CSCI944 – Perception and Planning

Assignment 3: Finger Processing Application

Group Q

Trine Herrmann (6575705)

Xiaoming Mo (7357564)

Wingsze Cheung (7068682)

Joanna Law (7063891)

23 October 2022

Introduction

In this paper we present a vision processing application implemented in LabVIEW, which accepts an image file as input and performs a set of operations in order to determine what hand is shown (left/right), count the fingers, and identify which fingers are present in the image. In the following sections, we will go provide detailed descriptions of the implementation and processing of our program, including the main program and a number of subVIs. Following this, we will evaluate the performance of the application with results from testing the application on both a set of images of artificial hands which were originally provided with the assignment and a set of images of real hands which we have captured. Finally, we will reflect upon the limitations of our solution and make suggestions for improvements for a later version.

Front Panel

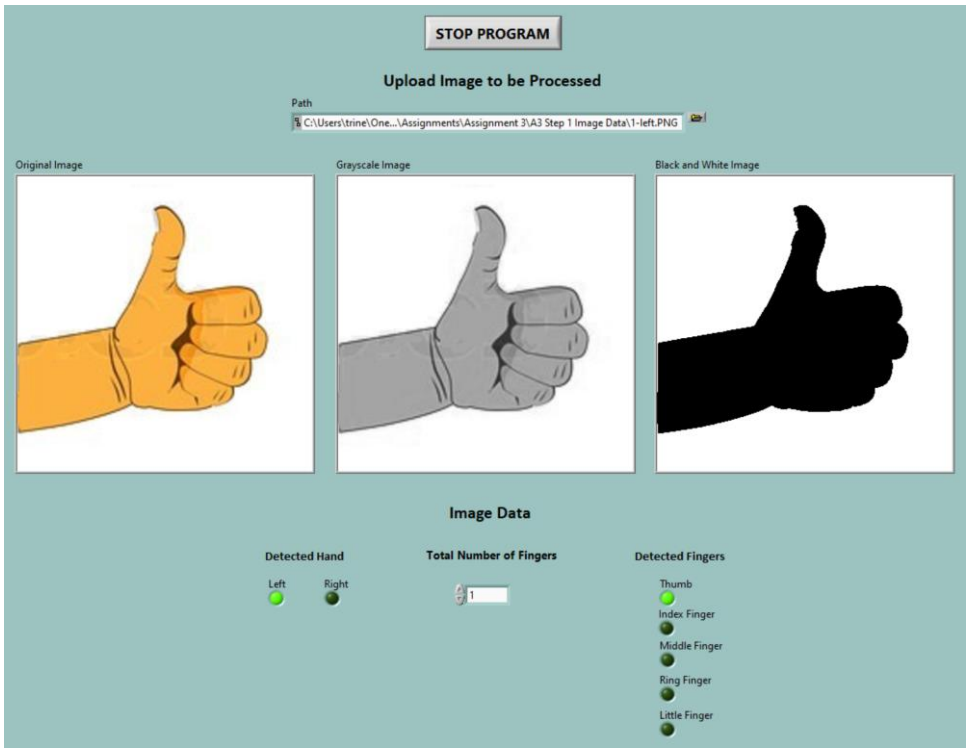

Figure 1: ProcessFingers.vi Front Panel.

The image above displays the front panel of the main program, **ProcessFingers.vi**. This is the interface through which the user can interact with the application. First, the user needs to upload

a PNG image file, and once this has happened, the application extracts information from that image, including whether it is left or right hand, the finger count, and which fingers are present in the image. While the finger count is displayed as a number, Boolean lights are used to indicate the information about what hand is present and which fingers are present. For example, if a left hand is detected, the light below “Left” will light up and the light below “Right” will remain off.

In addition to displaying this information about the image, the application also displays three different images to visualise some of the pre-processing steps involved. The first image is simply the original image which the user has chosen to upload. The second image is the same image after being converted to grayscale, whereas the last image is a binary mask where all pixels of the hand are black, and all other pixels are white.

The user can keep uploading as many images as they wish, and every time a new image is uploaded, the application displays the updated information about the image. Only when the user presses the button to stop the program, does it stop running.

Block Diagram

The main program can be divided into three sections: 1. pre-processing, 2. left/right hand detection, and 3. fingers count and detection.

