G.R.A.S.P.

Acknowledgement

First of all, we would like to thank all the professors and our facads to believe in our idea and giving us the chance to make this project in ITSP. We would also like to thank our WnCC managers , Kumar Ayush and Kalpesh Krishna for their constant guidance. Our ITSP mentor Arka Sadhu has also been quite helpful to us. A special thanks to our senior Ritwick Choudhry for guiding us and giving us ideas on how to proceed with the image processing techniques and enlightening us on both image processing as well as coding design techniques. Finally, thanks to all my friends who were a part of our project, directly or indirectly, by helping us in testing and motivating us and working alongside with us on their projects.

Introduction

In today's hugely tech-savvy world, improvements in image processing techniques and increasing accessibility using cool and awesome techniques is helping us just to make our lives easier.For example, there is a really nice tool called Air Bar, which converts your non-touch PC/laptop into a touchscreen PC simply by using a small device. This makes tasks a lot simpler and the device easier to use. Similar is the concept of using hand gestures to control certain functions of the computer. There are projects which make use of user's actions to perform various actions like opening a browser, deleting files, searching and playing your favorite songs, and so on. Sophisticated tools take this to an extent unimaginable. Machine learning techniques integrated into these applications make them effortless and seamless to use. Taking inspiration from these projects and keeping in mind the ease of use for the end-user, we decided to make a simple gesture recognition tool, which utilizes the webcam and tracks hand movements to perform certain actions. Now let us talk about what our tool does, and how.

Our project

We decided to make a simple hand recognizing and motion tracking tool, and use the movements to perform actions. Due to time constraints, we decided to fix to lateral, 8-direction movement, similar to what is found in a pattern lock in your smartphone. We are using color masks to isolate the hand, so wearing a glove gives the best results. Initially, the program accesses the webcam, and shows a window, and prompts the user to place his hand in the green rectangles shown in the window. It takes the colors and creates a color mask that separates out the hand from the rest of the frame. This image is then used to create the contours and the central point of the hand. This central point is tracked and the 'noticeable' lateral movements in the hand are recorded. A sequence of such movements is matched against some predefined data, and then if a match is found, the required command is executed and the recorded data is refreshed for new gestures to be recorded.

Working

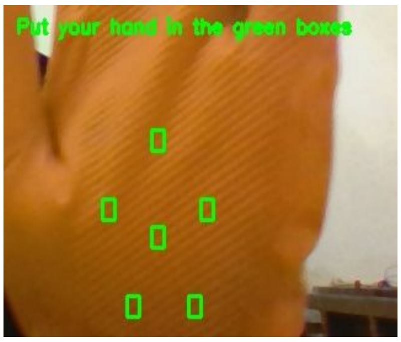
Here comes the interesting part. This tool is written in python using numpy and openCV libraries. Numpy is a library which provides a powerful N-dimensional array object used for complex manipulations. OpenCV uses numpy to store and manipulate image data using numpy arrays (ndarrays). OpenCV is an image processing library built for python and some other different languages. We developed in an Ubuntu 14.04 LTS system. Here are the pre-requisites for the tool to work -

1. Python 2.7.x (We used Python 2.7.6)
2. OpenCV 2.x.x (We used OpenCV 2.4.9)

Warning: Using Python 3.x and/or OpenCV 3.x.x may not work with this project. If you want to test it in a different (virtual) environment without disturbing your current python installation, we recommend you to use 'virtualenv' for setting up a virtual python environment.

3. Numpy (We used v1.8.2)
4. The following modules in python (although these modules are present by default in most or all python installations) – *os,math,subprocess,json,time,datetime*.

Once we have the tools ready, let us get started with reading and understanding the program.
First, we capture the default WebCam using the 'VideoCapture' method of OpenCV. This method takes the default camera of the computer, and if it does not exist, or is used by another process, it will raise an error. A screen similar to the ones below opens up. You need to cover the rectangles entirely with the hand so that the program picks up the right colors.



