c)/2
        ar = math.sqrt(s*(s-a)*(s-b)*(s-c))

        #distance between point and convex hull
        d=(2*ar)/a

        # apply cosine rule here
        angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57
```

5.comparing the angles and displaying the numbers

```
        # ignore angles > 90 and ignore points very close to convex hull(they generally come due to noise)
        if angle <= 90 and d>30:
            l += 1
            cv2.circle(roi, far, 3, [255,0,0], -1)

        #draw lines around hand
        cv2.line(roi,start, end, [0,255,0], 2)


    l+=1

    #print corresponding gestures which are in their ranges
    font = cv2.FONT_HERSHEY_SIMPLEX
    if l==1:
        if areacnt<2000:
            cv2.putText(frame,'Put hand in the box',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
        else:
            if arearatio<12:
                cv2.putText(frame,'0',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
            elif arearatio<17.5:
                cv2.putText(frame,'6',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

            else:
                cv2.putText(frame,'1',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==2:
        cv2.putText(frame,'2',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==3:

          if arearatio<27:
                cv2.putText(frame,'3',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
          #else:
            #     cv2.putText(frame,'ok',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
```

```
      elif l==2:
          cv2.putText(frame,'2',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      elif l==3:

            if arearatio<27:
                  cv2.putText(frame,'3',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)
            #else:
              #     cv2.putText(frame,'ok',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      elif l==4:
          cv2.putText(frame,'4',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      elif l==5:
          cv2.putText(frame,'5',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      elif l==6:
          cv2.putText(frame,'reposition',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      else :
          cv2.putText(frame,'reposition',(10,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

      #show the windows
      cv2.imshow('mask',mask)
      cv2.imshow('frame',frame)
  except:
      pass
```

6.Destroying all windows and releasing camera

```
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

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

CHAPTER 5

VARIABLE EXPLORER

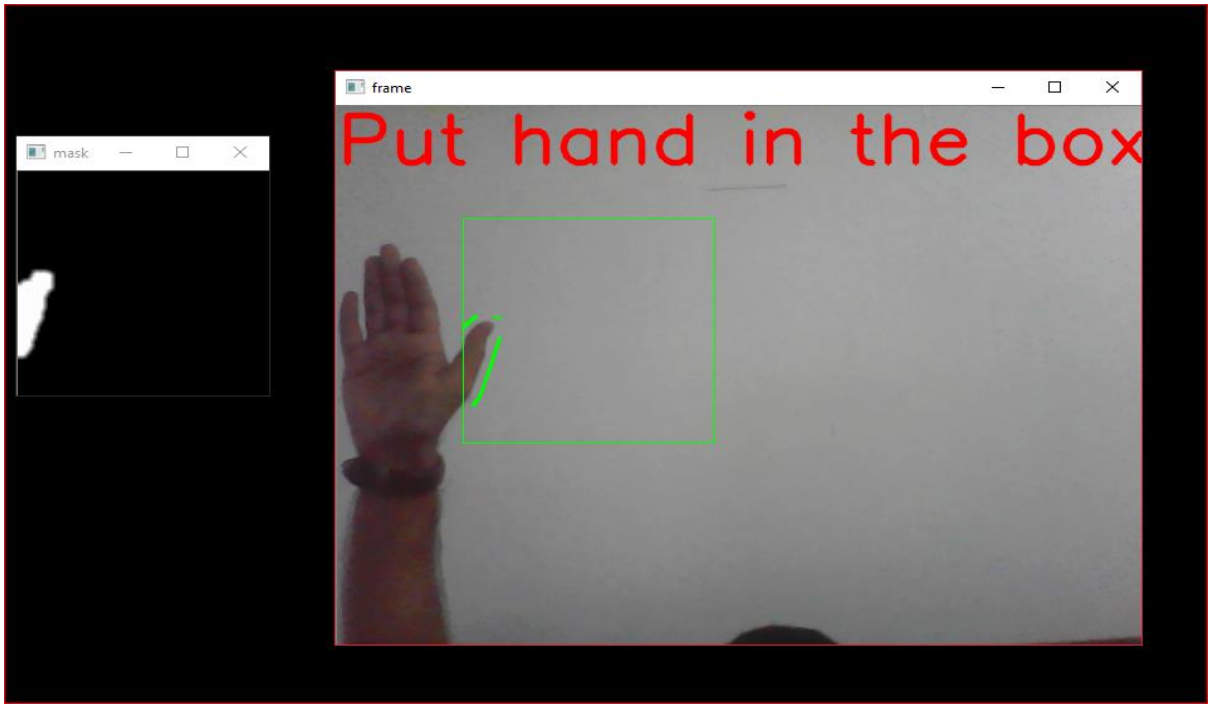
Name	Type	Size	Value
a	float	1	33.24154027718932
angle	float	1	104.44787042535032
approx	int32	(17, 1, 2)	ndarray object of numpy module
ar	float	1	91.49999999999997
areacnt	float	1	685.5
areahull	float	1	807.0
arearatio	float	1	17.72428884026258
b	float	1	6.082762530298219
c	float	1	31.144823004794873
cnt	int32	(17, 1, 2)	ndarray object of numpy module
contours	list	2	[Numpy array, Numpy array]
d	float	1	5.505160064005108

