# Rock-Paper-Scissors Detection System (June 2018)

Shaurya Dhankar, Daniel Hosseinian

***Abstract*— This paper will focus on a Rock- Paper-Scissor detection game designed to work in real time. The main purpose of this project is to detect hand symbols of between two players simultaneously from a video feed and update the score accordingly. The system was implemented using an OMAP L138 Development Kit and the vpif_lcd_loopback framework provided in the C6748 LCDK Starterware library. The video feed of the hand signals was processed to facilitate the extraction of feature vectors. The features were then used to train a single-hidden-layer neural net, whose coefficient would later allow for fast real-time detection of the hand symbols. We were able to successfully identify and display the corresponding hand symbol and update the score in real time. The whole process was implemented without the use of filters and was efficient and expeditious.**

## I. Introduction

The standard rock-paper-scissors game was implemented with image processing techniques that allowed for skin tone detection and neural nets that identified the hand detection. With RGB color code model and Breadth First Search algorithm this process was executed with speedy performance.

### *A. History and Sources*

Rock-Paper-Scissors is a classic hand game played by two people, in which each participant must simultaneously form one of three shapes with an outstretched hand. The three shapes are “rock” (a closed fist), “paper” (a flat hand), and “scissors” (a victory sign). It is a zero-sum game with only two possible outcomes - either there is a draw, or one player defeats another. “Rock” defeats “scissors” , which defeats “paper”, which in turn defeats “rock”.

Implementation of this game with a video recognition system provides an interesting signal processing challenge. Even though many image processing techniques may be conducted on CPUs, digital signal processing (DSP) chips may be advantageous in performing computations with DSP dedicated taks.

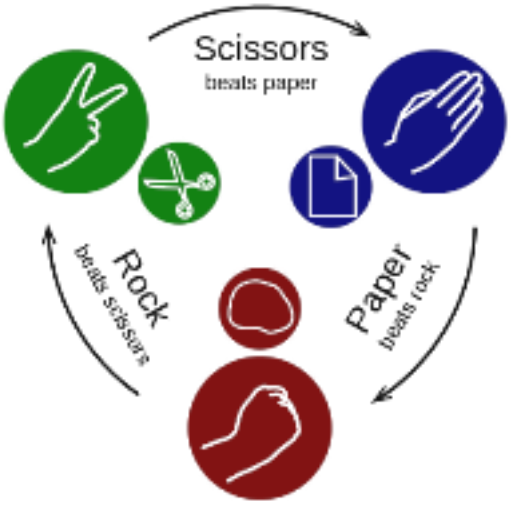


Figure 0: Rock Paper Scissor game dynamics
(Courtesy of Signatureforums.org)
http://www.signatureforums.org/amoeba-sisters-video-recap-biomolecules-worksheet-answers/amoeba-sisters-video-recap-biomolecules-worksheet-answers-luxury-massachusetts-dys-science-instructional-guide/

### *B. Global Constraints*

While DSP chips may prove to be advantageous in performing certain computations, they may still be limited in memory, requiring careful engineering of the product with lower space complexities.

## II. Motivation

This project was motivated by a desire to challenge ourselves with designing a real-time output system and to familiarize ourselves with various image processing techniques.

## III. Approach

### *A. Team Organization*

Daniel Hosseinian and Shaurya Dhankar familiarized themselves with operation the video camera and processing the hand symbols and observed the results in CCS via LCDK. Daniel Hosseinian researched and formed algorithms to process and display input data captured from hand gestures with hand symbol displayed in white. Daniel Hosseinian and Shaurya Dhankar researched and formed techniques to store YCbCr input image from video camera into RCB color code

model. Shaurya Dhankar and Daniel Hosseinian researched and implemented the breadth first search algorithm to allow skin tone detection and identify hand symbol displayed on the screen. Daniel Hosseinian formed the method to limit the original 320*480 screen into a smaller space bounding the hand using a Breadth First Search (BFS), and then researched the method to split that limited space into 16 smaller, distinct regions which are compiled and stored a feature vector used for training the model. Shaurya Dhankar formed the initial score tracker and Daniel Hosseinian compared and contrasted different hand gestures and built the score tracker in RGB color code model to be displayed on the screen. Shaurya Dhankar and Daniel Hosseinian integrated the overall workload and used NN tool of Matlab to train the saved data set and recompile it to improve efficiency and detection of the device.

