FINGER PLACEMENT CORRECTION FOR STATIC GESTURE RECOGNITION IN AMERICAN SIGN LANGUAGE

A thesis presented to the faculty of the Graduate School of
Western Carolina University in partial fulfillment of the
requirements for the degree of Master of Science in Technology.

By

Veronica Yenquenida Flamenco Cordova

Director: Robert D. Adams, PhD
Associate Professor
Department of Engineering and Technology

Committee Members:
Martin L. Tanaka, PhD
Department of Engineering and Technology
Paul M. Yanik, PhD
Department of Engineering and Technology

April 2014

This thesis is dedicated to my grandmother, Prudencia Cordova.

ACKNOWLEDGEMENTS

I begin by offering my sincerest gratitude to my main advisor, Dr. Robert D. Adams, who helped me throughout the completion of my thesis. I would also like to thank my other two committee members, Dr. Paul M. Yanik and Dr. Martin L. Tanaka, for their assistance and advice. I extend sincere thanks to Dr. James Zhang, for without his words and encouragement, I would not have considered and finished my undergraduate and graduate program. Lastly, I offer my warmest regards and appreciation to my parents, Blanca Nely Martinez, my mother, and Henry Martinez, my father. Their reassurance of their unconditional support throughout my undergraduate and graduate studies was a blessing. They offered me an opportunity that many do not have and I value and cherish them for everything they gave to help me achieve my greatest endeavor. God bless everyone who made this possible and believed in me, thank you again from the bottom of my heart.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

FINGER PLACEMENT CORRECTION FOR STATIC GESTURE RECOGNITION IN AMERICAN SIGN LANGUAGE

Veronica Yenquenida Flamenco Cordova, M.S.T.

Western Carolina University (April 2014)

Director: Robert D. Adams, PhD

Within the past few years, research involving gesture recognition has flourished and has led to new and improved programs assisting people who communicate with sign language [1–8]. Although numerous approaches have been developed for recognizing gestures [5, 6, 9], very little attention has been focused on American Sign Language (ASL) training for correcting the placement of individual fingers. Although, it is easy to mimic gestures, it is difficult to know whether or not you are signing them correctly. This is important in that most gestures, if made slightly incorrect, convey a completely different word, letter, or meaning [10]. This research involved developing a computer program to assist in teaching the correct placement of the fingers when performing ASL. Considering sign language has a wide range of gestures, the focus of the study is on static gestures which include a few letters of the alphabet. In order for the program to recognize finger placement, the user must wear colored latex over the fingertips. Then by using image processing techniques along with different algorithms, ASL hand gestures made by the user will be compared to standard images in a database. The program will provide feedback concerning how close the user is to the reference gesture as well as specific instructions concerning how to correct the gesture. This is the first step in developing a training/teaching program to help teach sign language accurately and precisely without the need of face-to-face instruction. Future studies could lead to more accurate training techniques for a wider range of ASL gestures.

CHAPTER 1: LITERATURE REVIEW

1.1 What is sign language?

Sign language is one of the most complex languages. It is so complex that it consists of approximately 6000 gestures of common words with finger spelling used to communicate obscure words or proper nouns [6]. There are also many other gestures that people do not consider sign language, but they allow for communication between people when a language barrier exists. Currently in the US, ASL is the third most used language [11]. ASL, although it is used by “English” speakers, is a language all on its own [3, 11]. It has all the characteristics that makes up a language, such as “having its own grammar, sentence structure, idiomatic usage, slang, style, and regional variations” [12]. ASL does not solely consist of static gestures and dynamic hand movements, but it also involves body language and facial expressions [3, 13].

According to [11], many organizations such as the American Sign Language and Interpreter Education (ASLIE), American Sign Language Teachers Association (ASLTA), and American Council on the Teaching of Foreign Languages (ACTFL) felt the need of a standard for ASL in 2007. These standards were completed and circulated in 2010 [11]. This standardization will aid with programs currently being used to detect certain gestures accurately because programs usually require parameters to be set when classifying and training the program to recognize certain gestures [6]. This would also help researchers and programmers to develop more accurate recognition software because it eliminates the time in building multiple libraries for each gesture being portrayed and what is trying to be communicated.

There are about 6,900 distinct languages as of today and of those, 200 are sign languages, according to discover.com. With such a wide variety of languages, it is important to have some type of bridge that can allow for more universality so everyone can communicate with one another. Technology is one of the tools that is allowing for this to happen. There are many computer programs being made (or improved) that are aiding in this when it comes to sign language [14]. Current methods that help teach sign language include videos /tutorials [15], books [16], and the more traditional approach, taking a class. Although these approaches have been useful, researchers are allowing for more interaction and feedback with new virtual programs. Ellis [1] presents research concerning a computer programming technique that would teach children sign language. Her experiments showed that the technology was identifying the signs the children were performing as well as correctly evaluating the signals in signs posed by different children in order to determine whether the sign was being performed well enough. Other programs being made range from helping deaf people communicate with hearing people, like a translator [2], to programs that help teach math to deaf children [8].

1.2 Different techniques used for gesture recognition

Researchers are developing new techniques for gesture recognition by using different devices such as sensor gloves (data-gloves) [3] or image capture techniques [6, 17]. With these devices there are many methods that can be implemented such as Hidden Markov Models [4, 14, 18] , Local Orientation Histograms [19, 20], Neural Network Models [3, 13], Bottom-up and Top-down Approach [4], Zernike moments [4], etc. [17]. What follows is a discussion of the various combinations of feature recognition techniques and classification methods that have been used.

1.2.1 Sensor gloves

Sensor gloves are one of the devices many researchers are using when dealing with real-time gesture recognition of dynamic gestures [1–3]. Ellis uses sensor gloves in her research in teaching children sign language using conditional template matching techniques [1]. This method is faster than using Hidden Markov Models (HMM), because it is easier to search for errors and to explain the outcome of the classification process [1, 21]. This method demonstrates a detection accuracy of 95 percent, when two children performed the test, on 175 different gestures [1]. Her research concluded that this was a useful method but considering the children's hands were small, the program could not recognize them all accurately. She also states that a learning system providing feedback to the user could help in future research for a wider range of learners. In Kadam's research, the main goal was accuracy of gestures made [2]. He considered sensor gloves the best way to implement a teaching program. He was able to recognize fourteen gestures 86 percent of the time; but this was not sufficient. There are many features to consider when performing sign language, and he did not take into account the probability of multiple gestures having similar characteristics to others which made the program confuse one for the other. Kadam also stated that they would need to reconsider using more sensors or a new approach, such as image capturing, for better accuracy.

1.2.2 Image capture

Another approach for gesture recognition is using image capture [4,6,7,14,17–20,22]. This approach can be used for both static and dynamic gestures. In Yang's research, he uses this approach by capturing frame by frame images of a user performing the word *cheerleader* [18]. His experiment as well as Starner's approach [6] show that image capturing can be used on videos, because they take still images when extracting and recognizing each gesture [6, 18]. Starner and Freeman prefer image recognition, because it avoids the use of expensive "data gloves" considering most signing does not involve finger spelling but instead, gestures which represent whole words [6, 19]. This will allow conversations when signed to proceed along at a pace similar to the normal spoken conversation.

1.3 Classification of images

Image classification is the "process of assigning a feature vector or a set of features to some predefined classes in order to recognize the hand gesture" [4]. When it comes to classification of the images, Starner [6], Tanibata [7], and Min [14] all use Hidden Markov Models (HMM). Hidden Markov Models are used for visual recognition of complex, structured hand gestures [6]. An HMM is a collection of finite states connected by transitions [23]. Each state is characterized by two sets of probabilities: a transition probability, and either a discrete output probability distribution or a continuous output probability density function [23]. Although HMMs are ideal for most projects, they encounter three problems: the evaluation, estimation, and the decoding [6]. Starner, conveniently in her research also provides us with ways to fix these types of problems [6]. Many researchers have used HMM's for gesture recognition [2–4, 6, 14]. A few have achieved recognition rates over 85 percent [6, 14].

Another way to classify images is by using Local Orientation Histograms [4, 19, 20]. The way orientation histograms work is that they provide more contrast within the colors of the image providing better detection of an object [19, 20]. Local Orientation Histograms are robust when dealing with lighting changes. This allows for higher recognition accuracy of the program when recognizing gestures, as shown in Zhou's [20] research, for example. Although Freeman [19] and Zhou [20] use local orientation histograms in their research, Messer [4] and Freeman found that many gestures which look different to the human eye might have an almost identical orientation histogram, and vice-versa. It is crucial for the program to recognize the gesture being made and know what image to correctly compare it with in order to train the user properly [2, 19]. If the program recognizes the image, but compares it to a different gesture because it recognizes it as the correct gesture made for that image, it will be a very flawed system.

1.4 Other devices used for gesture recognition

There are other devices that were found to be useful for sign language recognition. Kinect for Xbox 360 [5] has been used for real-time recognition, and is capable of tracking users' fingers. The lack of resolution of the Kinect cameras makes quickly moving hand features hard to distinguish. Another device is the Wii Remote. It has motion sensing capability which allows for gesture recognition [13]. It is a good approach for dynamic gesture recognition, because it follows the pattern of hand movement. The Wii remote is not ideal when performing the correction process of sign language gestures because although it recognizes the hands position as a whole, it does not detect position or movement of individual fingers.

Various other technologies for displaying and recognizing hand gestures are either is the research phase or have recently become available in the marketplace [24–26]. Leap Motion is a sensor as small as a USB flash drive that detects finger movements and displays them on a monitor screen [24]. This device is fairly new and non expensive, but many complaints have come up concerning the device's ability to detect all individual fingers. *Digits* by Microsoft [25] is a glove-less device which works with infrared sensors. It displays the hand on a monitor screen and can be used for gaming, sign language, mobile device interaction, etc. [25]. Currently Digits is a Microsoft research project and is not available on the market. MYO [26] is a wrist/arm band that detects the motions of our muscles and allows for computer interaction [26]. This is a unique idea and although it is for sale, shipment will not begin until mid-2014.

1.5 Advantages and disadvantages of different methods for gesture recognition

From the literature review, we can extract useful techniques for gesture correction. Data gloves seemed to be the better choice because they allow for real-time detection of gestures being made, but they have their disadvantages as well [2]. It is possible to make a glove-less sensor such as Microsoft did [25] or a data glove in [2] , but it would be another project all on its own, and thus, infeasable given the time limit for the research. Image capture recognition is sensitive to the environment, which may cause a challenge to the recognition process, such as bad illumination, irregular backgrounds, etc. [4]. There are methods, like

Local Orientation Histograms [20] and Hidden Markov Models [6] that can help improve the results. Edge detection methods [27] could be used for the purpose of comparing the user's gesture with the reference gesture. Another variable to be considered is the difficulty of recognizing open and closed finger gestures. Geetha [17] finds a method that is effective in recognizing 50 percent of the letters in the alphabet 100 percent accurately. All methods have their advantages for the type of application trying to be implemented. Yang [28] shows an explanation and a summary of this with different approaches that have been used in gesture recognition. From Yang's research we can get a better understanding of different methods that have been used and what approaches result in better outcomes for specific applications.

1.6 Key terms

ASL: American Sign Language.

Edge detection: An edge is where there is a significant change in the intensity of an image which occurs on the boundary between two different regions (edges) in an image [27]. Edge detection shows where pixels should be discarded as "noise" and which pixels should be retrained.

Image capture: Encyclopedia.com describes this as the process of getting a digital image from a vision sensor, such as a camera. Usually this entails a hardware interface known as a frame grabber, which captures single frames of video, converts the analogue values to digital, and feeds the result into the computer memory. The conversion process is often accompanied with image compression.

Static gesture: In reference to American Sign Language (ASL), static gesture is a particular configuration and pose, represented by a single image [19]. Letters as well as individual words in ASL can be static.

CHAPTER 2: METHODOLOGY AND RESULTS ANALYSIS

2.1 Summary and outline

In this research we used static images of ASL gestures in developing a program that would recognize the user's colored fingertips automatically in order to correct fingers position. We selected twelve letters from a database [29] to use for comparison with user gestures. We then took pictures of the user performing the letters in a lighted and shaded environment. Then we enhanced and analyzed the intensities of the images to derive unique ranges of red, green, and blue (RGB) colors for each finger. This was useful because it provided a color range that would be more accurately recognized in a normal environment when taking snapshots of images. We then computed the midpoint for each finger based on the pixels found using the set ranges. From these midpoints, we computed distances and angles between fingers. We then compared the user's correctly performed gestures to the database. We now had information of the user's gestures that were performed correctly to begin testing intentionally incorrectly made gestures to the "correct" gesture information. From the angles that were obtained, we could confirm that the program was indeed detecting the fingers that were misplaced correctly. Then we found centroids for each gesture and the corresponding reference image. Then we determined the conditions the program would use in correcting the *x* and *y* placements for each fingertip. Once this was established, we began to test the correction process of different incorrectly made gestures to make sure the program was correcting the fingers effectively. Finally a performance metric was established to show that once the correction process was complete, the user was indeed within a certain percentage to be considered correct.

What follows is a detailed description of the research process used to develop a computer program to assist in the learning of ASL.

2.2 Database construction

We needed a database of standard ASL gestures in order for the users to have a visual representation of the letters that they would be signing. It also provided a reference image in order to compare certain information to the user's image. Images were retrieved online from the database at lifeprint.com [29] and approval was obtained for using this database [30]. Some of the information that was found from the database images were the position of the midpoint on each fingertip, direction angles, distances between two fingers, angles formed between three fingers, as well as centroids, etc. (Finger 1 = pinky,...,Finger 5 = thumb). This information was used to compare the database and user gestures. Testing was performed on twelve letters (a, b, e, h, i, L, n, r, t, u, w, and y) selected from the database. These specific letters were chosen because all fingertips were visible in the images. Hand gestures that show all fingertips are called open hand gestures. The ASL hand gesture images extracted from the database [29] corresponding to each of the twelve letters chosen are shown in Figure 2.1.

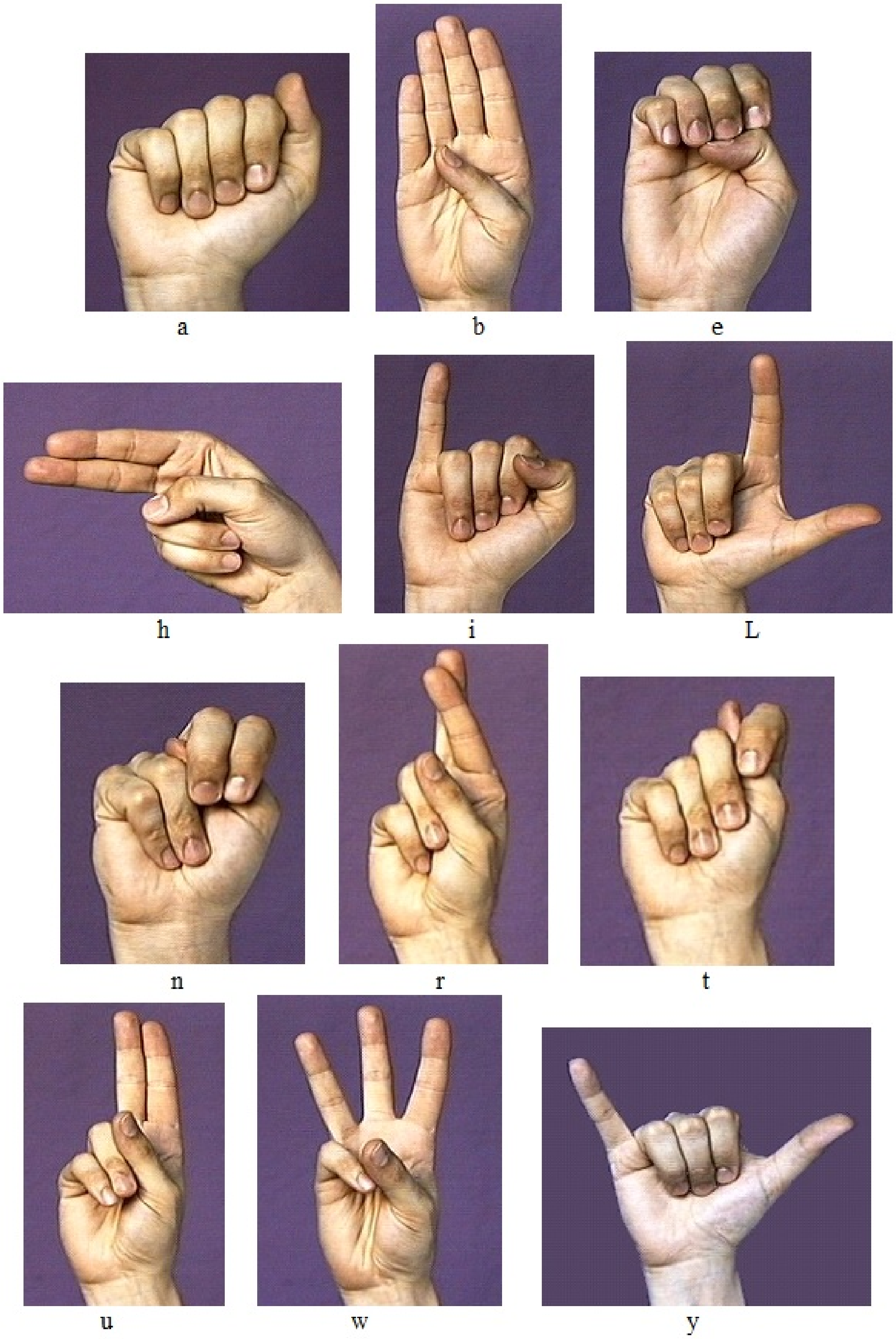


Figure 2.1: Database of twelve letters used

2.3 Lighting setup for image capture

One of the objectives was for the program to recognize fingertip colors automatically when taking snapshot images. In order for the program to recognize the colors in a "natural" environment, we asked the user to perform all letters in two different lighting conditions. We started by having the user (Veronica Flamenco) wear five different colored latex balloons on the tips of each finger using the right hand. Then, images of the user performing the twelve letters were taken in both an unshaded and shaded environment. Unshaded gestures were taken in a well-lit room with natural and artificial lighting. Shaded images of these letters were taken in a room with moderate lighting using a poster to block most of the light hitting the hand. These digital images were used to gather red, green and, blue (RGB) color ranges for both conditions which would then be used to find a set range of RGB colors for automatic detection of all five fingertips under different lighting conditions. All twelve letters were performed as closely as possible to the reference images, shown in Figure 2.1.

2.4 Procedure for extracting RGB color ranges

We needed to investigate the RGB values of the unshaded and shaded images in order to identify all five colors uniquely. The unshaded and shaded images all produced a variety of intensities of RGB values. A digital image is composed of 24-bits, or three 8-bit values. In a digital image, every pixel is identified with a red, green, and blue intensity which range from 0-255 because of the bits that compose the image. If the RGB value is closer to 0, the color will be darker but when closer to 255 the intensity of the RGB values will be brighter. We developed a program to recognize all these different intensities in these two different environments. The purpose of this was to see what these intensities were. The program allowed for us to click on each of the colored fingertips, three times per finger, of the unshaded and shaded images of the user. These three clicks provided minimum and maximum RGB values. Pixels within a selected user image that fell within the min and max values were recolored black. This was a visual representation to show if each fingertip color was being recognized. The program also found the midpoint of each fingertip based on the average of all rows and columns of the black pixels found.

After several experiments with different user gestures under various lighting conditions, we found that this procedure did not uniquely identify each fingertip, because of overlap between the RGB ranges. Although the program was recognizing each fingertip color correctly, it was also detecting the same colored pixels in other regions of the image. In Figure 2.2, we see the process the program went through in identifying each of the fingers. The program first identifies the pink on the pinky by recoloring the pink pixels with black pixels (a). Then when finding the yellow on the ring finger (b), other areas are also being recognized to have the same range of color. As the process continues with identifying the rest of the colors on the other fingers (c-e), we can see more misidentification of individual colors on other parts of the image. Therefore we began to investigate image enhancement techniques to help with unique recognition of each fingertip color.