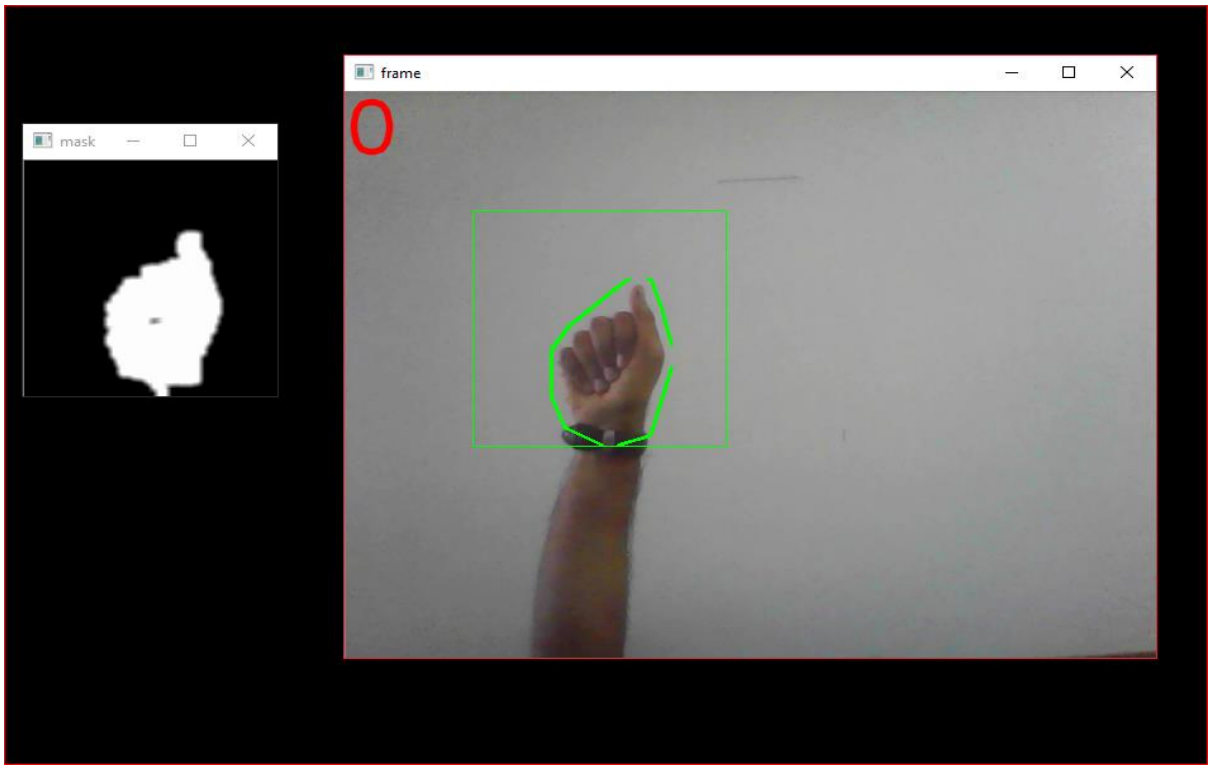
Name	Type	Size	Value
defects	int32	(3, 1, 4)	ndarray object of numpy module
e	int32	1	int32 object of numpy module
end	tuple	2	(int32, int32)
epsilon	float	1	0.06653553384542465
f	int32	1	int32 object of numpy module
far	tuple	2	(int32, int32)
font	int	1	0
frame	uint8	(480, 640, 3)	[[[138 140 129] [138 140 129]
hierarchy	int32	(1, 2, 4)	ndarray object of numpy module
hsv	uint8	(200, 200, 3)	[[[60 255 255] [60 255 255]
hull	int32	(7, 1)	ndarray object of numpy module
i	int	1	2