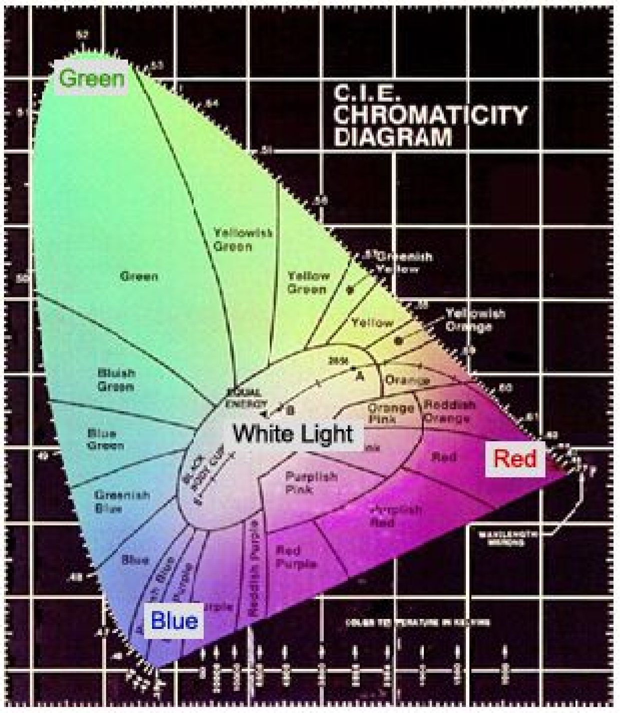
If we are done with the correct placing of the hand, we can simply press 'q' to close the window or wait for some time, it will close automatically in that case. Now, we have the ROI (rectangles of interest) regions extracted from the image. But there is a catch, and that is colorspaces.
Colorspaces, now what are they?

Colorspaces

A colorspace is simply a 3D coordinate system where each point is a different color. Aptly put by **ArcSoft,**

A range of colors can be created by the primary colors of pigment and these colors then define a specific color space. Color space, also known as the color model (or color system), is an abstract mathematical model which simply describes the range of colors as tuples of numbers, typically as 3 or 4 values or color components (e.g. RGB). Basically speaking, color space is an elaboration of the coordinate system and sub-space. Each color in the system is represented by a single dot.

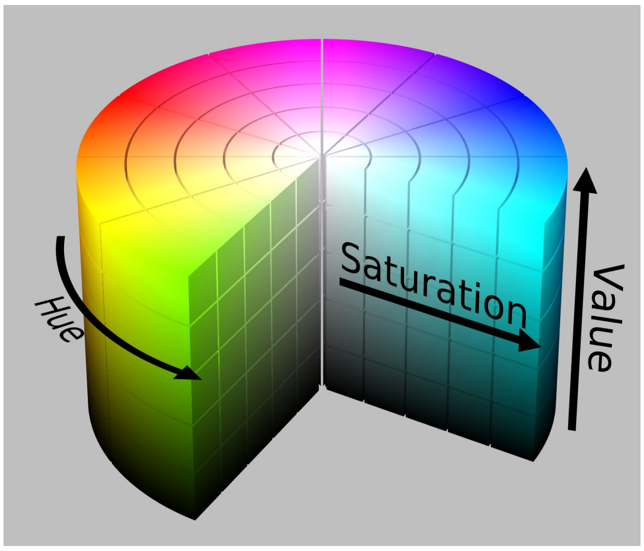
Below is an example of a graph depicting a colorspace. It's very complex and is not required to be fully understood. But there are some things which we should know about colorspaces. Like there are different representations of the 3D coordinate system (cartesian, cylindrical, and spherical), there are different representations of colorspaces as well



(RGB,HSV,CMYK,etc.).

