1. INTRODUCTION

1.1 Image Processing

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- ·Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

1.2 Purpose

The purpose of image processing is divided into 5 groups. They are:

1.Visualization-Observe the objects that are not visible.

2.Image sharpening and restoration-To create a better image.

3.Image retrieval-Seek for the image of interest.

4.Measurement of pattern–Measures various objects in an image.

5.Image Recognition–Distinguish the objects in an image.

1.3 Project Overview

In this project we perform all mouse functionalities without using mouse by image processing domain.

This project is divided into 4 modules:

1. Color Calibration
2. Morphological Transformation
3. Calculation of centroid
4. Controlling of mouse

By using pyautogui, openCV we can able to perform all functionalities like

a) Cursor movement- Based on the finger tip movement we can able to move the cursor
b) Clickings- Based on the distance between two respective colors we can able to perform left click or right click
c) Dragging-Based on the distance between three colors simultaneously we can able to perform drag operation
d) Scrolling- Based on respective color movement like up and down we can be able to perform scrolling operations.

1.3.1 Existing System

A **computer mouse** is a handheld hardware input device that controls a cursor in a GUI and can move and select text, icons, files, and folders. For desktop computers, the mouse is placed on a flat surface such as a mouse pad or a desk and is placed in front of your computer.was invented by Douglas Engelbart in 1963 while working at Xerox PARC There are different types of mouse has come into existence like wired mouse , wireless mouse, optical mouse, touchpad, trackball etc .This are connected to computer via ports like Serial port, USB Port, PS/2 port, bluetooth etc. Recently there arduino based glouse mouse has also come into existence.

1.3.2 Proposed System

Alternate Mouse is essentially a program which applies image processing, retrieves necessary data and implements it to the mouse interface of the computer according to predefined notions.

It uses the cross platform image processing module OpenCV and implements the mouse actions using Python specific library PyAutoGUI.

1.3.3 Scope of Proposed System

We intend to create a mouse simulation system for computers which would include gesture recognition system for spatial control. The system would include development of a program based on image processing for gesture recognition and implementing the processed data as an alternative for mouse.

Bases on the user finger gestures we can perform all functionalities of mouse like

- Moving the cursor
- Clicking
- Scrolling
- Dragging

2. SYSTEM REQUIREMENTS

2.1 Software Requirements

Supported Operating System:

- Windows 7(32 or 64 bit), Windows 8(32 or 64 bit), Windows 10(32 or 64 bit)
- Linux (Ubuntu Linux)

2.1.2 Supported Development Environment

- Python
- OpenCV
- PyAutoGUI
- Numpy

2.2 Hardware Requirements

- Processor: 1.2 GHZ
- RAM: 4 GB
- ROM: 15 GB

3. TECHNOLOGY

3.1 Image Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analyzing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis.

Types of Image Processing:

The two types of methods used for Image Processing are Analog and Digital Image Processing.

Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre-processing, enhancement and display, information extraction.

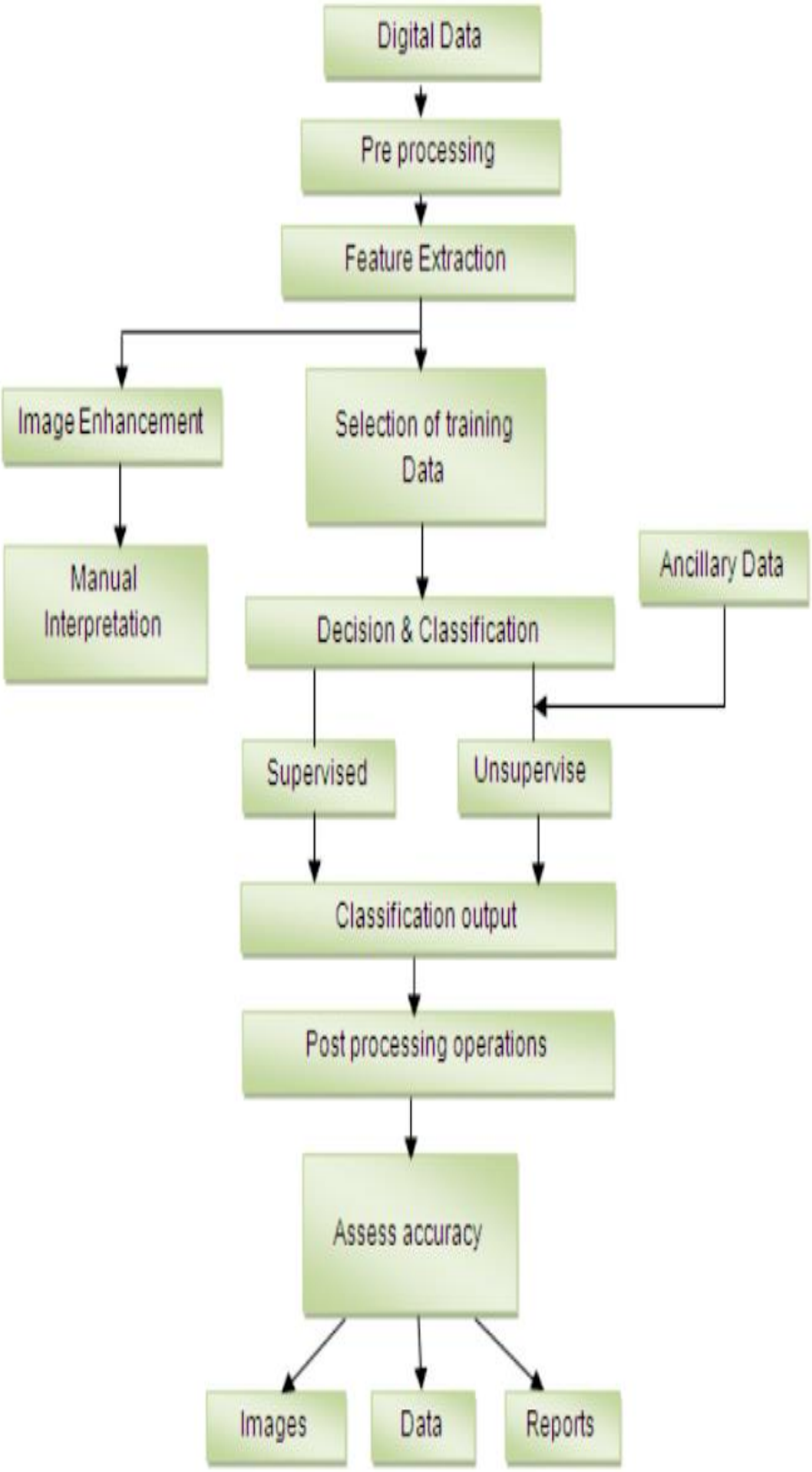


Figure 1:Image Processing

3.2 OpenCV

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of NumPy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from NumPy arrays. This also makes it easier to integrate with other libraries that use NumPy such as SciPy and Matplotlib. The following modules are available:

- **Core-functionality:** a compact module defining basic data the dense multi-dimensional array Mat and basic functions used by all other modules.

- **Image processing** - an image processing module that includes linear and non-linear image filtering, geometric image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

- **features2d** - salient feature detectors, descriptors, and descriptor matchers.

- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

- **highgui** - an easy-to-use interface to simple UI capabilities.

- **Video I/O** - an easy-to-use interface to video capturing and video codecs.**gpu** - GPU-accelerated algorithms from different OpenCV modules.

3.3 Python

Python is an interpreted high-level programming language for general purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython

is managed by the non-profit Python Software Foundation. Python is a programming language, which is fast to learn and has powerful Functions. It provides abundant basic modules and third party modules that could be directly used in programmers. It contains a series of functions like Database, web development, file operations and etc…

Python currently has two versions : Python 2 and Python 3, and They are code incompatible. Python 3 has improved a lot, and has access to most libraries.

3.4 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- A powerful N-dimensional array object
- sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is licensed under the BSD license, enabling reuse with few restrictions.

3.5 PyAutoGUI

3.5.1 Purpose

The purpose of PyAutoGUI is to provide a cross-platform Python module for GUI automation for human beings. The API is designed to be as simple as possible with sensible defaults.
For example, here is the complete code to move the mouse to the middle of the screen on Windows, OS X, and Linux:

```
>>> import pyautogui
>>> screen Width, screenHeight = pyautogui.size()
```

>>> pyautogui.moveTo(screen Width / 2, screenHeight / 2)

PyAutoGUI can simulate moving the mouse, clicking the mouse, dragging with the mouse, pressing keys, pressing and holding keys, and pressing keyboard hotkey combinations.