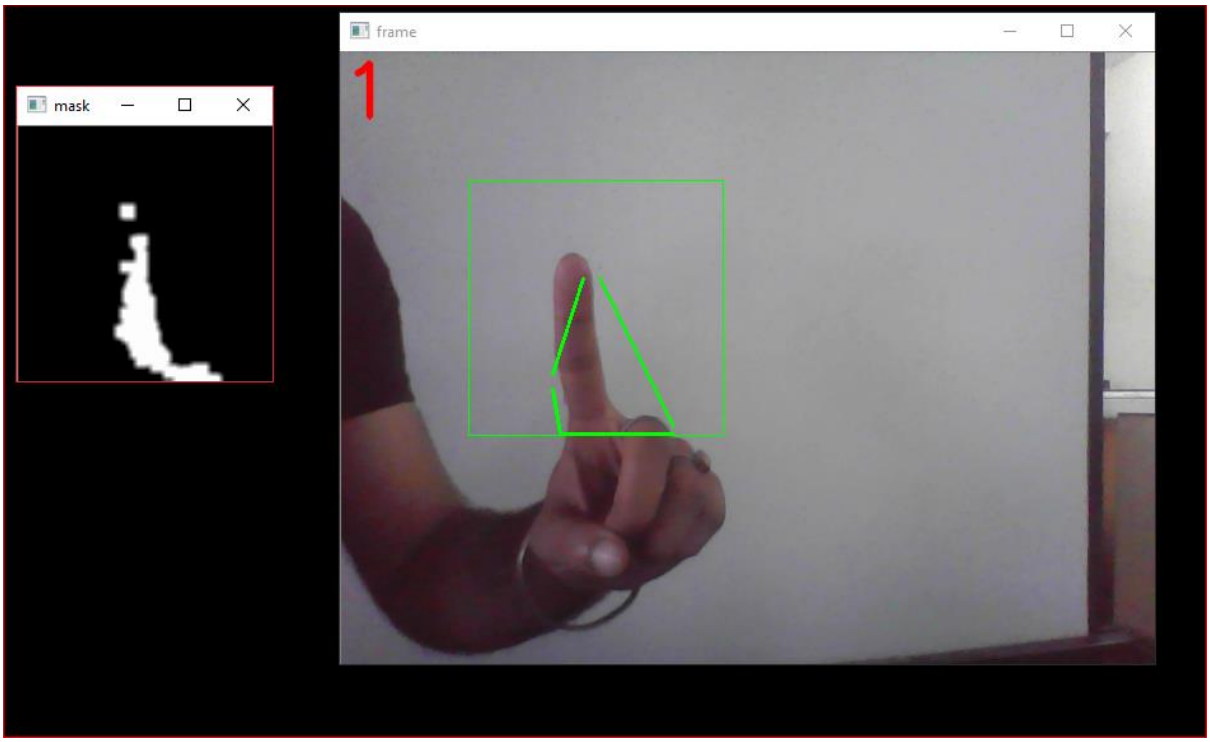
Name	Type	Size	Value
i	int	1	2
k	int	1	27
kernel	uint8	(3, 3)	[[1 1 1] [1 1 1]]
l	int	1	1
lower_skin	uint8	(3,)	[0 20 70]
mask	uint8	(200, 200)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]]
pt	tuple	2	(100, 180)
ret	bool	1	True
roi	uint8	(200, 200, 3)	[[[0 255 0] [0 255 0]]
s	float	1	35.23456290614121
start	tuple	2	(int32, int32)
upper_skin	uint8	(3,)	[20 255 255]

