EECE 401 Senior Design

Project Report

Sign Language to English Text (SLatE8)

A Prototype for Multi-Directional American Sign Language to Text and Speech to Text Translation.

Vanessa Galani
Michelle Warren

Advisor : Dr. Mohamed F. Chouikha

Department of Electrical & Computer Engineering

Howard University

April 20, 2016

Abstract

Our vision is to build a real-time, portable, cheap and intelligent cell phone application based multi-directional communication system which can successfully convert American Sign Language (ASL) to text and speech to text. We attempt to do this using image and video processing techniques namely: skin color detection and background elimination with NumPy and OpenCV, histogram of oriented vectors and support vector machines. In order to realize this vision, we need a comprehensive library of ASL.

The steps involved in realizing this vision were as follows: first we compiled a comprehensive ASL to English dictionary that we put on a remote database on a TCP socket based server. Next, we created and converted the images in usable data using skin detection, background elimination, and support vector machines and saved it on our server . We then do the same computation on images that we get from the user and find the best match in our dataset. The resulting word which compares more closely to the input image is then sent as output to the GUI of the communicating users' cell phones.

Currently, our program is able to successfully convert sign language to English text in a controlled environment for signing users who are wearing dark colors and long sleeve shirts. It is also only able to recognize the one word it is currently programmed to recognize: "WELCOME". This makes our recall ratio very high at 0.9, but reduces our precision to 0.000000975.

This system attempts to bridge the gap between close to 28 million people in the United States who are either deaf, mute or hearing impaired, and the rest of the population who can either speak or write English language.

Contents

Introduction

There are approximately 38,225,590 people in the United States who are either deaf or hard of hearing.[1] This number represents 13% of the population. There is an estimate of about 2,000,000 people who speak American Sign Language (ASL) in the United States[2]. In addition to the United States, ASL is spoken in most of Anglophone Canada. Also, ASL based dialects are spoken in several West African and Southeast Asian countries. The difficulty arises because most non-deaf people do not speak ASL, and this creates a communication barrier between the deaf and the non-deaf. However, both deaf and hearing people are able to read and write English, in the United States in particular.

Written English has been used as the communication bridge between the hearing and the hard of hearing or deaf. Unfortunately, the current state of the art is sufficient to bridge this gap effectively. People in the deaf and hard of hearing community are forced to use dictionaries or interpreters to communicate effectively with those who can hear. This is unfortunately either very costly with hiring an interpreter, or cumbersome with the sign language dictionary.

In an attempt to bridge this gap, we have designed a sign language to written English and speech to text translator which is cheap, portable and can do real time translation. The application is cell phone based and uses the camera on the user's cell phone. This application works on an algorithm that applies image processing techniques using NumPy and OpenCV, a connection oriented socket based server for fast, reliable and accurate computation.

Problem Statement

People in the deaf and mute community do not have a time efficient way of communicating with those who do not use ASL, thus obstructing and interfering with full integration into society. Our project is to build a portable device that converts ASL gestures to text and uses speech-to-text technology for successful communication between users and non-users of American Sign Language.

Current Status of Art

In 2012, for a capstone project, a group of engineering technology students and industrial design students designed the concept for and created a prototype for the My Voice device, which

[1] Harrington, Tom. "Deaf Statistics Tags: Deaf, Faq ." *Deaf Population of the U.S.* Gallaudet University Libraries, July 2014. Web. 19 Apr. 2016.

[2] Harrington, Tom. "Sign Language Tags: Deaf, Faq ." *ASL: Ranking and Number of Users*. Gallaudet University Libraries, May 2014. Web. 20 Apr. 2016.

takes in video input and translates it to audible words.[3] The conceptual design of My Voice is a handheld, portable device that uses a microphone, sound board, speaker, video camera, and monitor. When placed on a hard surface, it reads a user's sign language movements. Once My Voice processes the motions, it then translates sign language into an electronic voice that can be heard. It can also capture a person's voice and can translate words into sign language, which is projected on its monitor. Currently, this device can only translate one phrase. This design is a nice start to the world of translating sign language for non-ASL users. This design, however, is not very practical in that users have to worry about carrying and the possibility of forgetting another device on the daily basis.

The Microsoft Kinect Sign Language translator was developed by Microsoft China and uses the Kinect device that was originally created for Xbox360. The user stands in front of the camera on the Kinect device, which can detect the motion of anyone in front of the camera. On a screen, communication is realized by both parties, the sign language speaking constituent and their counterpart. Written and spoken translations of sign language are shown for the non-sign language speaking parties, while an avatar carries out gestures translated by the spoken words of the non-signing party.[4] This prototype is wonderful in that it bridges the gap between the two parties for fluid communication because of its capability to do real-time translations. The concept is not portable, however, which reduces the practicality of the system in that it can only be used where a computer, screen, and Kinect device are present, producing a very controlled and unrealistic situation.

