Background:

The aim of the assignment was to familiarize one with development of typical vision processing application with LabView. Image processing and gesture recognition in labview was done from image files on disk and real-time data input obtained from mac's camera.

Problem Statement: Identify various human hand gestures used in the rock, paper and scissors game.

Resources used:
Custom made user libraries/subVI's known as LabVVis were used for performing image processing functions such as thresholding,fetching input data etc. on the input image data.

Following are the general steps.

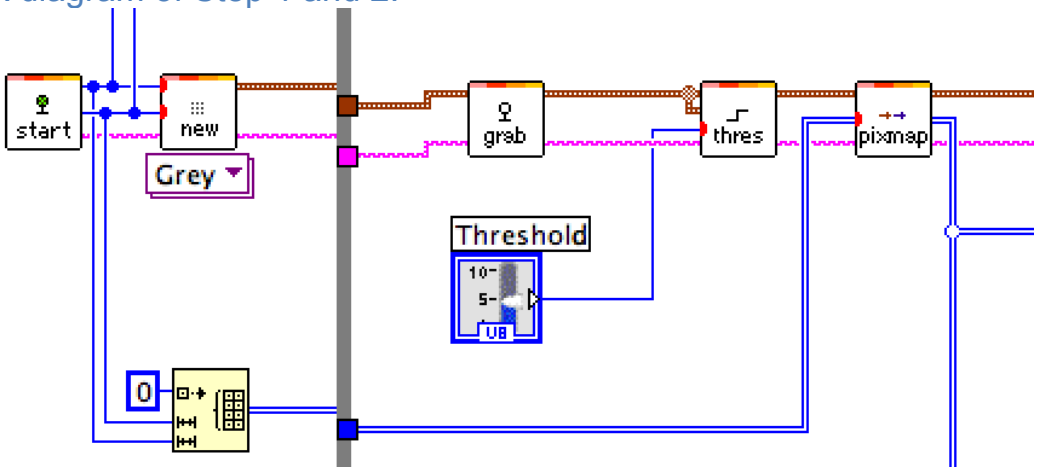
1. Fetching Input data:

A new image is created using the NEW subVI available in the labvvis library and is populated with the image data using either a file input or an input stream from the camera using the grab subVI.

2. Image Segmentation:

Thresholding technique is employed in this program for image segmentation. After converting the input image to a graysacle image, we applied thresholding filter at a given level to create a binary image that can be used for further processing. Thresholding subVIs available in the labVVIS were used to perform this step. The resultant binary images after applying the thresholding filter were as follows:

Block diagram of Step 1 and 2:



3. Image Cropping: For the purpose of simplifying the problem at hand we will only be focusing on right most part of the image as it contains the region of interest (hand's fingers in our case). So we are in a way feeding our simple gesture detection algorithm (described in the next section) a sub array whose width is 1/3 the width of the input image. While rest of the image data is being ignored.

4. Applying Gesture Recognition Algorithm:

General Approach:

The general approach used in our image detection algorithm is counting stream of zeros present in each column.

Let's assume that a black pixel is represent by number zero and white by 1 then a typical columns of our three gestures will look something this like in the diagram. Of course it's somewhat simplified. i.e Noise and other factors like imperfect cropping can hinder in successful gesture recognition. But we can always add buffer columns and do some calibration of input data before applying our algorithm.

Input: A single binary image/frame(produced as a result of applying the threshold filter).

Output: Based on the max probability of columns matching rock, paper scissor gestures that our algorithm will detects, an LED indicator of relevant gesture will light up (just like in the figure below):

Figure: Shows the input gesture and the LED indicator, indicating the recognized gesture.

Rock	Paper	Scissor
1	1	1
1	1	1
1	0	0
1	0	0
1	0	1
1	0	0
1	1	0

Shift Registers:
A number of shift registers were used to hold counter values during each iteration of the array traversal loops. Their brief explanation is as follows:

zero_stream_counter #used for keeping count of the number of streams of zeros in a single column of the cropped input image.