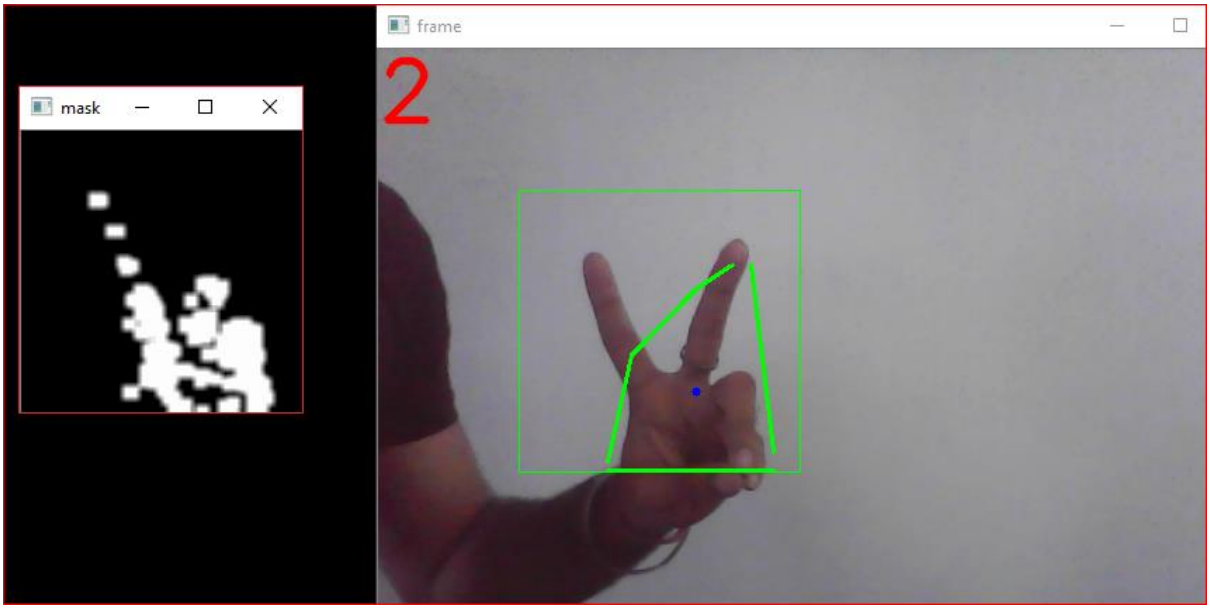
CHAPTER 6

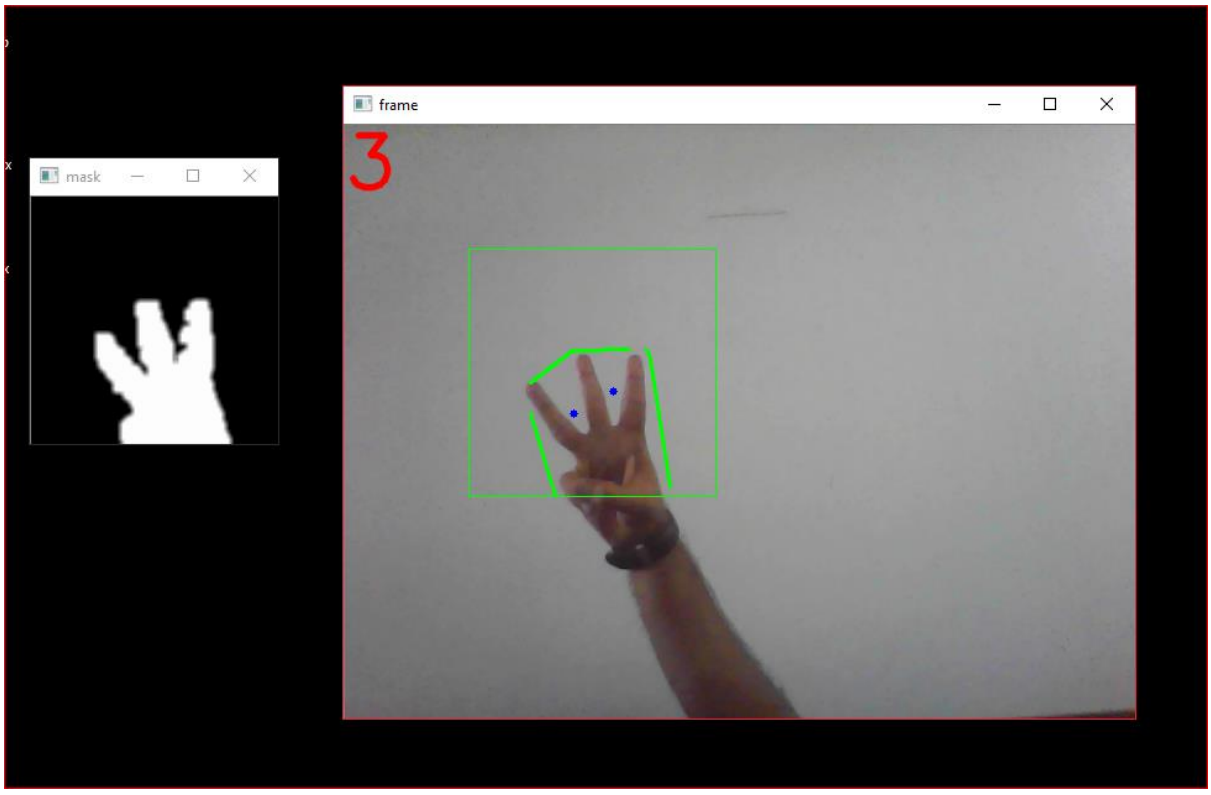
OUTPUTS

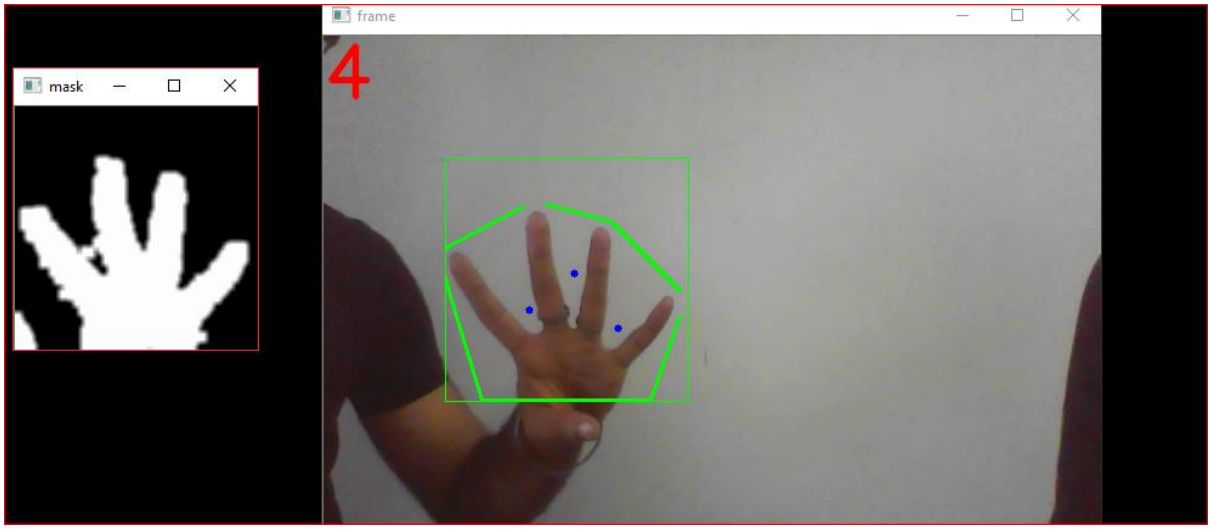


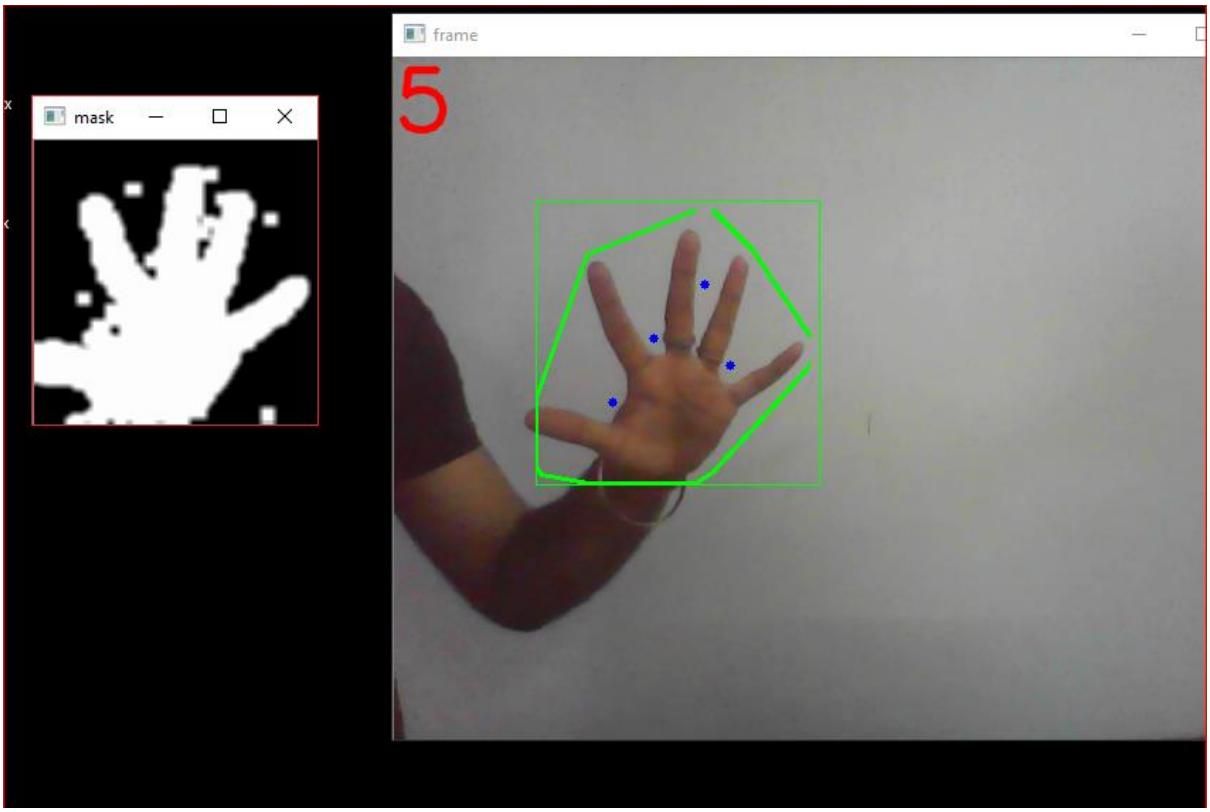


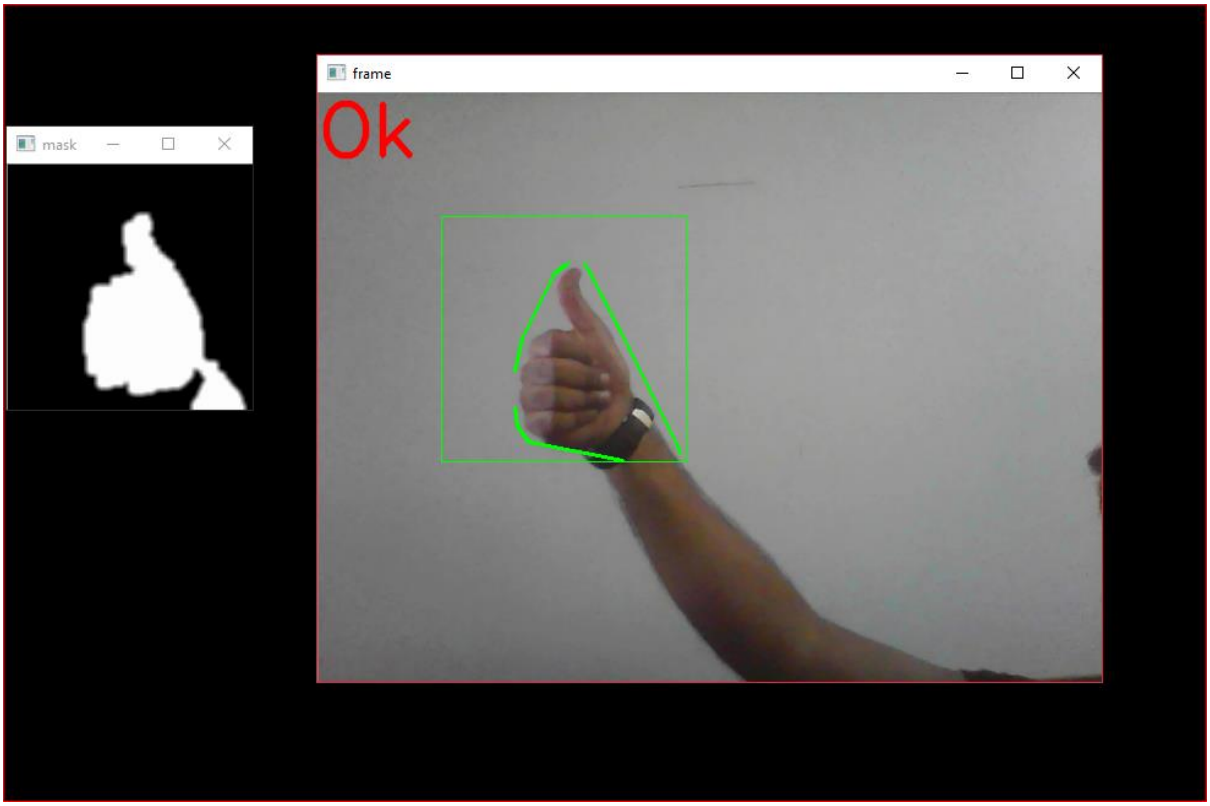

mask
frame
1


mask
frame
2


mask
frame
3


mask
frame
4


mask
frame
5


mask
frame
Ok

CHAPTER 7

Conclusion

7.1 Summary

The gesture keyboard produce a problem that in many regards fit thedescription of a problem that could be solvedusing machine learning.A pattern exists, the problem cannot be pinned down mathematically and a lot of data can be generated.One requirement of a gesture keyboard