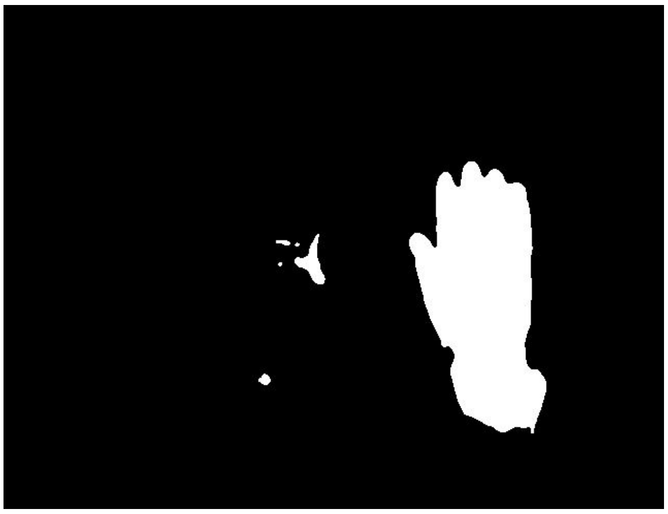
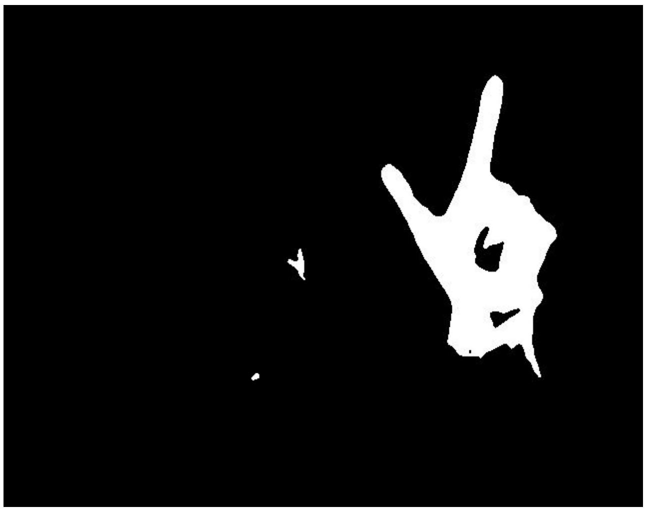
Each colorspace has its own pros and cons. The RBGA colorspace is a common one. Any color can be represented as a sum of Red, Blue, and Green components. In addition to that, there is a alpha blending process of mixing the translucent foreground color with a background color, which is handled by the Alpha channel. This colorspace is useful if we want the Red, Green, and Blue components separately. But it is not so useful when we are comparing the same image in different lighting because the RGBA channels do not give us information about lighting. Hence, we must use another colorspace to get the information. Here comes the HSV colorspace.

HSV colorspace- Hue,Saturation,Value



RGB colorspace can be visualized as a cartesian colorspace, with values ranging from 0 to 255. In contrary, HSV is a cylindrical colorspace which separates the *luma* (intensity values) from the *chroma* (color values). That is the reason why HSV colorspace is used in many image processing applications. In our project, we are going to convert the image from an RGB to HSV image for further processing.

After getting the ROIs from the image, we convert into HSV and then separate the Hue values, which essentially stores the color information as seen from the above diagram. For best approximate results, we sort the ROI arrays and take the median Hue value. We do this for all the 5 ROIs. Then, we create color masks using those colors and an OpenCV method. Every frame taken from the camera is passed through these masks (after some other processing) and we create a sum of those images. Then they are thresholded. Here are some results of the final masked and thresholded image.