From Texas A&M, a wearable device was created to translate sign language gestures. Although the device is in the prototype stage, it is already able to recognize 40 ASL gestures with about 96 percent accuracy. The inertial sensor in this device responds to motion, using an accelerometer and gyroscope to measure accelerations and angular velocities of the arm and hand. This sensor also uses the user's hand orientations to make accurate predictions. This design also uses an electromyographic sensor to measure the muscle activity, in order to be able to distinguish between gestures that may be similar in arm and hand movement but different in finger placement. The data from the device is sent via Bluetooth to a remote laptop to be computed through algorithms that interpret and translate these gestures into text.[5] This device uses a new approach, but fails to bridge the gap between communication between the ASL speaking community and the English speaking community. This device only provides one-sided understanding, which doesn't really benefit the ASL speaker but the English speaker.

---

[3] Emery, Mike. "UH Students Develop Prototype Device That Translates Sign Language". May 2012. Web. 20 Apr 2016. http://www.uh.edu/news-events/stories/2012/may/0529MyVoice.php .

[4] "Kinect Sign Language Translator expands communication possibilities". Web. 20 Apr 2016. Research.microsoft.com/en-us/collaboration/stories/kinectforsignlanguage_cs.pdf

[5] Garcia, Ryan. "New technology at Texas A&M could enable smart devices to recognize, interpret sign language". Aug 2015. Web. 20 Apr 2016. Engineering.tamu.edu/news/2015/08/20/slr-technology

There are of course other conventional ways to seek understanding that don't provide the speed that we would like to have. ASL dictionaries show pictorials of how to articulate different signs to ASL users, but don't prove to be as helpful for those on the other side of the situation. Flipping through pages to match a recently viewed gesture to the pictorials in a dictionary is not very efficient. The same is available as an application for Android and Apple platforms, but the same problems remain.

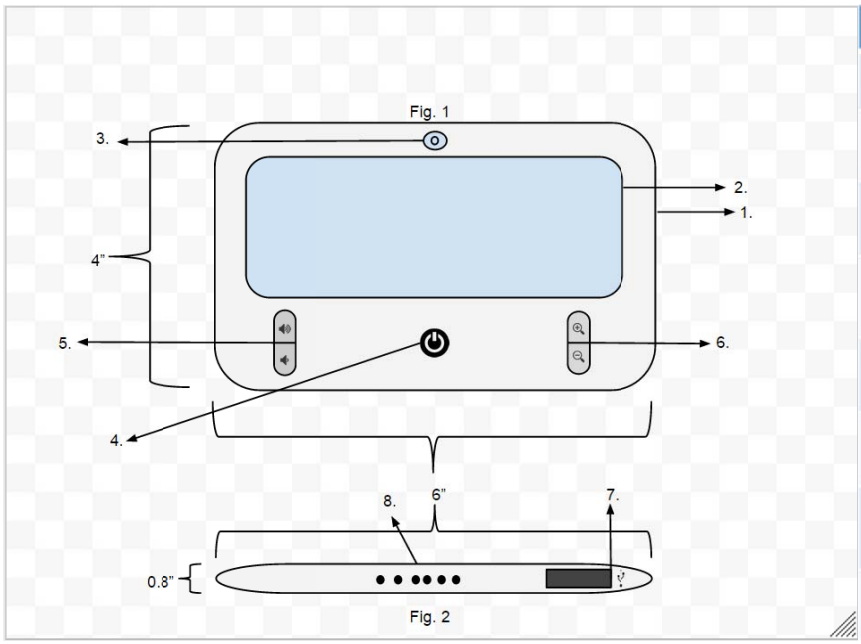
Design Requirements

Our design would be useless is it does not certain requirements and standards that allow it to serve its intended purpose. We want the communication between users of our system to be real-time, with as little delay as possible between the system input and output. This can be achieved if the delay between the signal and the written text is in the order of milliseconds. Also, in order to ensure reliability of translation, the system must be 100% precise (where precision is the ratio of the number of true positives to the number of translations). Furthermore, for a high accuracy in translation, the system is designed to have a recall ratio no less than 0.95(where the recall is the ratio of number of true positives to the number of translations). With the aforementioned precision and recall values, the system is expected to have an F1 score of at least 0.97 (where the F1 score is the harmonic mean of the precision and recall). The needs to capture enough frames from a given sign in order to correctly differentiate it from a similar sign. Our system ensures this by capturing input images at a rate of 30 frames per second. Additionally, our system needs to be portable and affordable, with each component costing less than $10, and the entire system with accessories weighing less than 0.25lbs.

Solution Generation

There were different possible approaches that we could have implemented in this design. Last year, the team came up with a different design. These two designs are similar in multiple ways, though distinctly different from each other. We did a comparative analysis of the two designs before selecting this year's team design.

Last year, the design included a portable device with a camera, as seen in Figure 1.0 below.



