Hand Gesture Recognition

Authors: Ryan McGinty, Nhat-Cuong Ly, Alexa Yao

Spring 2015 EE443

Department of Electrical Engineering

University of Washington

Seattle, WA 98195

I. Project Summary

An increasingly impactful part of everyday life is how we interact with our technological devices - most notably our computers. Much of the ingenuity stemming from human-computer interaction research focuses only on improving current mainstream devices out there today. Only a few modes of Human-Computer interaction exist today: namely through keyboards, mouses, touch screens, and other handheld helper devices. Each of these devices has been confronted with their own limitations when adapting to more powerful and versatile hardware in computers.

A commonality between the issues of these devices are this: providing an intuitive mode of human-computer interaction cheaply without the additive of extra devices. With that said, we felt there were ways we could build an intuitive gesture-control mode of human-control interaction without the need to necessarily build another device. When tackling this problem of creating a new mode of human-computer interaction we knew we could utilize a combination of built-in functions that is typically provided in most computer designs. Specifically we have exploited the built-in webcam that has become a standard feature in most computers. This feature provides us a way to track and respond user hand-free movements and gestures.

Using a combination object detection and recognition, the following project successfully builds a computationally inexpensive static hand gesture recognition system using a simple RGB webcam - creating a truly more natural form of human-computer interaction.

II. Problem Formulation

1. Skin Segmentation

The first step in implementing our particular gesture-recognition system is being able to effectively segment skin pixels from non-skin pixels. By using only a simple RGB-based webcam we are limited in methods for locating and distinguishing static hand gestures. For this reason, we have chosen to focus on segmentation using various color spaces. Skin segmentation methods are generally computationally inexpensive, and moreover, they can function robustly across many different models of simple webcams - an important feature given the notable difference in quality, color, etc. that can exist between webcams.

Specifically, we use a basic thresholding technique, choosing min and max threshold values which contain well-known and thoroughly tested skin pixel value regions. In total we use three different color spaces: 1) RGB (red-blue-green), 2) HSV (Hue-Saturation-Value), and 3) YCbCr (luminance, blue-difference and red-difference chroma). Using three color spaces is a computationally efficient procedure, especially given the added robustness it provides to distinguishing skin and non-skin values. HSV and YCbCr color spaces provide additional information about the separation between luminance and chrominance that RGB color space doesn’t provide. In all we use nine different pairings of min- and max- threshold values, each representing a given color space, then, applied certain boundaries rules introduced by Thakur et al. [2]. A particular pixel is part of skin region if and only if that pixel value passes boundaries rules.

2. Hand Detection

The next step in implementing this static gesture recognition system is locating the hand in the video frame. At this stage we are provided a segmented skin mask of the original image. While the segmented skin mask provides us the regions in which the hand could be, the hand is only part of all that is segmented from the mask. Since the mask captures skin regions - most notably the arm and face skin - we must do more processing to find the hand. Specifically, we want to find an indistinguishable characteristic to the hand, namely a consistent feature we can always find. With that mindset, we felt the palm of the hand provided that consistency. Much of the difficulty in finding the location of the palm is the variability in size. We didn’t have any information regarding the position of the user’s hand as well as size (i.e. the closer the user’s hand to the screen the larger hand appears). In addition, we were more interested in fingers which played a crucial role in static hand gesture recognition algorithm (discussed in the next section).

To solve this problem, we took advantage of the hand geometry. Any human hand’s largest cross section area lies somewhere near the middle of the palm and independent of size. With this property, we use the centroid equation to find the arithmetic mean position of the hand which corresponds to the center of the palm. We can easily differentiate between the top and bottom of the hand after finding center of the hand. By using this method, we provided a robust algorithm to detect the center of any human hand.

3. Static Hand Gesture Recognition

After the location of the hand is found, the challenge becomes identifying different gestures of the given hand. The main difficulty in this is what type of features to use in distinguishing various static gestures. We concluded that most static gestures relied on the hand’s fingers. Though a simple conclusion, it is this very deduction that our algorithm was built around. While more complex algorithms for contour detection and gesture recognition exist, we chose a simpler algorithm based on and adapted from Malima [3].

In our adapted model, we draw a rectangle around the center of the palm found in the preceding procedure. This rectangle’s size is proportional to the furthest skin pixel value from the center of the palm. Ideally the rectangle drawn does not entirely encapsulate the hand. Instead, the lines of the rectangle only contain the palm, cutting through the fingers and wrist. In this process we make a key assumption, in that the hand is upright (i.e. the line of the rectangle that cuts through the fingers is known.) Then we simply calculate the number of fingers that cross the line of the rectangle (calculations explained in greater detail in the section below.) This algorithm provides us a very simple, computationally efficient method for counting the number of fingers any given hand has up.