3.5.2 Dependencies

On Windows, PyAutoGUI has no dependencies (other than Pillow and some other modules, which are installed by pip along with PyAutoGUI). It does not need the pywin32 module installed since it uses Python's own ctypes module.

On OS X, PyAutoGUI requires PyObjC installed for the AppKit and Quartz modules. The module names on PyPI to install are pyobjc-core and pyobjc (in that order).

On Linux, PyAutoGUI requires python-xlib (for Python 2) or python3-Xlib (for Python 3) module installed.

3.5.3 Fail-Safes

Figure 2:Fail-Safes

Like the enchanted brooms from the Sorcerer's Apprentice programmed to keep filling (and then overfilling) the bath with water, your program could get out of control (even though it is following your instructions) and need to be stopped. This can be difficult to do if the mouse is moving around on its own, preventing you from clicking on the program's window to close it down.

As a safety feature, a fail-safe feature is enabled by default. When pyautogui.FAILSAFE = True PyAutoGUI functions will raise a pyautogui.FailSafe Exception if the mouse cursor is in the upper left corner of the screen. If you lose control and need to stop the current PyAutoGUI function, keep moving the mouse cursor up and to the left. To disable this feature, set FAILSAFE to False:

```
>>> import pyautogui
>>> pyautogui.FAILSAFE = False # disables the fail-safe
```

You can add delays after all of PyAutoGUI functions by setting the pyautogui.PAUSE variable to a float or integer value of the number of seconds to pause. By default, the pause is set to 0.1 seconds. This can be helpful when interacting with other applications so that PyAutoGUI doesn't move too fast for them. For example:

```
>>> import pyautogui
>>> pyautogui.PAUSE = 2.5
>>> pyautogui.moveTo(100, 100); pyautogui.click()   # there will be a two and a half second pause after moving and another after the click
```

All PyAutoGUI functions will block until they complete. (It is on the roadmap to add an optional non-blocking way to call these functions.)

It is advised to use FAILSAFE along with setting PAUSE.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

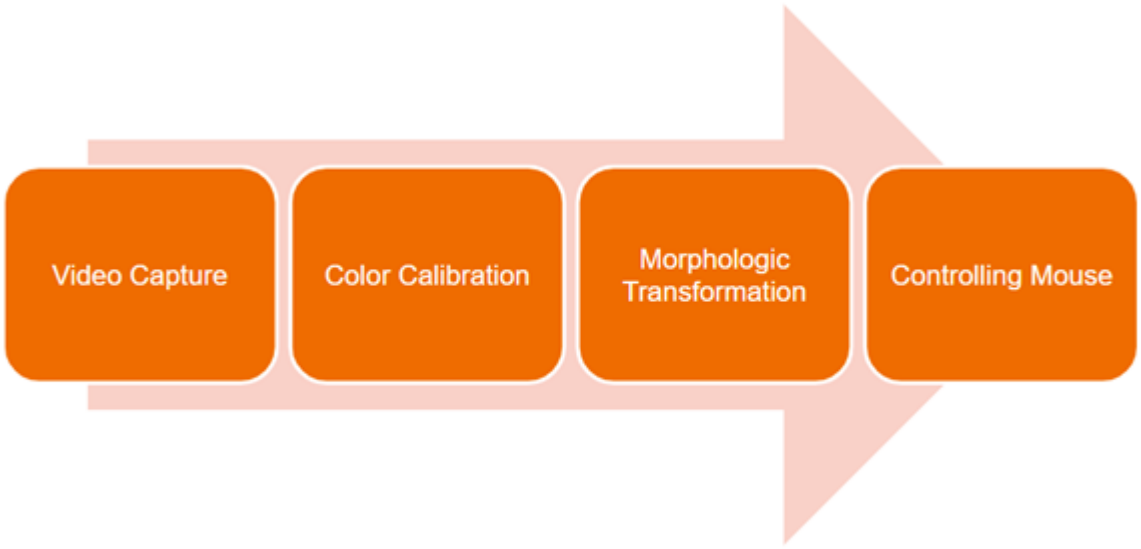


Fig 3: System Architecture

4.2 Data Flow Diagram

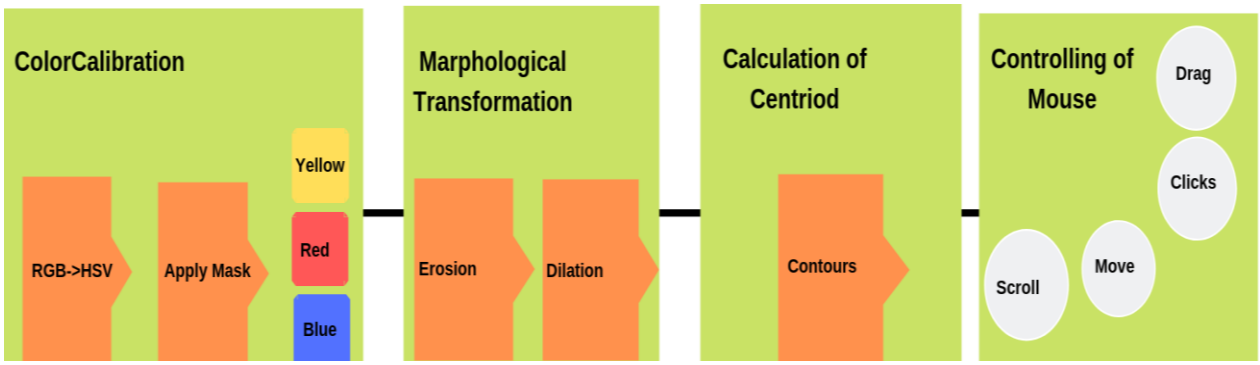


Figure 3:Data-Flow

1) The first thing that we do is convert the captured video into HSV format. Now the user gets to calibrate the color ranges for three of his fingers individually. This is done by calling the calibrateColor() function thrice right at the beginning of the program. The user has an option to use the default settings as well.

2) Depending on the calibrations, only the three fingertips are extracted from the video, one by one, using the cv2.inRange() function. In order to remove noise in the video feed, we apply a two-step morphism i.e. erosion and dilation. The noise filtered image referred to as mask in the program

Location of each of the three centers involves:

1. Finding contours in the mask relevant to that colour range.
2. Discarding contours of irrelevant areas using area filters.
3. Finding the largest contour amongst the remaining ones and applying method of moments to find its center.

Then comes the step for defining position of cursor on the screen. The thumb, with yellow color is responsible for position of the cursor.
The following techniques have been used in this end:

- Generally the webcams we use captures video at a resolution of 640x480 pixels. Suppose this frame was linearly mapped to the 1920x1080 pixel display screen. If we have a right-handed user, he would find it uncomfortable to access the left edge of the screen as compared to the right edge. Also accessing the bottom portion of the screen would build stress at the wrist. We realized that instead of mapping the whole video frame to the screen, we could rather consider a rectangular sub portion more biased towards right (considering right-handed user) and upper parts of the frame in order to improve comfort. This sub portion which measures 480x270 pixels is then linearly mapped to the screen with a scaling factor of 4.