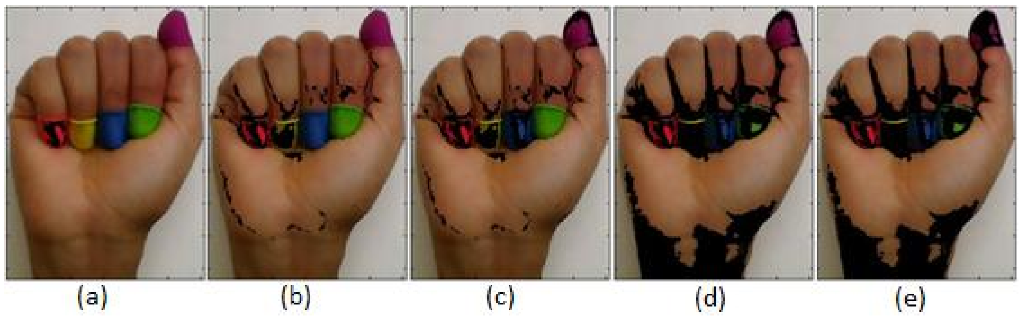


Figure 2.2: Identifying specific colors on each fingertip one by one; (a) Identifying pink on pinky finger, (b) Identifying yellow on ring finger, (c) Identifying blue on middle finger, (d) Identifying green on index finger, (e) Identifying purple on thumb

2.5 Image enhancement

Image enhancement was added to the original user image to help narrow the range for the RGB values for the purpose of identifying fingertips. Three different enhancement methods were tested on the original image: decorrelation, image color scale adjustment, and decorrelation with linear contrast stretching. Decorrelation methods assist in uniquely identifying colors in a digital image by stretching the color bands to “enhance the color separation of an image with significant band to band correlation” [31]. More information concerning decorrelation is found in Appendix A. The image color scale adjustment method provides more contrast enhancement.

Figure 2.3 (a) shows the original unshaded image of the user performing the letter ’a’ and (b-d) show the image enhanced using the three enhancement methods. It can be seen in the enhanced images that, of the two decorrelation methods, (b) and (d), seemed to provide more defined fingertip colors. After further examination of the data, the RGB values provided by the decorrelation with linear contrast stretching method (d) gave smaller ranges for each individual color. This meant that when viewing the enhanced images, this method provided better color separation which made the features on the image easier to distinguish. This also showed that the RGB values of all five different fingertip colors were not overlapping as much for this method, which meant each color was less likely to be mistaken with one of the other colors. Based on this, the decorrelation with linear contrast stretching method was chosen to assist in identifying fingertip locations.

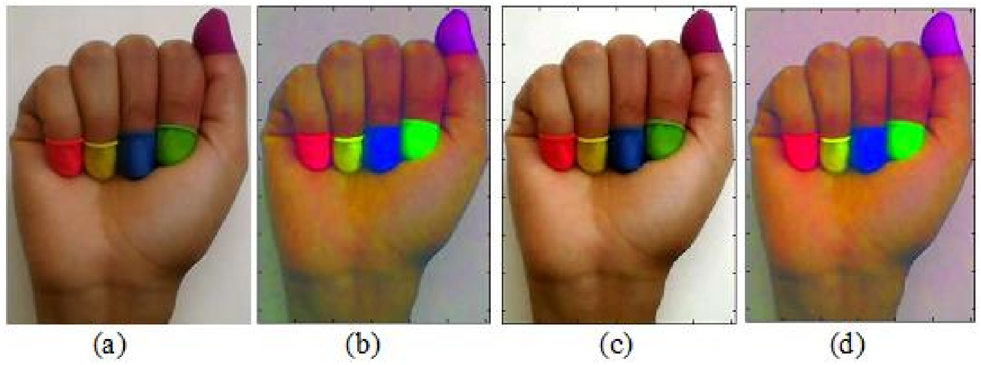


Figure 2.3: Original image of ’a’ compared to the three methods used: (a) Original image (b) Decorrelation (c) Image color scale adjustment (d) Decorrelation with linear contrast

2.6 Determination of overlap between RGB values

After finding the RGB ranges for both unshaded and shaded images, we examined the overlap between both images in order to recognize snapshot images. This was going to help produce a set range of RGB values in order for the program to recognize the colors in a "natural" environment. After analyzing the RGB values, we noticed six different cases of overlap between both images, which depended on the enhancement method chosen. From these cases, we got a set range of RGB values for every color on each fingertip. Comparisons of the six cases found can be seen in Figures 2.4-2.9.

2.6.1 Explanation of all cases

For each of the following cases:

$range(1)$ is the minimum intensity for a given finger in the unshaded image

$range(2)$ is the maximum intensity for a given finger in the unshaded image

$range(3)$is the minimum intensity for a given finger in shaded image

$range(4)$is the maximum intensity for a given finger in shaded image

Figure 2.4 shows Case 1 in which:

$range(4) \leq range(1)$

We will produce two min and max set values, which create two non-overlapping ranges, as seen in Figure 2.4. The first min will be $\frac{1}{2}(range(3) + range(4))$and the corresponding max will be $range(4)$. The second min will be $range(1)$ and the corresponding max will be $\frac{1}{2}(range(1) + range(2))$.

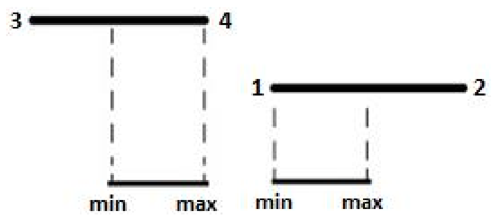


Figure 2.4: Case 1

Figure 2.5 shows Case 2 in which:

$range(2) \leq range(3)$

We will produce two min and max set values, which create two non-overlapping ranges, as seen in Figure 2.5. The first min will be $\frac{1}{2}(range(1) + range(2))$and the corresponding max will be $range(2)$. The second min will be $range(3)$ and the corresponding max will be $\frac{1}{2}(range(3) + range(4))$.

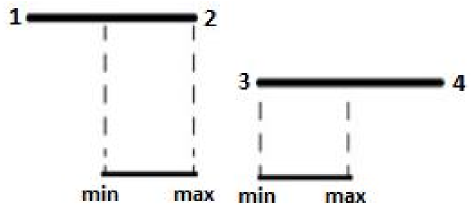


Figure 2.5: Case 2

Figure 2.6 shows Case 3 in which:

$range(3) \geq range(1)$ and $range(4) \leq range(2)$ are both true

Then if $range(4) - range(3) \leq \frac{1}{4}(range(2) - range(1))$

The min and max will be found using(1) shown in Figure 2.6. The min will be $\frac{1}{2}(range(1) + range(3))$.The max will be $\frac{1}{2}(range(4) + range(2))$.

But if $range(4) - range(3) \geq \frac{1}{4}(range(2) - range(1))$

Then the min and max will be found using(2) shown in Figure 2.6.

The min will be $range(3)$and the max will be $range(4)$.

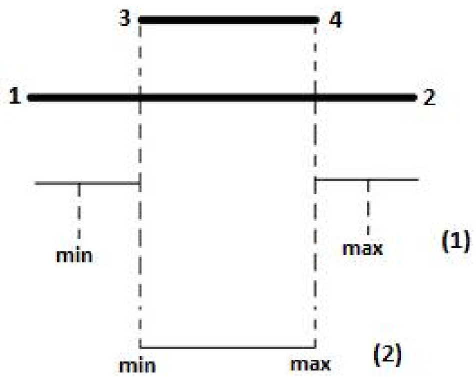


Figure 2.6: Case 3

Figure 2.7 shows Case 4 in which:

$range(1) \geq range(3)$ and $range(2) \leq range(4)$ are both true

Then if $range(2) - range(1) \leq \frac{1}{4}(range(4) - range(3))$

The min and max will be found using(1) shown in Figure 2.7. The min will be $\frac{1}{2}(range(3) + range(1))$.The max will be $\frac{1}{2}(range(2) + range(4))$.

But if $range(2) - range(1) \geq \frac{1}{4}(range(4) - range(3))$

Then the min and max will be found using(2) shown in Figure 2.7.

The min will be $range(1)$and the max will be $range(2)$.

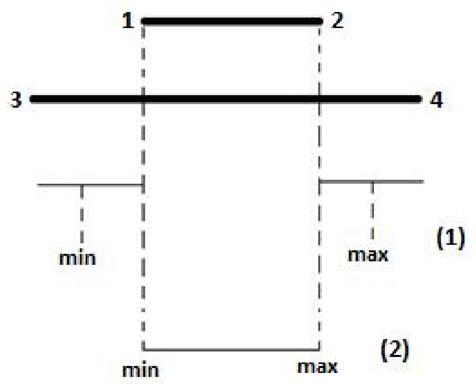


Figure 2.7: Case 4

Figure 2.8 shows Case 5 in which:

$range(2) - range(3) \leq \frac{1}{2}(range(4) - range(3))$

In this case, the min is set to $\frac{2}{3}(range(1) - range(3)) + (range(3)$ and the max is set to $\frac{2}{3}(range(4) - range(2)) + range(2)$.

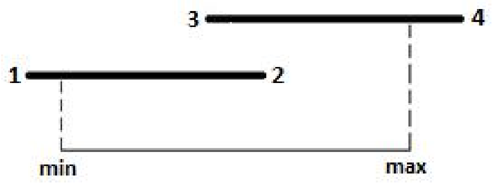


Figure 2.8: Case 5

Figure 2.9 shows Case 6 in which:

$range(2) - range(3) > \frac{1}{2}(range(4) - range(3))$

In this case, the min will be $range(3)$and the max will be $range(2)$.

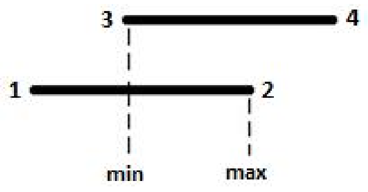


Figure 2.9: Case 6

An example of how these cases were used on both unshaded and shaded images using the decorrelation with linear contrast stretching method to find the set ranges is as follows. Table 2.1 shows the enhanced unshaded and Table 2.2 shows the enhanced shaded RGB values found for the letter ’a’ performed by the user.

Table 2.1: RGB values of enhanced unshaded letter ’a’ performed by user

Finger	R min	R max	G min	G max	B min	B max
Pinky	255	255	0	0	70	84
Ring	137	176	145	222	0	0
Middle	0	0	15	32	248	255
Index	0	29	255	255	0	0
Thumb	146	198	0	0	190	247
	range(1)	range(2)	range(1)	range(2)	range(1)	range(2)

Table 2.2: RGB values of enhanced shaded letter ’a’ performed by user

Finger	R min	R max	G min	G max	B min	B max
Pinky	255	255	0	0	94	102
Ring	114	121	242	246	0	0
Middle	0	0	0	12	255	255
Index	0	0	255	255	0	6
Thumb	64	132	0	0	171	242
	range(3)	range(4)	range(3)	range(4)	range(3)	range(4)

From Table 2.1 and 2.2 we can see all the different ranges of intensities, as well as the narrowed ranges between the min and max values of Red, Green, and Blue. Also notice that the Thumb, which is purple, has higher intensities of red and blue but absolutely no green, which would be expected when trying to obtain the color purple.

Table 2.3-2.5 show the set ranges found for the Red, Green, and Blue using the values from Table 2.1 and 2.2. The case that was used in determining the set ranges for each color on each finger is also provided. Notice that if the case was either Case 1 or Case 2, two min and max value sets would be provided, otherwise, only one set of min and max values would be found.

Table 2.3: Red min and max set ranges for letter ’a’ performed by user

Finger	min	max	min	max	Case
Pinky	252	255	255	255	1
Ring	113	143			5
Middle	0	0	0	0	1
Index	0	0	0	0	1
Thumb	153	196			5

Table 2.4: Green min and max set ranges for letter ’a’ performed by user

Finger	min	max	min	max	Case
Pinky	0	0	0	0	1
Ring	230	255			3
Middle	0	8			4
Index	255	255	255	255	1
Thumb	0	0	0	0	1

Table 2.5: Blue min and max set ranges for letter ’a’ performed by user

Finger	min	max	min	max	Case
Pinky	124	157			5
Ring	0	0	0	0	1
Middle	255	255	255	255	1
Index	0	0	6	14	2
Thumb	236	255	255	255	5

2.7 Automatic detection of fingertips

Using the set ranges of RGB values found in the previous section, we were ready to design a program that would automatically detect all the five colored fingertips. The cases provided criteria for locating the fingertip colors. The program recolored the fingertips black by using the set ranges to search for pixels meeting the criteria of the given case. This provided a visual representation to show if each fingertip color was being recognized. It also found the midpoint based on the average of the row and column values of the recolored pixels. Figures 2.10-2.21 show the colored fingertips and the midpoints of all twelve letters, found by the program.

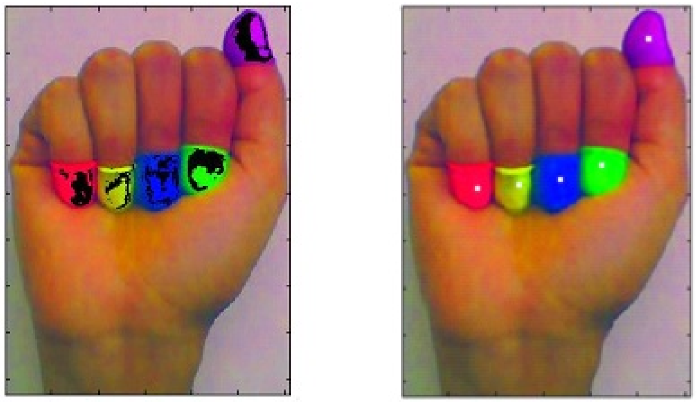


Figure 2.10: Detection of fingertips for Letter ’a’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

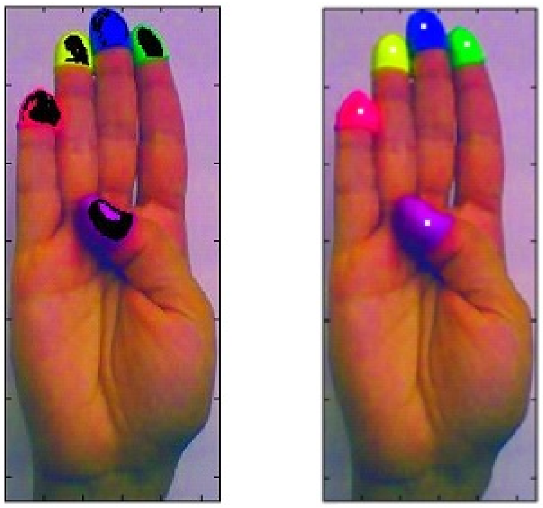


Figure 2.11: Detection of fingertips for Letter ’b’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

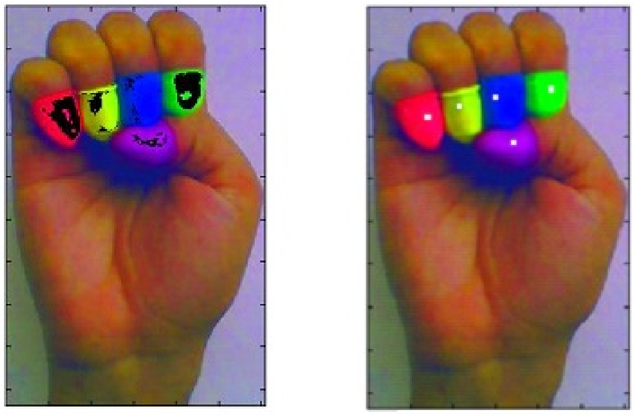


Figure 2.12: Detection of fingertips for Letter 'e'. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

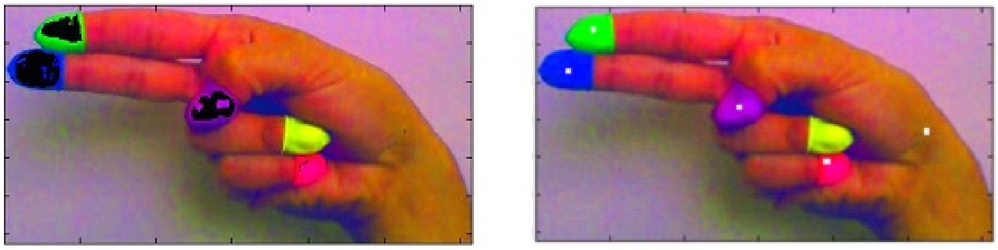


Figure 2.13: Detection of fingertips for Letter 'h'. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

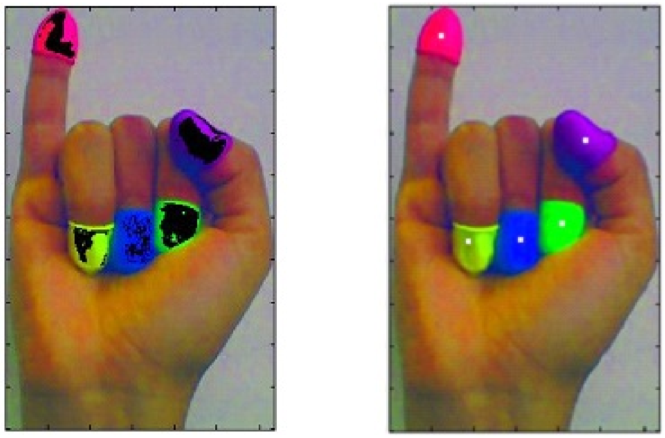


Figure 2.14: Detection of fingertips for Letter 'i'. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

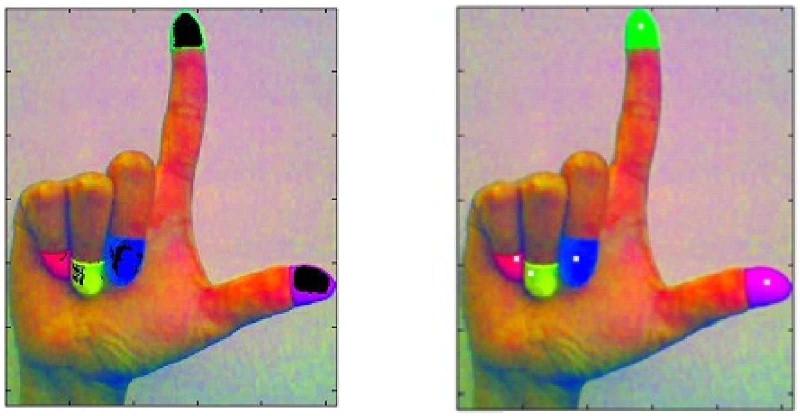


Figure 2.15: Detection of fingertips for Letter ’L’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

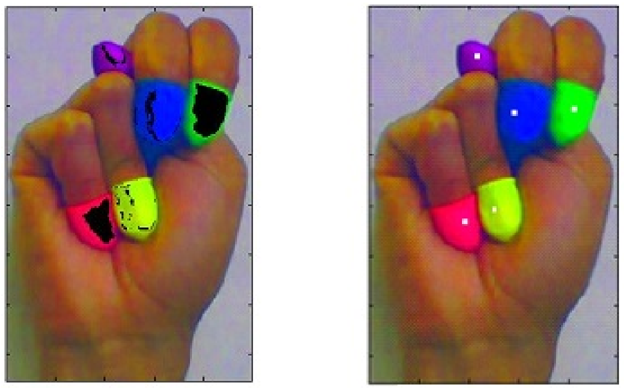


Figure 2.16: Detection of fingertips for Letter ’n’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

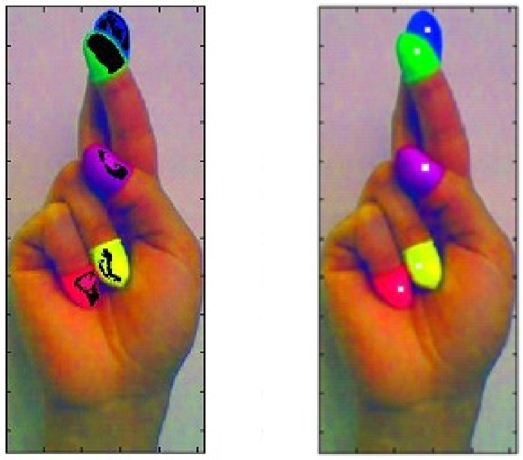


Figure 2.17: Detection of fingertips for Letter ’r’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

