ASL Detection
for
Practical Implementation

—

24th Annual Supercomputing Challenge
2013-2014

FINAL REPORT
Team №1

Team Members
Noah Caulfield, Academy for Technology and the Classics
Dmitri Malygin-Voorhees, Santa Fe High School

Sponsoring Teacher
Brady Gotcher, Academy for Technology and the Classics

Running tittle: ASL Detection for Real World Implementation

Table of Contents

Executive Summary

Deafness is a widespread problem that affects hundreds of thousands of Americans, restricting their access to basic means of communication and expression. Hundreds of thousands more are considered 'functionally deaf,' meaning that their hearing is impaired and they require assistance though the means of hearing aids and other devices whose high price tags could potentially limit their accessibility. Hearing is one of the body's five core senses and the primary means of conscious social interaction; individuals who can't process audible communication face great difficulty in interfacing with the outside world.

An attempt to alleviate some of this difficulty was the central aim of our project for the 24th Annual Supercomputing Challenge, in which we investigated whether it is feasible to have conventionally built computers and devices recognize American Sign Language (ASL) fingerspelling gestures and use them as input, an area which has received little to no real research in the past. Although there are a number of consumer solutions for those who rely on ASL, many of them are overly expensive or impractical to be considered adequately accessible. Our project was imagined with the core concept of it being a catalyst, so that others, even outside the Challenge, may view and build off of our work so that in the future a true method is successfully implemented for the benefit of the posterity.

We, Team No. 1, looked to utilize generic web cameras and the Xbox 360 Kinect sensor, connected to a computer, to investigate our selected methods of recognition after eliminating other options from a list of other means of creating accurate gestural recognition. Much ardor was put into tweaking on the Kinect via the Processing language, yet we finally deemed that there were too many core library issues at the time of development to be able to produce a finalized prototype -- although this would be entirely possible in the foreseeable future. Python was the other language implemented in the design and construction of our other prototype, a real time contour comparison system that receives users' gestural input via web camera and compares it to our own set of previously compiled images to determine peak similarities, which determines the actual gesture with a negligible margin of error when used in the correct environment. We relied heavily on the OpenCV library, which deals mainly with computer vision, to define a method of comparison to implement and to optimize the project in its entirety.

1. Introduction

American Sign Language, or ASL for short, was introduced in the United States nearly 200 years ago; today, and it serves as the most common means of face-to-face communication for the deaf of hearing impaired. These groups, however, have difficulty making use of technological resources without having the mastery of English that most of America takes for granted. Factually, computers and similar devices are not intuitive to the hearing impaired, and accessibility devices can be inconvenient, expensive, or both.

Through the last several decades, there have been many advancements in the areas of user accessibility and gestural recognition including the concept and now popular trend of wearable technology, yet the question of true user accessibility and functionality is constant. A device is only innovative if able to be successfully implemented among a significant group.

Our project follows the path of early speech recognition technology. With previous attempts to abet gestural recognition in computers, depth has been a constant factor within the camera's input, which limits processing and accuracy. We spent much time working on isolating the ASL fingerspelling alphabet, which consists of individual (usually static) signs representing the 26 Arabic letters used in English, as well as several integers and special-case devices. ASL fingerspelling was chosen due to serving as the foundation of the language, which made it the natural choice.

Other forms of past designs of ASL recognition have relied on instrumented gloves or a desktop based camera system, such as thermal cameras, to have the highest precision rates. After investigating this trend, we noticed that it could be deducted that many of these options, although some chosen for being case/design specific, were broadly chosen due to being of a high resolution which lowered the background visual noise. From that, we decided to attempt to have our program be, in theory, universally usable, which has impacted some of our coding techniques and hardware testing/choosing.

In its core, our project is to simply ask the question, is American Sign Language detection and implementation possible at a true consumer level, and if so, what is a valid method for doing such? If achieved, our program would serve as a basis for further implementation within consumer level operating systems in an attempt to create true user accessibility- disabled or not.