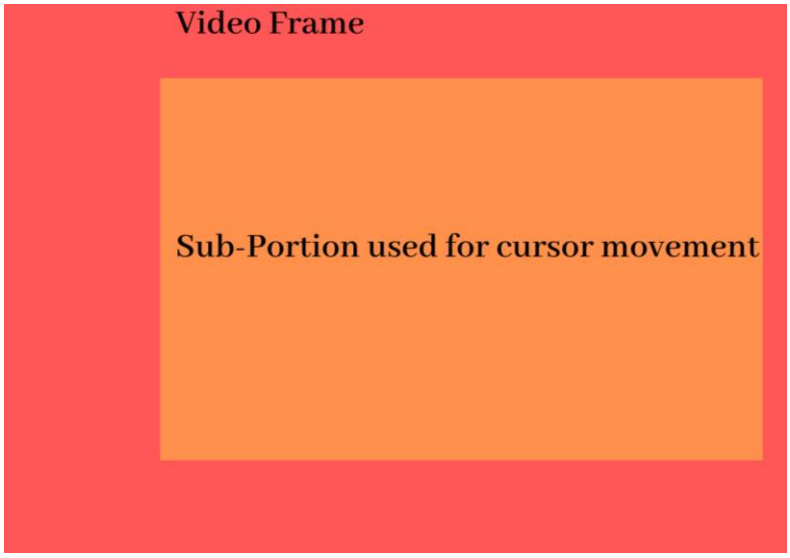


Figure 4:Sub-Portion

Due to noise captured by the webcam and vibrations in the hand, the centers keep vibrating around a mean position. On scaling up, these vibrations create a lot of problem with the accuracy of cursor position. To reduce the shakiness in cursor, we make use of differential position allocation for the cursor. We compare the new center with the previous position of the cursor. If difference is less than 5 pixels, it is usually due to noise. Thus the new cursor position is inclined more towards the previous one. However, a larger difference in previous position and new center is considered as voluntary movement and the new cursor position is set close to the new center. For details, go through the setCursorPosition() function in the code.

Now the three centers are sent for deciding what action needs to be performed depending on their relative positions. This is done in the chooseAction() function in the code. Depending upon its output, the performAction() function carries out either of the following using the PyAutoGUI library:

1. free cursor movement
2. left click
3. right click
4. drag/select
5. scroll up
6. scroll down

5. IMPLEMENTATION

5.1 MODULES

1. Color Calibration
2. Morphological Transformation
3.Calculation of Centroid
4.Controlling of Mouse

5.1.1 Color Calibration

The first thing that we do is convert the captured video into HSV format. Now the user gets to calibrate the color ranges for three of his fingers individually. This is done by calling the calibrateColor() function thrice right at the beginning of the program

STEP 1 - RGB TO HSV CONVERSION

Given RGB color range, our task is to convert RGB color to HSV color.

RGB Color Model: The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

HSV Color Model: HSV – (hue, saturation, value), also known as HSB (hue, saturation, brightness), is often used by artists because it is often more natural to think about a color in terms of hue and saturation than in terms of additive or subtractive color components. HSV is a transformation of an RGB colorspace, and its components and colorimetry are relative to the RGB colorspace from which it was derived.

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

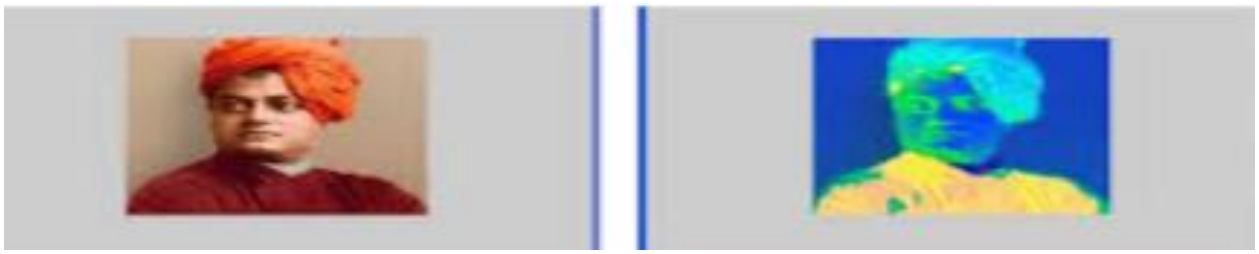


Figure 5:BGR-HSV

STEP 2 - APPLY A THRESHOLD MASK

Depending on the calibrations, only the three fingertips are extracted from the video, one by one, using the cv2.inRange() function. In order to remove noise in the video feed, we apply a two-step morphism i.e. erosion and dilation. To isolate the colors, we have to apply multiple masks. A low threshold and high threshold mask for hue, saturation and value. Anything pixel within these thresholds will be set to 1 and the remaining pixels will be zero

b_mask = makeMask(hsv, blue_range)

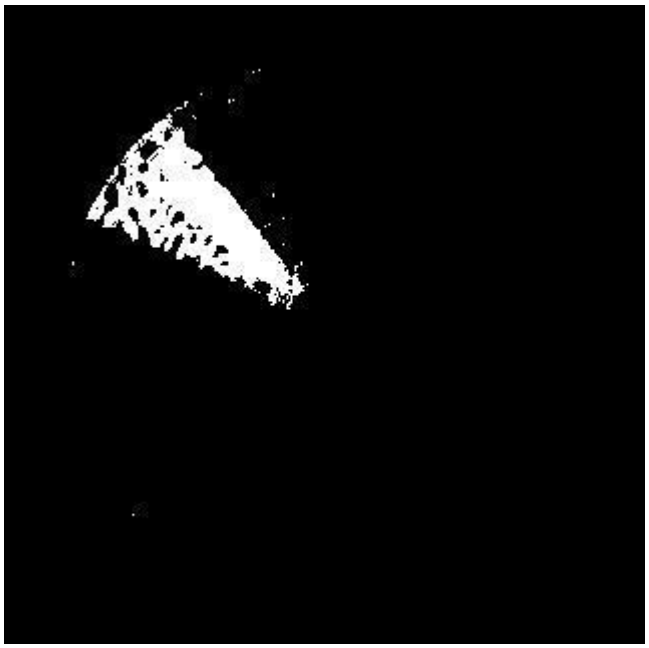


Figure 6:Threshold Image

5.1.2 Morphological Transformation

Binary images may contain numerous imperfections. In particular, the binary regions produced by simple thresholding are distorted by noise and texture. Morphological image processing pursues the goals of removing these imperfections by accounting for the form and structure of the image. These techniques can be extended to grayscale images.

The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :

- Removing noise
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image

We will explain dilation and erosion briefly, using the following image as an example: This operations consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle. The kernel B has a defined *anchor point*, usually being the center of the kernel.

Erosion: As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.

How Exactly it works?

Consider an image which is in binary so the matrix form of that particular is in 0's and 1's.Now Consider the structuring element in odd order matrix . Because for even order matrix we can't find the center of it .

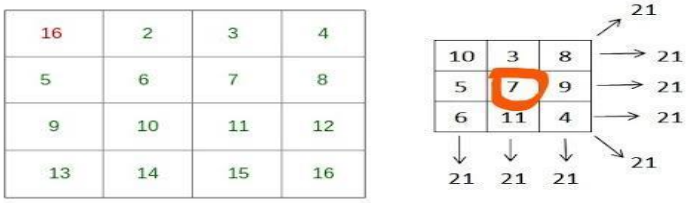
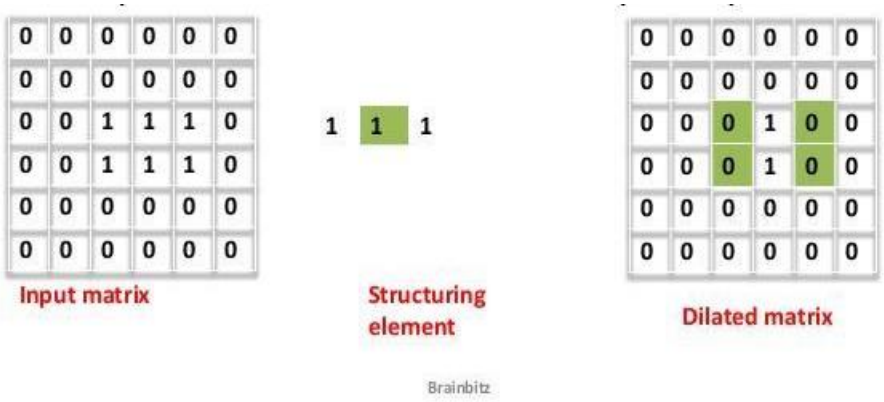


Figure 7:Even and Odd order Matrix

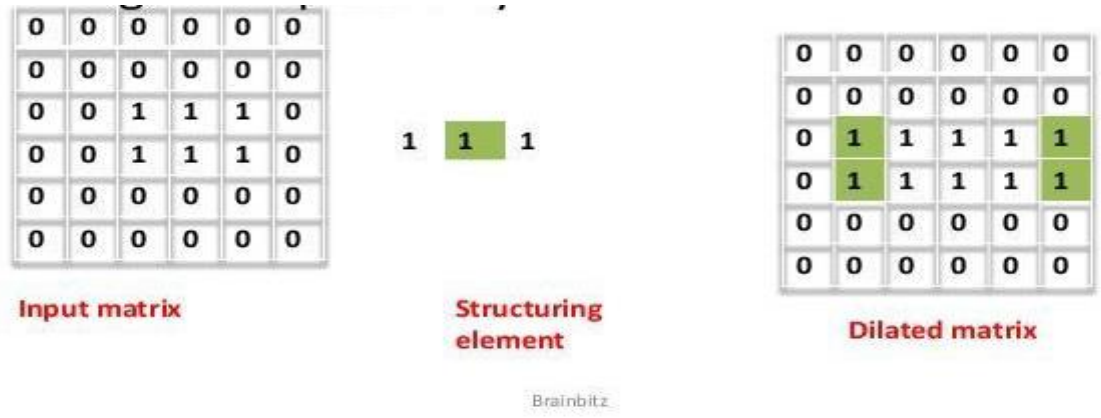
The structuring element moves over the binary image top, bottom, left, right. It multiplies with the image and replaces the center element with the minimum and edges are kept as it is.