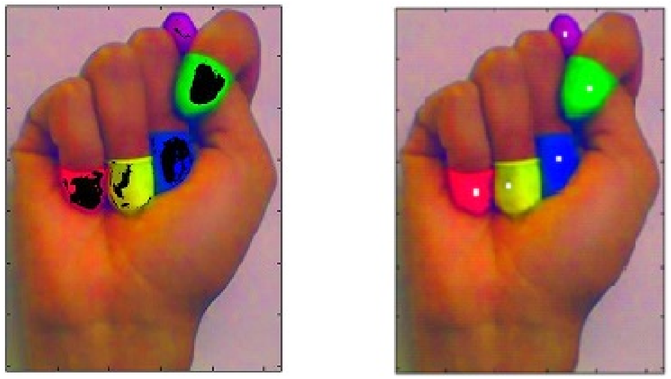


Figure 2.18: Detection of fingertips for Letter ’t’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

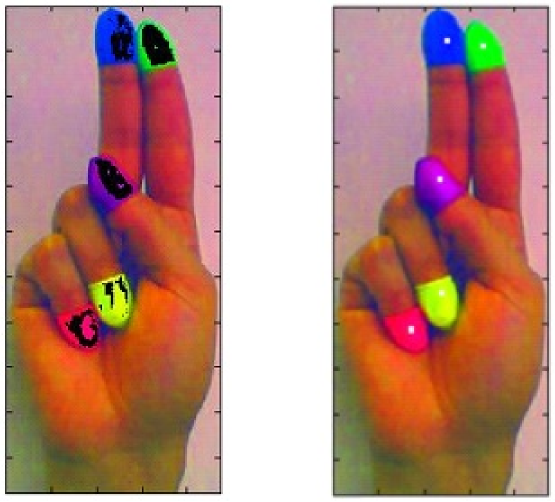


Figure 2.19: Detection of fingertips for Letter ’u’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

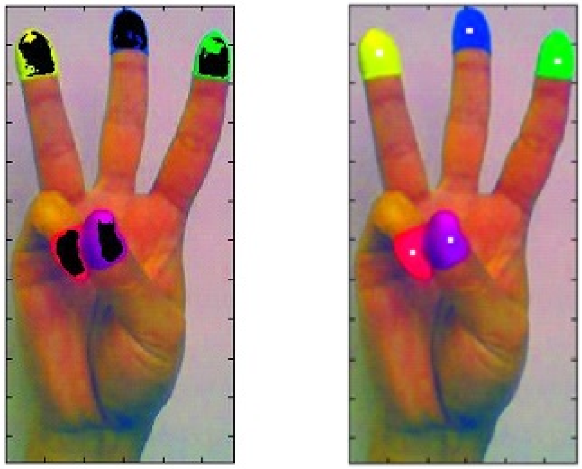


Figure 2.20: Detection of fingertips for Letter 'w'. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

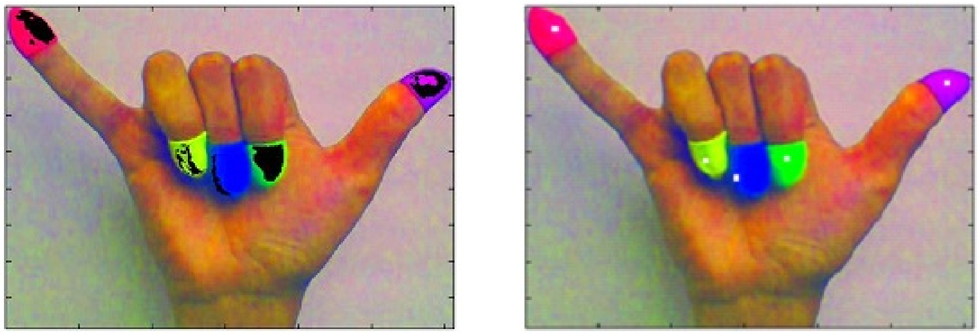


Figure 2.21: Detection of fingertips for Letter 'y'. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

Although most fingertips were identified, the program had difficulties identifying the midpoints of a few fingers on the images already stored. For example, in Figure 13 the ring finger is incorrectly identified. This was caused by the shaded images being too dark which affected the RGB ranges and caused them to be out of range for a "normal" image. Considering that most snapshots would be taken in normal lighting, this method was still seen to be valid. To fix the problem, shaded images were retaken to make them a little less shaded, then the process was repeated, and all the fingertip colors were recognized. New recolored fingertips and midpoints for each of the twelve gestures performed can be found in Appendix B.

2.8 Image scaling

As explained above, the distance between fingertips is a key measure for determining accuracy of the user's gestures. Before comparing the distances between fingers of both the database and the user's images, we needed to find a scaling factor for each letter, to use on the distance measurements. This way both images would be proportional in size for comparing. We decided to scale the user's images to match the width of the database images, because the user's images were already bigger than the database images. This way, the resolutions of the database images were maintained. Considering angles were not dependent on the images being proportional, they were not scaled.

The user began by performing each letter as close as possible to the images in the database. Then we detected both the left and right edges of where the hand began, in both the user's and database images, to find the width. This was performed by using Sobel edge detection [32]. We closed in on both the right and left side of both Sobel-transformed images until the edges were detected. An example of this can be seen in Figure 2.22 using the letter 'a'.

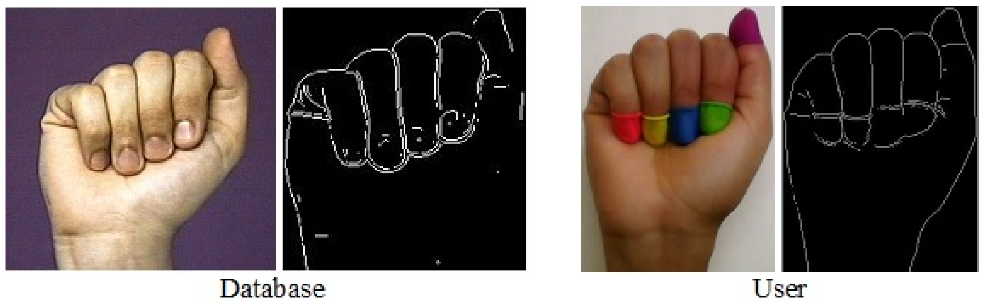


Figure 2.22: Edge detection process, between database (left) and user (right), using 'Sobel' edge detection, to find the width of 'a'

After obtaining both widths, (1) was used to find the scaling factor.

$$\text{Scaling Factor} = \frac{\text{database width}}{\text{user width}} \qquad (1)$$

A new scaling factor will not be found or changed each time a different variation of the letter is signed. The is because the perimeter of the hand gestures will be off, such as a finger extended too far out, and will make the edge detection process skew the width for the scaling factor.

2.9 Determination of angles and distances

While investigating methods in determining whether a gesture was correct or incorrect, an idea was to compare angles formed between the fingers as well as distances between fingers. Figure 2.23 shows the fifteen key angles that were chosen. In total there are sixty different possible angle formations among the five fingers. The process that was used for choosing these fifteen angles is as follows. We began by ruling out angles that were repeated but reversed. Then we noticed that bigger angles consisted of smaller angles added together; those were eliminated next. We decided to only use the smaller angles, because they covered all the angles possible in the gesture.

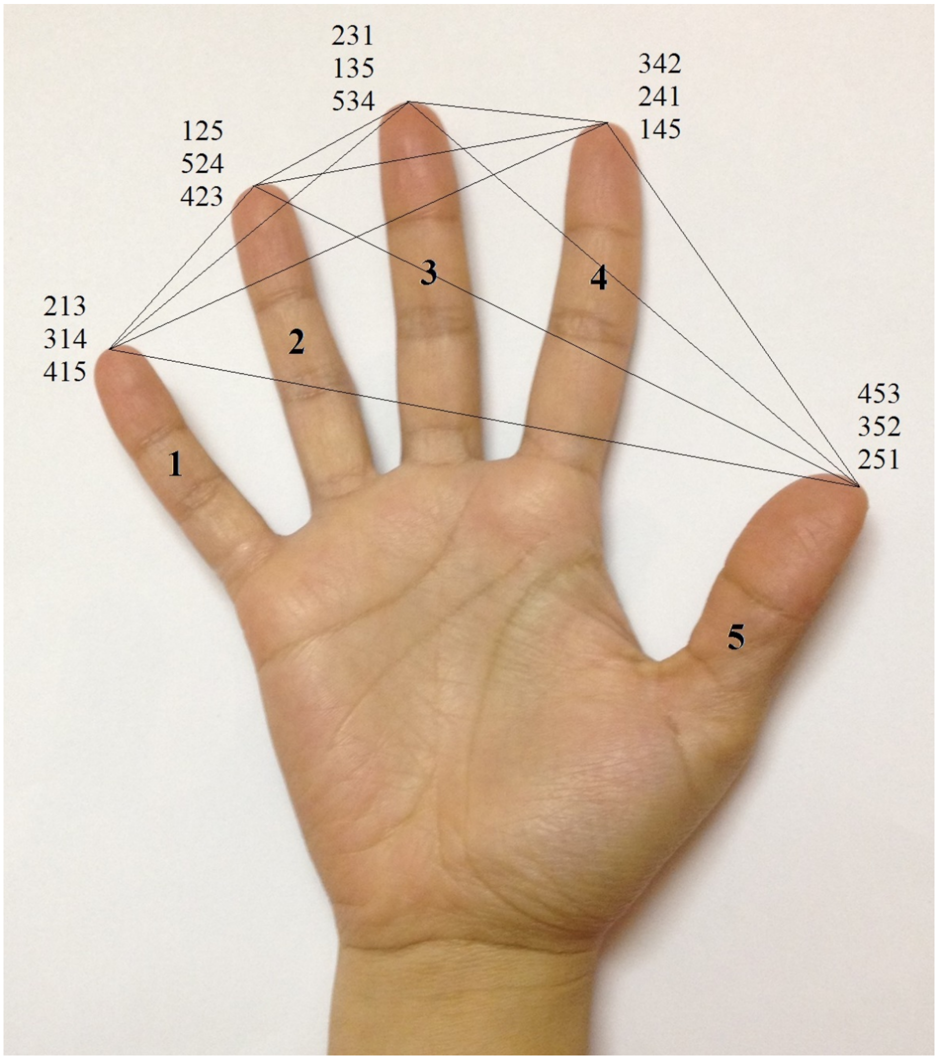


Figure 2.23: Angles formed between all fingers

Other angles that were used later on for comparison purposes were the outer angles that formed the perimeter of each hand. All angles were different for each letter signed because they all had unique shapes. An example of the outer angles that were used for the letter 'a' performed by the user can be seen in Figure 2.24.

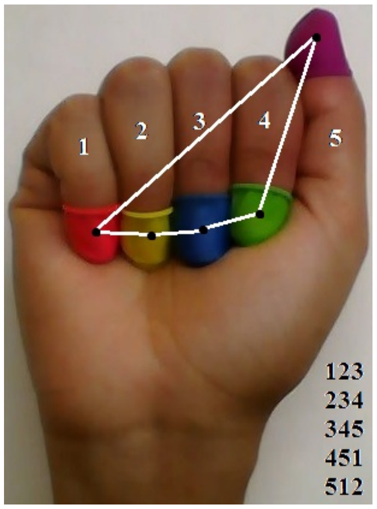


Figure 2.24: Outer angles of the letter ’a’

Distance calculations depended on the lines that formed all the fifteen chosen angles. For example, if Angle 213 was going to be used, the distance between finger 2 and finger 1 as well as the distance between finger 1 and finger 3 was going to have to be calculated. Considering all midpoints were previously found we had x and y coordinates for each fingertip. This information was used to find all the distances which were calculated using,

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2)$$

where x_1, y_1 are the coordinates of one fingertip and x_2, y_2 are the coordinates of another.

2.10 Angle comparisons

We compared the accuracy of the gestures by first comparing the fifteen key angles as well as the five outer angles between user and database images.

2.10.1 Fifteen key angles

As previously mentioned, fifteen angles were chosen to be the “Key angles” for all twelve letters. After calculating both database and user angles, differences between the images were calculated along with RMS errors of these angles. The RMS error is the root mean square error for all angles and is calculated using,

$$\text{RMS error} = \sqrt{\frac{\sum_{t=1}^{n}(a_t - \hat{a}_t)^2}{n}} \qquad (3)$$

where a_t is a particular angle on the scaled user image, $\hat{a}_t$ is the same angle on the database image, n is the number of angles used in the comparison.

Table 2.6 shows the angle differences between the user and the database gestures for each of the fifteen angles that were chosen (Finger 1 = pinky,..., Finger 5 = thumb). Based on the RMS errors of the differences between the user's gestures that were performed "correctly" and the database images, we found that the images had angle RMS errors between ten and twenty-three degrees. From this we chose a threshold of twelve degrees for determining gesture accuracy. In Table 2.6, we can see the angles that resulted in an angle RMS error greater than twelve degrees when performing correct gestures. Angles with a difference greater than twelve degrees are represented with an 'x' for all twelve letters. Table 2.6 shows that each individual letter had a total of about five or fewer angles that deviated by more than twelve degrees (refer to the count at the bottom of Table 2.6). We can also see that three out of the twelve letters (a, b, and n) were considered extremely close matches because they did not have angles greater than twelve degrees.

Table 2.6: Fifteen key angles that resulted in greater than a twelve degree difference between user and database images

Fingers forming the angles			a	b	e	h	i	L	n	r	t	u	w	y	count
2	1	3				x				x	x				3
3	1	4						x							1
4	1	5						x					x		2
1	2	5			x	x				x				x	4
5	2	4					x	x							2
4	2	3					x	x			x				3
2	3	1					x							x	2
1	3	5			x						x			x	3
5	3	4			x			x		x		x			4
3	4	2					x			x					2
2	4	1					x							x	2
1	4	5			x						x				2
4	5	3													0
3	5	2			x										1
2	5	1											x		1
count> 12^o			0	0	5	2	5	5	0	4	4	1	2	4	

Although all the images of the twelve letters were made as correctly as possible, some angles were considerably off. This could have been caused either by the location of the midpoint or considering some fingertips were close to one another, a slight movement of the finger could have caused more deviation than necessary.

2.10.2 Outer angles

Another idea that was considered to show how close the gestures were to the database was to find the outer angles of all the letters. The outer angles formed the unique outside perimeter shape of all twelve letters. Table 2.7 shows the five additional angles. Based on the data found, the outer angles were not useful. Although all twelve gestures were made as close as possible to the database images, the program considered many of the letters completely off. This could be explained by slight movement in adjacent fingers offsetting the angles even when the user's gesture appeared to be a close match to the database gesture.

Table 2.7: Five outer angles that resulted in greater than a twelve degree difference between user and database images

Corresponding angle for specific letter	a	b	e	h	i	L	n	r	t	u	w	y	count
1	x				x			x	x	x		x	6
2			x						x				2
3	x			x	x								3
4				x									1
5			N/A	N/A		N/A		x			N/A		1
count $> 12^o$	2	0	1	2	2	0	0	2	2	1	0	1	

2.11 Testing of intentionally incorrect user images

After finding differences between the user's "correct gestures" and the database images, we now had an angle threshold that we could use to start comparing incorrect gesture images. Considering the letter 'a' was a very close match, based on the information in Table 2.6, we decided we would use this letter as the preliminary testing image. Then, we took five different shots of the user performing the letter 'a' incorrectly. These five different versions of the letter 'a' can be seen in Figure 2.25 (Refer to Figure 2.1 to see the database image of the letter 'a' and Figure 10 for the user's correctly performed 'a'.) Images a01 through a03, in Figure 2.25, show the thumb gradually being moved farther to the observer's right. In a04 the thumb is moved slightly down to the observer's left and in a05 the pinky is moved up. By using my program we found the midpoints and calculated the angles of the incorrect gestures that were performed.

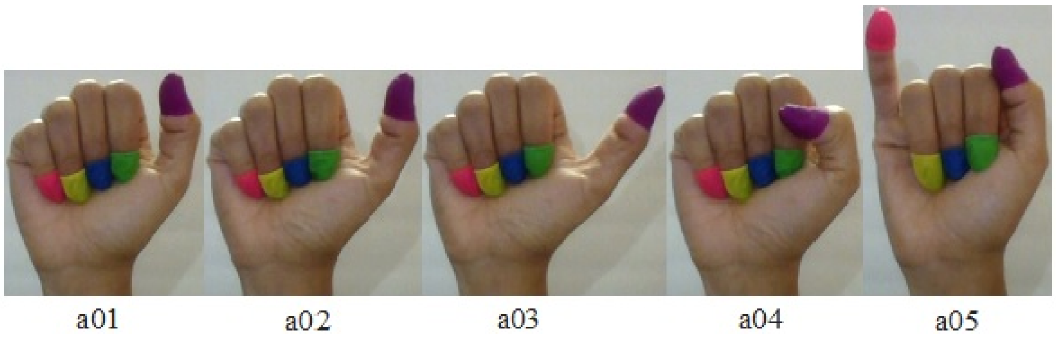


Figure 2.25: 5 Versions of the user performing the letter ’a’ incorrectly

2.11.1 Angle differences between correctly and incorrectly performed ’a’

After calculating all the angles for all five incorrect versions of the letter ’a’, we then tabulated them alongside the angles of the correct ’a’ gesture. These results are summarized in Table 2.8. It can be seen in Figure 2.25 that fingers 1 and 5 (pinky and thumb) were moved. The information in Table 2.8 shows that the angles that involve finger 1 and 5 have more deviation than other angles. Angles highlighted in red show angle deviations when finger 5 was an outer segment of the angle. Angles highlighted in green show angle deviations when finger 1 was an outer segment of the angle. This confirmed that the program was correctly detecting the angles which were incorrect.

Table 2.8: Difference of angles between the fingers of five different incorrect versions of ’a’ compared to the correctly performed ’a’

Fingers that form the angles			Degree difference of angles					
First	Second	Third	“Correct a”	a01	a02	a03	a04	a05
2	1	3	9.9	7.6	9.0	7.2	9.0	0.7
3	1	4	2.0	2.2	2.3	2.9	0.7	7.4
4	1	5	3.8	6.5	15.5	21.9	11.5	2.6
1	2	5	8.9	1.4	10.9	21.6	0.3	72.9
5	2	4	8.5	7.8	19.2	26.0	11.7	5.3
4	2	3	2.0	3.1	3.9	5.0	1.7	4.8
2	3	1	9.5	6.5	7.7	7.7	5.9	63.5
1	3	5	3.4	12.4	26.6	37.1	9.3	76.9
5	3	4	9.0	4.8	18.8	27.8	9.6	4.9
3	4	2	1.8	2.1	2.8	3.3	1.7	3.7
2	4	1	5.5	1.0	1.6	1.2	2.4	60.9
1	4	5	7.8	8.1	29.3	43.0	1.9	75.2
4	5	3	2.5	2.2	4.9	8.8	8.1	5.7
3	5	2	2.3	3.4	6.7	5.6	0.6	3.4
2	5	1	0.8	0.3	2.2	4.0	2.1	63.6

2.11.2 Distance differences between correctly and incorrectly performed ’a’

After calculating all the distances for all five incorrect versions of the letter ’a’, we then placed all the distances in a table along with the distances of the correct ’a’ gesture. These results are summarized in Table 2.9. In Figure 2.25, we can see that fingers 1 and 5 (Pinky and Thumb) were moved. The information in Table 2.9 show that the distances that involved finger 1 and 5 had more deviation than the other distances. Distances highlighted in red show large distance deviations when finger 5 was moved. Distances highlighted in green show large distance deviations when finger 1 was moved. This confirmed that the program was correctly detecting the fingers which were incorrect.