2. Background

2.1 American Sign Language Introduction

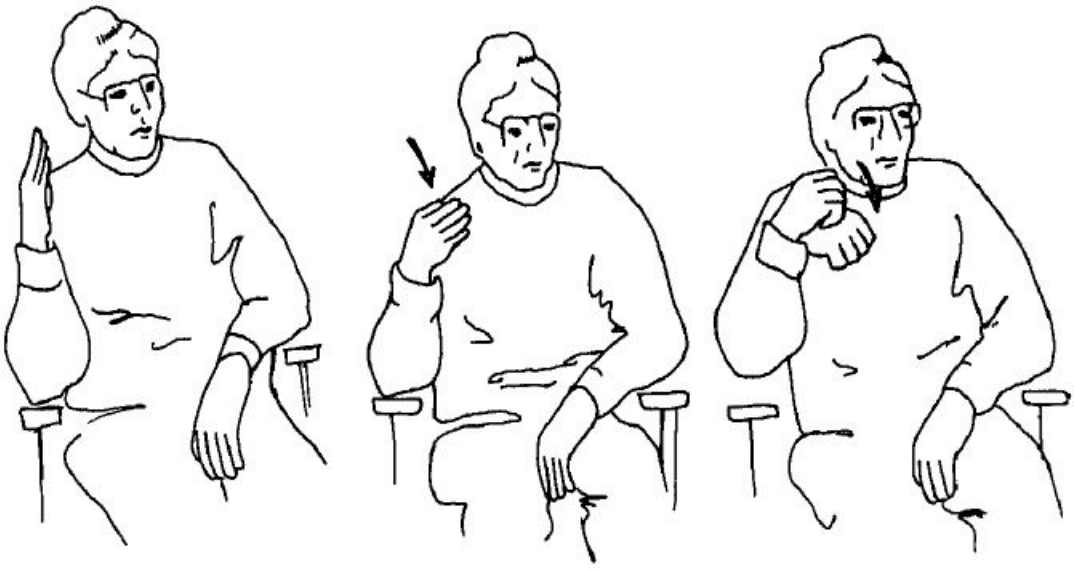
Sign Language was first introduced in America mainly by Thomas Hopkins Gallaudet, a Congressional minister whom originally sought a method of communication for his deaf neighbor. In 1815 he actually traveled to England to study the current methods of sign language, which were much different from today's standards, and returned with modified versions to meet America's reign specific colloquializations. He even then founded the nation's first school for the deaf in 1817, and he became our nation's first sign language teacher. Since then, sign language within the United States, specifically called American Sign Language or ASL for short, has become a uniform standard for which millions use daily.

The first concept to understand American Sign Language the Sign Space, which is an area of which most signs should be made for formal and informal conversations. Generally this space is centered to the left or right of one's middle torso, and is a rather natural feeling to the user. Main spelling is completed with the user's dominant hand, and generally one should not switch hands during simplistic signing or a conversation.

Figure 1. *Illustration of stereotypical personal gestural space.*

As of 1970, there was a record of over five hundred signs alone being used in everyday American Sign Language Usage. That number has steadily increased since then, yet an estimate has yet to be determined since.

Another important factor to remember about utilizing ASL is that the force and momentum of the gestures within sequence can indicate emotion, questioning, and even regional traits.



(1) [an' he ran a]cross
Right hand in B-shape moves up at right side, hand in front of face at eye level, palm toward center; holds this.
(2) [and he 'lectrocuted]
Right hand, in same shape and space, chops straight down to the front; holds this space.
(Experimenter: uh-huh how did that happen?)
(3) well by [pullin' the-] the sound-the signal
Right hand, still in right space, changes to loose A-shape for gripping the bell cord, and pulls down. Goes to lap.
(Experimenter: uh-huh and then? If you pull the signal, then what would happen?)
(4) n' [he fell off]

Figure 2. *Illustration of ASL conversation translation with movement.*

Due to there being such a broad library of American Sign Language Gestures, we determined that it was rather impractical to attempt to recognize all of them. As with most other traditional languages, their basis is within their alphabet, and all words/ phrases are derived from it. Fingerspelling, also known as the ASL alphabet, seemed like the natural solution to begin our

work with. We realized that due to the vast technological and programming steps necessary to determine and complete a working basis for such specific recognition, we may not be able to recognize all fingerspelling gestures, but we personally valued a working basis for further work.