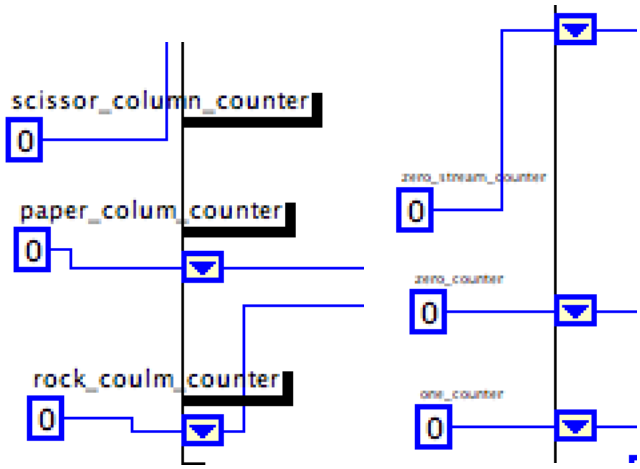
zero_counter #a temporary counter to keep track of the number of black pixels in the current detected stream.

One_counter #a temporary counter to keep track of the number of black pixels in the current detected stream.

Scirssor_column_counter # keeps track of the number of columns matching the criteria of a scissor gesture.

paper_column_counter # keeps track of the number of columns matching the criteria of a paper gesture.

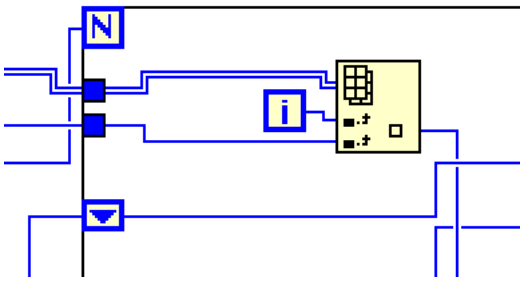
rock_column_counter # keeps track of the number of columns matching the criteria of a rock gesture.



Left figure show the Shift registers used by the outer for loop and right one is of the ones used by the inner for loop.

Algorithm:

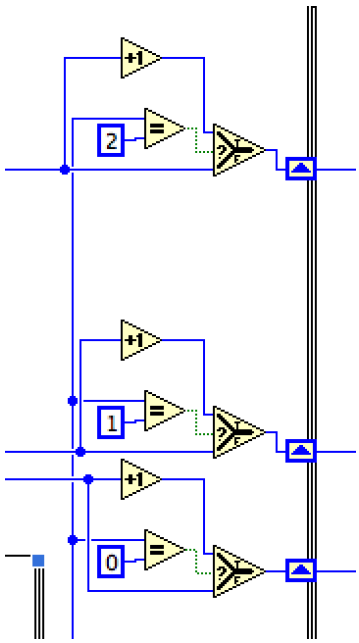
Step 1:
Scan or traverse each column from top to bottom. This is done by picking a column in our cropped region and then scanning each row of that particular column. If we encounter a stream of zeros then increment the value of zero_stream_counter by one(detecting stream is discussed later).



In our Lab View program we used two for loops along with Index Array function for traversing the input image. It was using the output from pixmap subVI in the form of an Ouput Pixmap and the row/column index of array was provided by the iteration counter of the For loops.

Step 2:
At the end of each column based on the gesture criteria we do the following;

if **zero_stream_counter** = 2
then increment the value of scissor_coulumn_counter(because its most likely that this column has two finger(represented by the zero streams) and rest of the image is binary.