III. Project Algorithms and Implementation

1. Skin Segmentation

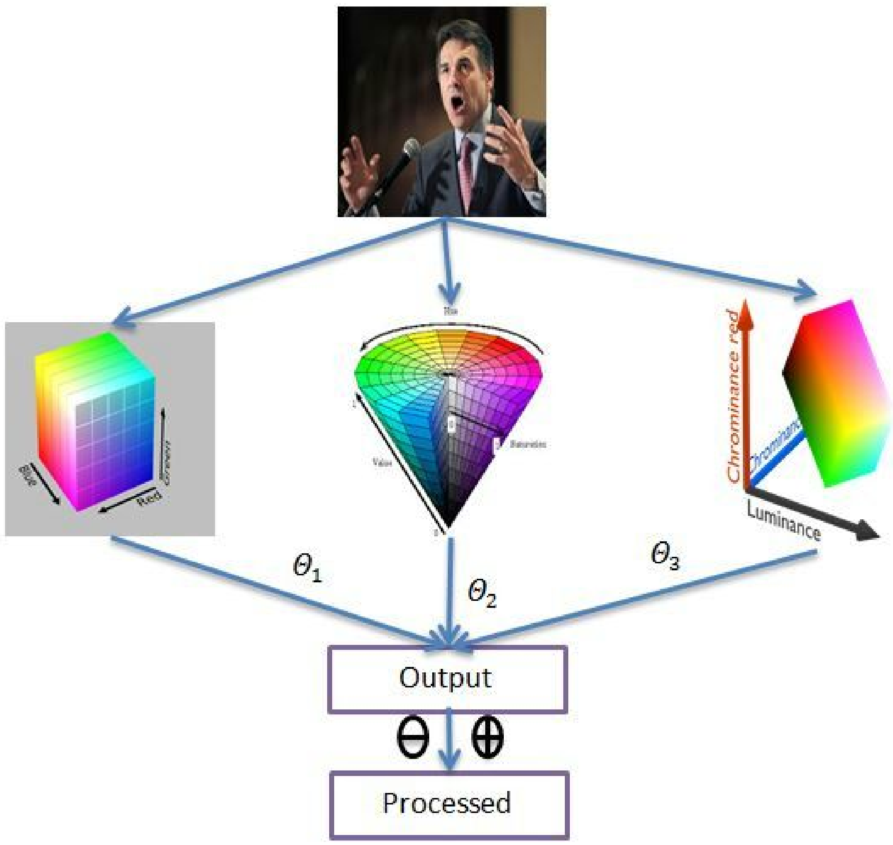

Figure 1. Skin segmentation system.

The figure above describes an overview of our skin segmentation system. Each input frame is converted to HSV and YCbCr color spaces (input frame is in RGB color space). Then, each color space is applied its thresholded using the specified boundaries found below. In RGB color space, the boundaries rules are the following:

$$R > 50 \;\&\&\; G > 40 \;\&\&\; B > 20 \;\&\&\; (\max(R,G,B) - \min(R,G,B)) > 10 \;\&\& \\ abs(R-G) \geq 10 \;\&\&\; R > G \;\&\&\; R > B \qquad (1)$$

$$R > 220 \;\&\&\; G > 210 \;\&\&\; B > 170 \;\&\&\; abs(R-G) < 15 \;\&\&\; R > B \;\&\&\; G > B \qquad (2)$$

In the HSV and YCbCr color spaces, the boundaries are described below. In each of these color spaces, we only use two channels: H and V, Cb and Cr for HSV and YCbCr color space, respectively.

$$Cb \geq 60 \;\&\&\; Cb \leq 130 \qquad (3)$$
$$Cr \geq 130 \;\&\&\; Cr \leq 165 \qquad (4)$$
$$H \geq 0 \;\&\&\; H \leq 50 \qquad (5)$$
$$S \geq 0.1 \;\&\&\; S \leq 0.9 \qquad (6)$$

With the given boundaries rules above, a particular pixel is classified as a skin color if and only if it passes the following conditional test:

```
If( (1) || (2) ){
      If( (3) && (4) ){
            If( (5) && (6) ){
                  Pixel is skin
            }
      }
}
```

In order to correctly distinguish skin from non-skin pixels using the given boundary conditions, the right conversion methods between RGB and the HSV and YCbCr color spaces is essential. In the HSV color space, hue channel has to be in range of 0^o-360^o whereas saturation channel has to be between 0 and 1. In case of YCbCr color space, both Cb and Cr channels have to be between 16 and 240. The equations below give the proper transformations from RGB color space to HSV and YCbCr color space.

$$Cb = 128 - 0.148 * R - 0.291 * G + 0.439 * B$$
$$Cr = 128 + 0.439 * R - 0.368 * G + 0.071 * B \qquad [4]$$

$$R' = \frac{R}{255}; G' = \frac{G}{255}; B' = \frac{B}{255}$$

$$C_{max} = \max(R', G', B'); C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

$$H = \begin{cases} 0^\circ & ;\Delta = 0 \\ 60^\circ * \left(\frac{G' - B'}{\Delta} \%6\right) & ; C_{max} = R' \\ 60^\circ * \left(\frac{B' - R'}{\Delta} + 2\right) & ; C_{max} = G' \\ 60^\circ * \left(\frac{R' - G'}{\Delta} + 4\right) & ; C_{max} = B' \end{cases}$$

$$S = \begin{cases} 0 & ; C_{max} = 0 \\ \frac{\Delta}{C_{max}} & ; C_{max} \neq 0 \end{cases}$$ [5]

There do exist some false positive skin pixel values. In order to eliminate those false positives - generally in the form of static white noise - we apply morphological operations. We use OpenCV functions *erode* and *dilate* to perform closing operation. For these functions, we chose structure element with size of 3x3 and 5x5 for *erode* and *dilate*, respectively.