Dilation: As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value.

How it works?

This is same as above process(i.e., erosion) but the center element is replaced with the maximum element like below



There are other different types of **Morphological** Transformations like

- Opening-Opening is just another name of **erosion followed by dilation**.
- Closing-Closing is reverse of Opening, **Dilation followed by Erosion**.
- Morphological Gradient-It is the difference between dilation and erosion of an image. The result will look like the outline of the object.
- Top Hat-It is the difference between input image and Opening of the image.
- Black Hat-It is the difference between the closing of the input image and input image.

5.1.3 Calculation of Centroid

The noise filtered image referred to as mask in the program is then sent for locating the centers. Location of each of the three centers involves:

1. Finding contours in the mask relevant to that color range.
2. Discarding contours of irrelevant areas using area filters.
3. Finding the largest contour amongst the remaining ones and applying method of moments to find its center.

Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

Let's see how to find contours of a binary image:

,contour,=cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

It has Three arguments

1.Mask

It consists of image which is converted from binary BGR image to either gray image or HSV image

2.cv2.RETR_TREE:(Contour Retrieval mode)

In the output, we got three arrays, first is the image, second is our contours, and one more output which we named as **hierarchy** Sometimes objects are in different locations. But in some cases, some shapes are inside other shapes. Just like nested figures. In this case, we call outer one as **parent** and inner one as **child**. This way, contours in an image has some relationship to each other. And we can specify how one contour is connected to each other, like, is it child of some other contour, or is it a parent etc. Representation of this relationship is called the **Hierarchy**

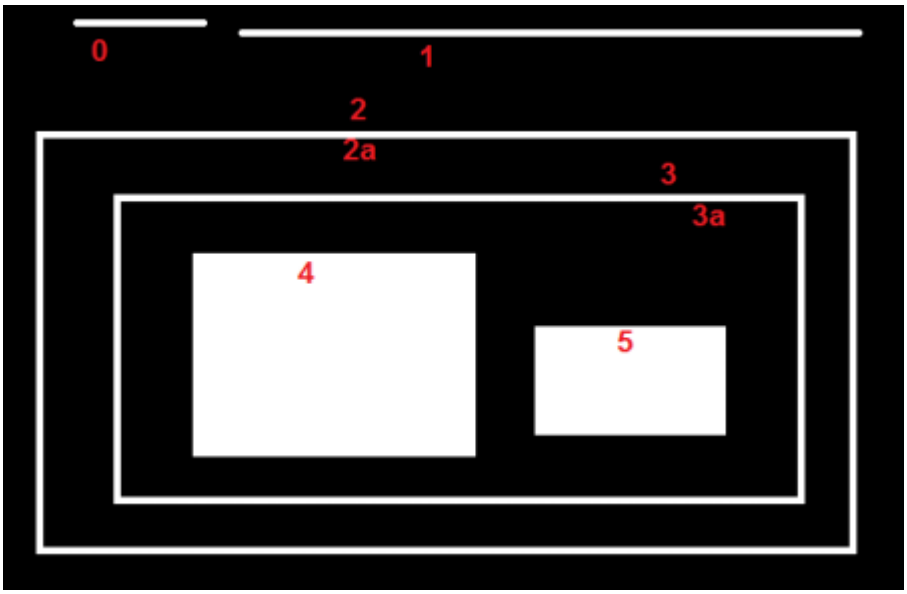


Figure 8:Hierarchy Image

Hierarchy Representation in OpenCV

[Next, Previous, First_Child, Parent]

There are different types of contour retrieval modes

A. RETR_LIST

This is the simplest of the four flags (from explanation point of view). It simply retrieves all the contours, but doesn't create any parent-child relationship. **Parents and kids are equal under this rule, and they are just contours**. ie they all belong to same hierarchy level

B. RETR_EXTERNAL

If you use this flag, it returns only extreme outer flags. All child contours are left behind. We can say, under this law, Only the eldest in every family is taken care of. It doesn't care about other members of the family

C. RETR_CCOMP

This flag retrieves all the contours and arranges them to a 2-level hierarchy. ie external contours of the object (ie its boundary) are placed in hierarchy-1. And the contours of holes inside object (if any) is placed in hierarchy-2. If any object inside it, its contour is placed again in hierarchy-1 only. And its hole in hierarchy-2 and so on.

D. RETR_TREE

And this is the final guy, Mr. Perfect. It retrieves all the contours and creates a full family hierarchy list. It even tells, who is the grandpa, father, son, grandson and even beyond.

3. cv2.CHAIN_APPROX_SIMPLE (contour Approximation Method)

It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates ? That is specified by this contour approximation method.

If you pass cv2.CHAIN_APPROX_NONE, all the boundary points are stored. But actually do we need all the points? For e.g., you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what cv2.CHAIN_APPROX_SIMPLE does. It removes all redundant points and compresses the contour, thereby saving memory.

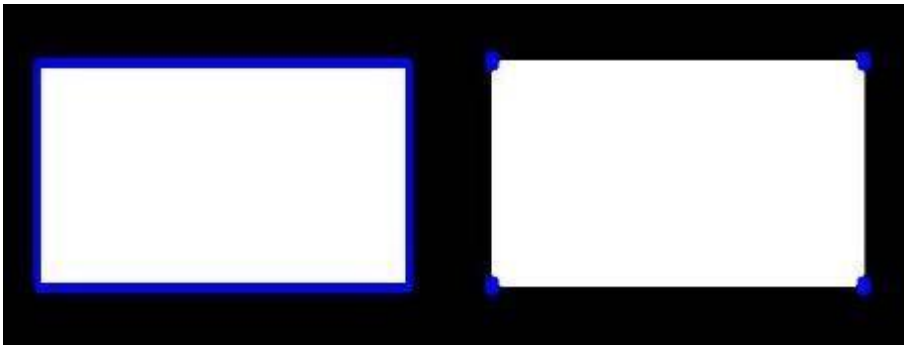


Figure 9:Bordering Images

Once we found the contours of object then we can calculate the centroid by using contour moments method

```
im2, contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
M = cv2.moments(contour[0])
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
center = (cx,cy)
```

Now the three centers are sent for deciding what action needs to be performed depending on their relative positions. This is done in the chooseAction() function in the code. Depending upon its output, the performAction() function carries out either of the following using the PyAutoGUI library:

1)free cursor movement
2)left click
3)right click
4)drag/select
5)scroll up
6)scroll down

5.1.4. Controlling of Mouse

This can be done by using a module called PYAUTOGUI in python. PyAutoGUI is a Python module for programmatically controlling the mouse and keyboard.

Locations on your screen are referred to by X and Y Cartesian coordinates.

The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down.

General screen resolution is 1920 x 1080

```
>>> pyautogui.size() It returns the screen resolution
(1920, 1080)
>>> pyautogui.position() It returns present mouse cursor position
(187, 567)
```

MoveTo

The moveTo() function will move the mouse cursor to the X and Y integer coordinates you pass it. The None value can be passed for a coordinate to mean "the current mouse cursor position"

ex: 1)pyautogui.moveTo(100, 200) *# moves mouse to X of 100, Y of 200.*

2)pyautogui.moveTo(100, 200, 2) *# moves mouse to X of 100, Y of 200 over 2 seconds*

Mouse Drags

PyAutoGUI's dragTo() and drag() functions have similar parameters as the moveTo() and move()functions.