Parts
1 – Protective Case
2 - Screen
3 - Camera
4 – Power Button
5 – Volume Control
6 – Zoom Control
7- USB Port
8-Speakers

***Figure* □.□** Dhungel, Etienne, Kolloh, Yilma "Sign Language to English Text (SLatE) - An embedded prototype for translating American Sign Gesture to English Text". April 2015

This design includes an Intel DE2i-150 Atom board which did all the computation. The algorithm for this design is based in OpenCV and NumPy in Python. The algorithm converts the images to the HSV (Hue Saturation Value) color space and does a basic hand extraction. The image of the hand is then used to compute basic image parameters: the eccentricity, convexity, perimeter, area, convex hull, and these values are superimposed on values of stored words in the data set for best fit. The candidate word with the best comparative result is then chosen as the translation to the signed word. Also, last year's approach was to translate the ASL alphabet.

This year however, we viewed the problem differently. First and foremost, in a practical setting, a sign language speaker signs using gestures, rather than spelling out each letter of the word in question. We decided to focus on translating complete words rather than characters to make our design more practical and marketable. Also, we considered that most people signing would have a background behind them which would potentially skew the results of the translator of it is not able to do a background elimination. We computed our images in the YCrCb color space because this color space is luma independent, resulting in a better performance.[6] Furthermore, we used a connection oriented socket based server to reduce computation time from last year's predicted 3 seconds to the order of milliseconds. We also eliminated the necessity for the users to buy a component by implementing our system as a cell phone

[6] Basilio, Jorge, Gualberto Torres, Gabriel Perez, Karina Medina, and Hector Meana. "Explicit Image Detection Using YCbCr Space Color Model as Skin Detection." *Explicit Image Detection Using YCbCr Space Color Model as Skin Detection* (n.d.): n. pag. CEMATH Conference, 2011. Web.

application. There could be no additional weight, and the cost incurred will just be that of purchasing the application from the Apple Store or the Play Store.

Final Goals and Deliverables

By the end of the Spring 2016 semester, our goal was to have a working algorithm and application. The server, algorithm, and application are all in communication. As time progresses, the goal by May 2017 is to have an optimized system altogether. The algorithm will be enhanced in order to provide more accurate and faster results. Next, more data, taken from different application users and a multitude of tests, will be added to the data set in order to provide precise results in diverse and unique cases, the cases being diverse in skin tone, expression, and relative distance of hand to face. Lastly, our application will be available in the Apple Store.

Implementation, Testing, and Evaluation

We are using a socket-based server and iPhone application. Oracle describes a socket as "one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to".[7] Sockets allow for bidirectional travel of data. Rather than an HTTP server, a socket based server does not require internet connection, data, or Wi-Fi. For our purposes, transmission control protocol (TCP) sockets are more useful for our design in that they provide the flexibility of coding in the language of the programmer's choice and sending the exact data needed to send in order to improve proficiency. Lastly, our server can track what clients are using the system.[8]

Our application has three main views: A view for the client to declare their nickname in the server's chat room space (joinView), a view for the user to be able to input and view text and/or translations between the two users communicating with one another (chatView), and a view that shows the viewfinder of the camera (cameraView).

[7] "The Java Tutorials". *What is a Socket?* 2015. Web. 20 Apr 2016. https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[8] Rocchi, Cesare. "Networking Tutorial for iOS: How To Create A Socket Based iPhone App and Server". June 2011. Web. 20 Apr 2016. https://www.raywenderlich.com/3932/networking-tutorial-for-ios-how-to-create-a-socket-based-iphone-app-and-server

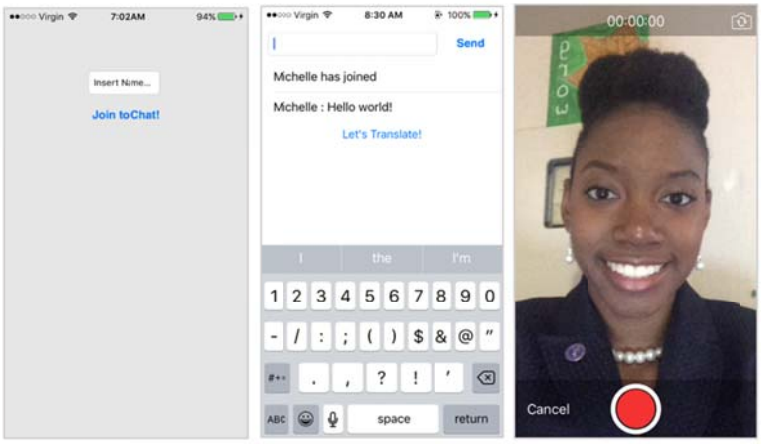


***Figure* □.□**: Screenshots of pages: joinView, chatView, and cameraView respectively

Currently, the application uses low amounts of storage, since all camera input is sent to the server for computation rather than being computed in the application itself.