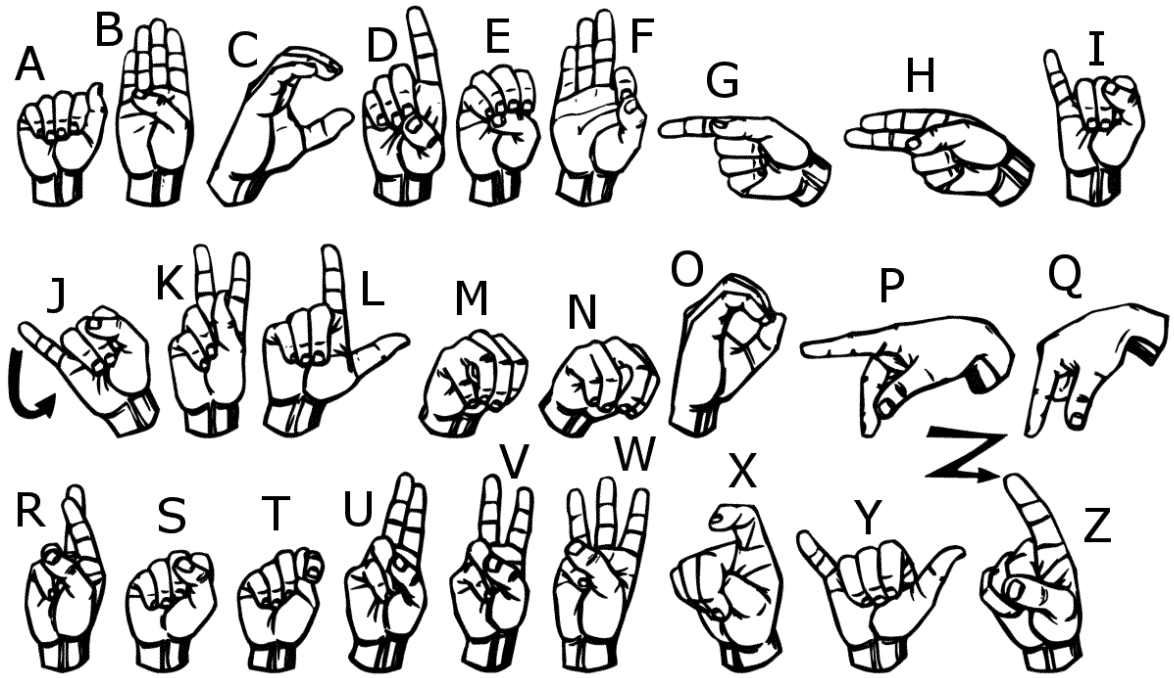


Figure 3. *Visualization of American Sign Language Alphabet.*

Another aspect we were determined to overcome, specifically within our Python build, was the aspect of moving gestures within the library, specifically J and Z, and visually similar gestures, such as M and N or U and V. These conflicts are further discussed and resolved within development.

2.2 Project Objectives

To summarize our project, we asked the Question, “Is American Sign Language detection currently possible for real world consumer level implementation?”, and we were successful in answering it by maintaining primary and secondary goals as displayed below.

PRIMARY	SECONDARY
• Define list of possible recognition techniques and determine best specifically concerning ASL. • Develop prototypes to test whether means would be successful or not. • Choose one final method to fully design and program.	• Study the fundamental concepts of ASL, specifically those necessary to incorporate. • Define regional differences within ASL to determine a standard to be used via models library/recognition technique. • Contact and work with NM School for the Deaf while creating models library (Technique dependent). • Take note of consumer electronics intake while determining hardware choices. • Optimize the program/build to be textually minimalistic to avoid slow performance or runtime errors.

3. Processing Development

3.1 Why Kinect?

Microsoft's Xbox 360 adaptor, the Kinect, was released on November 4th, 2010, and was marketed a motion adaptor that could interact with the user without the need of a physical controller. It was also produced to broaden the Xbox 360's audience beyond the typical gamer base. As of march, 2011, Microsoft announced that the sensor had already sold 10 million units, which is quite a notable feat for an adaptor with an MSRP of $149.99. The key hardware components to note are a RGB camera, depth sensor, multi-array microphones, a tilt monitor, and even a three-axis accelerometer for mechanical motion control.

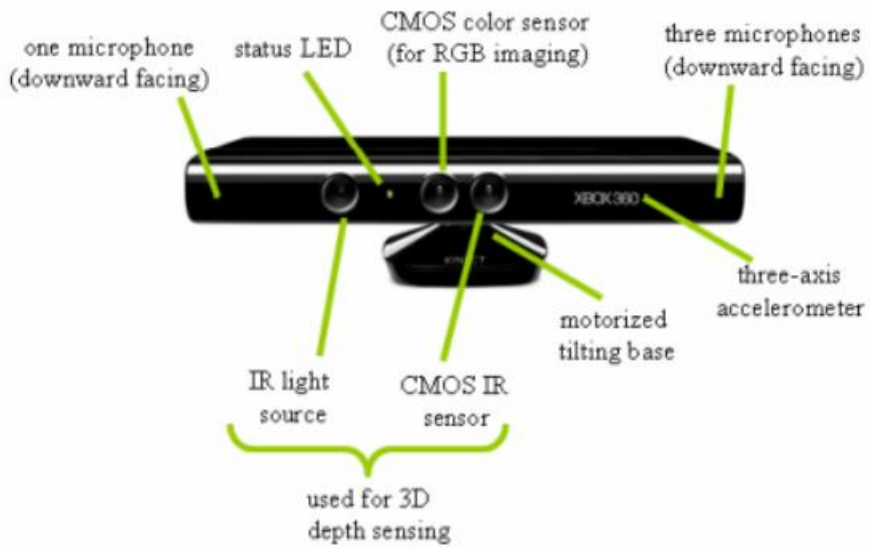


Figure 4. *Diagram of Kinect's components.*

At time of release, there was no open development kit, even for console development, yet within literally hours of its initial availability many open-source options arose, until Microsoft released the Kinect SDK for official development mainly within C, a language neither member was familiar with. The most notable libraries are OpenNI and NITE, OpenKinect, and CLNUI. By the winter of 2013 most of these libraries were either in the process of, or already ported to the Processing language, making them appear to be a perfect choice, MAC or PC. We chose to

use the CLNUI library, as the original library, before being ported, was actually distributed via Kinect's original manufacturer, Prime Sense. The Processing language is also designed primarily for visualizations, specifically in instances where user input via GUI is not entirely necessary. Along with many fantastic features, Processing also comes with fantastic exportation feature to have a project work on a number of devices- even Android, something we would like to see development on in the future for ASL. Although a bold decision to work with such hardware/software dependent on a rather underdeveloped library, we saw it as a solid possibility.

