An Analysis of Hand Gesture Recognition

Arpit Goyal
Manikanta Reddy D
Sourav Anand

ABSTRACT

In this era of where when we take a small step, technology takes a giant leap. Countless new innovation spawns each day which looks like something out of science fiction fantasy. Continuing on that trend, our current interface of communicating with computers have become quite old. Like the advents in technology, we need new technologies to interact with computers, technologies which can become more user friendly. Hence, we decided to make an interface through which we can easily communicate with the computer, *a Gesture.*
This analysis utilises opensource libraries of OpenCV available for C++, to implement Gaussian Mixture Models and Color based models for Background subtraction and feature extraction.

WHAT IS GESTURE?

Loosely speaking, Gesture is a form of non-verbal communication which uses various parts of the body like arms and face to communicate. Gesture as a language is found in nature all around us, as animals, who lacks many distinct sounds to communicate effectively, rely mainly on gesture to do so. It was also our primitive language before we started weaving sounds together.

To interact with the computer in a more natural way, we will go down to our basics – communicate with gesture. Give the computer a signal with hand, and the command will be executed.

DEVELOPMENT

SOFTWARE USED

1. Microsoft Visual Studio x64 2013 (Our IDE)

2. OpenCV 2.4.9 (Libraries and function)

The program was written in C++ language.

THE THOUGHT PROCESS

Prior and during development, the basic idea of the project was framed, the essential steps to make our idea a reality. We had certain checkpoints. Our task was

1. To capture the image as a video feed from the camera feed.

2. To track the gesture from the camera feed.

3. To recognize the gesture made.

4. To check the gesture made with the database.

5. Execute the function on the match.

THEORY

We focused mainly two main methods of gesture recognition, Gaussian model and color models.

Gaussian Mixture Model

Every image is made up of pixels and pixels are store of numbers that decode information about the image. These images can be modeled as a set of features. Let us delve a bit more into this. An image can have various components, say a straight line, a sloped line etc. These components can be approximated as a sum/mixture of Gaussians. This is a statistical model and when we say a mixture of Gaussians we imply that at every pixel the numbers it contains are approximately due to a linear combination of Gaussians. So it must be evident that if something is in background and it remains stationary for a long time then its mixture must be fairly constant over time. This is the whole crux of our method. Now we can apply a temporal filter which can subtract these background mixtures from every frame of the input feed and extract the feature as an alpha frame. The background should also be re-estimated at regular intervals so as to maintain a good approximation of the background pixels.

IMPLEMENTATION

OpenCV provides us with elegant mixture model based functions to estimate matrices of Gaussian's of the background, which can then be subtracted. For good tracking of features tweaks must be made to number of n-mixtures used in the mixture model and at $n=6$ we found an optimal balance between computational complexity and tracking precision any higher the computation is complex enough to slow do the program and an smaller gives us a very rough estimation which is prone to misinterpretation.

Color based tracking

The color based model relies on a much simpler mode of feature extraction which is computationally lighter. Every color image lies in the Red-Green-Blue (R,G,B) color space. So every pixel has a coordinate information and then a color information encoded as *(r,g,b)*. So if we are able to recognize the color of the tracking object fairly closely, we can track it to a very good approximation, by estimating its centroid.

It is much simpler to work in Hue-Saturation-Value (HSV) in order to extract bright colors. HSV is an other color space, into which RGB can be easily converted. A HSV space is better to use because unlike RGB, HSV seperates *luma*, or the imageintensity, from *chroma* or the color information.

This is very much useful as we want to seperate color components from intensity for various reasons, such as robustness to lighting changes, removing shadows,etc .However HSV is one of many color spaces that seperate *chroma* from *luma*, like LAB,YCbCr, etc.. HSV is being used because the code for converting from RGB to HSV is easily available and can be easily implemented.

Now by thresholding the HSV values by a low and high bars we can extract the desired color.

IMPLEMENTATION

In our project, we used colour markers as trackers. That means our software will track the colour movement and record it. To recognize the colour, we converted our webcam feed's RGB colour space into easy-to-manipulate HSV colour space by using cvtColor() function. Then by using inRange(), we assigned the range of desired colour between two adjustable H, S and V co-ordinates. This results in the formation of threshold matrix which can be interpretted as the alpha image of all points which have HSV value in the range of our two defined co-ordinates. Now as we have an alpha feed, we can start estimating the median in every frame and track it. This is indeed a very good approximation of the Gesture made by the hand.

RECOGNISING THE GESTURE

Now that we are done tracking and recording a gesture, we must get along with recognising the gesture. The current program can recognise only a few gestures. The gestures we are using are fundamentally quite easy to recognize, for all gestures in this projects are just permutations of 4 directional reading taken up to 3 times. That gives us more than enough, i.e. 64 gestures for this project.

To recognize the direction, we based it on the difference in the co-ordinates of initial point and the current point in the threshold matrix. The recognized direction is registered only when the previous direction registered is not same as the current direction awaiting registry. This way, we can also record the change in direction and string out a gesture.

To terminate the gesture mid-way, we just have to zero out the threshold matrix, i.e. remove the colour marker out of the webcam range.

FINDING A MATCH

The directions registered are labelled as '*l*', '*r*', '*u*' and '*d*' (quiet naturally they mean *left, right, up, down*). They are threaded together in a form of an array of characters for example {*urd*}. They are then compared to pre-defined strings with strcmp() function to find a match. On finding the match, the if condition is initialized and a command is executed.

EXECUTING A COMMAND

We also decide to for a recognized gesture to trigger an event. Upon recognition a gesture can perform a certain operation by utilizing keystrokes api provided by various Operating Systems.