When the data is received at the server, each image frame is converted to the YCrCb colour space.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112 \\ 112 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

We used the skin cluster at

$$Y > 80$$
$$85 < Cb < 135$$
$$135 < Cr < 180,$$
Where Y, Cb, Cr = [0, 255].

With this skin mask, we were able to successfully extract every skin color from a given frame. This skin mask however, because the color range is so large to accommodate every skin tone, picks up other colors, especially those with a high red color value as being skin color.

The next step in the computation is to track the hand and face and track the relative position and angle between the two. The assumption that the hand would be the smaller object in the image was made. When the hand is successfully detected, the properties of the image, namely the convex hull, the convexity defects, the eccentricity, the relative angle between the hand and the head, the area, the perimeter, and the aspect ratio of the image are computed. These values are then compared to values in our reference dataset. The dataset with which the subsequent images will be compared is reduced with each iteration for a given sign by elimination. If the input frame matches the the beginning of a set of signs, then only frames from those signs are kept for further comparisons in the next iterations. The set of plausible words are therefore reduced such that by the time the user is competing the word, there are only a few words left to compare the frames to. This technique increases the accuracy of our algorithm and reduces computation time and effort.

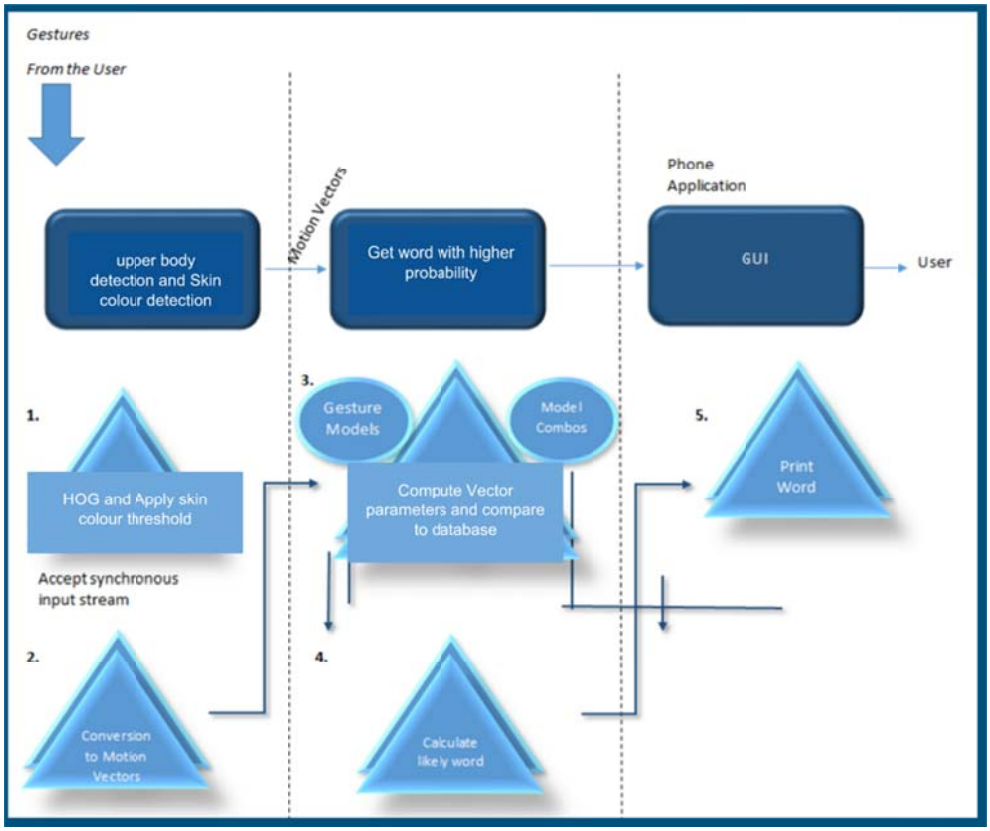


Figure 3.0. A visual representation of the algorithm.