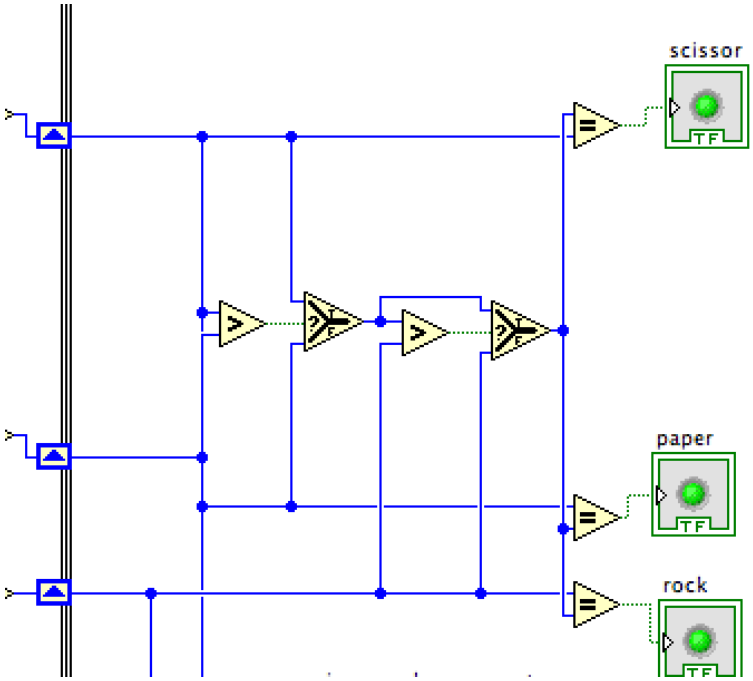


If **zero_stream_counter** = 1
then increment the value of paper_coulumn_counter(again following the same idea discussed above).

If **zero_stream_counter** = 0
then increment the value of rock_coulumn_counter

Step 3:

Compare the there columns counters. The highest number represents the highest probability of a particular gesture present in the input image. i.e the most columns encountered during our image scanning met the criteria(defined by us) of a particular gesture. After Comparison relevant LED is lit up to indicate on the interface, the gesture detected.

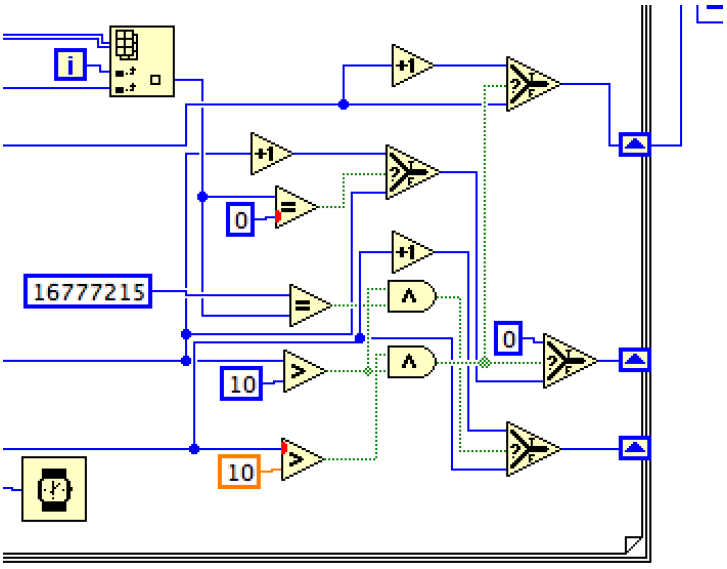


Detecting Streams of zeros:
Lets now discuss the algorithm for detecting a stream of zeros in a column needed in step 1. Traverse each column from top to bottom and do the following:

i. If pixel value is zero:
Increment Zero_counter

ii. If pixel value is white:
Increment one_counter

iii. And finally if one_counter and zero_counter are both greater than say 10. Then increment the value of zero_stream_counter by one(because the zeros stream just ended) and reset the zero_counter before moving on.



Problems faced and Conclusion: Real-time gesture recognition is possible after right calibration of the our hand and a white background to effectively segment our region of interest. Low light conditions also hinder in accurate detection. Even in ideal conditions real-time gesture recognition is a hard problem to solve.