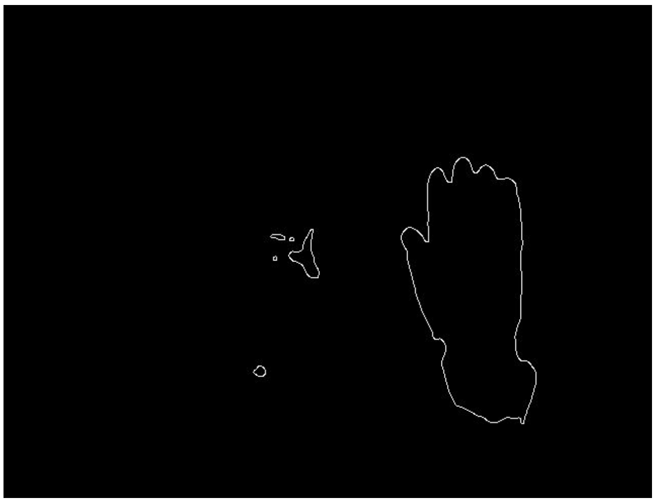
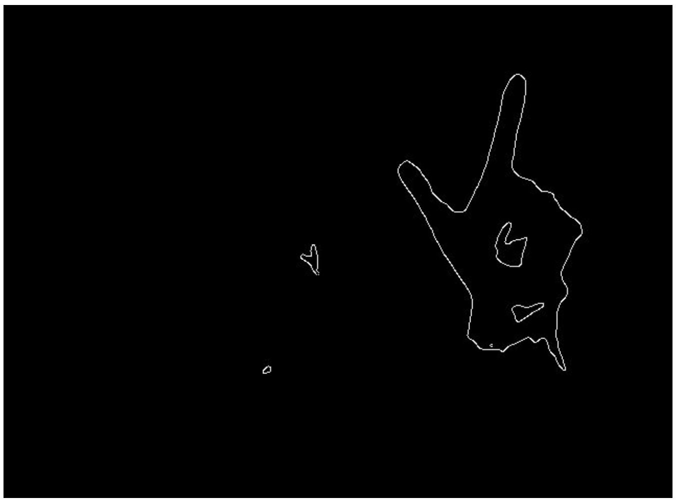



At the time of testing, we used an orange glove. Orange values are similar to skin color in the HSV colorspace, hence a bit of my face is also taken. But we will remove it later.

Before we proceed, we need to find the edges of the filtered image, and we do so with the Canny Edge detector.

Canny Edge Detector

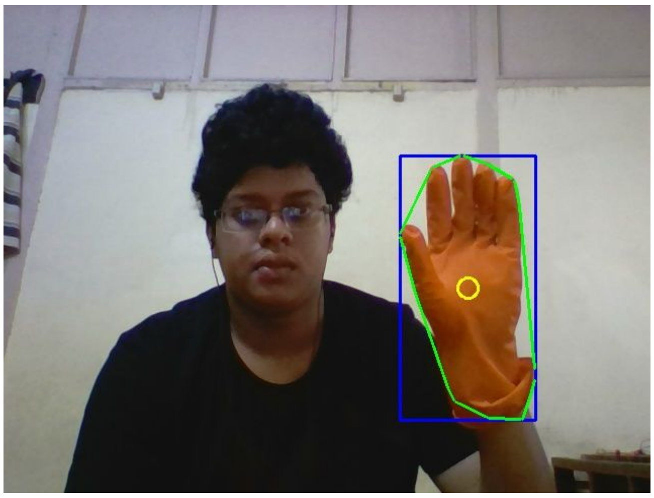
The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Here is an example of the Canny Edge detector algorithm applied to the above images.