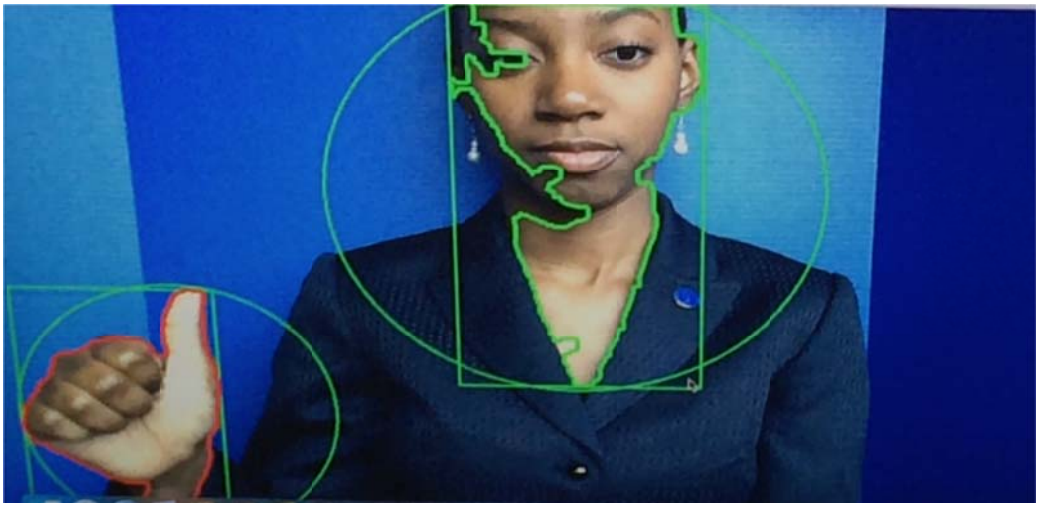


Figure 4.0 Hand and face tracking.

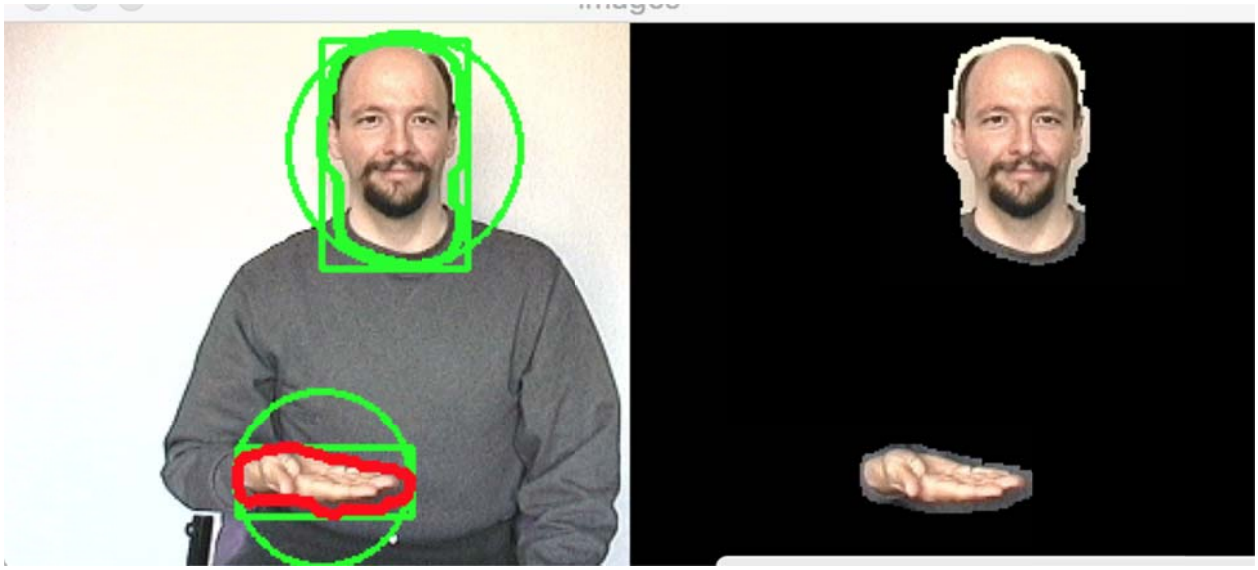


Figure 5.0. Skin colour detection, picture cutesy of http://www.lifeprint.com/asl101/

Thus far, our system has been trained only on one word, “WELCOME”. This greatly reduces the attainable precision, recall and F1 scores. Given that there are over 1025641 words in the English language and that our system is only trained to recognize one of those words, our performance is brought down to a recall ratio of 0.9 and precision as low as 0.000000975. These values are however not significant because our system is not yet sufficiently trained.

The system however meets the other design requirements: it is weightless at 0lbs, and the computation is done in the order of milliseconds. Besides, the total production cost is at $99 for a yearly subscription to Apple Developers, making it potentially cheap with an increased amount of users, and requiring no maintenance from the user.

Conclusion

Our system addresses the communication issue which has plagued and continues to plague the deaf and hard of hearing in our community. It doesn’t only translate ASL to English, but it provides multi-directional communication, making it suitable for the hearing, the hard of hearing and the deaf. Although our system is not yet fully trained, it has responded very positively to the little training that it has had, and produced the expected outcome. The implementation of this system using servers removes the problem of storage on the user's’ cell phone while simultaneously increasing computation speed, accuracy and reducing the likelihood of a complete system failure. This product will thrive in a market where the only other alternatives are either too expensive, not practical, or non-existent.