Mouse Clicks

The click() function simulates a single, left-button mouse click at the mouse's current position. A "click" is defined as pushing the button down and then releasing it up. For example:

1)pyautogui.click() *# click the mouse*

2)pyautogui.click(x=100, y=200) *# move to 100, 200, then click the left mouse button.*

3)pyautogui.click(button='right') *# right-click the mouse*

Mouse Scrolling

The mouse scroll wheel can be simulated by calling the scroll() function and passing an integer number of "clicks" to scroll.

pyautogui.scroll(10) *# scroll up 10 "clicks"*
pyautogui.scroll(-10) *# scroll down 10 "clicks"*
pyautogui.hscroll(10) *# scroll right 10 "clicks"*
pyautogui.hscroll(-10) *# scroll left 10 "clicks*

5.2 Code Snippets

```
import cv2
import numpy as np
import pyautogui
import time

blue_range = np.array([[88,78,20],[128,255,255]])
yellow_range = np.array([[21,70,80],[61,255,255]])
red_range = np.array([[158,85,72],[180 ,255,255]])

b_cen, y_pos, r_cen = [240,320],[240,320],[240,320]
cursor = [960,540]


r_area = [100,1700]
b_area = [100,1700]
y_area = [100,1700]


kernel = np.ones((7,7),np.uint8)

perform = False
showCentroid = False


def nothing(x):
        pass
def swap( array, i, j):
        temp = array[i]
        array[i] = array[j]
        array[j] = temp

def distance( c1, c2):
        distance = pow( pow(c1[0]-c2[0],2) + pow(c1[1]-c2[1],2) , 0.5)
        return distance
```

```
def changeStatus(key):
        global perform
        global showCentroid
        global yellow_range,red_range,blue_range

        if key == ord('p'):
                perform = not perform
                if perform:
                        print ('Mouse simulation ON...')
                else:
                        print ('Mouse simulation OFF...')

        elif key == ord('c'):
                showCentroid = not showCentroid
                if showCentroid:
                        print ('Showing Centroids...')
                else:
                        print ('Not Showing Centroids...')

        elif key == ord('r'):
                print
('*************************************************************************')
                print (' You have entered recalibration mode.')
                print (' Use the trackbars to calibrate and press SPACE when done.')
                print (' Press D to use the default settings')
                print
('*************************************************************************')

                yellow_range = calibrateColor('Yellow', yellow_range)
                red_range = calibrateColor('Red', red_range)
                blue_range = calibrateColor('Blue', blue_range)
```

```
        else:
                pass


def makeMask(hsv_frame, color_Range):

        mask = cv2.inRange( hsv_frame, color_Range[0], color_Range[1])

        eroded = cv2.erode( mask, kernel, iterations=1)
        dilated = cv2.dilate( eroded, kernel, iterations=1)

        return dilated

def drawCentroid(vid, color_area, mask, showCentroid):

        _, contour, _ = cv2.findContours( mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        l=len(contour)
        area = np.zeros(l)

        for i in range(l):
                if cv2.contourArea(contour[i])>color_area[0] and
cv2.contourArea(contour[i])<color_area[1]:
                        area[i] = cv2.contourArea(contour[i])
                else:
                        area[i] = 0

        a = sorted( area, reverse=True)

        for i in range(l):
                for j in range(1):
                        if area[i] == a[j]:
                                swap( contour, i, j)
```

```
    if l > 0 :
        M = cv2.moments(contour[0])
        if M['m00'] != 0:
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            center = (cx,cy)
            if showCentroid:
                cv2.circle( vid, center, 5, (0,0,255), -1)

            return center
    else:

        return (-1,-1)

def calibrateColor(color, def_range):

    global kernel
    name = 'Calibrate '+ color
    cv2.namedWindow(name)
    cv2.createTrackbar('Hue', name, def_range[0][0]+20, 180, nothing)
    cv2.createTrackbar('Sat', name, def_range[0][1]   , 255, nothing)
    cv2.createTrackbar('Val', name, def_range[0][2]   , 255, nothing)
    while(1):
        ret , frameinv = cap.read()
        frame=cv2.flip(frameinv ,1)

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        hue = cv2.getTrackbarPos('Hue', name)
        sat = cv2.getTrackbarPos('Sat', name)
        val = cv2.getTrackbarPos('Val', name)
```

```
            lower = np.array([hue-20,sat,val])
            upper = np.array([hue+20,255,255])

            mask = cv2.inRange(hsv, lower, upper)
            eroded = cv2.erode( mask, kernel, iterations=1)
            dilated = cv2.dilate( eroded, kernel, iterations=1)

            cv2.imshow(name, dilated)

            k = cv2.waitKey(5) & 0xFF
            if k == ord(' '):
                  cv2.destroyWindow(name)
                  return np.array([[hue-20,sat,val],[hue+20,255,255]])
            elif k == ord('d'):
                  cv2.destroyWindow(name)
                  return def_range


def setCursorPos( yc, pyp):

      yp = np.zeros(2)

      if abs(yc[0]-pyp[0])<5 and abs(yc[1]-pyp[1])<5:
            yp[0] = yc[0] + .7*(pyp[0]-yc[0])
            yp[1] = yc[1] + .7*(pyp[1]-yc[1])
      else:
            yp[0] = yc[0] + .1*(pyp[0]-yc[0])
            yp[1] = yc[1] + .1*(pyp[1]-yc[1])

      return yp


def chooseAction(yp, rc, bc):
      out = np.array(['move', 'false'])
```

```
    if rc[0]!=-1 and bc[0]!=-1:

        if distance(yp,rc)<50 and distance(yp,bc)<50 and distance(rc,bc)<50 :
            out[0] = 'drag'
            out[1] = 'true'
            return out
        elif distance(rc,bc)<40:
            out[0] = 'left'
            return out
        elif distance(yp,rc)<40:
            out[0] = 'right'
            return out
        elif distance(yp,rc)>40 and rc[1]-bc[1]>120:
            out[0] = 'down'
            return out
        elif bc[1]-rc[1]>110:
            out[0] = 'up'
            return out
        else:
            return out

    else:
        out[0] = -1
        return out

def performAction( yp, rc, bc, action, drag, perform):
    if perform:
        cursor[0] = 4*(yp[0]-110)
        cursor[1] = 4*(yp[1]-120)
        if action == 'move':

            if yp[0]>110 and yp[0]<590 and yp[1]>120 and yp[1]<390:
```

```
                pyautogui.moveTo(cursor[0],cursor[1])
            elif yp[0]<110 and yp[1]>120 and yp[1]<390:
                pyautogui.moveTo( 8 , cursor[1])
            elif yp[0]>590 and yp[1]>120 and yp[1]<390:
                pyautogui.moveTo(1912, cursor[1])
            elif yp[0]>110 and yp[0]<590 and yp[1]<120:
                pyautogui.moveTo(cursor[0] , 8)
            elif yp[0]>110 and yp[0]<590 and yp[1]>390:
                pyautogui.moveTo(cursor[0] , 1072)
            elif yp[0]<110 and yp[1]<120:
                pyautogui.moveTo(8, 8)
            elif yp[0]<110 and yp[1]>390:
                pyautogui.moveTo(8, 1072)
            elif yp[0]>590 and yp[1]>390:
                pyautogui.moveTo(1912, 1072)
            else:
                pyautogui.moveTo(1912, 8)

        elif action == 'left':
            pyautogui.click(button = 'left')

        elif action == 'right':
            pyautogui.click(button = 'right')
            time.sleep(0.3)

        elif action == 'up':
            pyautogui.scroll(10)


        elif action == 'down':
            pyautogui.scroll(-10)


        elif action == 'drag' and drag == 'true':
```

```
global y_pos
drag = 'false'
pyautogui.mouseDown()

while(1):

    k = cv2.waitKey(10) & 0xFF
    changeStatus(k)

    _, frameinv = cap.read()

    frame = cv2.flip( frameinv, 1)

    hsv = cv2.cvtColor( frame, cv2.COLOR_BGR2HSV)

    b_mask = makeMask( hsv, blue_range)
    r_mask = makeMask( hsv, red_range)
    y_mask = makeMask( hsv, yellow_range)

    py_pos = y_pos

    b_cen = drawCentroid( frame, b_area, b_mask, showCentroid)
    r_cen = drawCentroid( frame, r_area, r_mask, showCentroid)

    y_cen = drawCentroid( frame, y_area, y_mask, showCentroid)

    if      py_pos[0]!=-1 and y_cen[0]!=-1:
            y_pos = setCursorPos(y_cen, py_pos)

    performAction(y_pos, r_cen, b_cen, 'move', drag, perform)

    cv2.imshow('Frame', frame)

    if distance(y_pos,r_cen)>60 or distance(y_pos,b_cen)>60 or
distance(r_cen,b_cen)>60:
```

```
                    break

                pyautogui.mouseUp()



cap = cv2.VideoCapture(0)

print (' You have entered calibration mode.')
print (' Use the trackbars to calibrate and press SPACE when done.')
print (' Press D to use the default settings.')


yellow_range = calibrateColor('Yellow', yellow_range)
red_range = calibrateColor('Red', red_range)
blue_range = calibrateColor('Blue', blue_range)
print (' Calibration Successfull...')

cv2.namedWindow('Frame')

print (' Press P to turn ON and OFF mouse simulation.')
print (' Press C to display the centroid of various colours.')
print (' Press R to recalibrate color ranges.')
print (' Press ESC to exit.')


while(1):

        k = cv2.waitKey(10) & 0xFF
        changeStatus(k)


        _, frameinv = cap.read()

        frame = cv2.flip( frameinv, 1)
```

```
        hsv = cv2.cvtColor( frame, cv2.COLOR_BGR2HSV)

        b_mask = makeMask( hsv, blue_range)
        r_mask = makeMask( hsv, red_range)
        y_mask = makeMask( hsv, yellow_range)

        py_pos = y_pos

        b_cen = drawCentroid( frame, b_area, b_mask, showCentroid)
        r_cen = drawCentroid( frame, r_area, r_mask, showCentroid)
        y_cen = drawCentroid( frame, y_area, y_mask, showCentroid)

        if      py_pos[0]!=-1 and y_cen[0]!=-1 and y_pos[0]!=-1:
                y_pos = setCursorPos(y_cen, py_pos)

        output = chooseAction(y_pos, r_cen, b_cen)
        if output[0]!=-1:
                performAction(y_pos, r_cen, b_cen, output[0], output[1], perform)

        cv2.imshow('Frame', frame)

        if k == 27:
                break

cv2.destroyAllWindows()
```