1. **Pre-Processing**

The image pre-processing is handled by two subVIs: **Grayscale-convert SubVI** and **BW-convert SubVI**, which we use to convert an image to grayscale and to black and white, pixel by pixel. This allows us to better analyse the pixels in the image and perform the operations required to extract information about the image. The details of these subVIs will be explained in the SubVIs section.

We use the size of the hand in the input image to determine which path the program goes in the block diagram in order to be able to handle both images of full-sized hands and images where the hand is smaller.

To determine the size of hands, we count the number of black pixels in the image. If there are over 30,000 black pixels, the image is considered to be “big”. Otherwise, it is considered to be “small”.

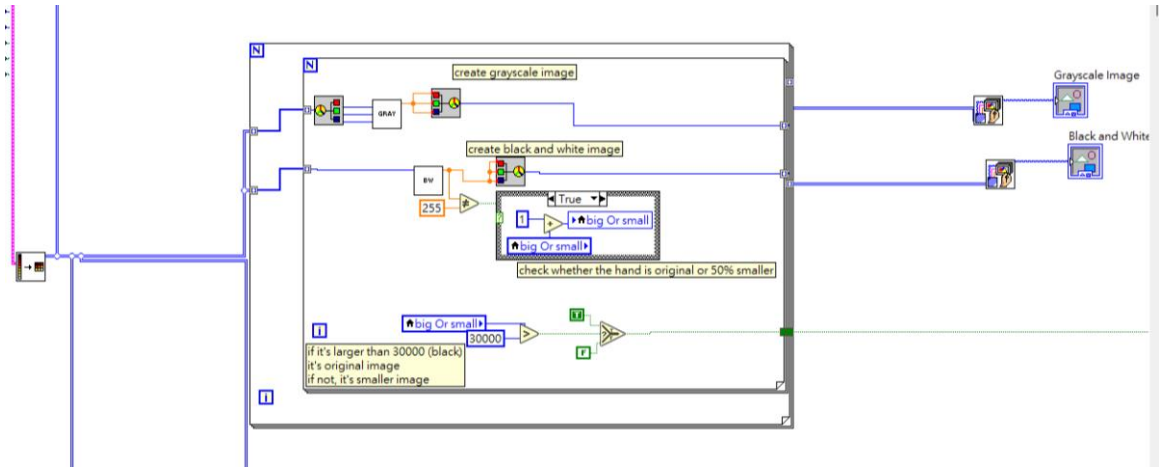

Figure 2: Section of block diagram that handles image pre-processing and determines the size of hands.

2. Left/Right Hand Detection

We count the black pixels at the edges of the image to determine the direction of where hands come in (left/right/top/bottom). To specify the target pixels, we initiate two variables: **i (row) and j (column)**. If the number of black pixels at an edge is greater than or equal to 10, we can conclude that the wrist comes from the corresponding direction. If a hand is coming from the top or left, the LED light of the left hand is on. If a hand is coming from the bottom or right, the LED light of the right hand is on. This based on the assumption of the patterns we learned from the given images. In these images, left hands only come in from top and left, and right hands from bottom and right.