Table 2.9: Pixel distance difference between two fingers of five different incorrect versions of ’a’ compared to the correctly performed ’a’

Distance between two fingers		Pixel distance difference between two fingers					
First	Second	“Correct a”	a01	a02	a03	a04	a05
1	2	5.3	1.7	1.7	2.1	0.1	106.4
2	3	0.7	2.8	2.9	1.2	3.7	2.8
3	4	1.7	8.7	8.4	7.7	5.1	7.0
4	5	7.9	2.6	8.1	30.3	37.4	4.2
1	3	6.8	2.8	2.5	1.6	2.1	88.7
1	4	8.7	11.9	11.3	9.7	7.1	70.2
1	5	6.0	12.6	30.1	53.3	25.2	14.1
2	4	2.1	11.3	11.0	8.6	8.8	9.5
5	3	5.0	5.7	19.9	42.4	31.6	1.0
5	2	0.9	11.1	26.3	46.9	25.5	1.9

2.12 Finding the centroids

In order to provide the user with instructions concerning how to correct misplaced fingers, we need to determine the amount of x and y displacement that each finger needs to make the gesture “correct.” The centroid of each fingertip location provides a central point to compare relative positions of the finger tips. Centroids are useful because they provide a mean position of selected points in a plane. All the gestures performed could be considered figures on a plane. We used the midpoints found on the fingertips to find a centroid for each of the gestures performed as well as the database images. The centroid position identified by the coordinates (x_c, y_c) is calculated using,

$$x_c = \frac{\sum_{i=1}^{5} x_i}{5} \qquad (4)$$

$$y_c = \frac{\sum_{i=1}^{5} y_i}{5} \qquad (5)$$

where (x_i, y_i) are the coordinates of the detected center position of the i^{th} fingertip. We used (2) to calculate the distances between each fingertip in relation to the computed centroid.

An example of this can be seen in Figure 2.26. It shows the centroid and the distances from each finger to the centroid of the database letter ’a’ (a), the user’s correctly performed ’a’ (b), and the user’s incorrectly performed ’a’ (c), which is the letter version a03 performed previously. Table 2.10 shows the pixel distances from each finger to the centroid of the database ’a’, the user’s correctly performed ’a’, and the user’s incorrectly performed ’a’.

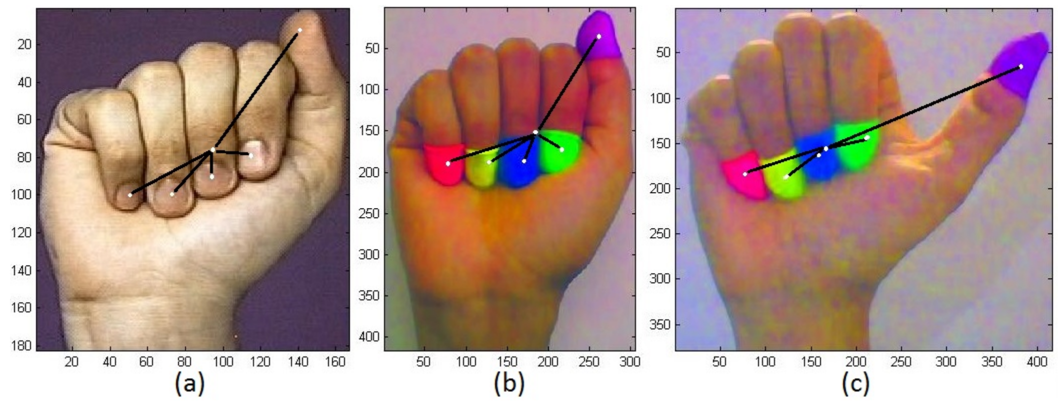


Figure 2.26: Centroid and distances from fingers to centroids of (a) database ’a’, (b) user’s correctly performed ’a’, and (c) user’s incorrectly performed ’a’

Table 2.10: Distances, in pixels, from fingers to centroids of the database ’a’, the user’s correctly performed ’a’, and the user’s incorrectly performed ’a’

Finger	Distance to Centroid of database ’a’	Distance to Centroid of user’s correctly performed ’a’	Distance to Centroid of user’s incorrectly performed ’a’
Pinky	49.56	54.21	64.49
Ring	31.84	29.95	43.21
Middle	14.22	17.01	19.27
Index	20.45	26.67	12.30
Thumb	78.77	82.01	114.25

From Table 2.10 we can see that when comparing the database ’a’ and the user’s correctly performed ’a’ that the distances are relatively close. But when comparing the database ’a’ to the user’s incorrectly performed ’a’ we can see that the thumb, which was the finger that was moved, now had the largest amount of deviation. Although the thumb was the only finger that was moved, we can still see some deviation in the other distances. This was caused by the centroid being shifted farther down which then caused there to be deviation from the other fingers as well.

2.13 Determining *x* and *y* correction

The correction process is the most important part of ASL training, because it provides the user with useful feedback concerning correct finger placement. We came up with nine different conditions that the program sifted through to determine if the position of the individual finger would or would not be considered within range of the database image. This process is similar to the Eight Nearest Neighbors technique. If the fingers are not considered within range, the program lets the user know which fingers are incorrect so correction can be performed. It also provides specific instructions on how to move the fingers so it is within a range in where the program will consider the finger “correct”. It goes through all the fingers and finds which finger is the worst (has the maximum amount of pixels off) and only tells you to correct that certain finger before correcting any other finger. Then the program will provide the user with the *x* direction of movement first, followed by the *y* direction of movement needed.

2.13.1 Testing using ’Eight Nearest Neighbors’ technique

A visual of the Eight Nearest Neighbor technique can be seen in Figure 2.27. The X represents the correct location of where the finger should be placed if the distance to the centroid is more than twelve pixels greater or less than that of the database. If the finger is any other location except at X, the program will prompt the user on how to move their finger accordingly. An example would be, if the finger fell within Region 3, the program would tell the user to move their finger left so many pixels then down so many pixels.

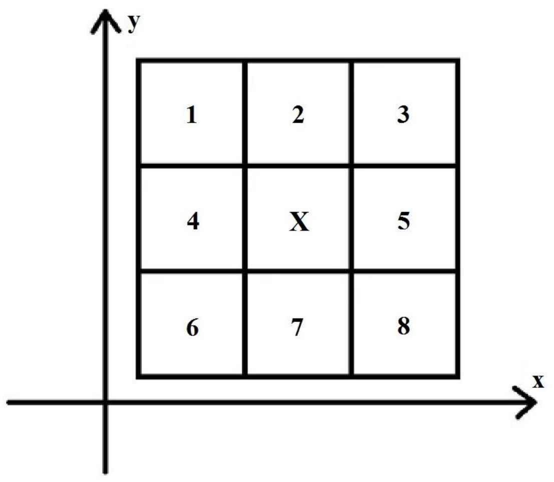


Figure 2.27: Eight Nearest Neighbor finger locations

2.13.2 Determining the amount of fingertip movement

A visual representation of the amount of movement that was needed in the x and y direction was desirable in order to show the user how to correct their finger positions.
A measure of accuracy for the correction process was found using the distance RMS error between the user and database images,

$$\text{RMS error} = \sqrt{\frac{\sum_{t=1}^{n}(d_t - \hat{d}_t)^2}{n}} \qquad (6)$$

where d_t is the distance between a particular fingertip and the centroid on the scaled user image, $\hat{d}_t$ the distance between a particular fingertip and the centroid on the database image.

We computed the RMS error for all the twelve letters and computed that twelve pixels was a good measure of accuracy. Figure 2.28 shows the distances from the centroid to the thumb as well as the x and y displacement for the database 'a' (a), the user's correctly performed 'a' (b), and the user's incorrectly performed 'a' (c). Notice that the x distance in the incorrectly performed 'a' is much larger than the x distance in the correctly performed 'a.' In this example, the amount of x movement needed to correct the thumb was determined by subtracted the x in the user image from the x in the database image. Similarly, the amount of *y* movement needed to correct the thumb was determined by subtracted the *y* in the user image from the *y* in the database image.

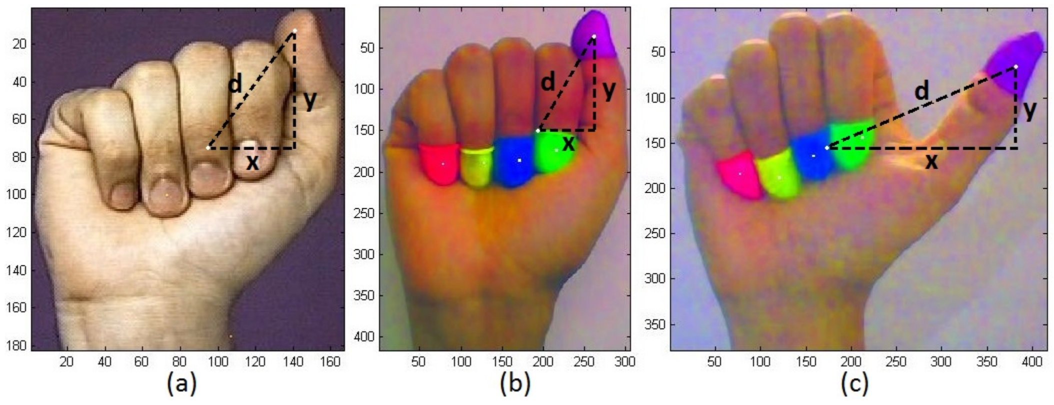
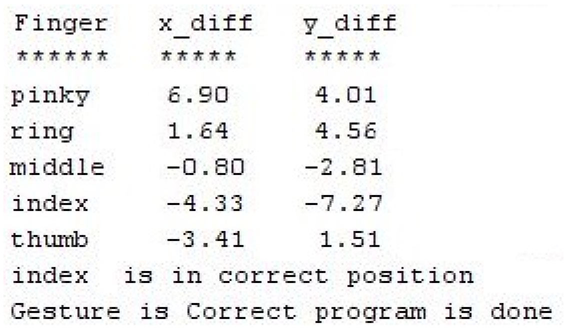


Figure 2.28: Centroids to Thumb for: (a) Database 'a', (b) User's correctly performed 'a', (c) User's incorrectly performed 'a'

Figure 2.29 shows the *x* and *y* pixel differences displayed on the screen for the user's correctly performed 'a' as compared to the database 'a' (refer to Figure 2.28). We can see that all the differences for both the *x* and *y* directions are all below the twelve pixel threshold. Considering all distances were under threshold, the program then provides the user with a message "Gesture is Correct program is Done."



```
Finger    x_diff    y_diff
******    *****     *****
pinky      6.90      4.01
ring       1.64      4.56
middle    -0.80     -2.81
index     -4.33     -7.27
thumb     -3.41      1.51
index  is in correct position
Gesture is Correct program is done
```

Figure 2.29: Screen shot of *x* and *y* distance differences between database 'a' and user's correctly performed 'a'; Indication of whether the gesture was performed correctly

Figure 2.30 shows the *x* and *y* pixel differences displayed on the screen for the user's incorrectly performed 'a' when compared to the database 'a' (refer to Figure 2.28). When comparing these two images, we can see that the differences increased for at least one of the directions on each of the fingers. But for the thumb, we notice that it has the greatest amount of difference, especially in the *x* direction. Considering that our program corrects one finger at a time, whichever has the most deviation, we can see that our program asks us to correct the thumb's position first. The program provides the user with a message to "Move Thumb -58.47 pixels (107 pixels unscaled) in the *x* direction" and "Move thumb -18.01 pixels (33 pixels unscaled) in the *y* direction." It also provides a stopping *x*, *y* coordinate with the message "Move thumb to the location $x = 275$, $y = 32$." Notice that the program displays scaled pixel values while the commentary provides unscaled pixel values. If you use the scaling factor particular for a, which was about .545, and you multiply by 107, you will get 58.3.

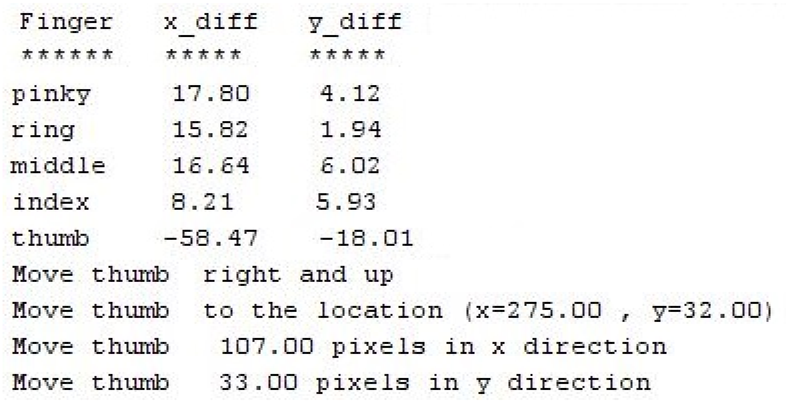

```
 Finger    x_diff    y_diff
 ******    *****     *****
pinky       17.80     4.12
ring        15.82     1.94
middle      16.64     6.02
index       8.21      5.93
thumb      -58.47    -18.01
Move thumb  right and up
Move thumb  to the location (x=275.00 , y=32.00)
Move thumb   107.00 pixels in x direction
Move thumb   33.00 pixels in y direction
```

Figure 2.30: Screen shot of *x* and *y* distance differences between database 'a' and user's incorrectly performed 'a'; Directions on whether the gesture was performed correctly when compared to database 'a'

It is difficult to know how much 59 or 18 pixels are when correcting for that amount. In order to help the user, we provided a visual on how to correctly move their fingers. This is explained in the following section.

2.13.3 Correction of user's five incorrectly performed 'a'

The finger with the most deviation, based on the twelve pixel threshold, will always be used first when determining the location in which the finger needs to be moved. When performing the correction process, the user will be provided with three images: the database image of the letter they are signing, the image of the gesture they signed, and an image of the gesture they signed with a white line, which demonstrates the direction the finger needs to be moved. As previously explained, the program will provide the user with a certain amount of pixels that should be moved in the *x* and *y* direction. But to help the user, we made the program provide the user with a white line to show the direction the finger should be moved as well as a white dot at the end of the line to show the stopping location of the midpoint. In Figures 2.31-2.35 we can see these three images of the five different versions of ’a’ that were performed.

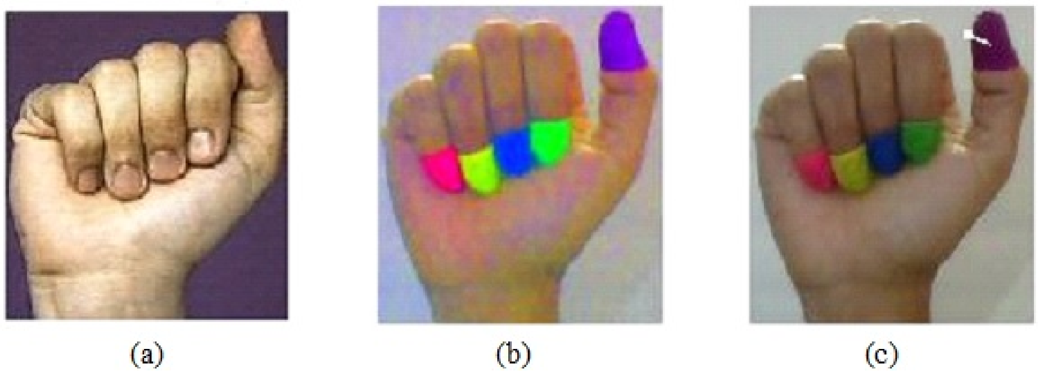


Figure 2.31: Correction of fingertip placement for ’a01’: (a) Database Image, (b) User Image, (c) Corrected User Image

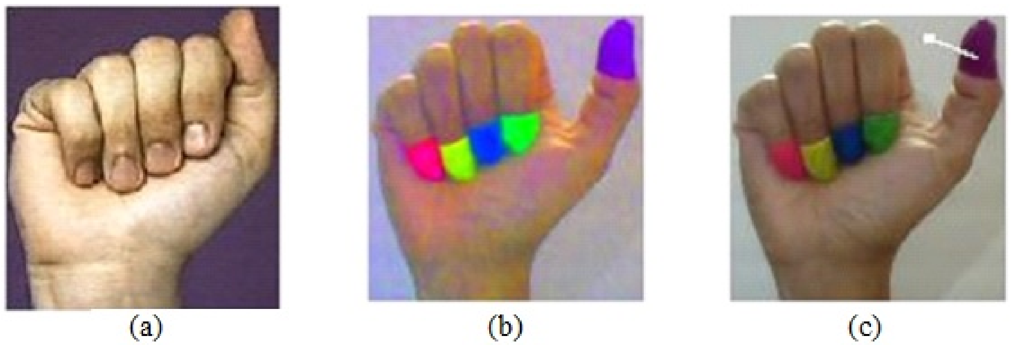


Figure 2.32: Correction of fingertip placement for ’a02’: (a) Database Image, (b) User Image, (c) Corrected User Image

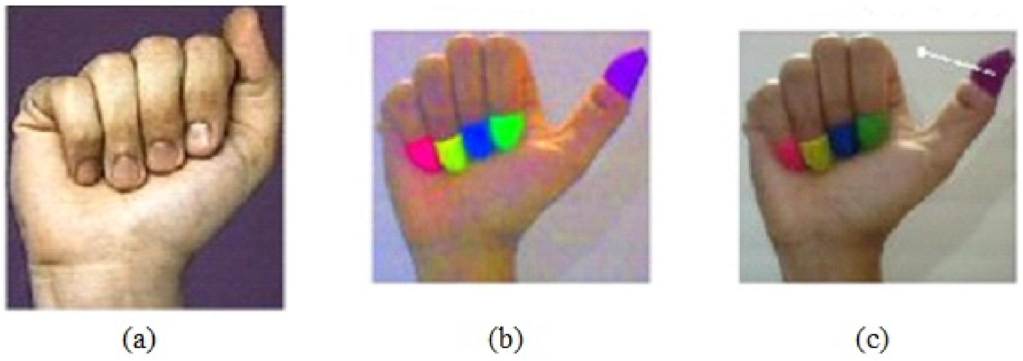


Figure 2.33: Correction of fingertip placement for ’a03’: (a) Database Image, (b) User Image, (c) Corrected User Image

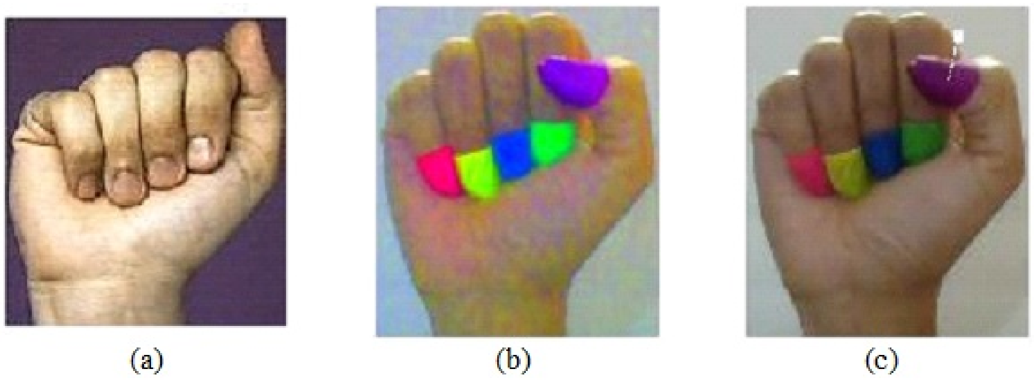


Figure 2.34: Correction of fingertip placement for ’a04’: (a) Database Image, (b) User Image, (c) Corrected User Image

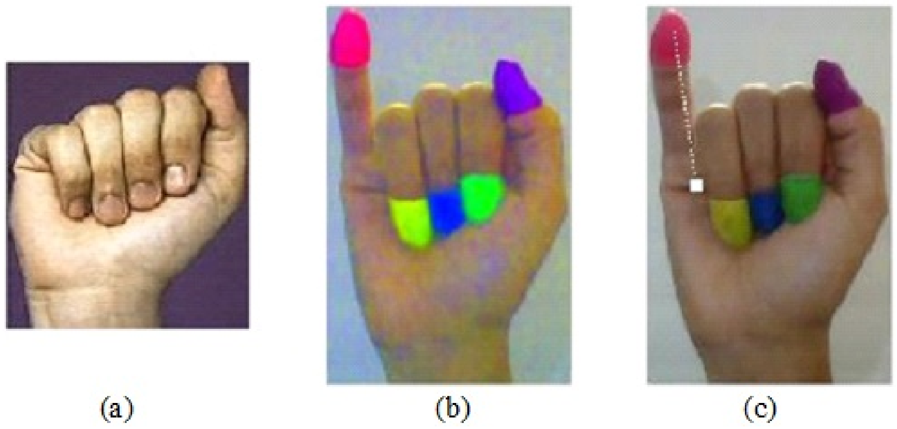


Figure 2.35: Correction of fingertip placement for ’a05’: (a) Database Image, (b) User Image, (c) Corrected User Image

Based on the location of the white dot, for all five versions of the letter ’a’ that were performed, we can see that the program gives an accurate location of the general area of where the fingertip should be moved. Considering the program corrects one finger at a time (whichever has the most deviation when compared to the database image) it could possibly show incorrectness of the ideal location because other fingers could be offsetting the one needing to be corrected first. An example of this can be seen in Figure 2.35. The location of the white dot is not precisely where it needs to be placed but once the user has corrected the finger with the greatest error; the next finger with the worst error will be asked to fix. An example of this can be seen in Figures 2.36-2.38 which show the correction process of version 05 of letter ’a’, from Figure 2.35. In Figures 2.36 and 2.37, if the user needs to redo their finger placement, the program shows the database image, the user’s gesture, and how they need to correct their gesture. In Figure 2.38, the user’s gesture was considered “correct” or within threshold, so the program outputted the database and the user’s “correct” image. Considering the program is using the location of the centroid to correct the user’s finger placement, a second (or as many times needed) correction could possibly be on the same finger until that finger is considered correct or until it moves on to another finger that has more displacement. The program will go through all the fingers and correct their location (if needed) until the gesture as a whole is considered to be within threshold.