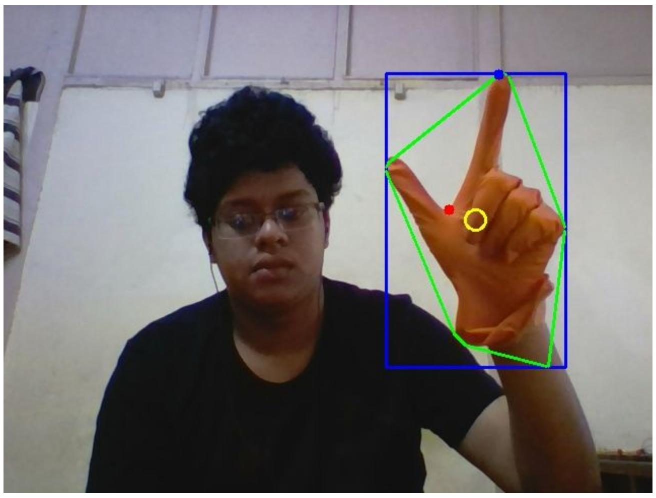



This helps us to visualize the contours very nicely. After that, we can use this image to find the contours and then select the biggest contour which will be the hand. To do this, we pass the frame into the in-built *findContours* function with “RETR_TREE” retrieval mode and “CHAIN_APPROX_NONE” method. “RETR_TREE” mode retrieves all of the contours and reconstructs a full hierarchy of nested contours. “CHAIN_APPROX_NONE” method takes absolutely all the contours without compressing horizontal, vertical or diagonal contours. As the name suggests, it does no approximation. After getting the contours, we use the *arcLength* of the contours to determine the biggest one. We could have used *contourArea* as well, but we found it good (since we are assuming that all other objects will be neglected during the masking stage). After finding the biggest contour, we find the convex Hull (the convex shape that covers the contour) and the bounding rectangle of the biggest contour. This will help us in finding the convexity defects, which can help us find the fingers. After finding the convexity defects, which contains 3 points – *start*,*end* and *far.* The points *start* and *end* are the points on the convex hull and *far* is the point of defect. We discard the defects which might be due to small foldings in the glove or the shape that brings in small defects. We remove them by using a small algorithm. The one line statement of the algorithm is that the minimum of distance(start,far) and distance(end,far) must be greater than 0.25 times the height of the bounding rectangle and the angle SFE (start,far,end) must be less than 80 degrees. These values are the just the results of hit and trial.

Tracking

After finding the contour, we can find the center using *moments* method of OpenCV. This gives us a 'dict' of values, which can be used to find the center. We can also just find the center of the bounding rectangle, but the former gives more accurate results. The final result is this (center marked in yellow) -

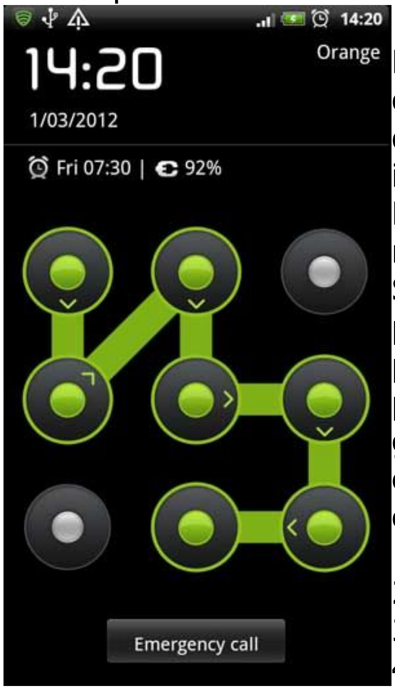




See that tiny little red dot in the second dot? It is the defect that we filtered out. Now, we can just track this point and record the gestures. Now the question is, how do we record?

Lateral movements

Given the time constraints, we decided not to use image templates for comparing gestures since it would involve things like resizing the templates, aligning to match with the recorded gesture, since the user can move his hand from anywhere, not just the center, and proper matching would be more than just a case of image subtraction (one of my friends just told me to subtract the template image from the recorded gesture, but that wasn't the case). So we decided to use 8-direction movement and got inspired from this common technique.



This made our work a lot more simpler. Now all we had to do was track the relative movement of the hand from one frame to another and find out the direction in which the hand goes, then we approximate it with the closest direction and add it to a python list. But what if the user makes some casual hand movements, and they would be recorded by mistake? So we restricted the distance between the tracking point in consecutive frames to be greater than 80 pixels (again, a hit and trial). This makes the user perform relatively *'swift'* hand movements for the gesture to register. And these movements were compared to some data we had in a file. The convention we used was as follows -

1. North-West
2. North
3. North-East
4. East
5. South-East
6. South
7. South-West
8. West

Storing new gestures

If you look carefully at the source code, you will find a file called *gesturedata.json* . This file stores the gestures in a JSON format. The entries will look something like this -

```
{
        "gesture" : [2,4,6,7],
        "command" : "rhythmbox",
        "default" : "true"
}
```

Every gesture stored has a "*gesture*" attribute to store the code of the gesture(more on it later), a "*command*" attribute, which displays the terminal command to perform, and a "*default*" attribute, which specifies whether the attribute is coded into the source code. It is recommended that any other gesture you might want to save must contain these three attributes and the "*default*" be set to "*false*". This will tell the code that it is a user added script.