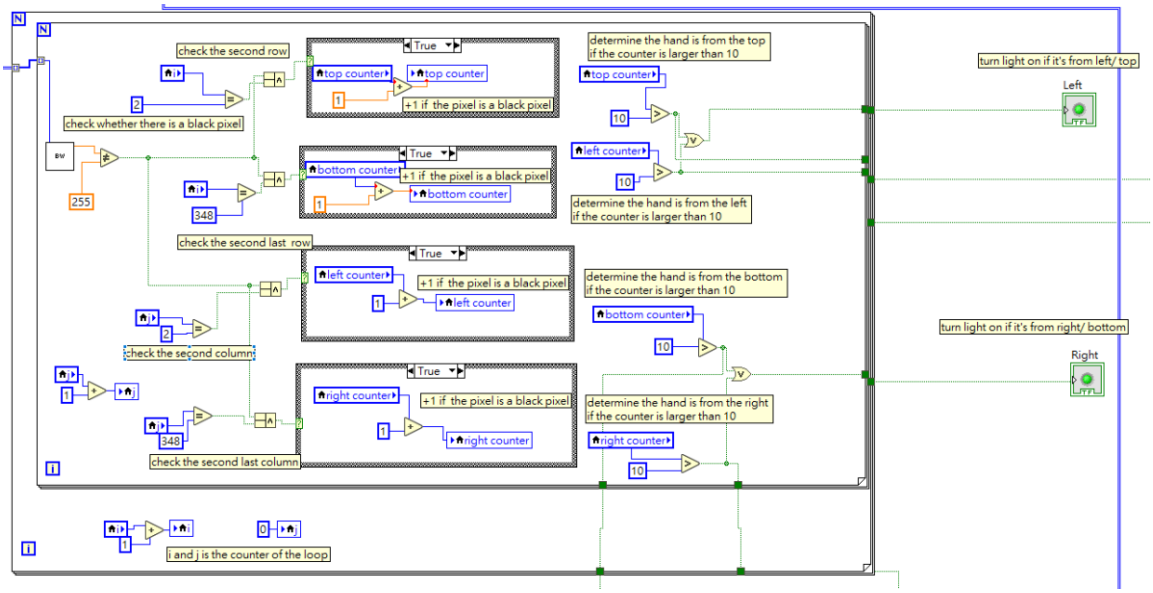

Figure 3: Section of block diagram that determines the direction of hands.

3. Fingers Count and Detection

This section is divided into 8 cases: 4 directions for both big hands and small hands. Different thresholds are set in different cases to detect and count fingers. For each case, the program handles thumb and other fingers separately.

For thumbs, if there are more than 100 black pixels in the specific threshold, the LED light for thumb is on, and the thumb counter increases by 1.

To count other fingers, we first initialise a “found finger” flag. At specific thresholds, if there is a black pixel and “found finger” flag is false, the finger counter increases by 1 and “found finger” flag is set to true. If a white pixel is detected, the “found finger” flag changes to false. Since the “non-thumb” fingers appearing must start from the index finger and must be consecutive (an assumption based on the given images), we can determine which fingers are shown by the value of the finger counter. Finally, the indicator Total Number of Fingers sums fingers counter and thumb counter and displays its value.

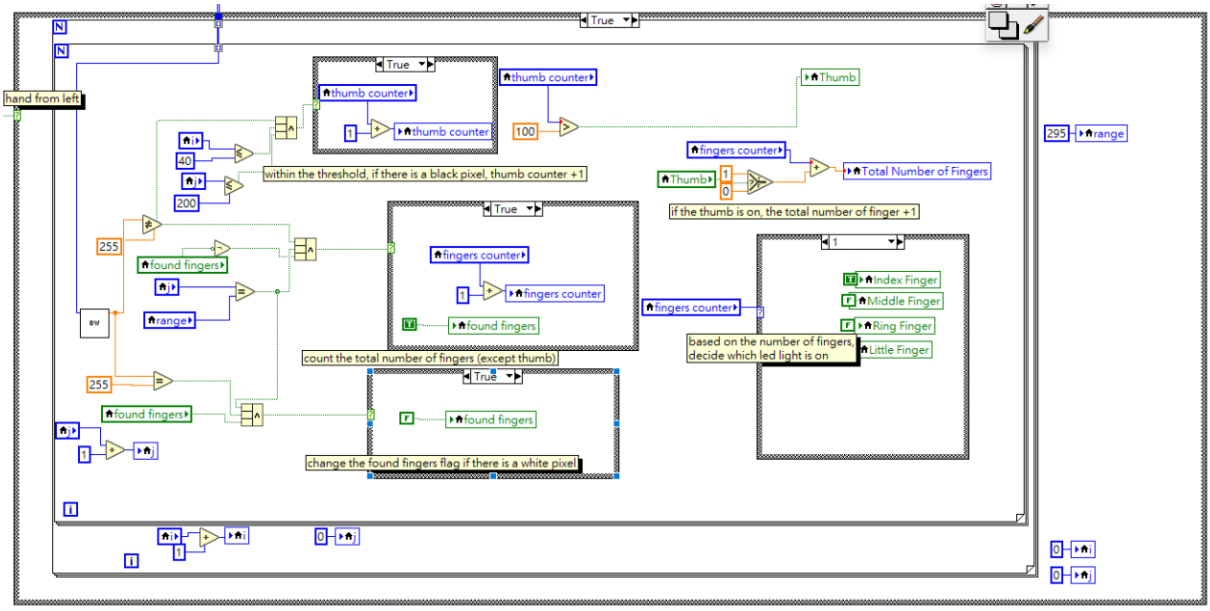

Figure 4: Section of block diagram for finger count and detection in the case of big hand from left

SubVIs

In this section, we will present the subVIs which we have created and utilised for our application. These include a subVI for converting pixels to grayscale and another for converting pixels to black and white (binary mask).