### *B. Plan and Implementation*

Week 3: In the first three weeks, we planned to familiarize ourselves with the video camera and start processing hand symbols via the LCDK on the computer screen. Research and form algorithm to process data from captured image and display skin tone in white with a blue/red background. Process and store YCbCr input data into RCB color code model.

Week 7: From week 4 to week 7, our aim was to form a search algorithm to implement skin tone detection and handle salt and pepper noise effectively. Next, we planned to extract the detected hand symbol and break into smaller, explicit areas that would be used for feature extraction. Furthermore, we planned to form an algorithm that would be able to compare and contrast between different hand gestures without processing it via a gaussian/median filter.

Week 10: In the last three weeks, our goal was to extract our feature vector and form a data set for all three hand symbols (rock, paper, scissors) and train our device from NN tool of Matlab. The next goal, was to compare and contrast the results obtained from the trained data sets of different number of hidden neurons and find the optimum number of hidden neurons required to form our final set of values for each hand symbol. In the final step, our goal was to compare and contrast different hand gestures using our machine learning algorithm and form a score tracker and symbol display using RGB color code model and update it accordingly.

In the first four weeks, we struggled with issues pertaining to captured image not being displayed on the computer monitor. We tried to use Facedetect Software of the TI integrated software library. We realized that the project took too long to compile and process and we were facing constraints regarding inefficient detection of hand symbols and skin tone. This led us to switch to vpif_lcd_loopback framework provided in the C6748 LCDK Starterware library and we worked with this program going forward.

Even though our early troubles prevented us from meeting our ambitious milestones, our work accelerated upon switching to the new framework. After effectively starting on week four, the skin detection module was implemented and optimized by week seven, feature extractions was engineered by week 8, and the neural net was trained by week 9, thus allowing us to conduct refinements on week 10. Ultimately, the project was complete on time and all specifications had been met.

We initially struggled with real time skin detection because the original plan to use a median or Gaussian filter proved to be unreliable and too computationally heavy. Instead, we pivoted in preference of a simple searching algorithm that was faster. Yet, again we faced the challenges of the large memory space needed for the data structures of the searching algorithms. Ultimately, we traded off a small amount of performance for the ability to store a large queue on external memory.

The rest of the implementation proved to be successful and introduced no significant roadblocks.

### *C. Standard*

Various image processing and machine learning techniques were used throughout the project.

YCbCr and RGB color code models were used to set the heuristics for skin tone detection. The video feed provided image data in YCbCr color space, which was converted to RGB with a standard color space conversion.

The Breadth First Search (BFS) algorithm was used to identify the boundaries around the hand to allow for variable hand shapes and distances.

NNtool of Matlab was used to train a single-hidden-layer neural net with the captured data set, a program often used in IEEE papers[5].

### *D. Theory*

To allow for efficient real-time processing of the video input, the system modules were designed to allow for future parallelization optimizations by reducing the dependencies amongst iterations of the video input processing. Furthermore, compile time optimizations allowed for efficient addition, multiplication, and storage processes.

Furthermore, we opted to avoid conventional image processing techniques such as convolutional filtering in preference for a searching algorithm, thus further improving performance.

### *E. Software/Hardware*

The system requires integration between the LCDK with a standard video camcorder with an RCA connector allowing for a composite video connection.

The software for the LCDK was written using the C programming language within the CCS integrated development environment. MATLAB was used to train a neural net, whose coefficients were later exported and hard-coded inside the board.