is to support a large dictionary. The results in the study show that with a growing numberof words the accuracy as well as the speed of the network decreased. Using75 words the percentage of misclassified words was 56%, combined with a training time of 56 seconds,whichsuggests thatthe approach tried in this study fails to reach the desired performance for larger dictionaries. A further requirement is that gesture keyboards have to be able to handle all categories of words. When testing joined words the network delivered satisfyingperformance with a misclassification level of about 1%. However, quite surprisingly,varyingwords resulted inloweraccuracywith a misclassification of about 3%.This is due to the fact thatthe network handles sequences of different lengths well whilesequences of similar lengths are harder to distinguishbetween.The study found thatthe sequences collected for each word varied due to variations in the gestures,causing the rest of the sequence to be displaced. This problem was emphasized intestswith similar words where

sequences of different words could actually, from the neural network's point of view,be more similar than sequences of the same word. Machine learning offersmany different approaches to problems and some have been considered inthis study. The results presented here indicate that our implementation of a MLP is not the optimal approach for this problem even though optimizations could be made to improve performance. However, other methods such as a SVM or a SOM would need to be tried before making any final conclusions regarding the performance of machine learning as a way of creating a gesture keyboard.

CHAPTER 8

REFERENCES

1. https://opencv-python-tutroals.readthedocs.io
2. https://www.geeksforgeeks.org
3. https://www.python.org
4. https://www.opencv.org
5. https://www.pyimagesearch.com
6. https://www.github.com