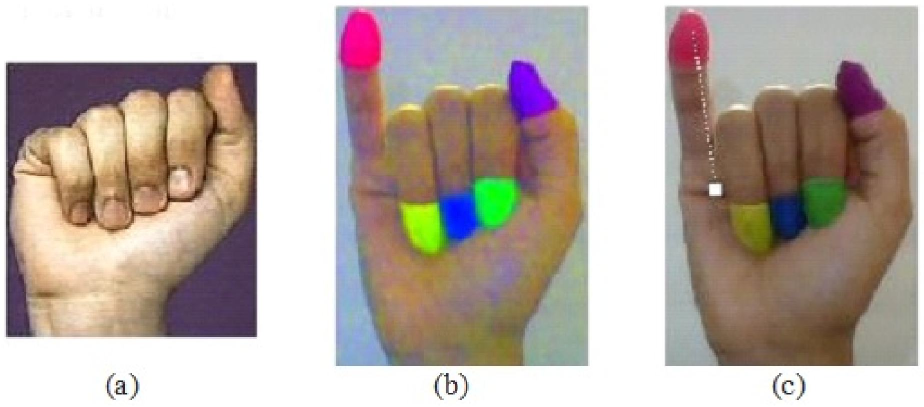


Figure 2.36: First correction process of version 05 letter ’a’, (a) Database ’a’, (b) User’s incorrect ’a’, (c) Correction for user’s incorrect ’a’

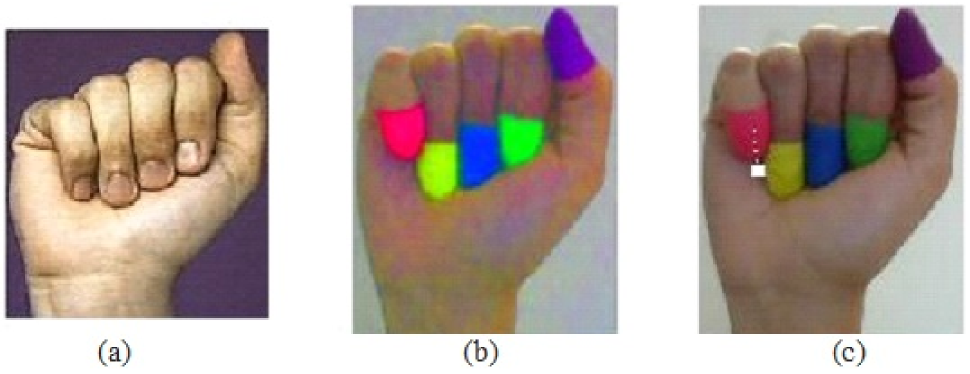


Figure 2.37: Second correction process of version 05 letter ’a’, (a) Database ’a’, (b) User’s incorrect ’a’, (c) Correction for user’s incorrect ’a’

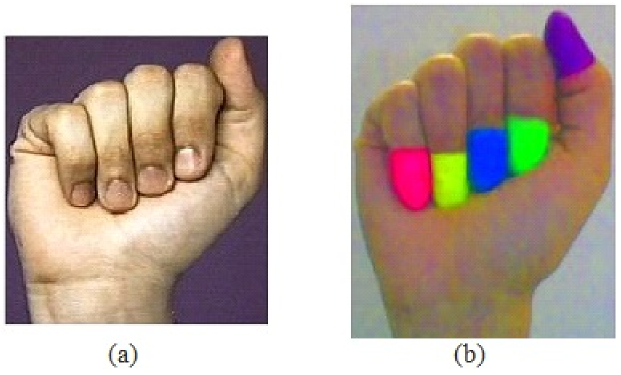


Figure 2.38: Corrected ’a’ of version 05 letter ’a’, (a) Database ’a’, (b) User’s corrected ’a’

2.14 Correction process of different letters and versions

Figures 2.39-2.92 show the correction process of different letters and versions of the letters. In each figure, we first see the database image of the letter that is being signed followed by the incorrect gesture, and then the correction(s) performed to get the gesture within threshold. There were a total of forty-seven different versions of the letters performed incorrectly, but only twenty-four are shown as a summary of the correction process. We randomly chose the versions of the letters shown. This is why some versions will be missing from certain letters.

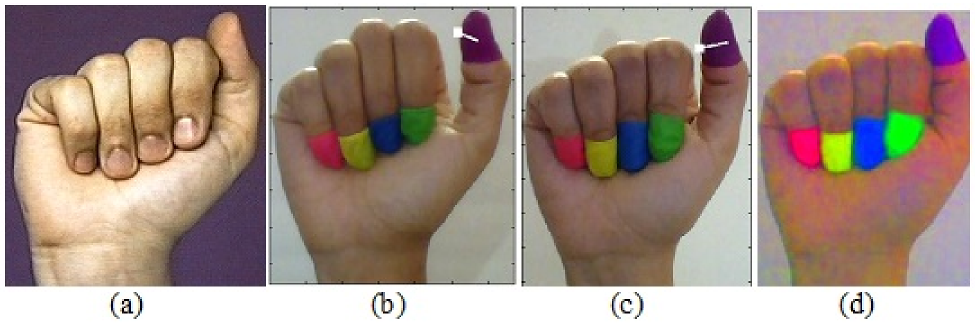


Figure 2.39: The correction process of a01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

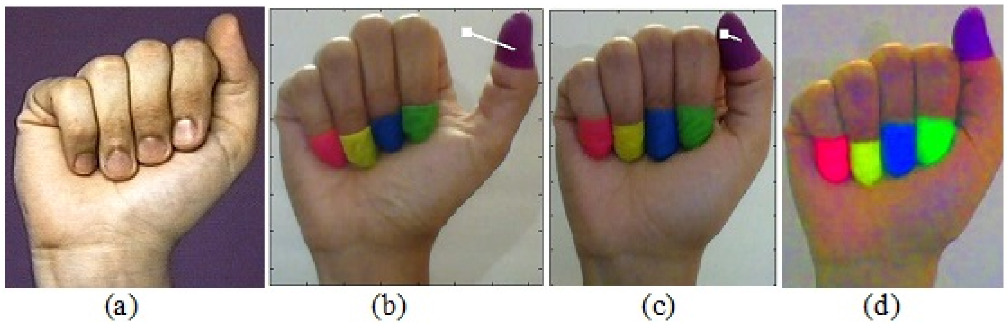


Figure 2.40: The correction process of a02; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

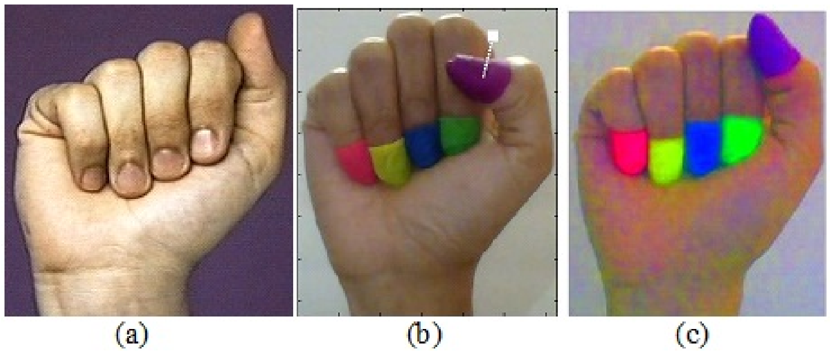


Figure 2.41: The correction process of a04; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

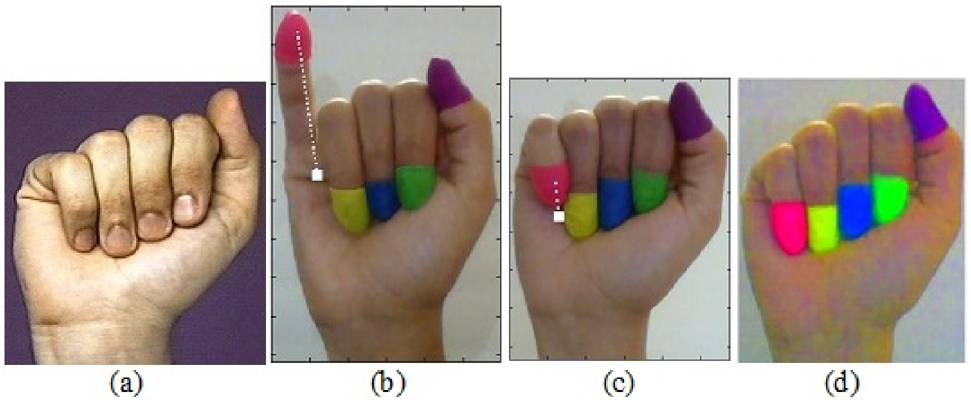


Figure 2.42: The correction process of a05; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

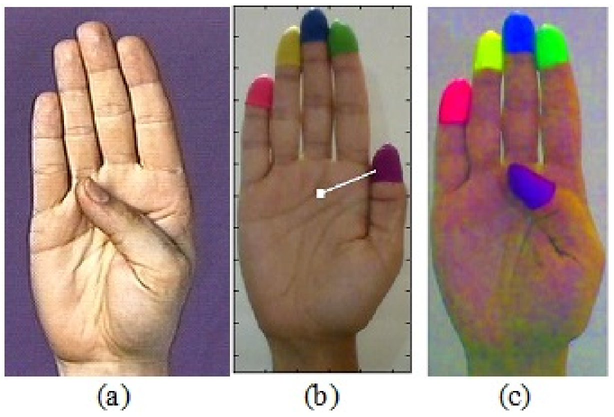


Figure 2.43: The correction process of b13; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

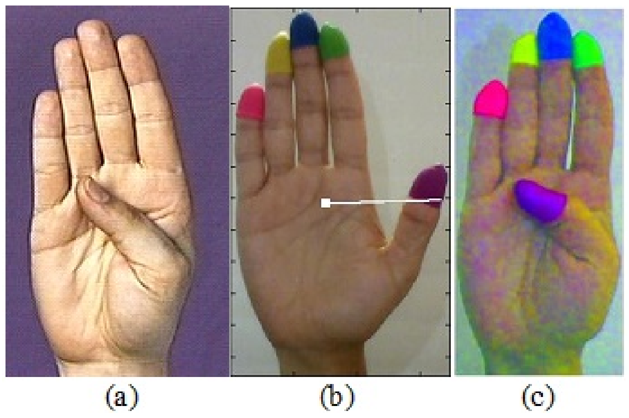


Figure 2.44: The correction process of b14; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

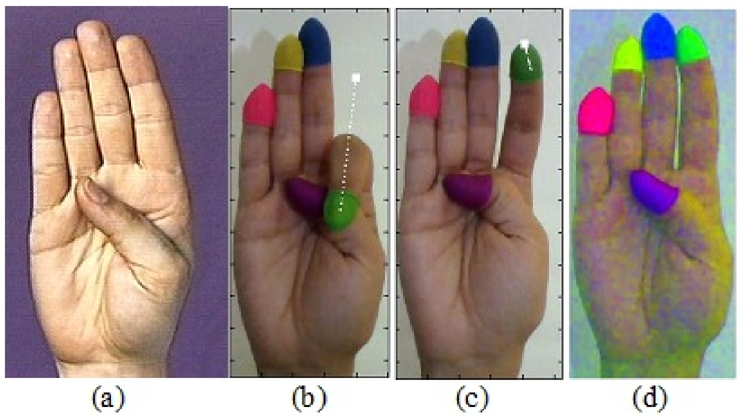


Figure 2.45: The correction process of b15; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

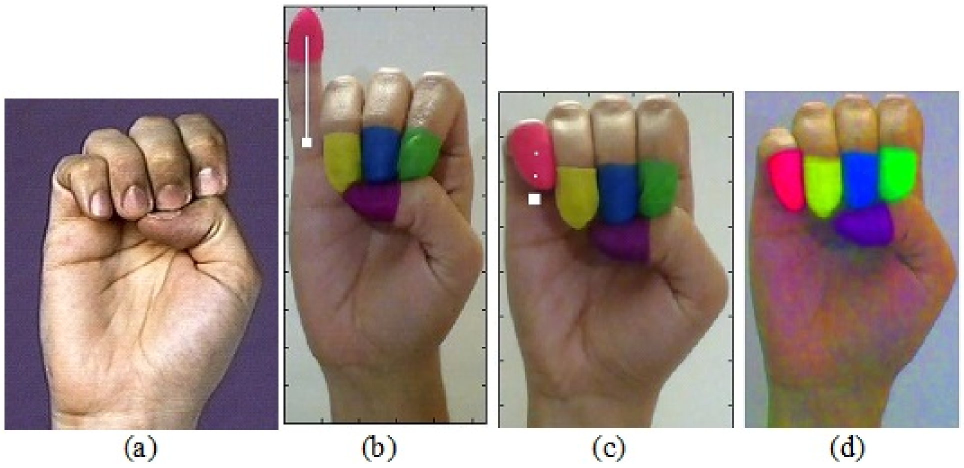


Figure 2.46: The correction process of e01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

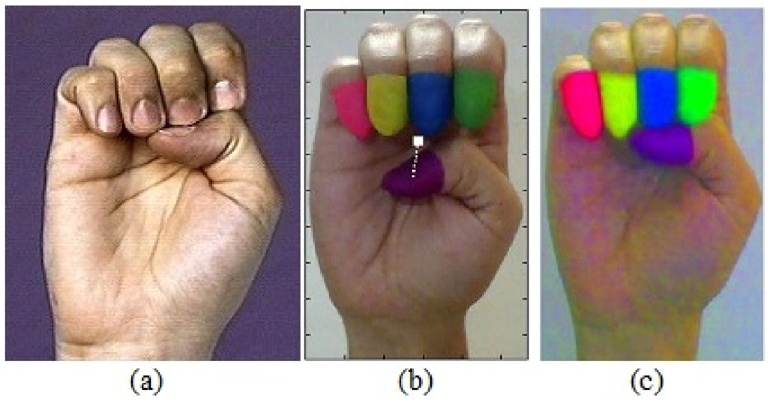


Figure 2.47: The correction process of e02; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

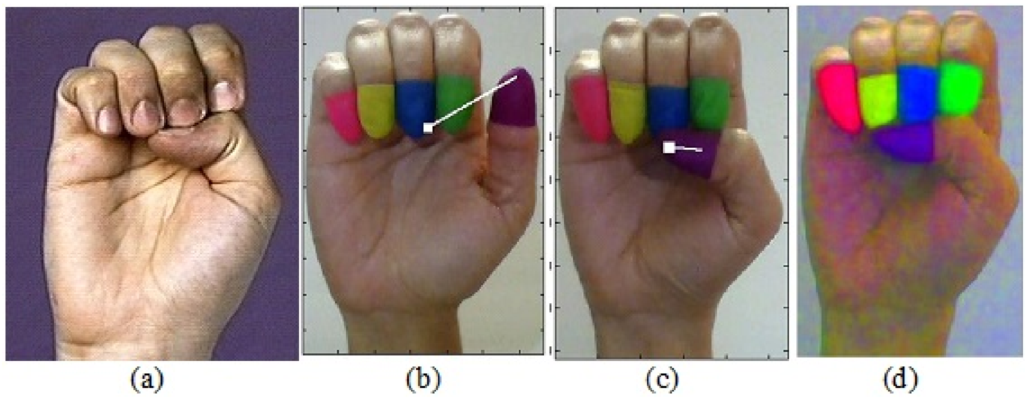


Figure 2.48: The correction process of e03; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

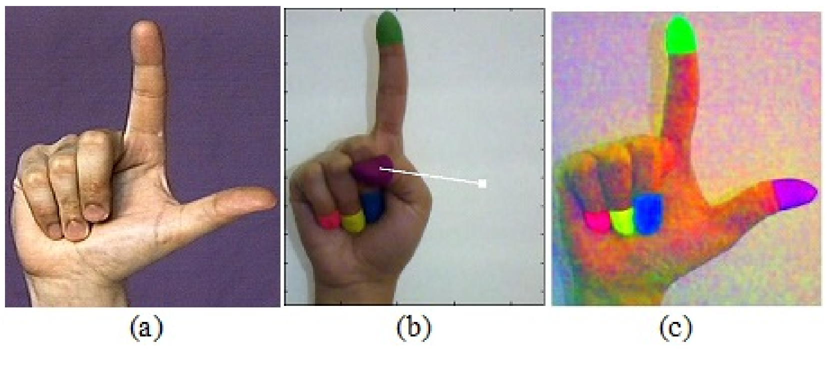


Figure 2.49: The correction process of L01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

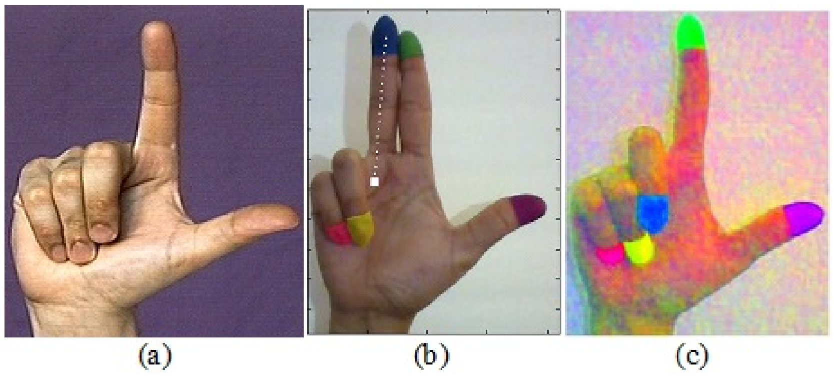


Figure 2.50: The correction process of L03; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

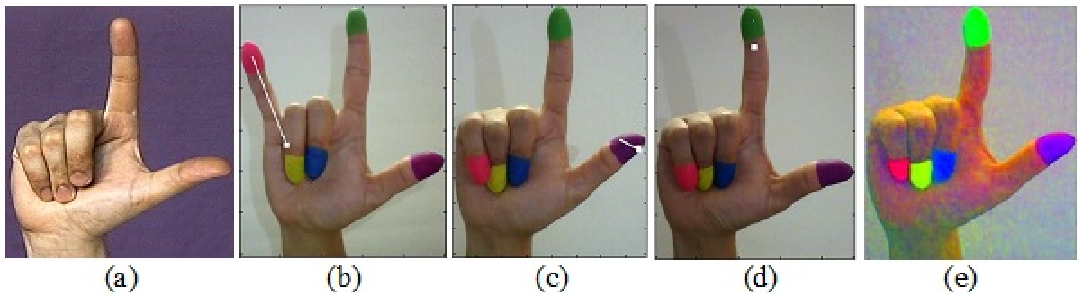


Figure 2.51: The correction process of L05; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) 3rd correction needing to be performed, (e) Corrected gesture