**Note: In order to ease the amount of processing power necessary for our program, we designate only a subregion of original input frame to do skin segmentation. This also provides the user with a specific region to actually present static gestures.

2. Hand Detection

From the procedure above, we are ensured a skin segmented mask. With that given segmented mask, we apply the centroid equation explained in the problem formulation section above. These equations ideally outputs the (x,y) coordinates for the center of the palm. Below you will find these equations.

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_i}{i}; \bar{y} = \frac{y_1 + y_2 + \cdots + y_j}{j}$$ [6]

Because the pixels in the mask are either white or black (i.e. 1 or 0), the equation is polarized to only skin pixel values.

3. Static Hand Gesture Recognition

As described in the problem formulation, we draw a rectangle around the center of the hand, whose coordinates are approximated by the centroid equation above. The size of this rectangle is calculated by finding distance from the center coordinates and the last counted skin pixel value in the matrix. Note that the “last” skin pixel value is simply the last set of coordinates with a white pixel when the matrix is initially traversed during the centroid calculations. i.e. Since the matrix is traversed using for loops - rows and then columns - the bottom row is usually where the last white pixel value resides.

From this distance *d* calculated above, we draw a rectangle that has length of twice the calculated distance *d* with center of that rectangle corresponds to the center of the segmented hand. This proportion was tested and most often drew the edges of the rectangle in the correct location.

Here, we make one large assumption: the user uses his right hand. Given that the box for computations is positioned to the right of the user, it is more intuitive. Still it is necessary to note.

Assuming the right hand is used and the rectangle is drawn such that its edges cut across the fingers of the hand - when the fingers are fully extended - we can simply count the number of white & black changes happen across the line:

```
Given:      Line of top edge of rectangle $(x_0,y_0) \rightarrow (x_0,y_N)$
     N = length of line
     SkinMask = Matrix corresponding to mask image
     pixCount = 0 , counts number of skin pixels in a row
     FingerThreshold = 10 , # of consecutive skin pixels needed in order to
            classify as a finger.
for j = 0 --> N
     if SkinMask($x_0$,$y_j$) == 1:
          pixCount++
     else:
          if pixCount > FingerThreshold:
               number of fingers++
```

This method is not limited to horizontal counting, as it can also be applied to vertical lines of the rectangle. From this method, we can count the number of finger the user is holding up. From this feature, we can create a multitude of different gestures.

4. Application

To better showcase the result of our hand gesture recognition, we integrated the recognition into a Snake game (developped in the Qt Creator), and translated each supported gesture into a game command. Our hope was to provide a better computer-user interactive experience by enabling user to play intuitively instead of relying on a keyboard.

As shown below, our complete application runs on the operating system side of the DE2 Intel board. With the addition of a Logitech C110 camera and a monitor, user can interact with the game without touching the mouse or keyboard.