*F. Operation*

The overall process is a six step procedure that captures the hand gesture, processes the captured image and compares the hand symbol between the two players and updates the score subsequently.

**Overall Implementation**

The screen monitor was split into two separate regions (red and blue) that indicated the playing area of player 1 and player 2 respectively. The score for each player was displayed at the bottom of the screen and the hand symbol was indicated at the top. A RGB green color dot was at the center of both screens indicated the point where the search for skin tone began. The overall data pipeline is indicated in Figure 1.

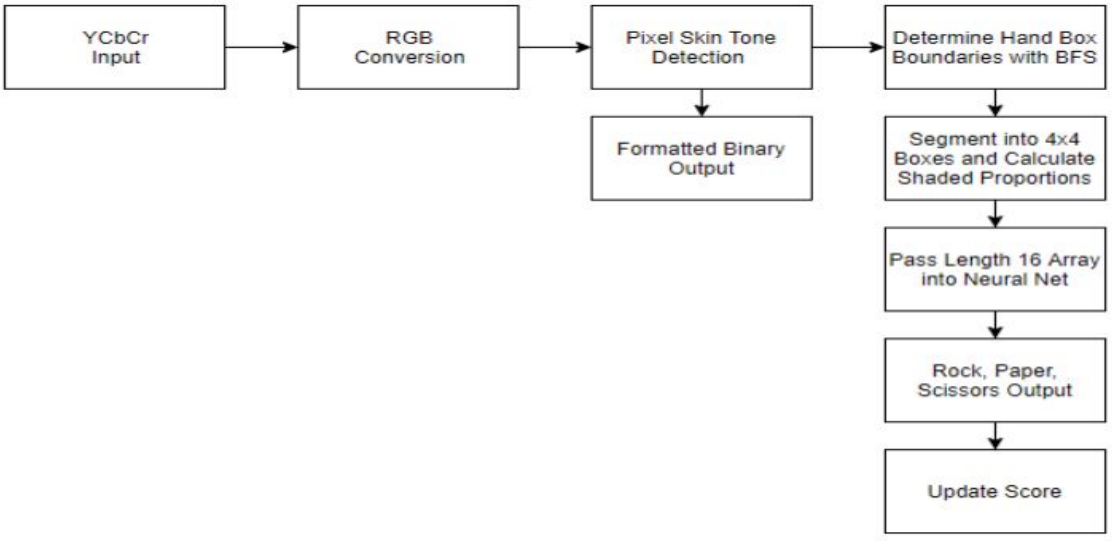


Figure 1: Project Data Pipeline

**Skin Tone Detection**

In the first step, we converted the YCbCr input into its corresponding RGB color code values. The RGB color code model comprises of three colors (red, green and blue) that combine to form a broad range of colors as shown in Figure 2. The matrix used to perform the conversion is displayed in figure 3.

A skin tone pixel was identified to be one in which the difference between red and green values (R-G) between the range 20 < R-G < 80 and the value of blue did not matter [1]. All other values outside the range were considered to be non-skin. The detected skin tone was displayed in white on the screen and the background was made blue/red based on the player. Other more accurate skin tone heuristics were considered [4], but they were found to be too complex for high-frame-rate real-time processing.

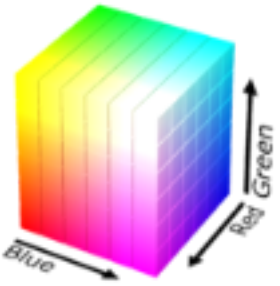


Figure 2: range of colors, RGB color code model
(Courtesy of NOAA and NASA)
http://rammb.cira.colostate.edu/training/visit/quick_guides/Simple_Water_Vapor_RGB.pdf