Future Works

The next steps in our design are as follows: first, compute the parameters of the remaining data in the training set. Next, establish successful communication between the cell phone application and the server. We will then go on to train the dataset, and build a classifier that can recognize and track hands and faces in an image. Finally, building the accessories and establishing communication between each accessory, the application and the server.

References

- Padden, Carol (2010), "Sign Language Geography", in Mathur, Gaurav; Napoli, Donna, *Deaf Around the World* (PDF), New York: Oxford University Press, pp. 19–37, ISBN 0199732531, retrieved November 25,2012
- Harrington, Tom. "Deaf Statistics Tags: Deaf, Faq ." *Deaf Population of the U.S.* Gallaudet University Libraries, July 2014. Web. 19 Apr. 2016.
- Harrington, Tom. "Sign Language Tags: Deaf, Faq ." *ASL: Ranking and Number of Users*. Gallaudet University Libraries, May 2014. Web. 20 Apr. 2016.
- Emery, Mike. “UH Students Develop Prototype Device That Translates Sign Language”. May 2012. Web. 20 Apr 2016. http://www.uh.edu/news-events/stories/2012/may/0529MyVoice.php .
- “Kinect Sign Language Translator expands communication possibilities”. Web. 20 Apr 2016. Research.microsoft.com/en-us/collaboration/stories/kinectforsignlanguage_cs.pdf
- Garcia, Ryan. “New technology at Texas A&M could enable smart devices to recognize, interpret sign language”. Aug 2015. Web. 20 Apr 2016. Engineering.tamu.edu/news/2015/08/20/slr-technology
- “The Java Tutorials”. *What is a Socket?* 2015. Web. 20 Apr 2016. https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
- Rocchi, Cesare. “Networking Tutorial for iOS: How To Create A Socket Based iPhone App and Server”. June 2011. Web. 20 Apr 2016. https://www.raywenderlich.com/3932/networking-tutorial-for-ios-how-to-create-a-socket-based-iphone-app-and-server
- Basilio, Jorge, Gualberto Torres, Gabriel Perez, Karina Medina, and Hector Meana. "Explicit Image Detection Using YCbCr Space Color Model as Skin Detection." *Explicit Image Detection Using YCbCr Space Color Model as Skin Detection* (n.d.): n. pag. CEMATH Conference, 2011. Web.
- “The Java Tutorials”. *What is a Socket?* 2015. Web. 20 Apr 2016. https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
- http://www.lifeprint.com/asl101/

- Dhungel, Etienne, Kolloh, Yilma “Sign Language to English Text (SLatE) - An embedded prototype for translating American Sign Gesture to English Text”. April 2015

Source Code Listing

Algorithm Implementation:

```
from __future__ import print_function
import argparse
from os import listdir
from imutils.object_detection import non_max_suppression
from imutils import paths
import time
import math
# imaging engine to use to process images
# import the necessary packages
ENGINE = 'opencv_engine'
from os.path import isfile, join
import numpy as np
import cv2
#from pyimagesearch import imutils
import argparse
import imutils

# Create a window to display the camera feed
cv2.namedWindow('Camera Output')



# define the upper and lower boundaries of the HSV pixel
# intensities to be considered 'skin'
lower = np.array([0, 48, 80], dtype = "uint8")
upper = np.array([20, 255, 255], dtype = "uint8")

Parameters = []
cnt2 = None

mypath='/Users/Galani/Desktop/africa'
onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]

def compare(Parameters, contour, onlyfiles):
```

```
        Welcome1 = [5.430795917066408, 0.5985512771635532, 2.0721788886407637,
0.8758479217091408, 0.8986834573554665, 63.856762487299818, 184.5670609832643]
        Welcome2 = [5.970398357341481, 0.5761633011413521, 2.2514900654859424,
0.89595229018531142, 0.8539362394274561, 48.682292722461796, 155.04837954651444]
        Welcome3 = [7.7432667864466005, 0.6516775396085741, 3.385763570031182,
0.95538777869037417, 0.8870916587377101, 39.454477941643056, 143.33875958721003]

        dic = {}


        for n in range(1, len(onlyfiles)):
                firstFrame = None
                max_index = 0
                scnd_max_index = 0
                cnt = None
                img = cv2.imread( join(mypath,onlyfiles[n]) )
                imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
                blur = cv2.GaussianBlur(imgray,(5,5),0)
                image = imutils.resize(img, width=min(400, img.shape[1]))
                orig = image.copy()
                if firstFrame is None:
                        firstFrame = imgray
                # continue
                converted = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
                skinMask = cv2.inRange(converted, lower, upper)
                kernel = np.ones((3,3),np.uint8)
                skinMask = cv2.dilate(skinMask, kernel, iterations = 2)
                skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)
                skin = cv2.bitwise_and(img, img, mask = skinMask)
                im_gray = cv2.cvtColor(skin,cv2.COLOR_BGR2GRAY)
                (thresh, im_bw) = cv2.threshold(im_gray, 128, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
                se = np.ones((12,12), dtype='uint8')  #Return a new array of given shape and type,
filled with ones. numpy.ones(shape, dtype=None, order='C')[source]
                thresh_open = cv2.morphologyEx(im_bw, cv2.MORPH_OPEN, se)
                thresh_close = cv2.morphologyEx(im_bw, cv2.MORPH_CLOSE, se)   #It is
useful in closing small holes inside the foreground objects, or small black points on the object
                im_bw =  cv2.dilate(im_bw,kernel, iterations = 3)
                contours, hierarchy =
cv2.findContours(thresh_close,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)   #Finds
```