Grayscale-convert SubVI

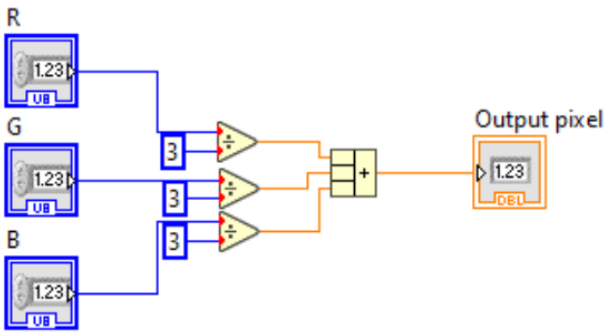

Figure 5: Grayscale-convert.vi Block Diagram.

Grayscale-convert is a subVI which has the purpose of converting a pixel to grayscale. It accepts three numbers as input, representing RGB values, then divides each of these numbers by 3 before summing the three resulting values into one value to be outputted as the new grayscale pixel.

BW-convert SubVI

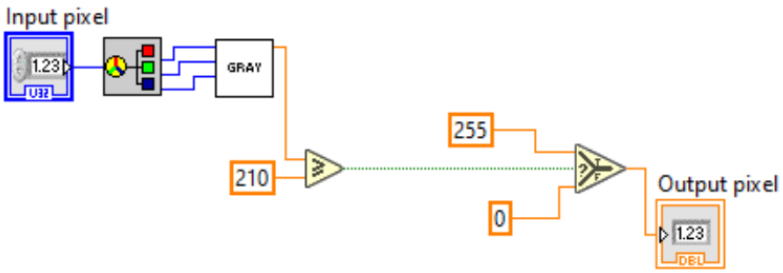

Figure 6: BW-convert.vi Block Diagram.

The BW-convert subVI is a function to be used for transforming a colour image into a black and white image as a binary mask in order to extract the hand from the background. This function does not transform the entire image at once but must be used on each pixel in an image. BW-convert accepts a numeric value as input, representing one colour pixel. It separates this pixel into three values representing RGB and utilises the grayscale-convert subVI to convert them into a single grayscale pixel. After this, it checks whether this pixel is larger than or equal to a certain threshold (in this case, 210, determined by trial and error), and sets its value to 255 if that is true and 0 if it is false. This means that each pixel which this subVI is applied to will be transformed to white if it meets the specified threshold or to black if it is below the threshold. That way, the hand can be extracted from the background which enables us to perform operations to extract the relevant information, such as determining what hand is shown and counting the fingers.

Performance

To test how well our finger processing program can perform its operations of detecting what hand is present, how many fingers are present, and which fingers are present, we are utilising two sets of images. The first set includes all the 16 images which were provided with the assignment. The other set includes 29 of our own images which we have captured of real hands and edited slightly to prepare for our program. We will display the test results of each set of images in the two tables below and reflect upon what these results tell us about the performance of our program.

Tests on Provided Images

Name	Hand (True)	Hand (Detected)	Count (True)	Count (Detected)	Fingers (True)	Fingers (Detected)
0-left.png	Left	Left	0	0	None	None
0-right.png	Right	Right	0	0	None	None
1-left.png	Left	Left	1	1	Thumb	Thumb
1-right.png	Right	Right	1	1	Thumb	Thumb
2-left.png	Left	Left	2	2	Thumb, index	Thumb, index
2-right.png	Right	Right	2	2	Thumb, index	Thumb, index
2a-right.png	Right	Right	2	2	Index, middle	Index, middle
3-left.png	Left	Left	3	3	Thumb, index, middle	Thumb, index, middle
3a-left.png	Left	Left	3	3	Thumb, index, middle	Thumb, index, middle
3-right.png	Right	Right	3	3	Thumb, index, middle	Thumb, index, middle
4-left.png	Left	Left	4	4	Index, middle, ring, little	Index, middle, ring, little
4a-left.png	Left	Left	4	4	Index, middle, ring, little	Index, middle, ring, little
4-right.png	Right	Right	4	4	Index, middle, ring, little	Index, middle, ring, little
5-left.png	Left	Left	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little
5-right.png	Right	Right	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little
5a-right.png	Right	Right	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little

Table 1: Tests Results from Provided Images.