3.2 Method

Most sensors of the 21st century use either the Charged Coupled Device (CCD) or the Complementary Metal-oxide-semiconductor (CMOS) technology due to the vivacity and color clarity offered. There are several monocular and binocular hues for depth sensory along with infrared technologies in common practice, yet there currently stands no universally precise method for depth interpretation visually for machine vision/learning.

The Kinect, with its infrared sensor, delivers a 640*480 depth-map static image via live feed that produces not only a relatively accurate depth sensor, but also background detection and extraction, and even blob detection for moving aspects, more specifically the user.

Since we are dealing with specifically the hands of the user, hand extraction had to be implemented, and the following concept was used due to having previous success shown within sources:

- *Hand and wrist detection and extraction*
- *Hand isolation from wrist via X and Y axis from creating projections*
- *Filtering to remove extra pixels, blank areas/distortions, and perimeter*
- *When completed, image was smoothed to perform trace search with an averaging lenses, and removing layer due to enlargement*

- *Search algorithm used for redefinition of perimeter*

- *Features of hand defined by "Seeing" the angle between three points on the smoothed trace to determine if a threshold requirement was met*

- *Neural network designed to take and classify inputs as static ASL hand gesture library*

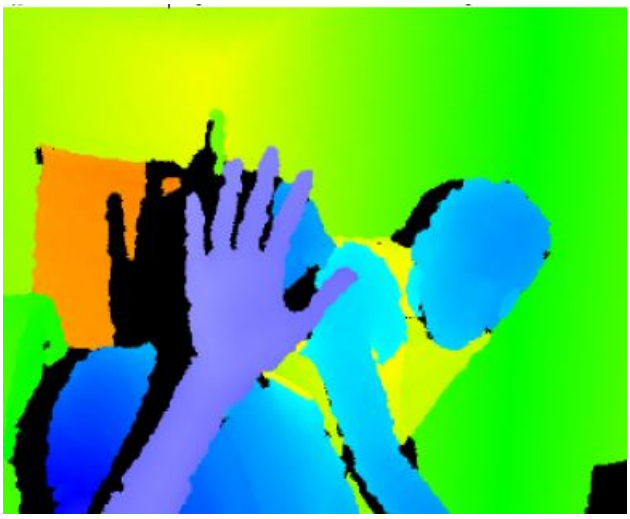


Figure 5. *Color Visualization for Kinect raw depth sensor input.*

Figure 6. *Hand and wrist Detection and Extraction within input.* ***(Step 1)***

While extracting the hand, the thickness on the Y axis at all positions can be created, and the X axis merely corresponds to the total number of pixels in that single row, considering a row was found. Vertical projections may be created by the same process by continuing the X coordinate of each of the hand's pixels, but changing the Y value so that it corresponds to the number of X coordinate pixels.

```
PFont ft;
PFont ft2;
// x projection variables
int xOfXProj=0;//x position of x projection pixels
int yOfXProj=0;//y position of x projection pixels
int smoothYAmount = 30;
int [] xPos = new int [smoothYAmount];//array to hold 10 values x values of x projecton
int maxXOfXProj; //maximum x value in x projection
int maxYOfXProj; //max y valu in x projeciton
int xProjXAve=0;//mean of values in array
int prevXProjXAve=0;//previous mean of values in array
int relMinXOfXProj=w;//x coord of relative minimum of x projection
int relMinYOfXProj=h;//y coord of relative minimum of x projection
int tempRelMinYOfXProj=h-1;


// y projection variables
int xOfYProj=0;//x position of y projection pixels
int yOfYProj=0;//y position of y projection pixels
int smoothXAmount = 30;
int [] yPos = new int [smoothXAmount];//array to hold 10 values x values of x projecton
int maxYOfYProj; //maximum x value in x projection
int maxXOfYProj; //max y valu in x projeciton
int yProjYAve=0;//mean of values in array
int prevYProjYAve=0;//previous mean of values in array

//finds mid point on hand y coord

int tempMaxXOfXProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify midpoint of hands
int tempMaxYOfXProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify midpoint of hands

//finds mid point on hand x coord
```

Figure 7. *Screen capture of code (Processing 2.1.1) that deals with X and Y axis projections.*