To display colors, we had to pack the standard 24-bit RGB888 (in which each color is assigned 8-bits) to a 16-bit RGB565 (in which green is assigned 6 btis while red and blue are assigned 5 each). For instance, if we desired to display green, we identified its RGB888 color code: (102,255,51). To convert this into RGB555 format and to display the desired green color we obtain the binary values for rgb = (102/255 * 31, 255/255 * 64, 51/255 * 31) = (13, 63, 6) which in binary is assigned the values (01101, 111111, 00110) and is a 16 bit number and when converted to hex gets the value 0x6FE6.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix} \quad \begin{matrix} R \in [0, 255] \\ G \in [0, 255] \\ B \in [0, 255] \end{matrix}$$

Figure 3: YCbCr input conversion matrix to form RGB values

(Courtesy of what-when-how.com)
http://what-when-how.com/introduction-to-video-and-image-processing/conversion-between-rgb-and-yuvycbcr-introduction-to-video-and-image-processing/

Furthermore, we implemented certain skin tone detection optimization methods. Skin tone can be computationally heavy and each cycle requires 640*480 (=307200) pixels to be processed. This huge load of image processing data was optimized using C intrinsic function such as the following:

```
double _mpyu4 (unsigned s1, unsigned s2)
_amem2(&destination) = variable.
```

Functions such as the aforementioned ones perform specialized operations in compile-time instead general non-optimized operations. For example, when multiplying two 4-byte unsigned integer types, a specified function would notify the compiler to use an optimized operation.

**Find Box Boundaries**

In the second step, we attempted to crop the input image to the size of the hand. We considered many options, such as detecting the first skin-tone pixel from each side of the input image, but this strategy is prone to salt and pepper noise. Therefore, we attempted to cancel the noise through filtering. First, we tried to process the image via median and Gaussian filters that resulted in slow and unreliable results with crashing at times and still retaining significantly large artifacts. Convolutional filters have debilitatingly large runtimes; convolving an MxN image with an mxn filter would yield a runtime of O(MNmn).

Consequently, the idea of noise removal was abandoned in preference of a searching algorithm. We tried to form our search algorithm using a recursive depth first search. However, this algorithm resulted in stack overflow due to the many instances of the recursive search function generated in runtime. The program crashed and was not able to handle searching through the large image.

Finally, we formed our search algorithm using a queue based breadth first search algorithm. The large size of the queue required it to be stored on external memory.

**Breadth First Search (BFS)**

A point was marked as visited with '#' to indicate it detected skin tone. BFS began from the centre green dot highlighted Figure 4. The starting green point was inserted onto the queue and marked as discovered. Next, if point on the east was skin tone and undiscovered we marked that point as discovered using '#' and inserted it into the queue [2].

In a similar fashion, we proceeded to for the points in the west, north and south directions as displayed in Figure 4. If skin tone was not detected then we went back to the previous point searched for skin tone in the other directions.

Figure 4: BFS performed on hand symbol

**Feature Vector Extraction**

In this step, the 320x480 array of captured image of the hand symbol was compressed into a length-16 array. The captured image was separated into 16 distinct regions as shown in Figure 5. Each region was assigned its own value and these value were saved in an array.

Our modified technique for feature vector creation was inspired by Htet and Cendejas-Cardenas [3]. As can be seen in Figure 5, the top left corner region is assigned a value of 0.00 since there was no skin tone detected in that region while the region assigned a value of 1.00 is completely covered in skin tone. Henceforth, with the increase in skin tone value the value of each region increase up to 1.00 for completely white. These values were used to form length-16 feature vector that we used to train our machine.