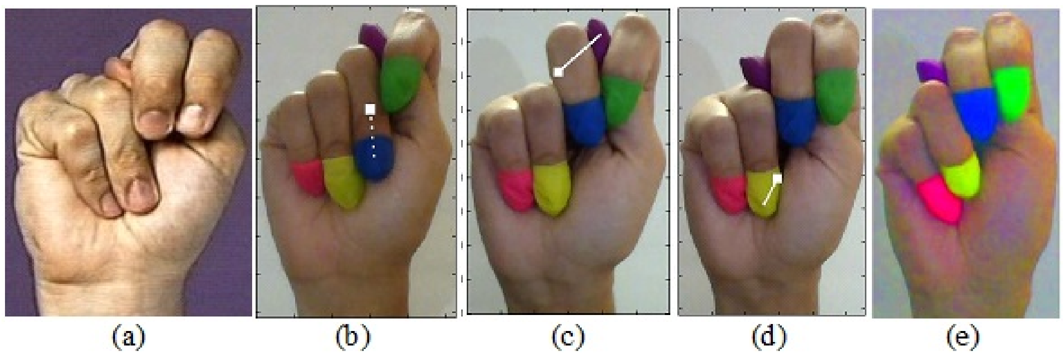


Figure 2.52: The correction process of n01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) 3rd correction needing to be performed, (e) Corrected gesture

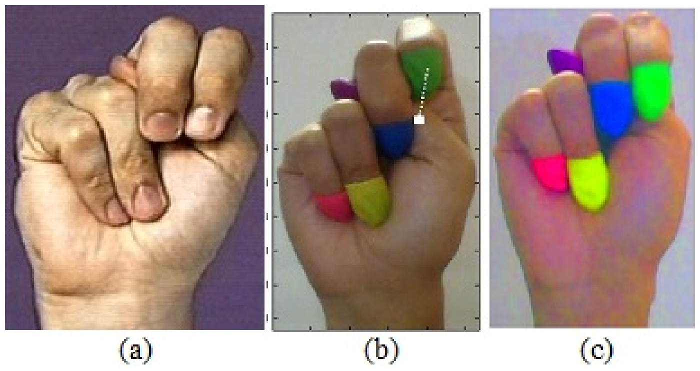


Figure 2.53: The correction process of n02; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

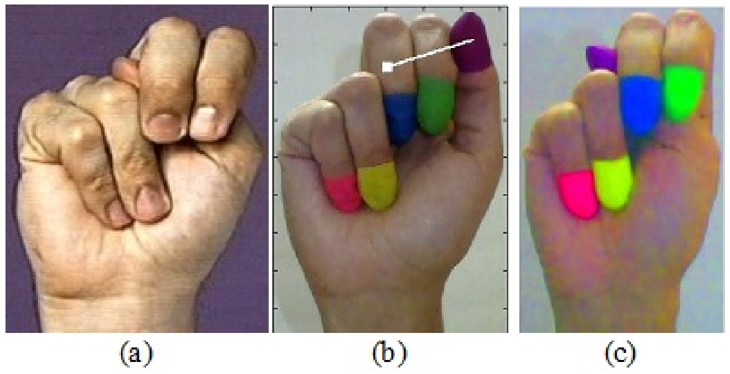


Figure 2.54: The correction process of n03; (a) Database image, (a) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

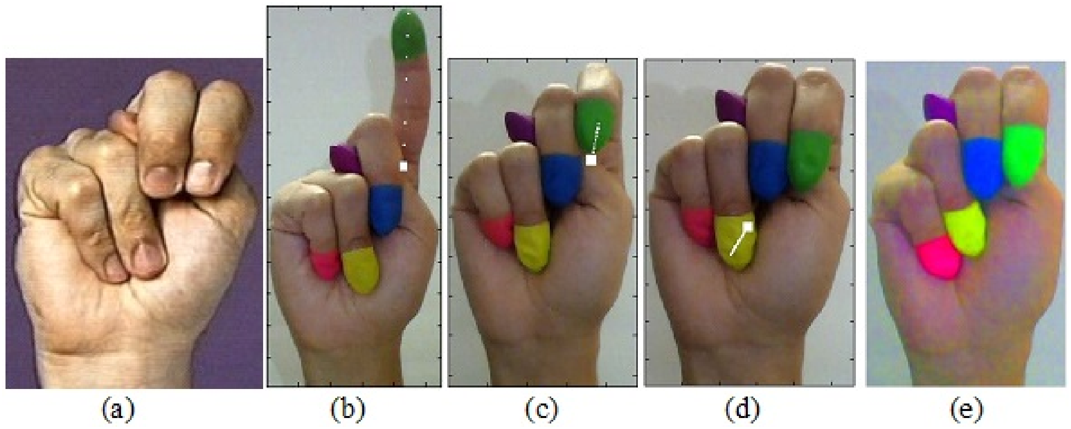


Figure 2.55: The correction process of n04; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) 3rd correction needing to be performed, (e) Corrected gesture

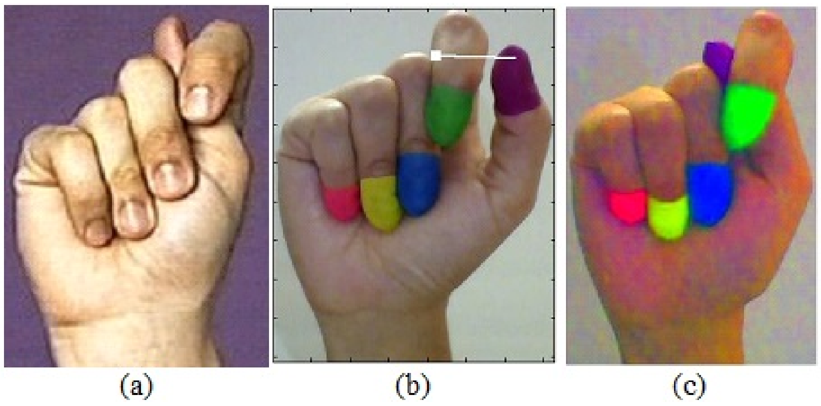


Figure 2.56: The correction process of t01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

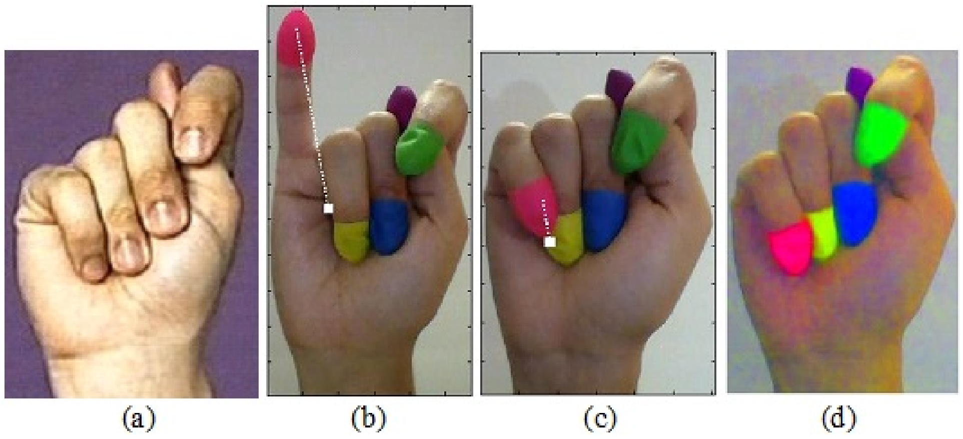


Figure 2.57: The correction process of t02; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

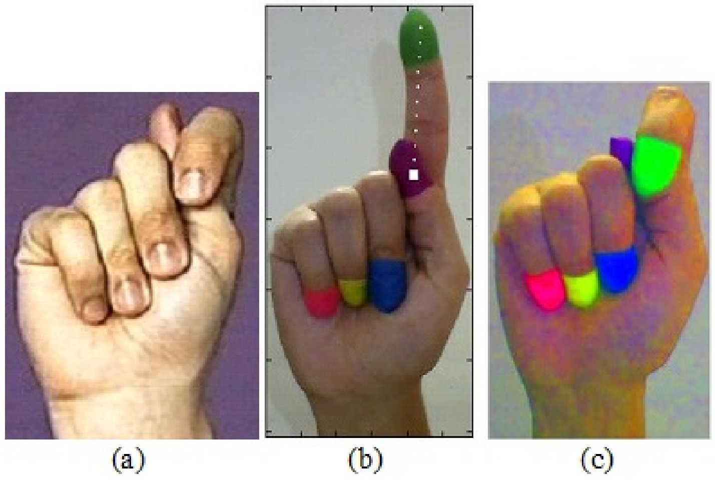


Figure 2.58: The correction process of t03; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

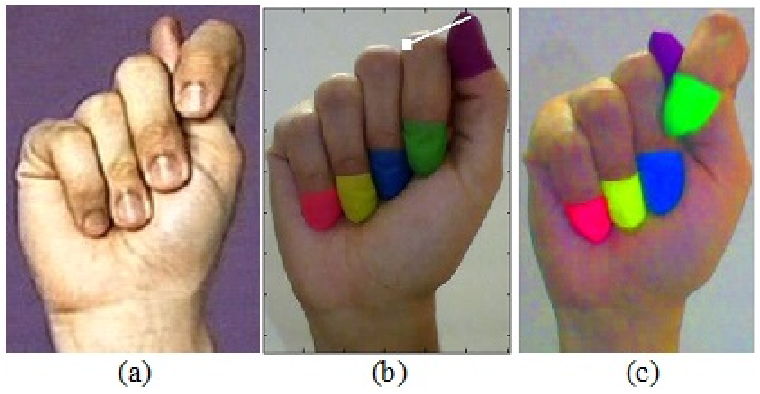


Figure 2.59: The correction process of t05; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

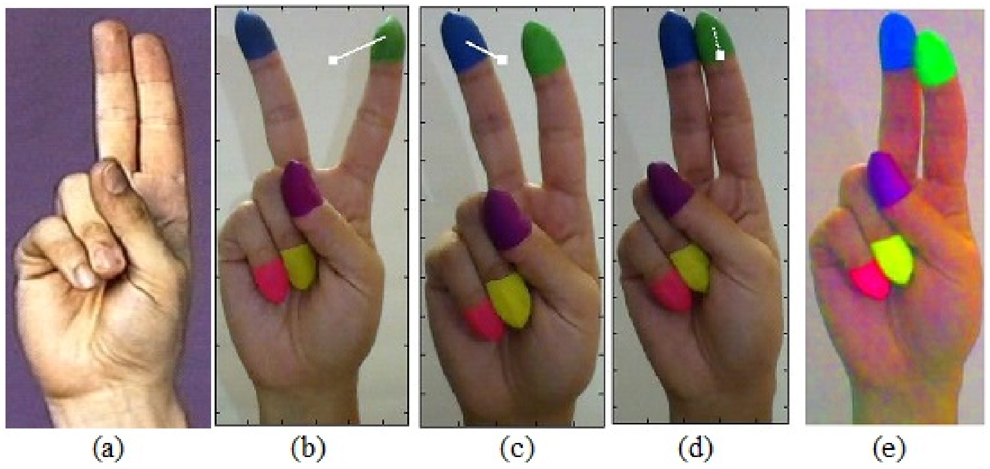


Figure 2.60: The correction process of u01; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) 3rd correction needing to be performed, (e) Corrected gesture

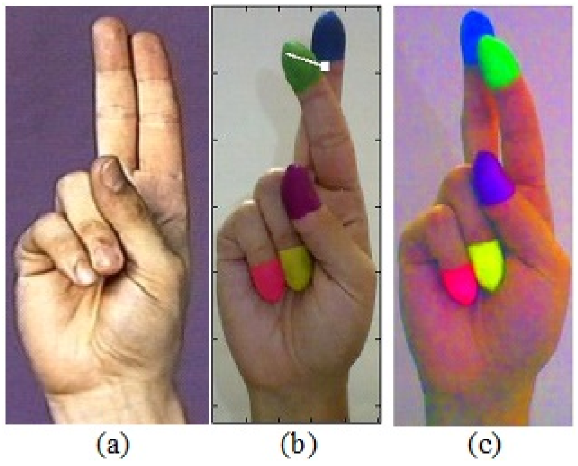


Figure 2.61: The correction process of u02; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) Corrected gesture

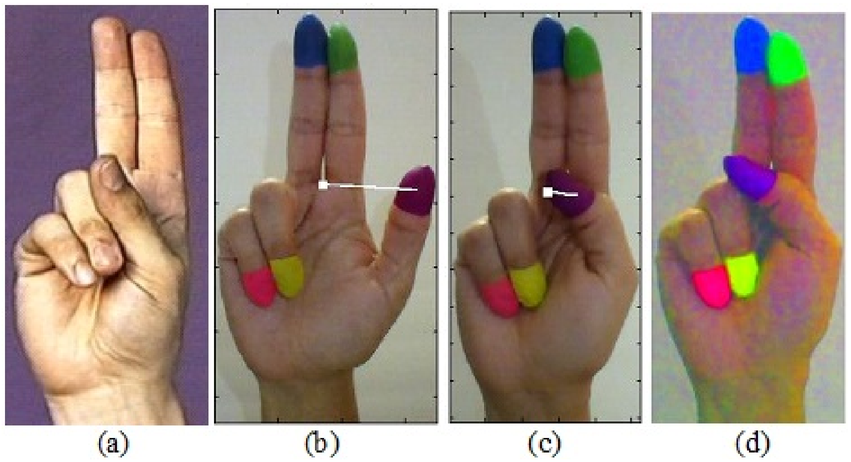


Figure 2.62: The correction process of u04; (a) Database image, (b) Initial incorrectly performed gesture and 1st correction, (c) 2nd correction needing to be performed, (d) Corrected gesture

2.15 Performance metric

Now we needed to find a performance metric for all the incorrect gestures that were signed. Our goal was to find a performance metric that would give us an indication of accuracy. We chose a measure that would produce a value of 90 or above, out of 100, to indicate a gesture that did not need any more correction. However, if it was below 90, the program would indicate that the gesture needed further correction. Equation (7) shows the initial equation that was used to find the performance metric.

$$\text{Initial performance metric } = 100 - \frac{100 \times \text{rms error}}{\text{database width}} \tag{7}$$

The RMS error (3) and the database width both depend on the letter that is being signed. After evaluating the performance metric using the full database width, we found that the values of the performance metrics for the gestures that had high deviation, resulted in having high performance values. Then we decided to use half the database width but this resulted in the performance values being slightly low for the gestures that were considered correct. We finally decided on using three fourths of the database width, which gave us (8); our final performance metric equation.

$$\text{Performance metric} = 100 - \frac{100 \times \text{rms error}}{\text{database width} \times .75} \qquad (8)$$

Using (8), we calculated the performance metric for all the versions of the letters that were incorrectly signed, as well as their corrections, Figures 2.39-2.62. We tabulated the data and the results can be found in Table 2.11. From Table 2.11 we can see that after the last correction was made for all letters and versions, all performance metrics were indeed 90 or higher.

Table 2.11: Letter versions corrections and their corresponding performance metric

Figure	Letter version	Performance metric	Correction number and performance metric (P-M)		
			1st correction and P-M	2nd correction and P-M	3rd correction and P-M
39	a01	96.0	91.7	97.3	
40	a02	91.9	95.7	96.5	
41	a04	90.5	95.7		
42	a05	76.6	95.7	96.2	
43	b13	90.3	90.9		
44	b14	78.9	95.6		
45	b15	77.2	92.9	91.0	
46	e01	82.4	94.0	95.9	
47	e02	90.2	96.0		
48	e03	77.7	93.5	94.9	
49	L01	73.2	97.2		
50	L03	78.6	95.3		
51	L05	79.6	94.0	90.3	95.9
52	n01	89.7	79.3	83.3	93.3
53	n02	83.1	93.6		
54	n03	81.0	94.6		
55	n04	41.5	87.9	87.3	92.4
56	t01	87.9	95.7		
57	t02	67.9	90.6	97.1	
58	t03	49.4	90.1		
59	t05	84.9	94.4		
60	u01	78.8	85.6	88.8	91.6
61	u02	88.2	92.5		
62	u04	61.0	87.8	92.8	

2.16 Flow diagram of algorithm

A flowchart of the entire process of determining whether or not a static ASL gesture is correct and providing the user with instructions for correcting a gesture is shown in Figure 2.63. We begin the process by asking the user to place colored latex on each fingertip. The program then asks the user the letter that they would like to sign. A display of the database gesture chosen appears and the web camera is turned on. The user then tries to make the gesture as close to the database image as possible and presses enter to take a picture of their hand. The image is then enhanced using the decorrelation and linear contrast stretching method. The program then searches for all five colors in the image. If all five colors are not found, the program will ask the user to redo their gesture. If all five colors are found, the program continues with finding midpoints on each fingertip. Once the fingertips are found, the program then calculates the distances and angles between all the fingers as well as the centroid, etc. The user's centroid is then compared to the one from the database. If all distances are equal to or less than the threshold, of twelve pixels, the gesture will be considered correctly performed. The program will then show the database and use's gesture side by side and the program will be done. If the distances are greater than twelve pixels, the program will first let the user know which finger needs correction and how much movement is required in the x and y direction. The program will correct one finger at a time, whichever finger it finds with the highest amount of deviation off. The program will then display the database image, the user's image, and an image of the correction of the fingertip placement needing to be performed. The program will then return to the step where the program provides a display of the database image of the letter the user chose and the web camera will start up again to take a new image of the correction the program provided to the user. The process will then repeat until the distance difference of the user's image is less than the twelve pixel threshold that was set.