Figure 8. *Horizontal X projection of Hand.*

Figure 6. *Vertical Y projection of Hand.*

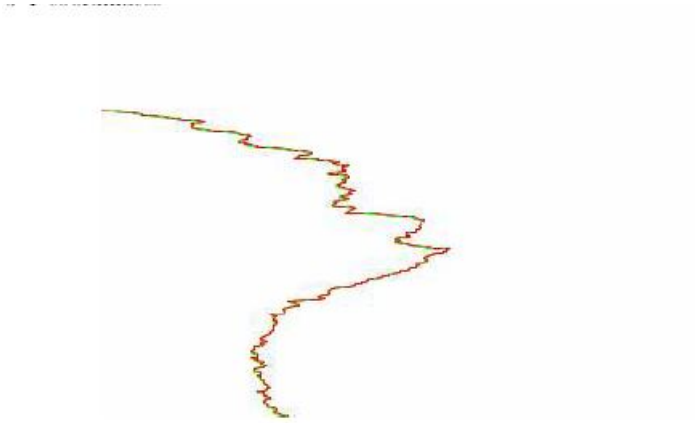


Figure 7. *Unsmoothed X Values with an average range of 3 Y coordinates.*

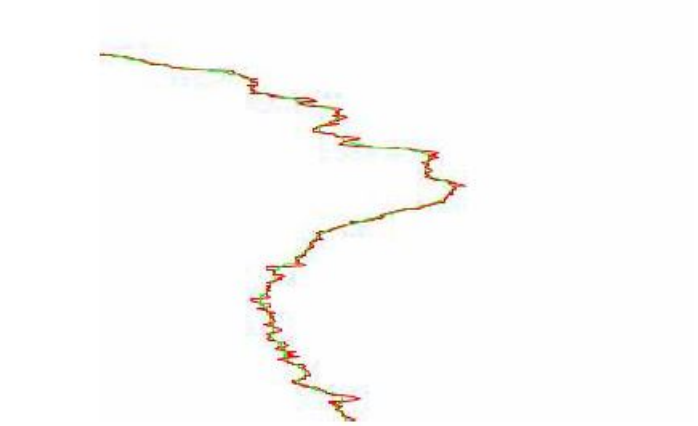


Figure 8. *Unsmoothed X values with an Average range of 11 Y coordinates in movement. Difference is distinguishable when compared to Fig. 7.*

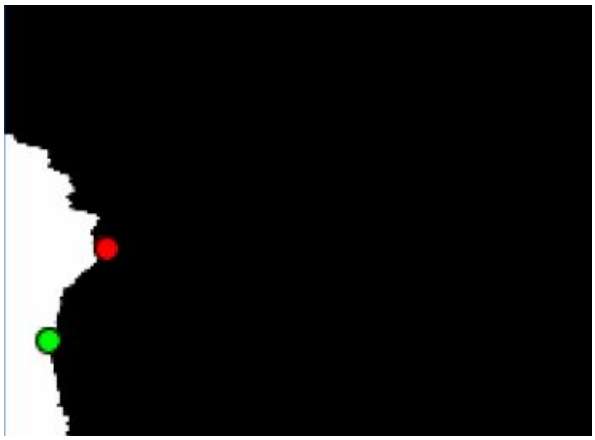


Figure 9. *Horizontal projection of Fig.8 with the red dot corresponding to the max. and the green responding to the set min. Max. = Palm of hand; Min. = Wrist of isolated wrist.*

Figure 10. *Finalized hand isolation from the input.*

3.2 Errors and Resolution

Although we were able to isolate the hand from the arm/wrist, and also able to fix deformities within the image, our development was halted when utilizing projections to display the hand's movement over time. Originally we considered the possibility of it possibly being a hardware issue, so we moved our development from the Windows Platform to OSX, but were only met with similar issues. Although CLNUI, our chosen library for using the Kinect sensor with Processing, is a universal build, it is also completely open-sourced and rarely updated alongside hardware, which proved to be the most rational cause.

We considered porting our previous work to the OpenKinect library, another universal build which is even supported within Processing's official Library plug-in tab, making it quite a choice. OpenKinect also would have given us more hardware freedom by allowing for even more physical hardware feature support, such as rotation, tilt, and indicator lights. This would have been quite efficient while tracking gestures in real time as the Kinect's limits are +/-30 degrees.

KinectTiltCamera

```
import org.openkinect.processing.*;

Kinect kinect;
float deg = 15;

void setup() {
  size(640, 480);
  kinect = new Kinect(this);
  kinect.start();
  kinect.enableRGB(true);
  kinect.tilt(deg);
}

void draw() {
  background(0);
  image(kinect.getVideoImage(), 0, 0);
}

void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
      deg = constrain(deg + 2,-30,30);
      kinect.tilt(deg);
      println(deg);
    }
    if (keyCode == DOWN) {
      deg = constrain(deg - 2,-30,30);
      kinect.tilt(deg);
      println(deg);
    }
  }
}
```

Figure 11. *Screen capture of source code example of Kinect tilt/rotation.*

Our decision to halt development was finalized after our February evaluation, where it was recommended to us to focus more on our Python development, due to reasons noted in Section 4 of this final report. Although the work put into this rather extensive method of American Sign Language recognition using generic consumer hardware did result in being inconclusive, and we were not able to find a finalized result, we personally view the work compiled to be something to explore more outside of the Challenge (due to time restraints), for when working, we can easily estimate an upwards of 90% accuracy- something most other methods simply can't currently realistically compare to.