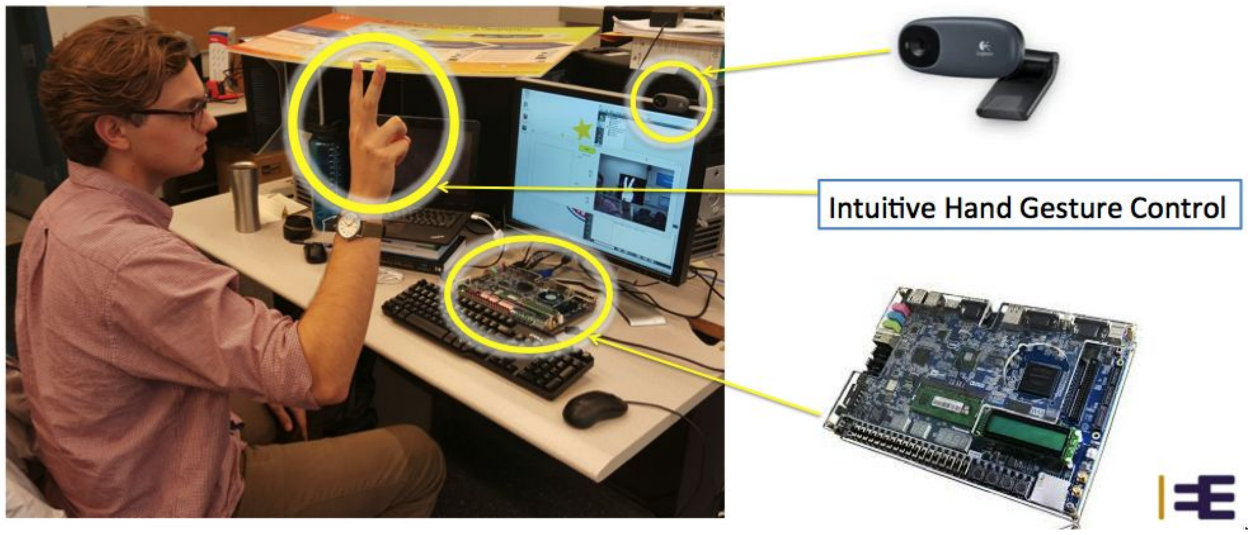

Figure 2. Hand gesture control game setup

4.1. The Graphical User Interface (GUI) - Snake Game

After we determined to build the game GUI in Qt Creator (development platform that supports c++ and OpenCV library and offers a large GUI library), we researched online for a suitable open source game GUI. We tested several snake games developed in Qt, and narrowed it down to this version - [7] which is a keyboard controlled snake game. We made more adjustments to improve the overall look of the GUI so it is tailored to a complete gesture-control game experience.

The Snake game GUI is divided into three sections (as shown in *Figure 3*). The left section is the snake game board, where the fruit is being generated and placed within the board, and snake can be moved within this region. The middle section provides all game related information, such as a “START” game button that can be selected to begin a new game, a “High Scores” button that allows user to check the highest score so far, a score display region that shows score earned in current round, and a command diagram to promote users which hand gestures are available to interact with the game. The right section of the GUI displays the real time video captured by the camera. This allows user to easily identify the hand placement in front of the camera. We have placed the skin detection box (filled in black for background, and plotted in white for skin region) slightly on the right side of the video display box (assuming right hand interaction). In order for the game to correctly respond to user's hand gesture command, the hand has to fit within the detection region. This skin detection box shows the posted processed skin detection in real time so user can easily calibrate their hand placement to ensure proper detection. The improvements we made includes but not limited to: increased game board size for readability and making the game more challenging, added color to the snake and the fruit, added gesture control command diagram, integrated real time video display, integrated our skin detection and gesture recognition to the back end of the game and linked all hand gesture commands in place of the keyboard commands.

To start the game, user should keep the right hand up to the camera in a fist shape and adjust its position until the fist fits just within the blue box (as shown in the figure below). Then click START button, a snake (head is in blue, the rest of its body is in green color) and its fruit (in red color dot) are immediately shown on the game board. The snake is in static position since the fist gesture is associated to the command "pause." User can now change to any available gestures to move the snake. Every time the snake walks through a fruit, it grows 1 unit length and a new fruit will be dropped randomly on the game board. The goal of the game is to "eat" as many fruits as possible. We added one penalty to the game, if user tries to move the snake against its current direction the snake will move forward instead. For an example, like the snake's orientation in the GUI snapshot, the snake intends to head left because its head is pointing to that direction. If user attempts to instruct the snake moving to the right by using the v-shape gesture, the snake will continue moving to the left instead.