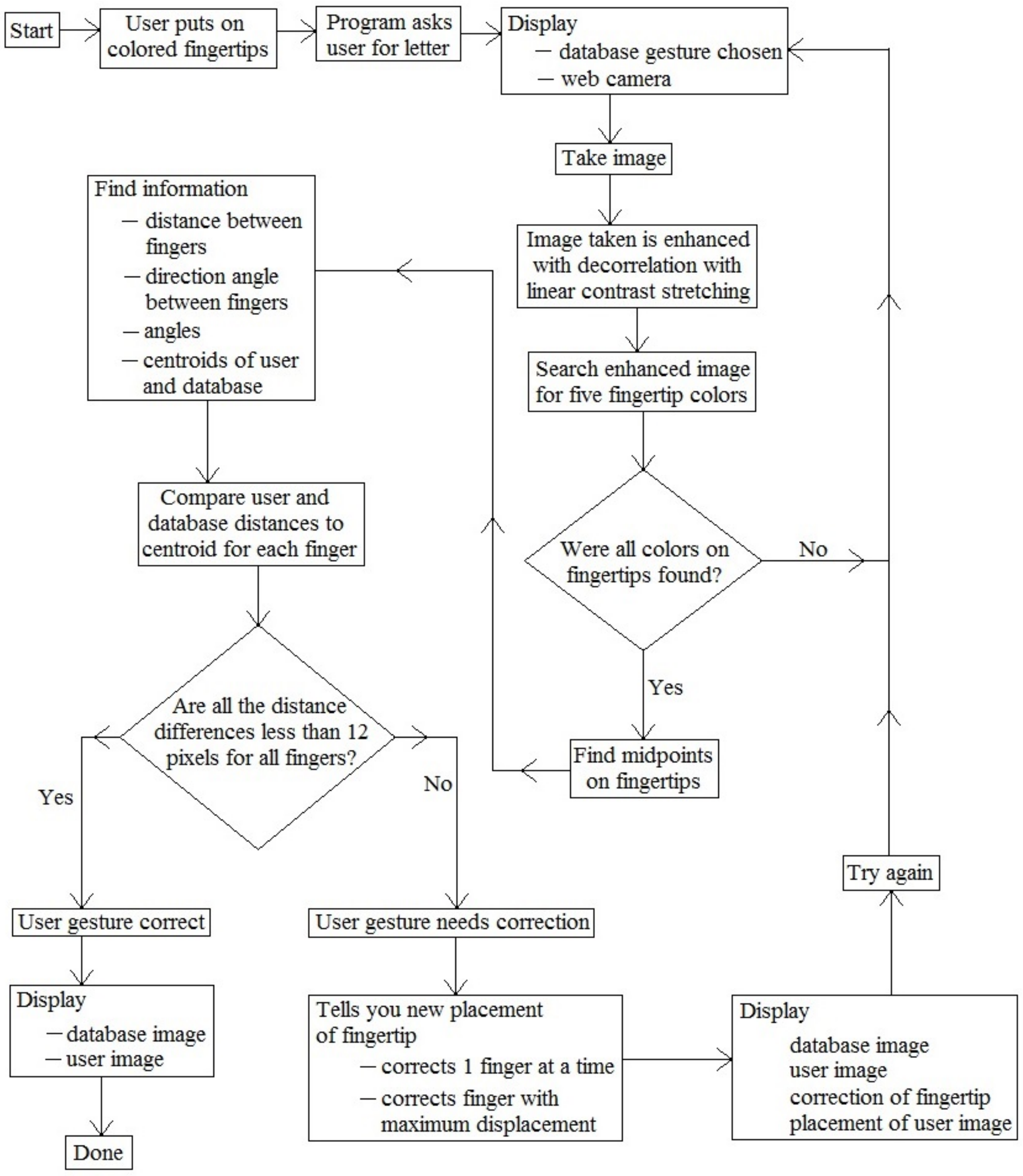

Start
User puts on colored fingertips
Program asks user for letter
Display
— database gesture chosen
— web camera
Take image
Image taken is enhanced with decorrelation with linear contrast stretching
Search enhanced image for five fingertip colors
Were all colors on fingertips found?
No
Yes
Find midpoints on fingertips
Find information
— distance between fingers
— direction angle between fingers
— angles
— centroids of user and database
Compare user and database distances to centroid for each finger
Are all the distance differences less than 12 pixels for all fingers?
Yes
No
User gesture correct
Display
— database image
— user image
Done
User gesture needs correction
Tells you new placement of fingertip
— corrects 1 finger at a time
— corrects finger with maximum displacement
Display
database image
user image
correction of fingertip placement of user image
Try again

Figure 2.63: Flow diagram of Algorithm

2.17 Participant testing

Four volunteers were recruited to perform testing of the ASL training program. Each participant completed a demographic data sheet and signed an informed consent form approved by the institutional review board (IRB) at Western Carolina University prior to participating in the study. Each participant was asked to sign two different letters of the American Sign Language. The participants were asked to sign both letters correctly in a shaded and unshaded environment. This was to set up a color range and to get scaling factors specific for the user in order to compare incorrectly performed gestures. The participant was then asked to perform the two letters incorrectly, three times per letter. The program then provided the user with instructions on how to correct each version of the letter that was performed incorrectly. Each participant was asked to follow directions as close as possible rather than correcting their gestures visually using the reference image. Each experiment lasted about an hour per person. The results of the participant testing are shown below.

2.17.1 Participant 1

Participant 1 was asked to sign the letter ’a’ and ’e’. In Figures 2.64-2.69 we can see all the incorrectly performed versions of each letters and the correction process for each as well as the performance metric associated with the correction being performed.

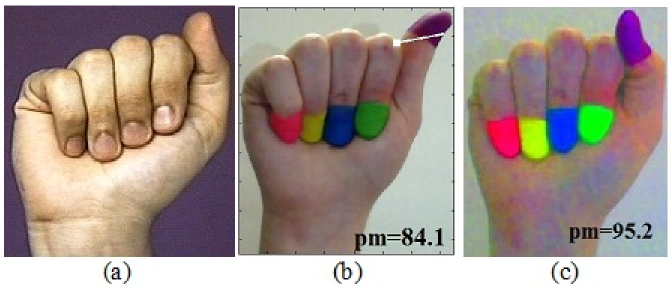


Figure 2.64: Correction process with performance metric of a01, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

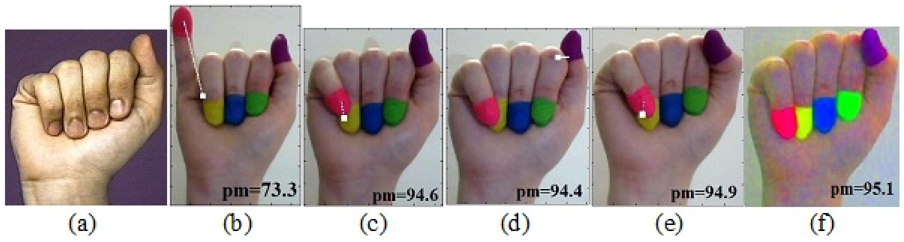


Figure 2.65: Correction process with performance metric of a02, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) 3rd correction needed, and performance metric, (e) 4th correction needed, and performance metric, (f) Corrected gesture and final performance metric

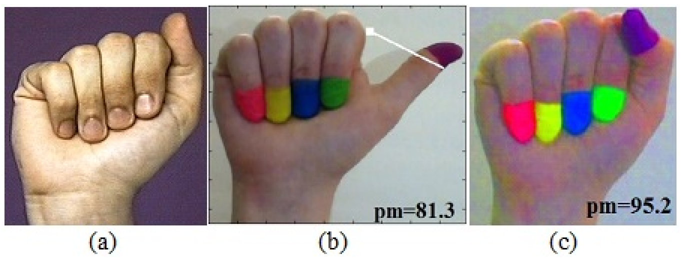


Figure 2.66: Correction process with performance metric of a03, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

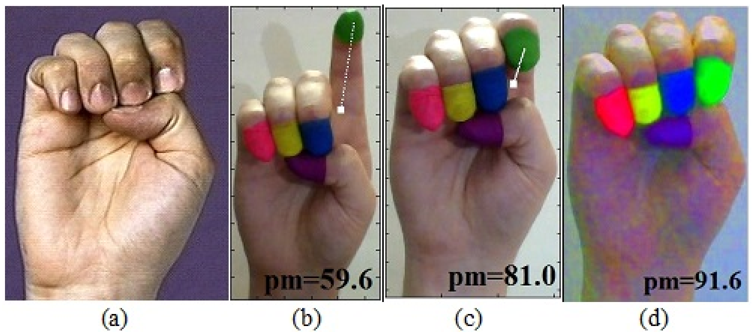


Figure 2.67: Correction process with performance metric of e01, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

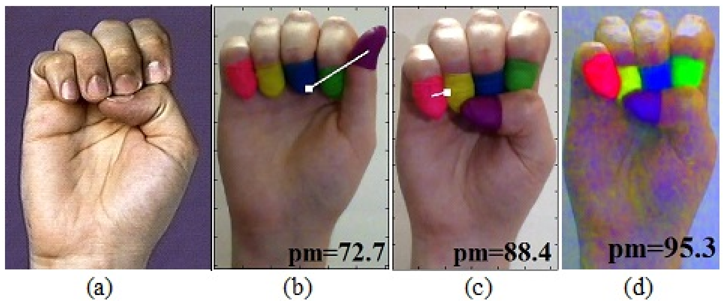


Figure 2.68: Correction process with performance metric of e02, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

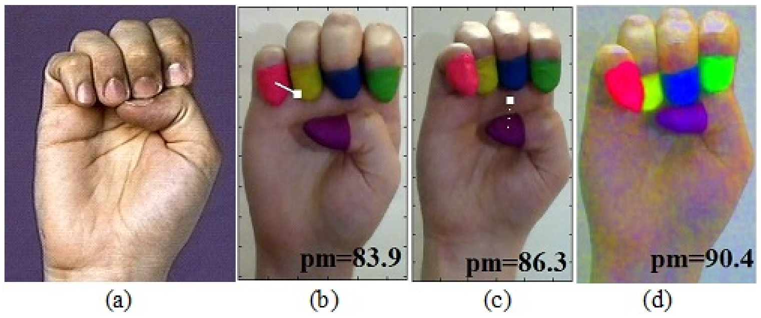


Figure 2.69: Correction process with performance metric of e03, Participant 1; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

2.17.2 Participant 2

Participant 2 was asked to sign the letter ’a’ and ’b’. In Figures 2.70-2.75 we can see all the incorrectly performed versions of each letters and the correction process for each as well as the performance metric associated with the correction being performed.

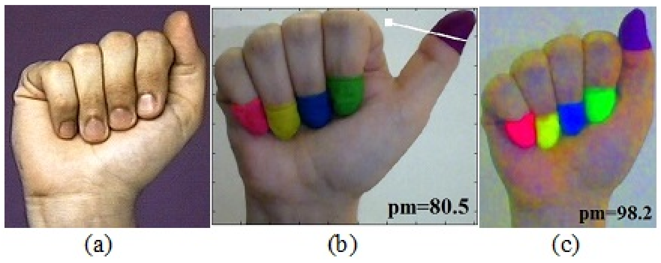


Figure 2.70: Correction process with performance metric of a01, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

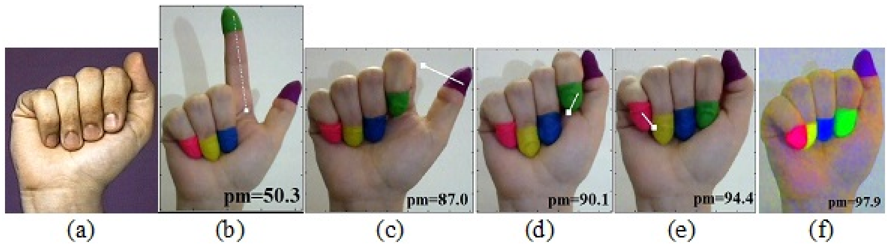


Figure 2.71: Correction process with performance metric of a02, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) 3rd correction needed, and performance metric, (e) 4th correction needed, and performance metric, (f) Corrected gesture and final performance metric

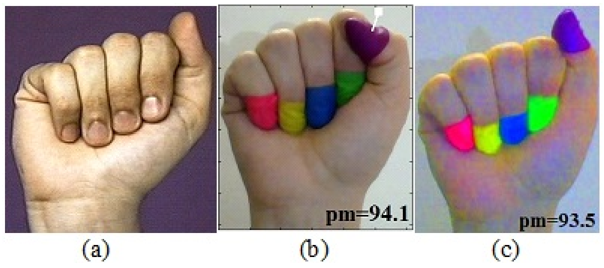


Figure 2.72: Correction process with performance metric of a03, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

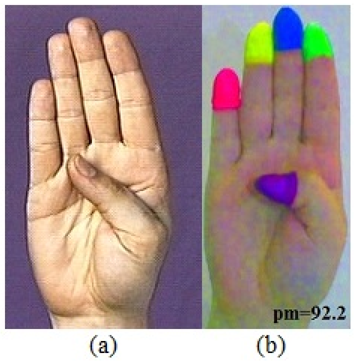


Figure 2.73: Correction process with performance metric of b01, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture and initial performance metric (no corrections were needed)

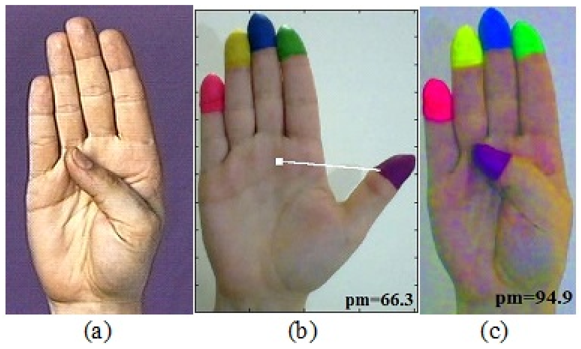


Figure 2.74: Correction process with performance metric of b02, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

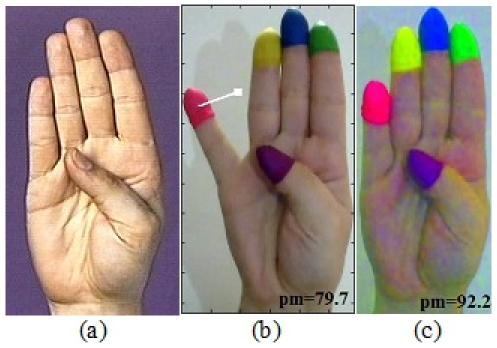


Figure 2.75: Correction process with performance metric of b03, Participant 2; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

2.17.3 Participant 3

Participant 3 was asked to sign the letter 'a' and 'L'. In Figures 2.76-2.81 we can see all the incorrectly performed versions of each letters and the correction process for each as well as the performance metric associated with the correction being performed.

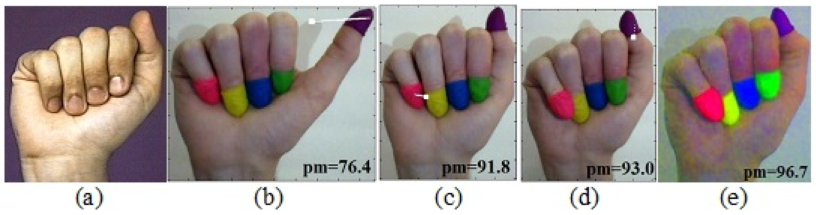


Figure 2.76: Correction process with performance metric of a01, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) 3rd correction needed, and performance metric, (e) Corrected gesture and final performance metric

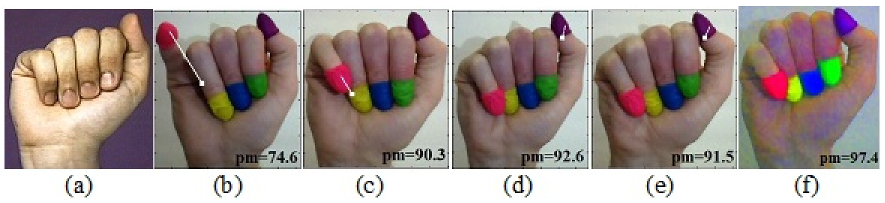


Figure 2.77: Correction process with performance metric of a02, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) 3rd correction needed, and performance metric, (e) 4th correction needed, and performance metric, (f) Corrected gesture and final performance metric

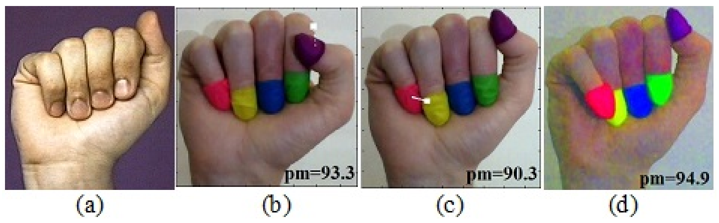


Figure 2.78: Correction process with performance metric of a03, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

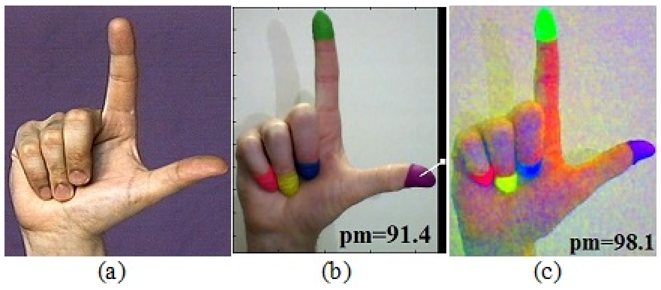


Figure 2.79: Correction process with performance metric of L01, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

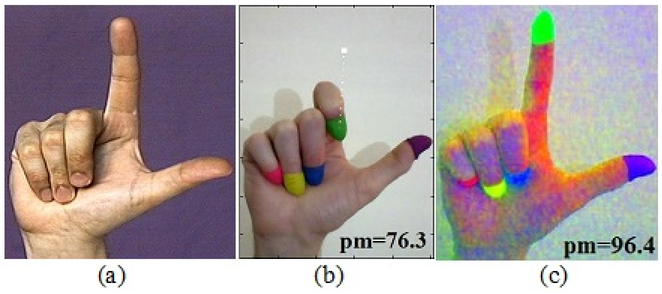


Figure 2.80: Correction process with performance metric of L02, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

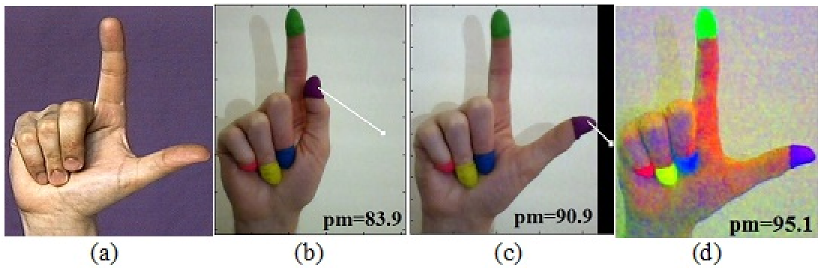


Figure 2.81: Correction process with performance metric of L03, Participant 3; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

2.17.4 Participant 4

Participant 4 was asked to sign the letter ’a’ and ’t’. In Figures 2.82-2.87 we can see all the incorrectly performed versions of each letters and the correction process for each as well as the performance metric associated with the correction being performed.

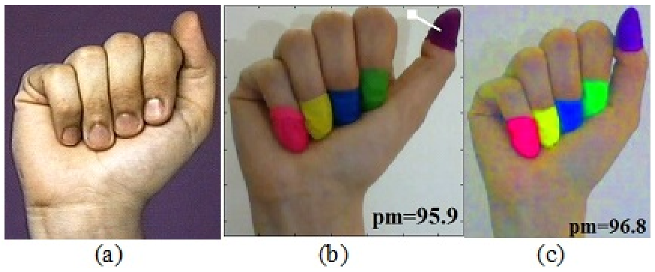


Figure 2.82: Correction process with performance metric of a01, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

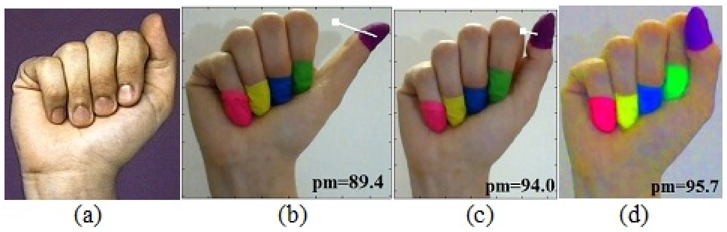


Figure 2.83: Correction process with performance metric of a02, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) 2nd correction needed, and performance metric, (d) Corrected gesture and final performance metric

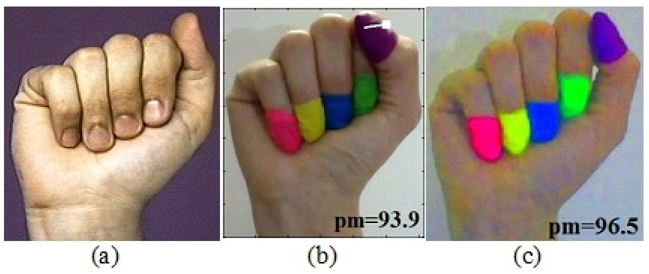


Figure 2.84: Correction process with performance metric of a03, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

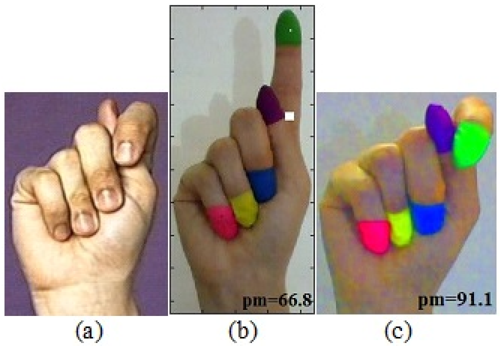


Figure 2.85: Correction process with performance metric of t01, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

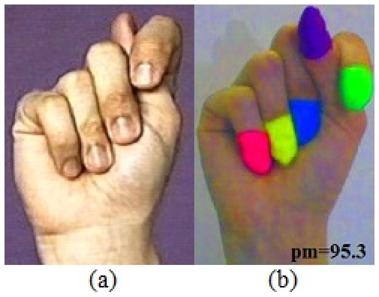


Figure 2.86: Correction process with performance metric of t02, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture and initial performance metric (no corrections were needed)

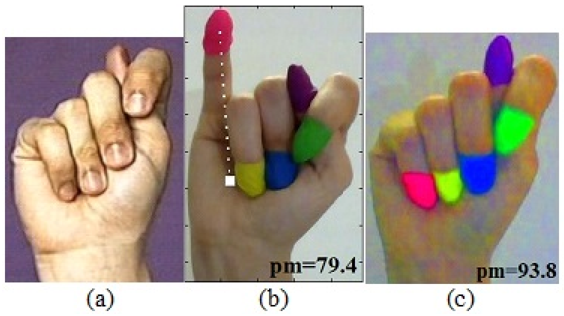


Figure 2.87: Correction process with performance metric of t03, Participant 4; (a) Database image, (b) Initial incorrectly performed gesture, 1st correction needed, and initial performance metric, (c) Corrected gesture and final performance metric

CHAPTER 3: RESULTS AND CONCLUSION

In this study we researched a method for developing a computer program to teach people American Sign Language. We developed and tested an interactive program that is able to detect correct placement of fingers for 12 static ASL gestures and provide the user with instructions in cases when the user's static ASL gesture is incorrect. We proved that our method of using colored fingertips combined with image enhancement techniques allowed for color recognition under "normal" indoor lighting conditions. Angle differences between user and database provided adequate data for identifying misplaced fingertips and determining when gestures were considered incorrect or correct. We also used this information to determine a threshold when comparing the user's gestures to the database images. We found that our method of comparing x and y displacement between each fingertip and the centroid of all 5 fingertip locations provided accurate measurements for providing visuals and descriptive information on how to correct the finger placements to match the database images. We developed a performance metric to provide the user with a measure of accuracy. We showed that our methods proved to be very accurate in providing corrections of all twelve static open hand gestures.