4. Python Development

4.1 Concept

From the original concept, we knew that Computer Vision would be heavily utilized in the development and creation, if possible, of detecting American Sign Language without various uncommon hardware. Python, is a dynamic, object oriented multipurpose programming language which was initial designed to be quick and efficient when in uniform syntax. OpenCV, also known as Open Source Computer Vision, is a library of programming functions which were developed for the purpose of real-time Computer Vision. Originally developed by Intel for C/C++, it is now supported by Willow Garage and Itseez. Due to being focused on real-time image processing, it also has the ability to optimize routines to accelerate itself if the Library finds Intel's Integrated Performance Primitives. In recent years of computer vision, even in other languages like C/C++ and Java, OpenCV has become one of many standards in open-source solutions in development.

4.2 Design

Our original concept when concerning Python actually stems from the Histogram Comparison tutorial within the OpenCV library. The example uses the compareHist function to get a numerical parameter that expresses how well tow histograms match with eachother. To compare two histograms, (H_1 and H_2), one must choose a metric ($d(H_1, H_2)$) to demonstrate how well they match. OpenCV also offers four different metrics to compute the matching:

a. **Correlation (CV_COMP_CORREL)**

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

Where:

$$\bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

and N is the total number of histogram bins.

b. **Chi-Square (CV_COMP_CHISQR)**

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

c. **Intersection (method=CV_COMP_INTERSECT)**

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

d. **Bhattacharyya distance (CV_COMP_BHATTACHARYYA)**

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H_1}\bar{H_2}N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

A program like this can do a number of things including: loading two base images to be compared, generate one image with the lower half of the base image, conversions to HSV formatting, compare the histogram of the base while concerning two or more histograms, and even display the numerical matching parameters obtained per cycle.

For our program, we take this technique, after much work, and have the comparison come from a live static input via web camera, and compare it to a .jpg located within a model library. Also, rather than having the entire input compared, we rather just compare the gesture space, which is visualized by its blue outlines within the GUI, which gives the program a rather efficient feel. Also, we have our shell giving the output success rate within the live comparison

with a good comparison being <.1, bad being >.1, and an exact copy =0 (Which is highly unlikely due to input environment).

Also, the ROI (Region of Interest) within our code stands as our gesture space, which a window of 400,200,200,200 given normal startup within Python. The ROI is converted to contour, along with our image library to ensure a higher rate of comparison and eliminate color errors rather than gestural. Being we chose to work with Python, 2.7.3. specifically, we also took the liberty of importing cv2.cv, the most updated build of OpenCV, and calling upon it as simply "cv"- a previous build. This was to ensure stability and being able to utilize the libraries optimization techniques rather than implementing our own, which would enlarge the code and possibly cause issues.

Our Program can currently recognize six gestures with a 74% rate of success. These gestures, A to F, were compiled in contour with the help of Aryssa Baca, a volunteer at New Mexico School for the Deaf along with a number of other notable institutions. With her help, we learned many aspects of fingerspelling and learned many techniques when dealing with capturing ASL gestures for a model library.