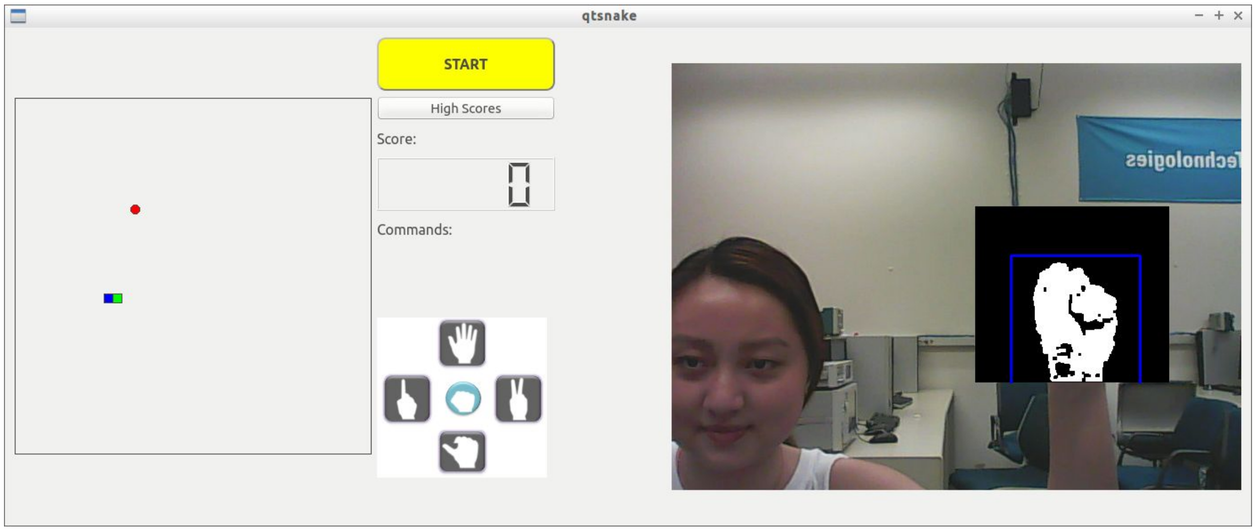

Figure 3. Graphical User Interface of the Snake game

4.2. Game Interaction - Gesture Control

The supported commands of the Snake game are shown in the customly created gesture command diagram below, with assumption that user will use right hand to play the game and all fingers are spread apart if more than 1 finger are held up for a gesture. We have experimented with different gesture-command pairs, and we decided on these gesture-command pairs for final demo because we found them to be the most intuitive to users. As illustrated in the diagram, each hand gesture is placed in the position that corresponds to its game control direction in respect to the snake game board. The middle fist gesture is provided for command - "pause" in case user would like to leave the game or take a break. The "up" and "down" directions are triggered by palm and thumb gestures respectively. With this setup, user would pull up all four fingers (leave thumb out) to instruct the snake moving upward, or pull down all four fingers to instruct the snake moving downward. The "left" and "right" directions are triggered by index finger gesture and v-shape gesture respectively. To make sure the snake continues moving in the desired direction, user can simply change to or hold the associated hand gesture to the camera.

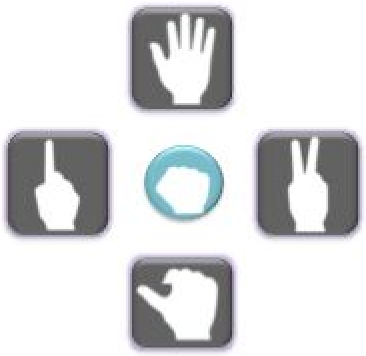
Figure 4. Snake game gesture command diagram

IV. Results and Discussion

1. Skin Segmentation

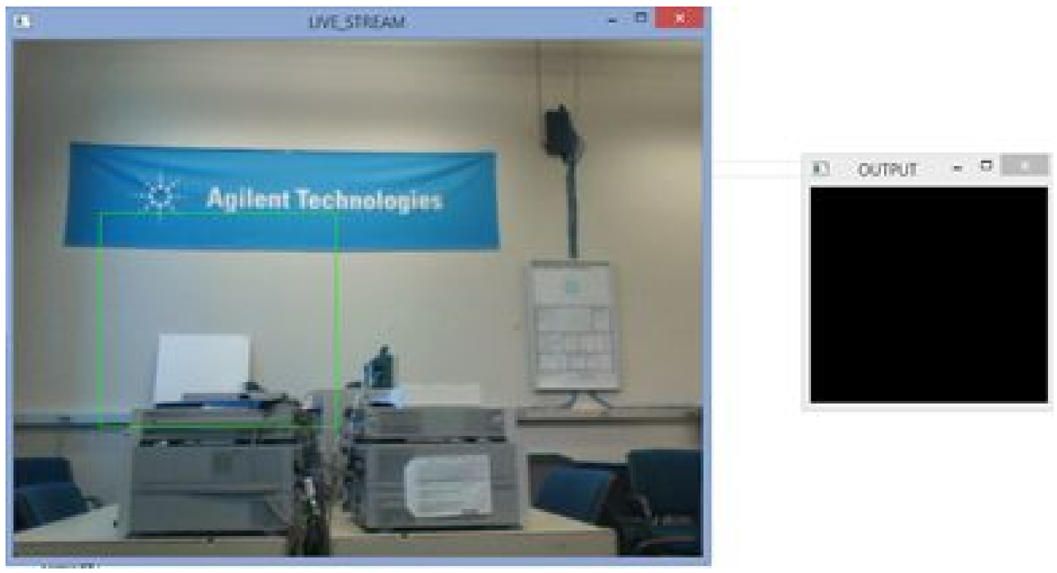

Figure 5. Input video frame (left) and output video frame (right) without a hand inside designated green window.

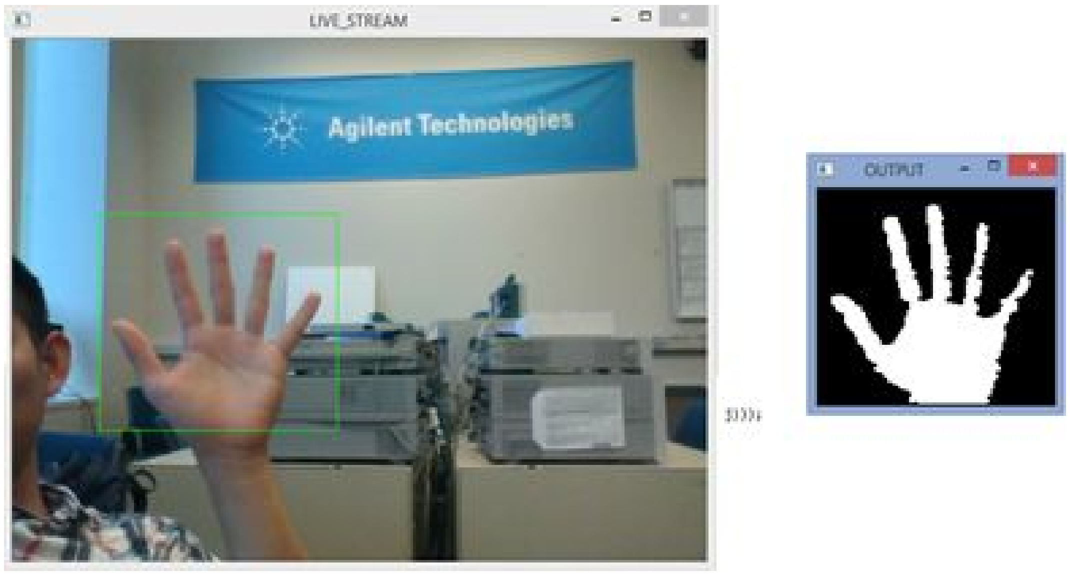

Figure 6. Input video frame (left) and output video frame (right) with a hand inside designated green window.