contours in a binary image.retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.c demo.

```
areas = []
largest = 0
secondlargestcontour = 0
largestcontour = 0
for i, j in enumerate(contours):
    area = cv2.arcLength(j, True)
    areas.append(area)
    if area > largest:
        largest = area
        index = i
if (areas != []):
    new_areas = areas
    max_index = np.argmax(areas)
    cv2.drawContours(img,contours, max_index ,(0,255,0),3)
    cv2.imshow("binary", im_bw)
    cv2.imshow("images", np.hstack([img, skin]))
    print(max_index)
    cnt=contours[max_index]
    new_areas = areas[:max_index] + areas[max_index+1 :]
    second_max_index = np.argmax(new_areas)
    scnd_max_index = 0
    for i, j in enumerate (areas):
        if j == new_areas[second_max_index]:
            scnd_max_index = i
    min_area = areas[scnd_max_index]
    cnt2 = contours[scnd_max_index]

    dic.update({n: cv2.matchShapes(cnt2, contour,
CV_CONTOURS_MATCH_I1, 0.0)})

for key, value in d.iteritems():
    if min(dic.values()) == dic[key]:
        return i
```

```
cap = cv2.VideoCapture(0)
```

```
while( cap.isOpened() ) :
        firstFrame = None
        max_index = 0
        scnd_max_index = 0
        cnt = None
        ret,img = cap.read()

        imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(imgray,(5,5),0)
        # resize it to (1) reduce detection time
   # and (2) improve detection accuracy
        image = imutils.resize(img, width=min(400, img.shape[1]))
        orig = image.copy()
        if firstFrame is None:
                firstFrame = imgray
                # continue

        converted = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        skinMask = cv2.inRange(converted, lower, upper)

        kernel = np.ones((3,3),np.uint8)
        skinMask = cv2.dilate(skinMask, kernel, iterations = 2)


        # blur the mask to help remove noise, then apply the
        # mask to the frame

        skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)
        skin = cv2.bitwise_and(img, img, mask = skinMask)
        im_gray = cv2.cvtColor(skin,cv2.COLOR_BGR2GRAY)
        (thresh, im_bw) = cv2.threshold(im_gray, 128, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)




        se = np.ones((12,12), dtype='uint8')  #Return a new array of given shape and type, filled
with ones. numpy.ones(shape, dtype=None, order='C')[source]
        thresh_open = cv2.morphologyEx(im_bw, cv2.MORPH_OPEN, se)
        thresh_close = cv2.morphologyEx(im_bw, cv2.MORPH_CLOSE, se)   #It is useful in
closing small holes inside the foreground objects, or small black points on the object
```

```
        im_bw =  cv2.dilate(im_bw,kernel, iterations = 3)
        contours, hierarchy =
cv2.findContours(thresh_close,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)   #Finds
contours in a binary image.retrieves all of the contours and reconstructs a full hierarchy of nested
contours. This full hierarchy is built and shown in the OpenCV contours.c demo.
        areas = []
        largest = 0
        secondlargestcontour = 0
        largestcontour = 0
        for i, j in enumerate(contours):
                area = cv2.arcLength(j, True)
                areas.append(area)
                if area > largest:
                        largest = area
                        index = i
        if (areas != []):
                new_areas = areas
                max_index = np.argmax(areas)
                cv2.drawContours(img,contours, max_index ,(0,255,0),3)
                cv2.imshow("binary", im_bw)
                cv2.imshow("images", np.hstack([img, skin]))
                print(max_index)
                cnt=contours[max_index]
                new_areas = areas[:max_index] + areas[max_index+1 :]
                second_max_index = np.argmax(new_areas)
                scnd_max_index = 0
                for i, j in enumerate (areas):
                        if j == new_areas[second_max_index]:
                                scnd_max_index = i
                min_area = areas[scnd_max_index]
                # maximum_area = areas[max_index]
                # minimum_area = areas[max_index]/3
                cnt2 = contours[scnd_max_index]
                x,y,w,h = cv2.boundingRect(cnt)
                cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
                (p1,q1), radius1 = cv2.minEnclosingCircle(cnt)
                center1 = (int(p1), int(q1))
                radius1 = int (radius1)
                cv2.circle(img, center1, radius1, (0, 255, 0), 2)
                a,b,c,d = cv2.boundingRect(cnt2)
```

```
                cv2.rectangle(img,(a,b),(a+c,b+d),(0,255,0),2)
                (p2,q2), radius2 = cv2.minEnclosingCircle(cnt2)