From the results above, we can see that our program was able to achieve a high performance on correctly detecting the hand, number of fingers, and names of fingers shown in all the images that were provided with the assignment. However, these were quite simple images of artificial hands. For that reason, it will be interesting to see how well our program can perform its operations on images of real hands as we will see below.

Tests on Own Images

Name	Hand (True)	Hand (Detected)	Count (True)	Count (Detected)	Fingers (True)	Fingers (Detected)
big-0-bottom.png	Right	Right	0	0	None	None
big-0-left.png	Left	Left	0	0	None	None
big-0-right.png	Right	Right	0	0	None	None
big-0-top.png	Left	Left	0	0	None	None
big-1-left.png	Left	Left	1	1	Thumb	Thumb
big-2-bottom.png	Right	Right	2	2	Thumb, index	Thumb, index
big-2-left.png	Left	Left	2	2	Thumb, index	Thumb, index
big-2-right.png	Right	Right	2	2	Thumb, index	Thumb, index
big-2-top.png	Left	Left	2	2	Thumb, index	Thumb, index
big-3-bottom.png	Right	Right	3	3	Thumb, index, middle	Thumb, index, middle
big-3-left.png	Left	Left	3	3	Thumb, index, middle	Thumb, index, middle

big-3-right.png	Right	Right	3	3	Thumb, index, middle	Thumb, index, middle
big-3-top.png	Left	Left	3	3	Thumb, index, middle	Thumb, index, middle
big-5-right.png	Right	Right	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little
small-0-left.png	Left	Left	0	0	None	None
small-0-right.png	Right	Right	0	0	None	None
small-1-left.png	Left	Left	1	1	Thumb	Thumb
small-1-right.png	Right	Right	1	1	Thumb	Thumb
small-2-left.png	Left	Left	2	2	Thumb, index	Thumb, index
small-2-right.png	Right	Right	2	2	Thumb, index	Thumb, index
small-2-top.png	Left	Left	2	2	Thumb, index	Thumb, index
small-3-bottom.png	Right	Right	3	3	Thumb, index, middle	Thumb, index, middle
small-3-right.png	Right	Right	3	3	Thumb, index, middle	Thumb, index, middle
small-4-bottom.png	Right	Right	4	4	Index, middle, ring, little	Index, middle, ring, little
small-4-left.png	Left	Left	4	4	Index, middle, ring, little	Index, middle, ring, little
small-4-right.png	Right	Right	4	4	Index, middle, ring, little	Index, middle, ring, little
small-4-top.png	Left	Left	4	4	Index, middle, ring, little	Index, middle, ring, little
small-5-bottom.png	Right	Right	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little
small-5-right.png	Right	Right	5	5	Thumb, index, middle, ring, little	Thumb, index, middle, ring, little

Table 2: Test Results from Own Images.

The results above show that our program is also able to correctly detect left/right hand, count the number of fingers, and identify which fingers are shown in images of real hands. This suggests that we have been able to develop a robust application for processing all the images which we have used for testing, including images of artificial and real hands.

Limitations

For this assignment, we have focused on building a program that can correctly perform its operations on specific images, including those initially provided as well as our own images. While our program was able to perform well on these images, that does not necessarily mean that this program would also perform well if tested in a real-world scenario where we cannot control what hand gestures people might do. This is because our solution is based on certain assumptions, for

example, that the middle finger will never be present without the index finger, and the ring and little finger will never be present without the index and middle finger. However, in reality, this might not always be the case. For example, a user may choose to upload an image to the program where only the middle finger is showing or where the index and little finger are showing. In such situations, our program would fail to correctly identify the fingers. For this reason, we may consider our program an initial exploration of the possibilities and the challenges of developing a finger processing application. In a later version, we can improve the functionality of the program to tackle these challenges. A potential solution to the problems related to correctly recognising the fingers is to find the centre point of the palm as well as a point at each fingertip present, and then getting the angles of those fingertips and using this information to determine which fingers are present. However, due to time constraints, it was not possible for us to implement and test the performance of this solution at this time.