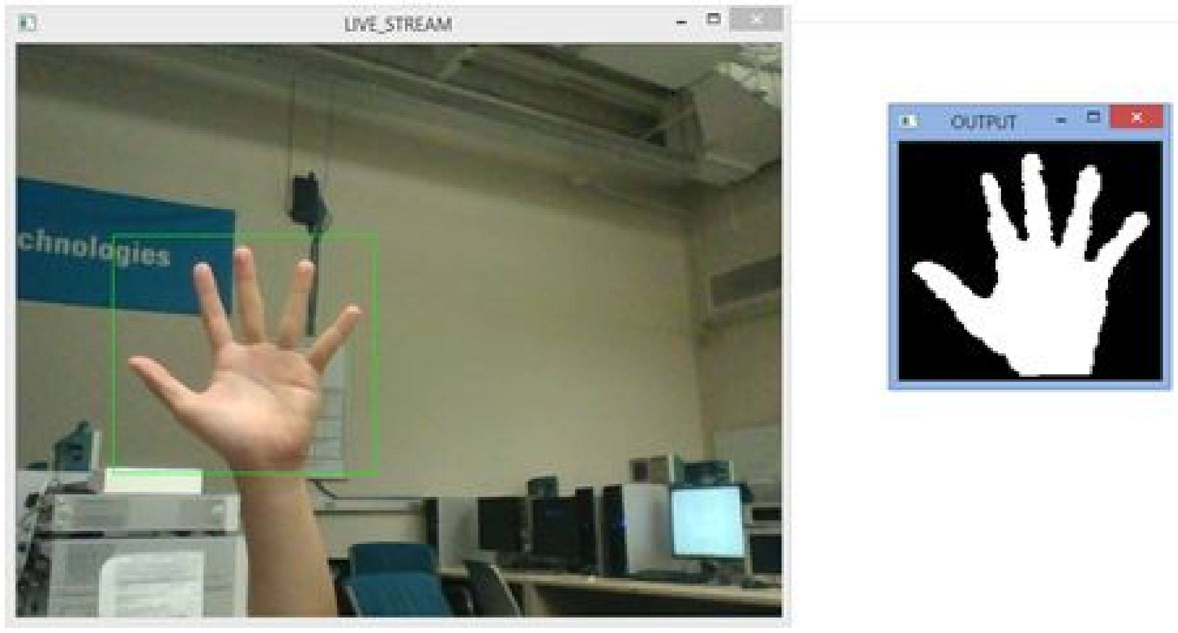

Figure 7. The input video frame (left) and segmented hand (right) with lower lighting and different background setting comparing to Figure 6.

The figures above shown the output of skin segmentation algorithm. As seen above, the skin segmentation algorithm works quite well for any given background and lightning condition. It can easily differentiates between skin and non-skin value. In addition, the computation time is quite fast. This algorithm is processed in real-time without any noticeable delay.

However, there are still some limitations. The most notable one is false positive which is mention in implementation section above (section *III.1.*, shown in *Figure 8*). In order to eliminate the false positive result, we use morphological operation - closing. The morphological operation properly removes these false positive. But, it also alters the shape of fingers as seen in *Figure 6* and *Figure 7*. The shape of one of the fingers in those image slightly different than the actual shape of the fingers. In addition, this algorithm also encounters some false negative values. As seen in *Figure 9*, small region of the hand didn't get detected by this algorithm. In *Figure 9*, the top half of the hand created a shadow that covered the region of the arm near the wrist. This shadow changed the illuminance value of these pixel values in this region which made them lie outside of skin tone color range. To reduce the false negative value, a better method was proposed by Vezhnevets et al. [1]. They proposed to use Bayes classifier. In Bayes classifier, the skin segmentation algorithm is used to differentiate skin and non-skin region for training data. Then, it uses that information to estimate the skin color of input image. This method has shown to produce less false negative results. Unfortunately, with time constraint, we couldn't build a large enough training dataset to use Bayes classifier.