6. TESTING

6.1 SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTS

Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach:

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7. SCREENSHOTS

Signs

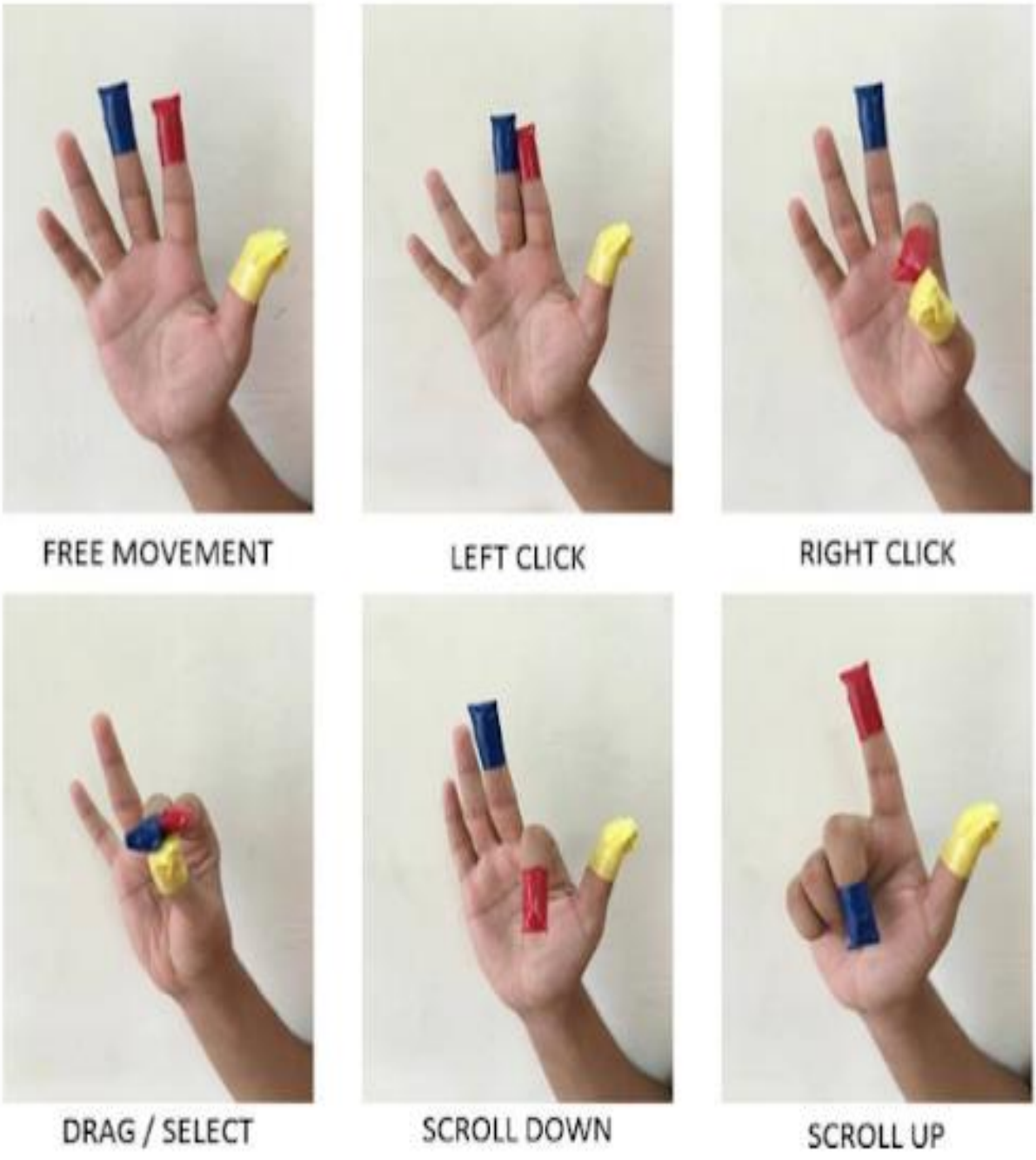
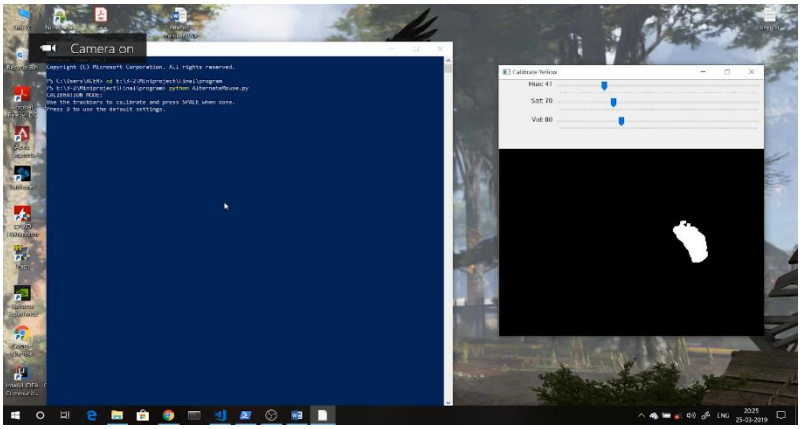