We tested our program on four different individuals who tested two different letters which were performed incorrectly three ways. The results of participant testing showed that the program correctly guided the user to perform all incorrect gestures close to the reference images. All final performance metrics obtained from the participant's gestures were 90 or higher. Although some of the corrections were very complex, the program was able to guide the user through them to correct the finger close to the reference images. This result, combined with visual verification, indicated that our program did correct all the gestures accurately.

It is important to note that this project will aid in helping with the placement of the fingers when learning to sign the alphabet. When in actual conversation with deaf speakers, one will need to take into account that the direction and placement of the hand, in context of the conversation itself [33]. When one expresses several gestures in sequence, the position does not necessarily have to be directly facing the person being addressed.

CHAPTER 4: FUTURE WORK

The ASL training program developed in this study is designed for use on static gestures in which all the fingertips are visible. Further work could focus on closed hand gestures in which not all of the fingertips are visible. In addition, research involving real-time could also be looked into. This will eliminate the need of taking multiple pictures of the user's hand when correcting the placement of the fingers. Future work could also focus on the training of dynamic gestures. This work could involve the use of data-gloves or image capture combined with image recognition and interpretation techniques. Considering sign language also consists of body and facial language as well, future work could involve the analysis of body, facial, and hand gestures together.

BIBLIOGRAPHY

[1] K. Ellis and J. C. Barca, "Exploring sensor gloves for teaching children sign language," *Advances in Human-Computer Interaction*, pp. 1–8, June 2012.

[2] K. Kadam et al., "American sign language interpreter," in *2012 IEEE Fourth International Conference on Technology for Education*, 2012, pp. 157–159.

[3] Y. N. Khan and S. A. Mehdi, "Sign language recognition using sensor gloves," in *Proceedings of the 9th International Conference on Neural Information Processing ICONIP 2002*, Lahore, 2002, pp. 2204–2206.

[4] T. Messer, "Static hand gesture recognition," M.s. thesis, Department of Informatics, University of Fribourg, Fribourg, Switzerland, 2009.

[5] A. Samir and M. Aboul-Ela, "Error detection and correction approach for arabic sign language recognition," in *Seventh International Conference on Computer Engineering and Systems*, Cairo, Egypt, 2012, pp. 117–123.

[6] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *Proceedings of the IEEE International Symposium on Computer Vision*, Coral Gables, FL, November 1995, pp. 265–270.

[7] N. Tanibata N. Shimada and Y. Shirai, "Extraction of hand features for recognition of sign language words," in *Proceedings International Conference on Vision Interface*, Osaka, Japan, 2002, pp. 391–398.

[8] N. Adamo-Villani J. Doublestein and Z. Martin, "The mathsigner: An interactive learning tool for american sign language," in *Eighth International Conference on Information Visualization Proceedings*, Lafayette, Indiana, July 2004, pp. 713–716.

[9] A. Licscar and T. Sziranyi, "Dynamic training of hand gesture recognition system," in *Proceedings of the 17th International Conference on Pattern Recognition, ICPR*, Hungary, 2004, pp. 971–974.

[10] C. T. Best et al., "Effects of sign language experience on categorical perception of dynamic asl pseudo-signs," *The Psychonomic Society*, vol. 72, no. 3, pp. 747–762, April 2010.

[11] G. Ashton et al., "Standards for learning american sign language," Tech. Rep., A Project of the American Sign Language Teachers Association, March 2011.

[12] R. A. Tennant and M. G. Brown, *The American Sign Language Hand Shape Dictionary*, Gallaudet University Press, 2nd edition, 2010.

[13] F. Arce and J. M. G. Valdez, "Accelerometer-based hand gesture recognition using artificial neural networks," in *Soft Computing for Intelligent Control and Mobile Robotics*, pp. 67–77. O. Castillo J. Kacprzyk and W. Pedrycz, Tijuana,Mexico, 2011.

[14] B. W. Min et al., "Hand gesture recognition using hidden markov models," in *IEEE International Conference on Computational Cybernetics and Simulation Systems*, Orlando, Fl, October 1997, pp. 4232–4235.

[15] J. Lapiak, "Handspeak: Sign language resource," `http://www.handspeak.com/`, 1996-2013.

[16] T. L. Humphries and C. A. Padden, *Learning American Sign Language: Levels I and II–Beginning and Intermediate*, Pearson Edu. Inc., New Jersey, 2nd edition, 2004.

[17] M. Geetha et al., "Gesture recognition for american sign language with polygon approximation," in *IEEE International Conference on Technology for Education (T4E)*, Chennai, Tamil Nadu, July 2011, pp. 241–245.

[18] M. Yang and N.Ahuja, “Recognizing hand gesture using motion trajectories,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999, pp. 466–472.

[19] W. T. Freeman and M. Roth, “Orientation histograms for hand gesture recognition,” *International Workshop on Automatic Face and Gesture Recognition*, pp. 296–301, June 1995.

[20] H. Zhou D. J. Lin and T. S. Huang, “Static hand gesture recognition based on local orientation histogram feature distribution model,” in *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, June 2004.

[21] S. Simon and K. Johnson, “Improving the efficacy of motion analysis as a clinical tool through artificial intelligence techniques,” in *Pediatric Gait: A New Millennium in Clinical Care and Motion Analysis Technology*, 2000, pp. 23–29.

[22] C. Vogler and D. Metaxas, “Parallel hidden markov models for american sign language recognition,” in *Proceedings of the International Conference on Computer Vision*, Kerkyra, Greece, September 1999, pp. 22–25.

[23] J. Yang et al., “Human action learning via hidden markov model,” in *IEEE Trans. on Systems, Man, and Cybernetics*, January 1997, pp. 34–44.

[24] M. Buckwald and D. Holz, “Leap motion,” `http://www.leapmotion.com`, 2013.

[25] “Digits: Hands-free 3-d from microsoft,” `http://research.microsoft.com/en-us/projects/digits/`, 2013.

[26] Thalmic Labs Inc., “Myo,” `https://www.thalmic.com/en/myo/`, 2013.

[27] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2001.

[28] M. H. Yang N. Ahuja and M. Tabb, "Extraction of 2d motion trajectories and its application to hand gesture recognition," in *IEEE Trans. Pattern Anal. Machine Intell.*, August 2002, pp. 1061–1074.

[29] W. Vicars, "American sign language," `http://lifeprint.com/`, 2012.

[30] B. Vicars, "Database approval [online]," Available e-mail: vyflamenco@gmail.com Message: Veronica Y. Flamenco/ Master's thesis, January 9, 2014.

[31] MATLAB version 7.8.0., "The math works inc.," `http://www.mathworks.com/discovery/image-enhancement.html`, 2013.

[32] MATLAB version 7.8.0., "The math works inc.," `http://www.mathworks.com/help/images/ref/edge.html`, 2013.

[33] Boinis et al., "Self-paced modules for educational interpreter skill development, ohio school for the deaf, 1996.," `http://www.ohioschoolforthedeaf.org/resources/3/Resources/MRID\%20Modules/MRID%20Self%20Paced%20Manuals%20-%20Fingerspelling.pdf`, August 28, 2013.

Appendices

APPENDIX A: DECORRELATION

A method of image enhancement already embedded into Matlab which was used is Decorrelation with Linear Stretching. This can be found in the Image Processing Toolbox. Here one can find many different methods and techniques that can be used when performing image enhancement. "Decorrelation stretching enhances the color separation of an image with significant band-band correlation" [31]. When viewing the enhanced images, this method provides exaggerated color which improves visual interpretation making the features on the image easier to distinguish. The number of color bands in this function are three (for Red, Green, Blue); but more color bands can be applied. The mean and the variance in each band remain the same. In this process, the original color values are mapped on to a new set of color values with a wider range. "The color intensities of each pixel are transformed into the color Eigen space of the NBANDS-by-NBANDS covariance or correlation matrix stretched to equalize the band variances, then transformed back to the original color bands" [31]. Linear contrast stretching (also known as Normalization) was added to the decorrelation method in order to expand the color range more. The same method can be applied alternatively using the MATLAB functions 'stretchlim' and 'imadjust' together. This however limits the pixel values in certain images with unsigned integers of X amount. The 'Tol' option bypasses this and is used in the 'decorrstretch' function. The 'Tol' chosen was .01; which meant that the transformed color range was mapped within each band to a normalized interval between .01 and .99, saturating it two percent. Increasing the 'Tol' gave the image too much saturation, causing the colors to blend into each other which in essence did the opposite and made it more difficult for the program to distinguish the set range for the colors being found. Decorrelation involves reducing the time lag between two signals when using autocorrelation or cross-correlation while still preserving other aspects of the signal [31]. Most of the decorrelation algorithms are linear but they can also be non-linear [31].

APPENDIX B: NEW MIDPOINTS USING LESS SHADING ON THE SHADED IMAGES

In section 2.7, we presented a method for detecting fingertips. We obtained a slight improvement in the results by re-taking the photographs of the shaded images. Recall that the color ranges are extracted from unshaded and shaded images and then divided into six different cases. By using a shaded image that is not so extremely shaded, this generated color ranges that better matched the colors of a typical user's photograph. The results using this modification are shown in Figures B.1-B.12.

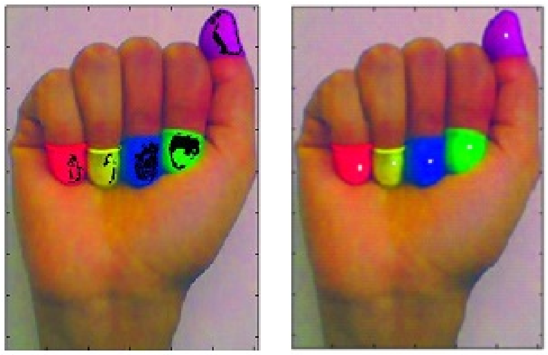


Figure B.1: Detection of fingertips for Letter ’a’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

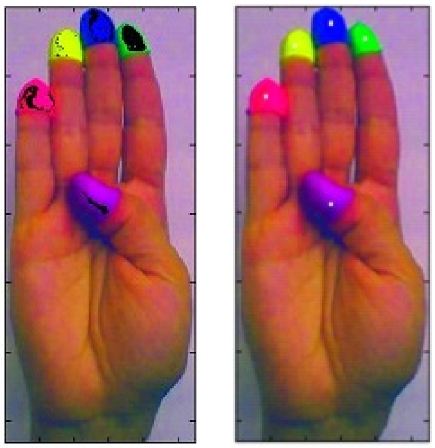


Figure B.2: Detection of fingertips for Letter ’b’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

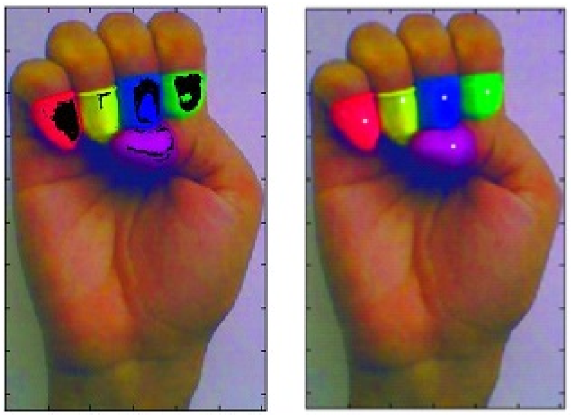


Figure B.3: Detection of fingertips for Letter ’e’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

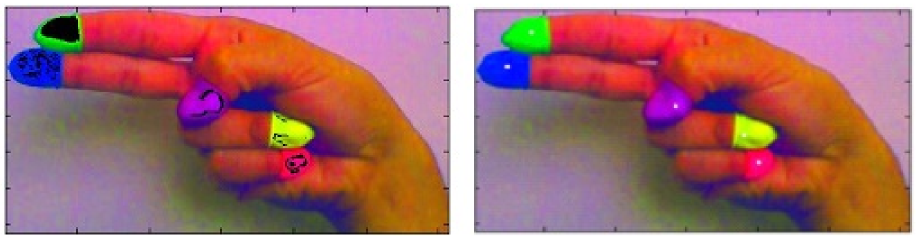


Figure B.4: Detection of fingertips for Letter ’h’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

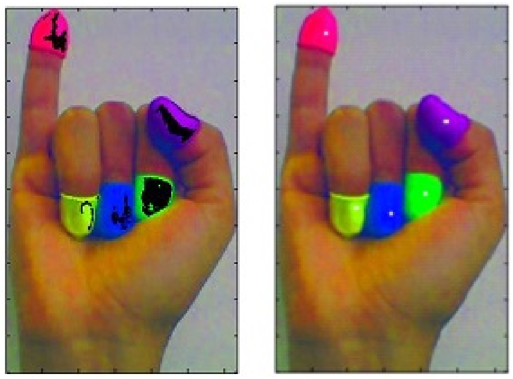


Figure B.5: Detection of fingertips for Letter ’i’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

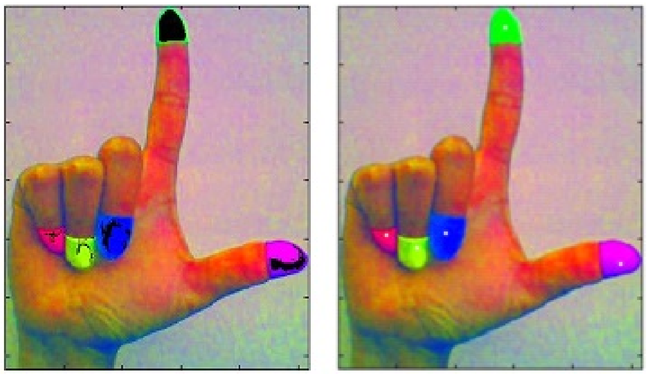


Figure B.6: Detection of fingertips for Letter ’L’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

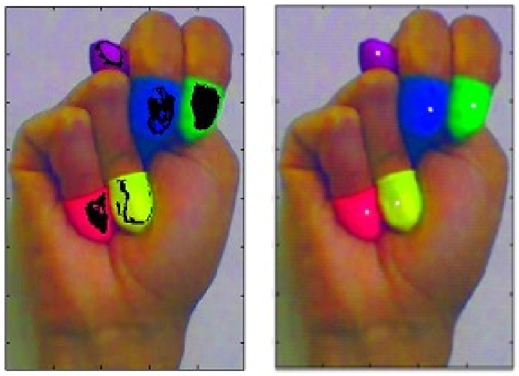


Figure B.7: Detection of fingertips for Letter ’n’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

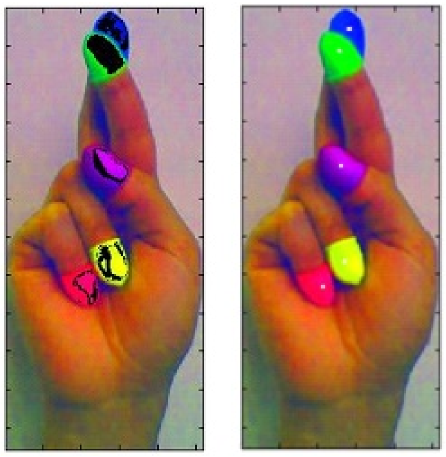


Figure B.8: Detection of fingertips for Letter ’r’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

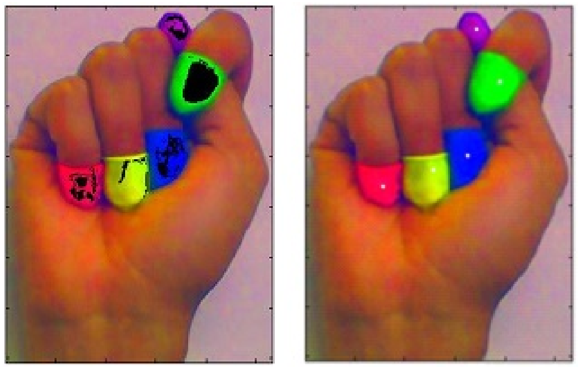


Figure B.9: Detection of fingertips for Letter ’t’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

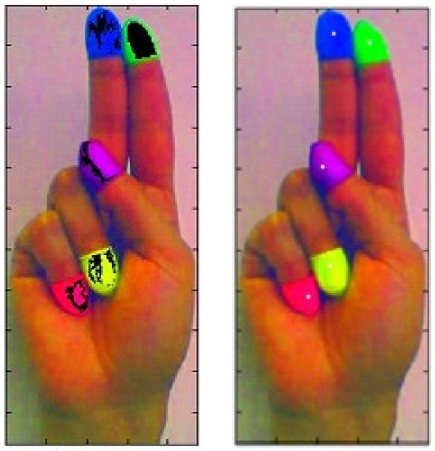


Figure B.10: Detection of fingertips for Letter ’u’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

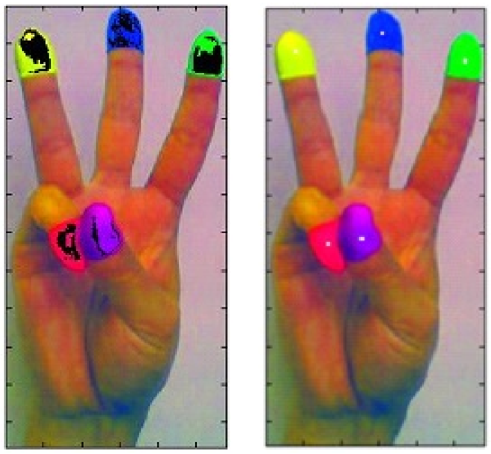


Figure B.11: Detection of fingertips for Letter ’w’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip

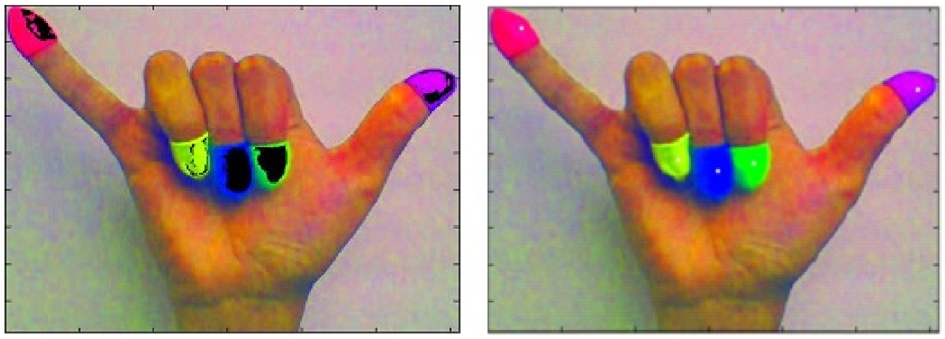


Figure B.12: Detection of fingertips for Letter ’y’. Right: Selected pixels for each fingertip; Left: averaged midpoint for each fingertip