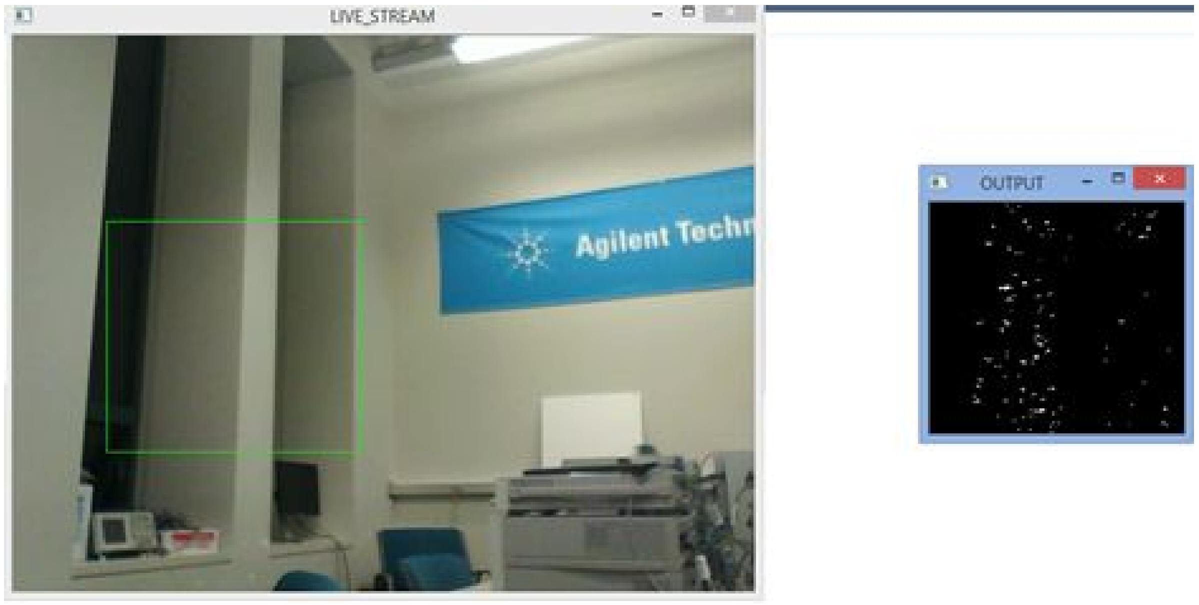

Figure 8. Input video frame (left) and output video frame without performing morphological operation - closing.

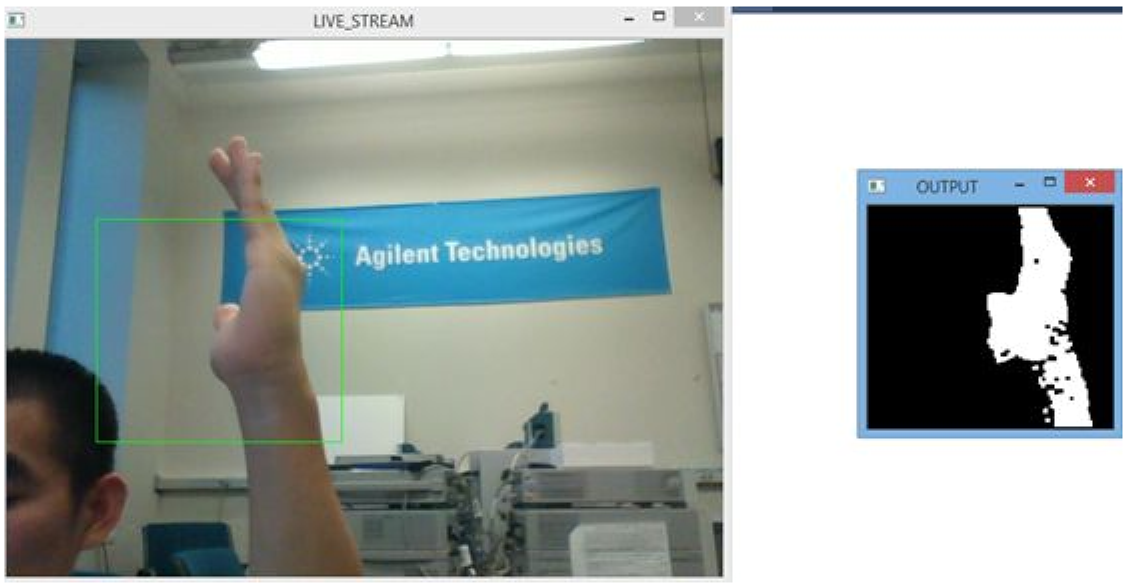

Figure 9. Input video frame (left) and output segmented frame (right) with some false negative.