                center2 = (int(p2), int(q2))
                radius2 = int (radius2)
                cv2.circle(img, center2, radius2, (0, 255, 0), 2)
                cent1 = np.array(center1)
                cent2 = np.array(center2)
                distance = np.linalg.norm(cent1-cent2)
                print("distance = ", distance)
                # compute the absolute difference between the current frame and first frame
                frameDelta = cv2.absdiff(firstFrame, imgray)
                thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]
                # dilate the thresholded image to fill in holes, then find contours
                # on thresholded image
                thresh = cv2.dilate(thresh, None, iterations=2)
                (cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
                # loop over the contours
                c = None
                for c in cnts:
                # if the contour is too small, ignore it
                        if cv2.contourArea(c) < args["min_area"]:
                                continue
                # compute the bounding box for the contour, draw it on the frame,
                # and update the text
                if c != None:
                        (r, s, t, u) = cv2.boundingRect(c)
                        cv2.rectangle(frame, (r, s), (r + t, s + u), (0, 255, 0), 2)
                dotProduct = sum((a*b) for a, b in zip(center1, center2))
                cos = dotProduct/ ( np.linalg.norm(cent2)* np.linalg.norm(cent1))
                angle = np.arccos(cos)
                angle = angle * (180 / ((22/7)))
                print("cosine = ", cos)
                print("angle = ", angle)

                m = cv2.moments(cnt2)
                Area          = m['m00']
                Perimeter     = cv2.arcLength(cnt2,True)
                BoundingBox   = cv2.boundingRect(cnt2)
                Centroid      = ( m['m10']/m['m00'],m['m01']/m['m00'] )
                EquivDiameter = np.sqrt(4*Area/np.pi)
```

```
                Extent       = Area/(BoundingBox[2]*BoundingBox[3])
                ConvexHull    = cv2.convexHull(cnt2)
                ConvexArea    = cv2.contourArea(ConvexHull)
                Solidity      = Area/ConvexArea
                centre,axes,angle2 = cv2.fitEllipse(cnt2)
                MAJ = np.argmax(axes)
                MIN = 1-MAJ
                MajorAxisLength = axes[MAJ]
                MinorAxisLength = axes[MIN]
                ratio = MajorAxisLength/MinorAxisLength
                Eccentricity    = np.sqrt(1-(axes[MIN]/axes[MAJ])**2)
                Orientation     = angle2
                EllipseCentre   = centre
                Parameters =  [Perimeter/MinorAxisLength, Extent, ratio, Eccentricity, Solidity,
angle, distance]
                print("Parameters " ,Parameters)




        cv2.waitKey(100/30)

    # if the 'q' key is pressed, stop the loop
        if cv2.waitKey(1) & 0xFF == ord("q"):
                break

        cv2.drawContours(img,contours, max_index ,(0,255,0),3)
        cv2.drawContours(img,contours, scnd_max_index ,(0,0,255),3)
        #c = contours[ind]


        # show the skin in the image along with the mask
        cv2.imshow("binary", im_bw)
        # cv2.imshow("contours", cnt)

        cv2.imshow("images", np.hstack([img, skin]))
        cv2.waitKey(100/30)

    # if the 'q' key is pressed, stop the loop
```

```
        if cv2.waitKey(1) & 0xFF == ord("q"):
                break
        # compare(Parameters, cnt2, onlyfiles)

        Welcome1 = [5.430795917066408, 0.5985512771635532, 2.0721788886407637,
0.87584979217091408, 0.8986834573554665, 63.856762487299818, 184.5670609832643]
        Welcome2 = [5.970398357341481, 0.5761633011413521, 2.2514900654859424,
0.89595229018531142, 0.8539362394274561, 48.682292722461796, 155.04837954651444]
        Welcome3 = [7.7432667864466005, 0.6516775396085741, 3.385763570031182,
0.95538777869037417, 0.8870916587377101, 39.454477941643056, 143.33875958721003]
        answer = []
        for i, j in enumerate(Parameters):
                found = [abs(Welcome1[i]-Parameters[i]), abs(Welcome2[i] - Parameters[i])]
                if min(found) == abs(Welcome1[i]-Parameters[i]):
                        answer.append(1)
                elif min(found) == abs(Welcome2[i]-Parameters[i]):
                        answer.append(2)
                # elif min(found) == abs(Welcome1[i]-Parameters[i]):
                #         answer.append(3)

        if all(answer[i] <= answer[i+1] for i in xrange(len(answer)-1)):
                print("the Sign says  WELCOME!", )
        print (answer)



cv2.destroyAllWindows()
system.exit()
```