Gestures having "*default*" value of "*true*" have been given special commands that can be executed, for example, Rhythmbox has full support for play/pause, Volume control, song control and so on. So unless you are a developer and are ready to make changes to the code, it is recommended to keep the "*default*" to "*false*". Plus, the gestures are based on "*prefix-free code*", i.e. the greediest one executes first. For example, if a gesture is [2,3,4] and another gesture is [2,3,4,5], the latter one will never be executed since the former one will come first and it will be compared. This behavior can be overcome by performing some action like closing the fist, but haar cascading techniques aren't that accurate (we will look at it in improvements, maybe). As of now, if you wish to add new gestures, make sure not to overlap them with the previous ones.

With this, the working part comes to an end. This project was a fairly simple one we made, but we have plans to revamp it into a full-fledged gesture recognition app that can perform more complicated actions by incorporating machine learning and more sophisticated and accurate image processing techniques.

How to use

As an end-user, all you need to do is, go to the "*main*" folder, and in the terminal, type "*python main.py*" to open the main process. Make sure there are at least the following files in the directory - *"main.py , functions.py, gestures.py, gesturedata.json".* A small window will open with a text prompting you to place your hand to cover all the green rectangles. Once you are done, you can press '*q*' to exit the window or wait for it to close itself after a few seconds.

Then, a couple of windows open showing the final result in a window named "*frame*" and the mask named "*mask3*". Keeping any window active, press '*t*' to go to "*track mode*", '*i*' to go to "*idle mode*", and '*s*' to save a screenshot. The idle mode does no tracking, and in track mode, the terminal shows the movements you have made so far. Making some swift movements will show the new movement made along with the gesture recorded. In case you make the wrong gesture, simply press '*t*' to reset, or '*i*' to go to idle mode. Once the gesture is recorded , the action will be performed and you will be informed of it via the terminal. Just to be safe, you are sent to the idle mode after the gesture is executed. Press '*t*' again and perform another gesture, and so on. If the keypresses do not work, make sure your Caps Lock is turned off.

To exit the program, press the '*q*' button. If you are still confused, you can check out the video under the "*A demo?*" section below.

Here is a list of gestures by default -

Gesture	**Command**
[2,5,3,6]	Opens firefox with 4 tabs by default – Google,Gmail,YouTube,Quora. URLs can be added, deleted, or modified by changing the "*urls*" property in "*gesturedata.json*",

[2,6,2,6]	Opens Google Chrome with 4 tabs by default – Google,Gmail,YouTube,Quora. URLs can be added, deleted, or modified by changing the "*urls*" property in "*gesturedata.json*",
[2,4,6,7]	Opens RhythmBox
[3,2]	Volume Up in rhythmbox, can also be mapped to other processes.
[7,6]	Volume Down in rhythmbox, can also be mapped to other processes.
[3,5]	Play next song in Rhythmbox.
[1,7]	Play previous song in Rhythmbox.
[2,6,4]	Play Rhythmbox (if paused).
[2,6,8]	PauseRhythmbox (if played).
[5,2,7]	Closes the current process.

Cost

The only thing we had to buy was a glove for color based recognition. We plan to develop the algorithms that can help us in finding the hand independently and separately. Other than that, all software used was open source, so there were no costs involved in it.

Improvements

ITSP isn't the end of this project. It is just the beginning. What you just saw was a small attempt to recognizing gestures and performing small operations. Topics like image processing can be of great use in making our lives easier and designing new software and gadgets that can change the world altogether. We plan to dive more into the formal and conceptual topics of image processing to improve this project.

A demo?

Here is a youtube link showing how the project works - G.R.A.S.P. Demonstration Video 1
Here is a blog post, which is mostly the same as this documentation - G.R.A.S.P. blog post

Feedback/suggestions

We are always ready for your feedback and suggestions. Feel free to contact us.

The team members are -

1. Rohit Kumar Jena
2. Maitreya Verma
3. Abhishek Kumar

----------Thank You!----------