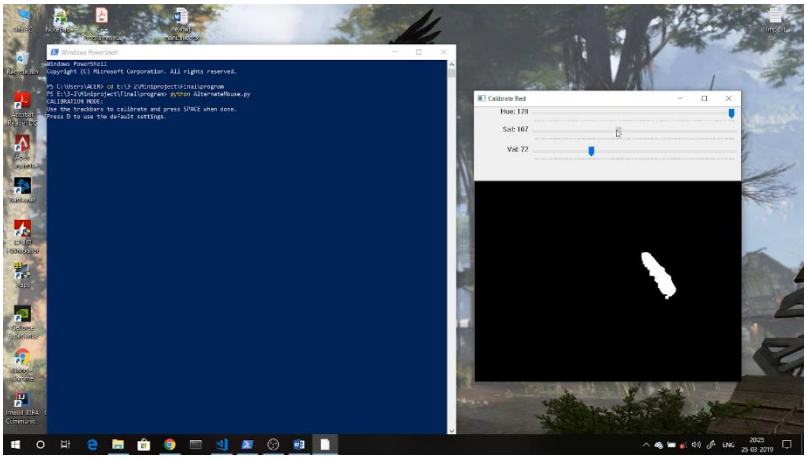
Figure 10:Signs

7.1 COLOR CALIBRATION

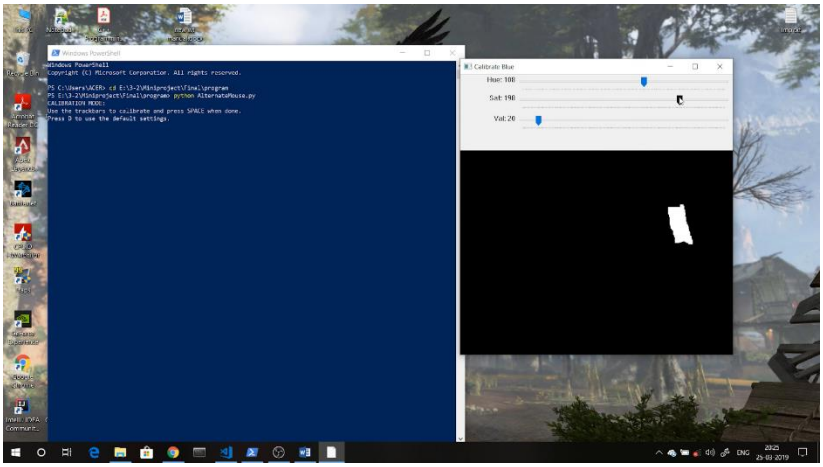
Yellow



Red

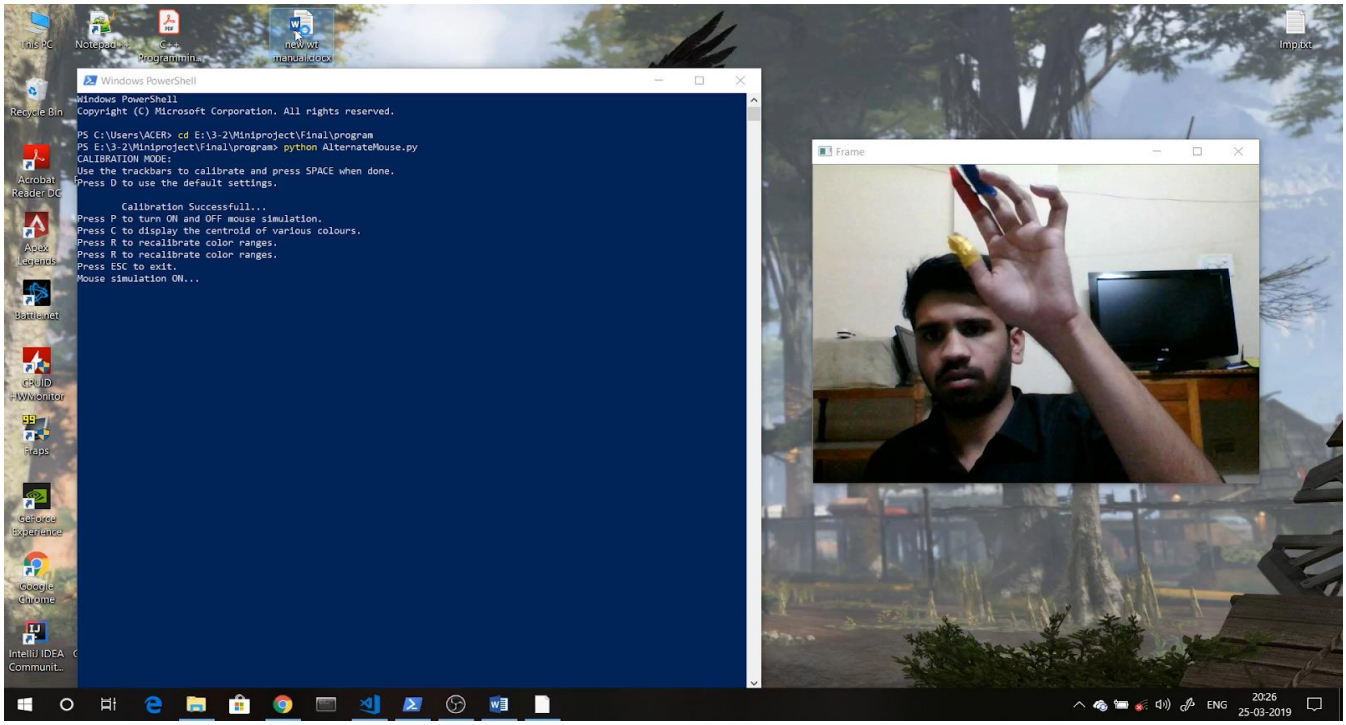


Blue

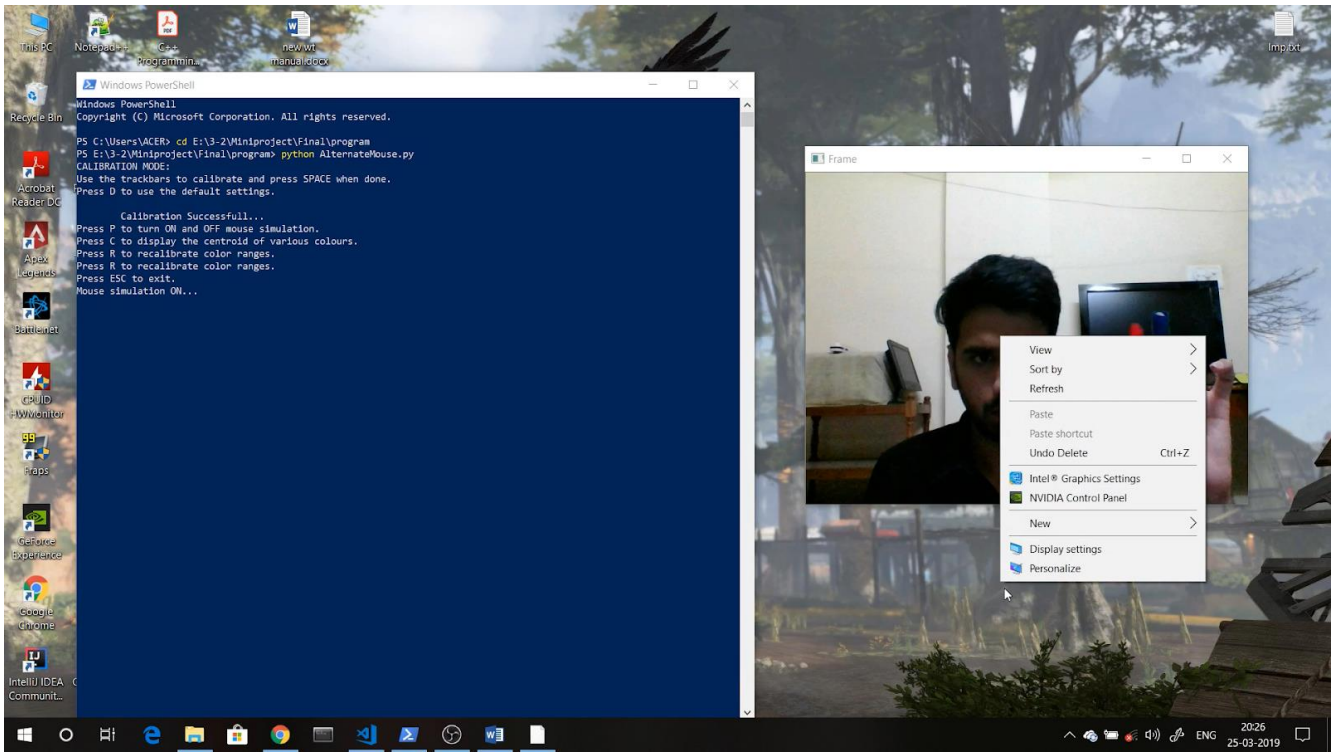


7.2. Clickings

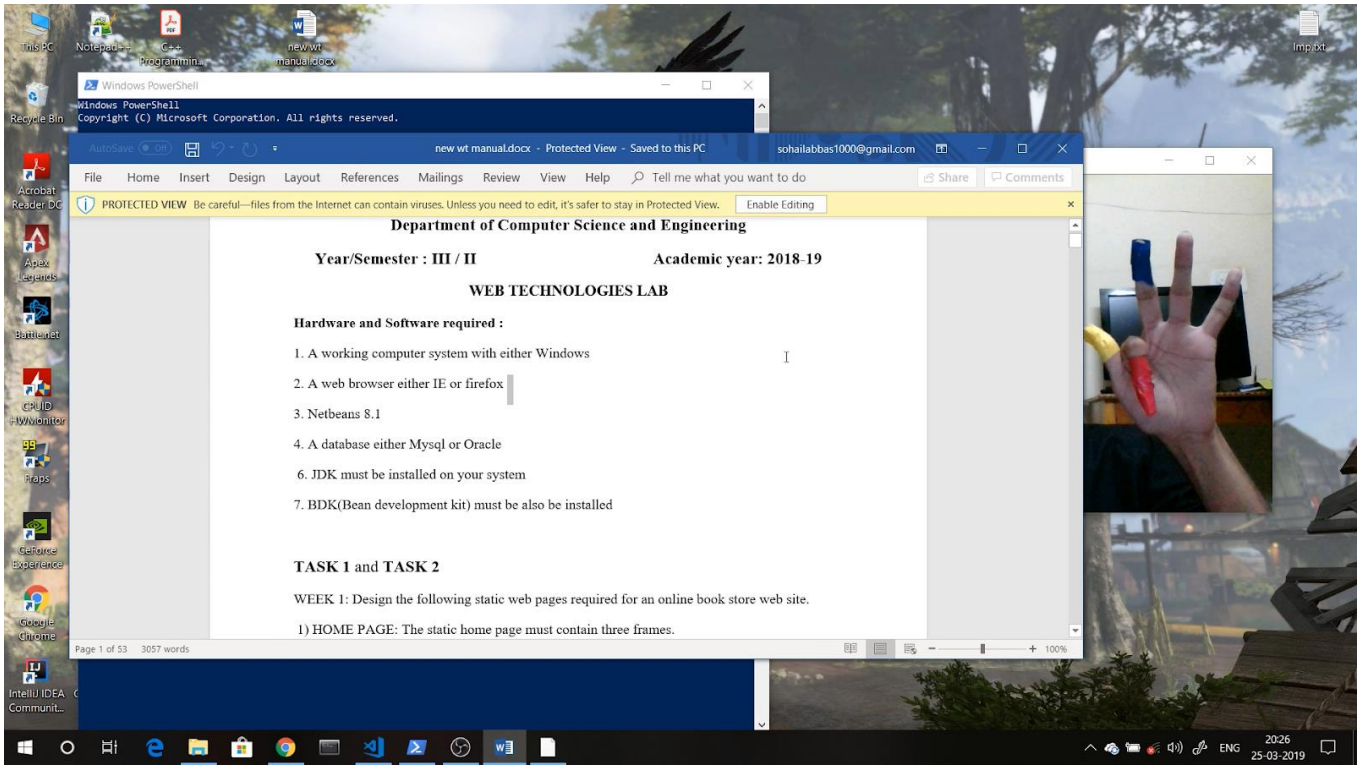
Left-Click



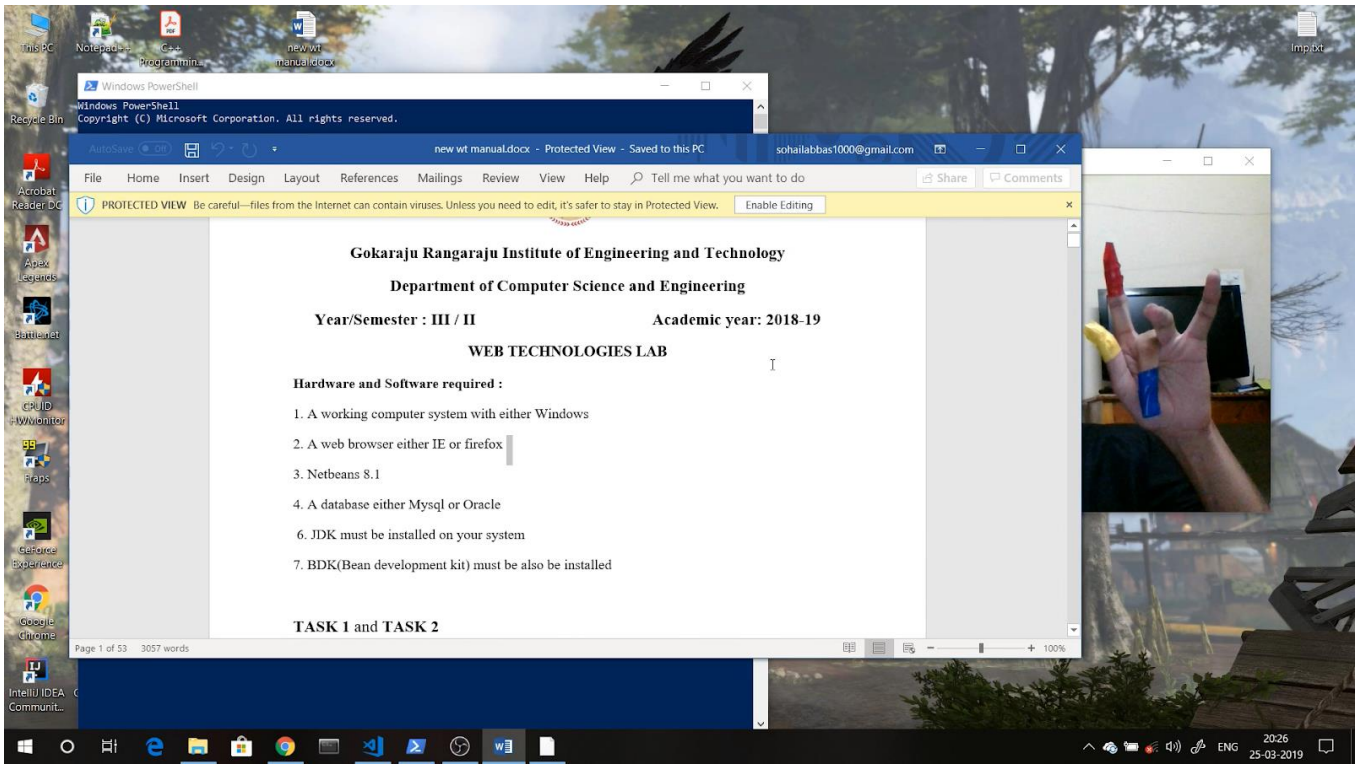
Right-Click



Scroll-Down



Scroll-Up



8. CONCLUSION & FUTURE ENHANCEMENT

The entire project has been developed and deployed as per the requirements stated, it is found to be bug free as per the tested standards that are implemented. Any specification untraced errors will be concentrated in the coming versions, which are planned to be developed. Due to the limitations like hardware problems of mouse we came up with this idea of developing an application which ensures communication between user and system without touching mouse but performs all functionalities . As we are also providing all the functionalities this project can will comes as alternative solution for users so that will be useful where the mouse is not in a condition to work.

We use colors to recognize actions and move the cursor. To further improve the accuracy and simplicity of recognition it can implemented by using the angles between fingers. This would reduce the process of color decoding.

9. BIBLIOGRAPHY

Basic Image Processing-
http://www.tutorialspoint.com/dip/index.htm

OpenCV Python Tutorials-
htstps://docs.opencv.org/3.0beta/doc/py_tutorials/py_tutorials.html

PyAutoGUI-
https://pyautogui.readthedocs.io/en/latest/

MorphologicalTransformation-
http://www.justinliang.com/tutorials/hsv_color_extraction

Contours-
https://docs.opencv.org/3.1.0/d9/d8b/tutorial_py_contours_hierarchy.html

Color_Spaces-
https://opencvpythontutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html