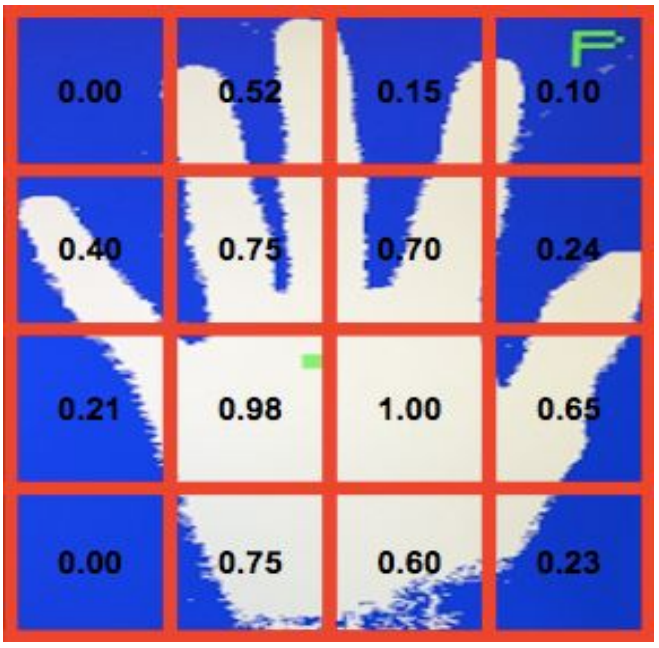


Figure 5: Areas associated with 16 distinct regions used to form feature vector

**Neural Net**

In the following step, we first imported the input data set for scissors, hand and rock values with 40 values for each symbol. Next, we assigned the output according to the input row values and then used the NN tool of Matlab to train a neural network.

The number of hidden neurons value was described as a hyperparameter, which is a variable whose value is set prior to training a machine learning model. Therefore, we had to train multiple models with different number of hidden neurons, and then compare their performances. We experimented between various number of hidden neuron numbers ranging from 3 to 10 and compared their results by observing the corresponding confusion matrices. Ultimately, we concluded that the best results were obtained at **number of hidden neurons = 7**. The plot of the confusion matrix for this value is indicated in Figure 6.

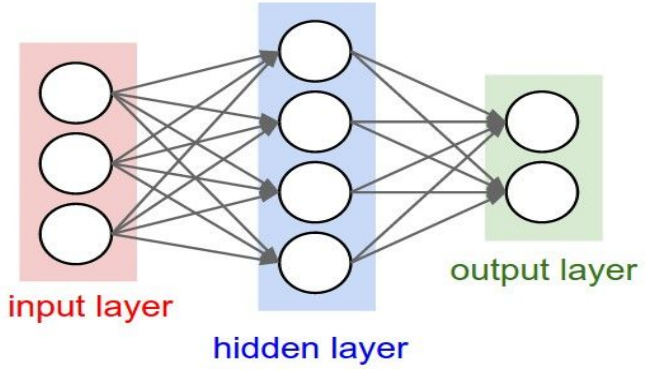


Figure 9: Three layers of a neural network


Figure 6: Plot of Confusion Matrix, number of hidden neurons = 7.

**Score Tracker and Hand Display**

In the final step, we built the score tracker and hand symbol display. The alphabets P, S and R were used to indicate Paper, Scissors and Rock respectively. The score tracker was displayed on the top right and left corner for both players as can be seen in Figure 7.

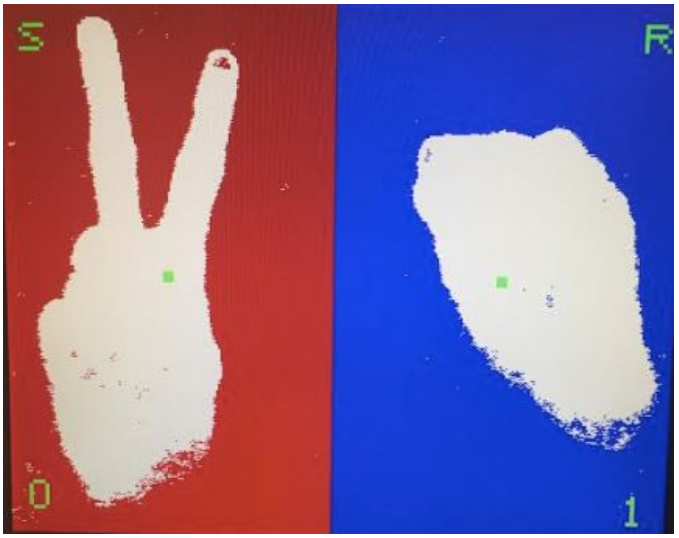


Figure 7: Snapshot of rock paper scissors game in action in real time.