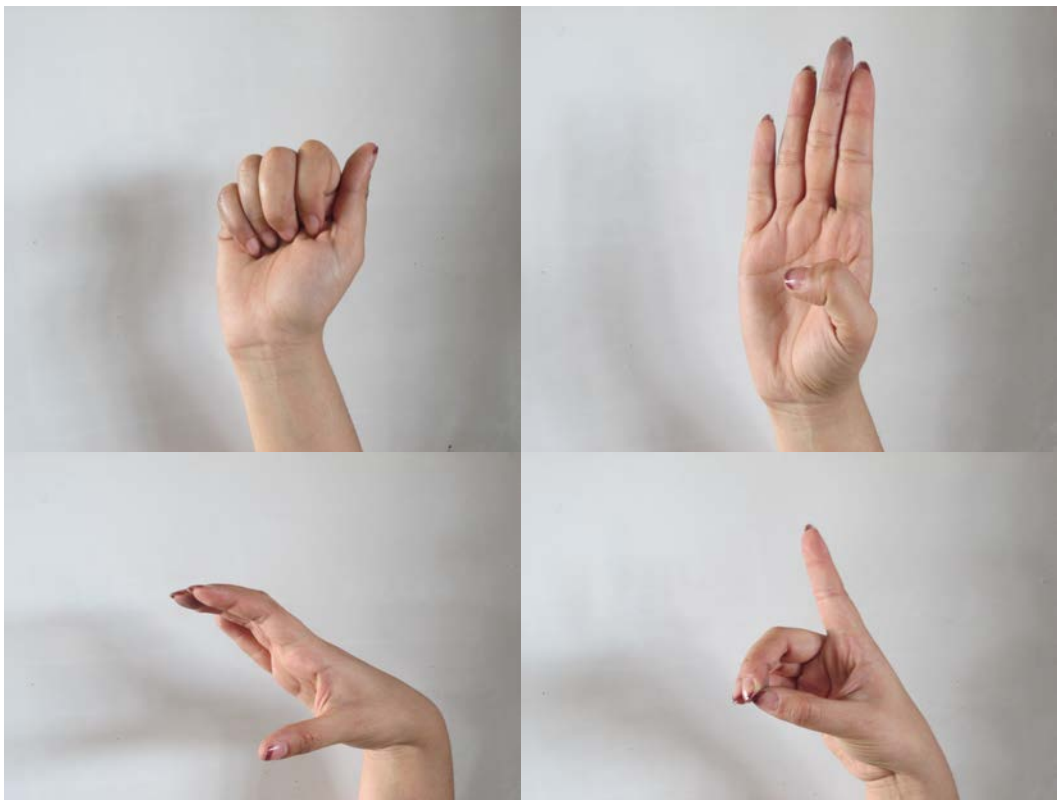


Figure 12. *Aryssa Bacca's hands displaying the first four gestures of fingerspelling.*

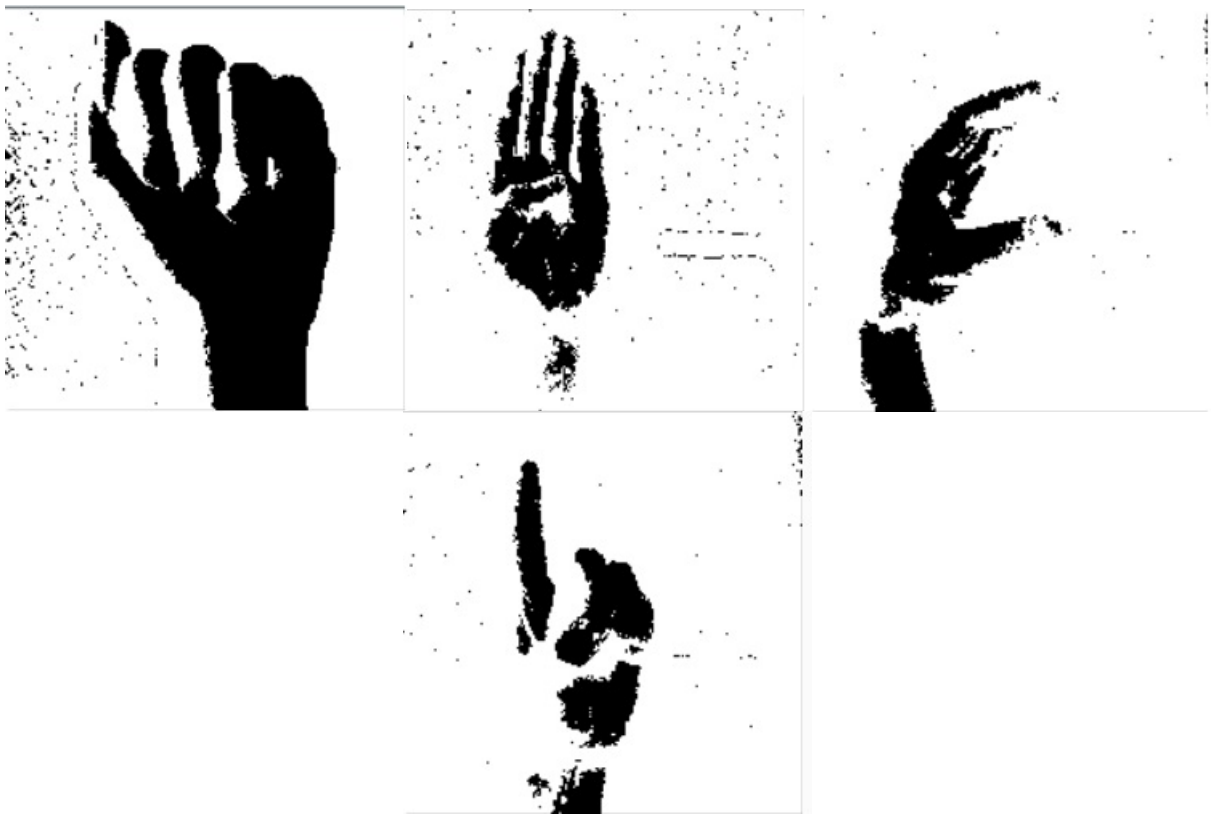


Figure 13. *First four ASL fingerspelling gestures in contour.*

We plan on releasing all 26 gestures in contour on our website, *asldetection.com* when completed and shown to be successful. Currently the library is compiled, but due to grainy images due to environmental factors we do not feel confident on their release just yet. We also intend on releasing our Processing attempt on our site, but do not consider it to be a notable aspect of our project no a finalized product due to library issues previously mentioned.