2. & 3. Hand Detection & Hand Gesture Recognition

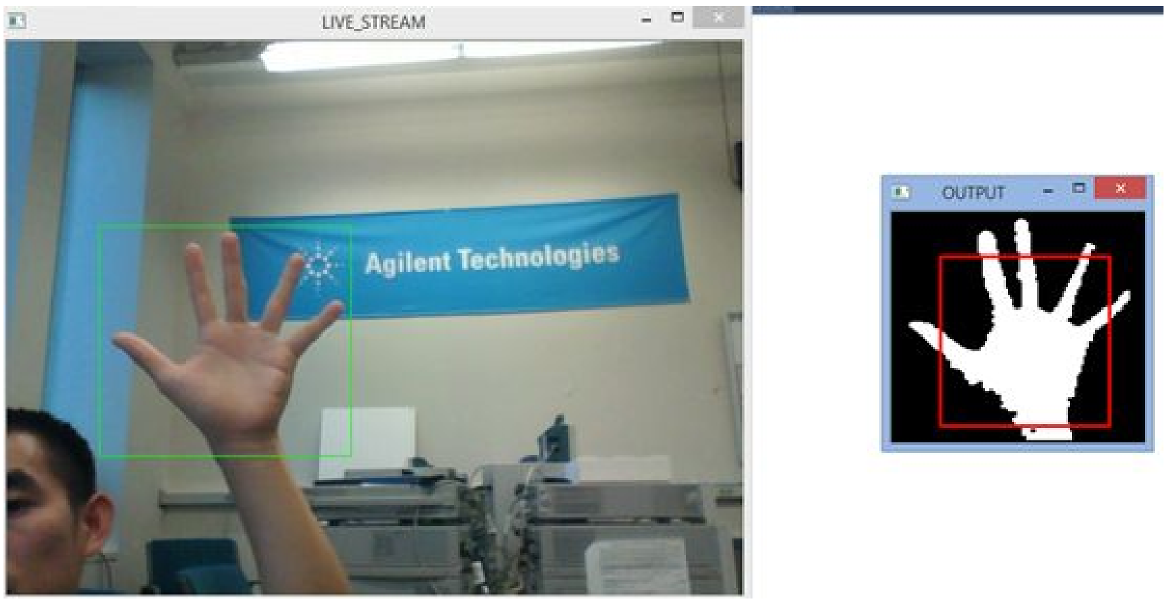

Figure 10. Input video frame (left) and segmented hand (right) with a rectangle that has its center corresponds to the center of segmented hand.

The centroid algorithm proves to be really efficient and robust method to locate the center of the palm. However, this method doesn't give a flexibility in detecting the hand at any position on the captured video frame. It requires the user to put his hand in a pre-defined region of the video frame range.

As shown in *Figure 10*, the rectangle is drawn such that its edges cut across all expanded fingers. The distance usually measures up correctly, however the size can be skewed if more of the user's forearm is included.

With regards to the finger counting, the output frame on the right in Figure 10 shows a clean segmented hand and correctly drawn rectangle. From this our algorithm would dedect fingers present based on the rectangle edges (we use left edge and top edge only in our gesture detection implementation).

4. Application

After integrating the hand gesture recognition with the Snake game, we had a challenge of handling the gesture command processing delay. In the game implementation, every unit step movement of the snake is instructed by user's input in real time. In the earlier version, every frame is interpreted as a command, we found it creates some error as the unintended gesture can be captured during the finger movements of gesture change. Additionally, it is very hard to change hand gesture at the right moment in order to move the snake to the desired spot which leads to a less friendly control experiences.

To overcome this challenge, we first analyzed the frame-to-frame elapse time, which starts from the back-end of the game reading in one frame, performing skin detection and gesture recognition until the interpreted command is concluded, and then the front-end of the game updates the GUI. This process is then repeated again and again between frames. By testing and running the game with timestamps, we found the average frame processing time is about 130 milliseconds. We then adjusted the game to accept a command if the number of repeated hand gesture frames has met a threshold. For an example, if the threshold is set to 1, then one consecutive hand gesture frame with the same command as the previous hand gesture frame meets the threshold, thus will be send as a single command to the front-end of the game to move the snake. According to Humanbenchmark.com[6], a site has collected over 18 million human clicks for reaction benchmarks, the average human reaction time is 266 milliseconds and the median lays at 256 milliseconds. Based on this statistical data and our testing, setting the threshold to either 1 (every 2 repeated frames -> 260 milliseconds per command) or 2 (every 3 repeated frames -> 390 milliseconds per command) gives a comfortable command elapse time for user to change their hand gesture. We finally settled with repeated frame threshold of 1 because this is much closer to the human reaction time which makes the game fun and challenging. Also, for continuous snake movement in the same direction, this setting allows the GUI to refresh at a reasonable speed (which is roughly the same as the command elapse time of 260 milliseconds) compared to the threshold of 2 which refreshes quite slow and makes the game appears laggy.

V. References

[1]V. Vezhnevets, V. Sazonov and A. Andreeva, 'A Survey on Pixel-Based Skin Color Detection Techniques'.

[2]S. Thakur, S. Paul, A. Mondal, S. Das and A. Abraham, 'Face Detection Using Skin Tone Segmentation', *IEEE*, 2011.

[3]A. Malima, E. Ozgur and M. Cetin, 'A Fast Algorithm for Vision-based Hand Gesture Recognition for Robot Control'.

[4] Wikipedia, 'YCbCr', 2015. [Online]. Available: http://en.wikipedia.org/wiki/YCbCr.

[5] 'RGB to HSV color conversion', 2015. [Online]. Available: http://www.rapidtables.com/convert/color/rgb-to-hsv.htm.

[6] Humanbenchmark.com, 'Human Benchmark - Reaction Time Statistics', 2015. [Online]. Available: http://www.humanbenchmark.com/tests/reactiontime/statistics.

[7] Snake, 'Qt Snake', *SourceForge*, 2013. [Online]. Available: http://sourceforge.net/projects/qtsnake/.