The green display for score and hand symbol was achieved using the RGB green color code values. The letters and numbers were assigned this value were saved in the corresponding action's implementation and the following array in Figure 8 was used to invoke the correct value being displayed on the screen. As can be seen in Figure 8, the 1s in

the array outline a P symbol that would be displayed on successful detection of paper.

```
unsigned char P[] = {1,1,1,1,1,0,
                     1,0,0,0,0,1,
                     1,1,1,1,1,0,
                     1,0,0,0,0,0,
                     1,0,0,0,0,0,
                     1,0,0,0,0,0};
```

Figure 8: Array used to display alphabet P

**GUI Composer Switch**

The on and off switches of GUI Composer were used to record and save values for the neural network setup in Matlab. Values were recorded when switch was held in on position and device stopped recording when switch was turned off. This simplified the process of saving feature vector values and the 40 values for each hand symbol were stored in this fashion.

## IV. Results

### A. *How well it works*

The overall performance of the project was quick and efficient in real time detection after optimization. Neural Nets increased the performance of the design and the trained data set worked with 100% as shown in Figure 6 of the confusion matrix plot. Score tracker and hand symbol display worked efficaciously in real time, taking no more than one second to change the score and update the hand gesture. We achieved 90-95% accuracy in hand symbol detection with few instances of overlapping detection between paper and scissors. However, this performance may be further improved with larger training databases with more diverse hand shapes.

### B. *Discussion*

The system can be used in a variable background, meaning that skin-tone can be detected in front of anything. Furthermore, the feature extraction is scale-invariable, meaning that the size or distance of the hand need not be considered during use.

We came to the conclusion that median/gaussian filters not only slow down the project but are inefficient in their main purpose of cancelling out salt and pepper noise. We also realized that a recursion based depth first search algorithm lead to a stack overflow, eventually crashing the program. We perfected our search algorithm by choosing a queue based breadth first search algorithm running on external memory that simplified our design and made it swift and proficient.

Moreover, we realized that the overall performance of the design can be improved by generating more data set values for the feature vector and train it using neural nets. Henceforth, we decided to use NN tool of Matlab and by contrasting various confusion matrix plots, we finalized our hidden neuron value at 7 and used this to train our data set.

We would have like to test different machine learning models and compared their performances. Even though the neural net worked accurately and efficiently, nearest neighbor and support vector machines were other options that may have been explored.

The system is modularized into a video input routine and a image identification routine. Therefore the modularity would allow for alterations and improvements of different routines without needing to restructure the entire system. For instance, the video input and skin detection routine may be kept, while slightly altering the image identification routine, to convert the system into a sign language detector or any other system requiring skin detection. We are excited about the diverse applications of this project.

## V. References


[1] A. Saleh, *A Simple and Novel Method for Skin Detection and Face Locating and Tracking*, Berlin, Germany: Springer-Verlag, 2004, pp 1-8..

[2] C. Nachenberg., Lecture 5: *Stacks and Queues*. Lecture Notes in CS 32 Winter 2018,. Los Angeles, California. UCLA.

[3] P. Zin Htet, M. Cendejas-Cardenas, *American Sign language Recognition System,* EE 113DA Project, March 2017, Los Angeles, California. UCLA.

[4] Kolkur, S., Kalbande, D., Shimpi, P., Bapat, C., & Jatakia, J. (2017). Human Skin Detection Using RGB, HSV and YCbCr Color Models. *arXiv preprint arXiv:1708.02694*.

[5] Z. Raida, *Modeling EM structures in the neural network toolbox of MATLAB,* IEEE Xplore Digital Library, Dec 2002, pp 46-47.


Link to Youtube video:
https://www.youtube.com/watch?v=NvAWEgTGzkg