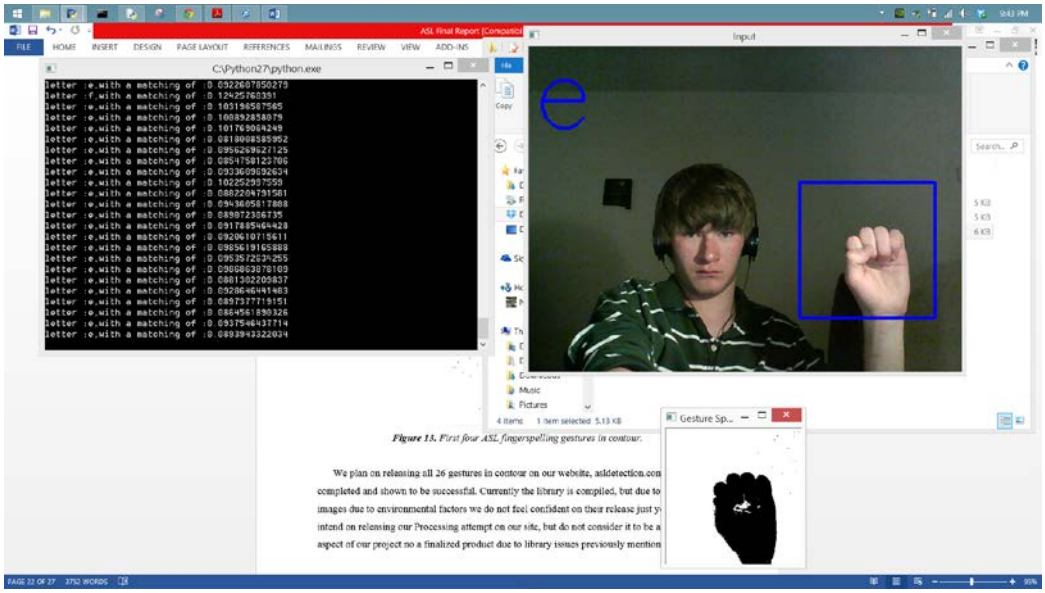


Figure 14. *Screen Capture of Python build.*

5. Conclusion

In conclusion, We Team #1 can confidently state that ASL detection for real world consumer level implementation is completely possibly, and we have accurately designed what we refer to as a prototype to demonstrate such. Although our Processing/Kinect design was not successful within the challenge, we do completely value the technique and plan on continuing its development to release alongside future builds of our Python program on our website *asldetection.com*, which will go live on April 14th, 2014 with the aim of not only having our web-based presentation, but also serve as a center for anyone curious about ASL detection using computer vision. We realize that our project is far away from a final product for all to use, but we want it to serve as a catalyst for future research in this intriguing field.

6. Significant Achievement

It's safe to say that our significant achievement was the addition of Dmitri Malygin-Voorhees virtually days before our December report was due. We lost time by catching him up to speed with the project's development, but gained true teamwork- which is important in a project of this magnitude.

7. Acknowledgement

We would like to personally thank Supercomputing Challenge Alumni Sara Hartse and Bjorn Swenson for not only inspiring us to compete, but also being available at the most inconvenient of times to help us out on whatever it may be.

We also would like to acknowledge Aryssa Baca for her much needed help on our project. If it wasn't for her knowledge of ASL, our project would not be where it is today.

Finally, we would like to thank not only our February Evaluation judges for their motivational help, but also everyone involved with the New Mexico Supercomputing Challenge for making this incredible program possible.

8. Source Code

```
#--------------------------------------------------------------------------------
# Name:        Contour ASL Gesture Recognition
# Purpose:     NM Supercomputing Challenge
#
# Author:      Noah Caulfield
#
# Created:     31/03/2014
#--------------------------------------------------------------------------------


import sys
import cv2
import cv2.cv as cv
import numpy as np
from time import clock




# function to compare two forms and returns the result of comparing
# Good result <0.1
# Bad result> 0.1
# exact matching =0


def compare_2_formes(Image1,Image2):
    mincontour=500 # minimum size of a form to be detected
    CVCONTOUR_APPROX_LEVEL=5# parameter for call contour
    img_edge1=cv.CreateImage(cv.GetSize(Image1),8,1) #egde image
```

```
#       img1_8uc3=cv.CreateImage(cv.GetSize(Image1),8,3)

        img_edge2=cv.CreateImage(cv.GetSize(Image2),8,1)
 #      img2_8uc3=cv.CreateImage(cv.GetSize(Image2),8,3)

        cv.Threshold(Image1,img_edge1,123,255,cv.CV_THRESH_BINARY) # filter threshold
        cv.Threshold(Image2,img_edge2,123,255,cv.CV_THRESH_BINARY)



        storage1=cv.CreateMemStorage()
        storage2=cv.CreateMemStorage()

        first_contour1=cv.FindContours(img_edge1,storage1) # pointer to the first edge of the form 1
        first_contour2=cv.FindContours(img_edge2,storage2) # pointer to the first edge of the form 2



        newseq=first_contour1
        newseq2=first_contour2

        if not(first_contour1) or not(first_contour2):
                return 0

        current_contour=first_contour1
        while 1:
                current_contour=current_contour.h_next() # path in the sequence of edges of the first form
                if (not(current_contour)): # stop condition if the contour pointer = NULL
```

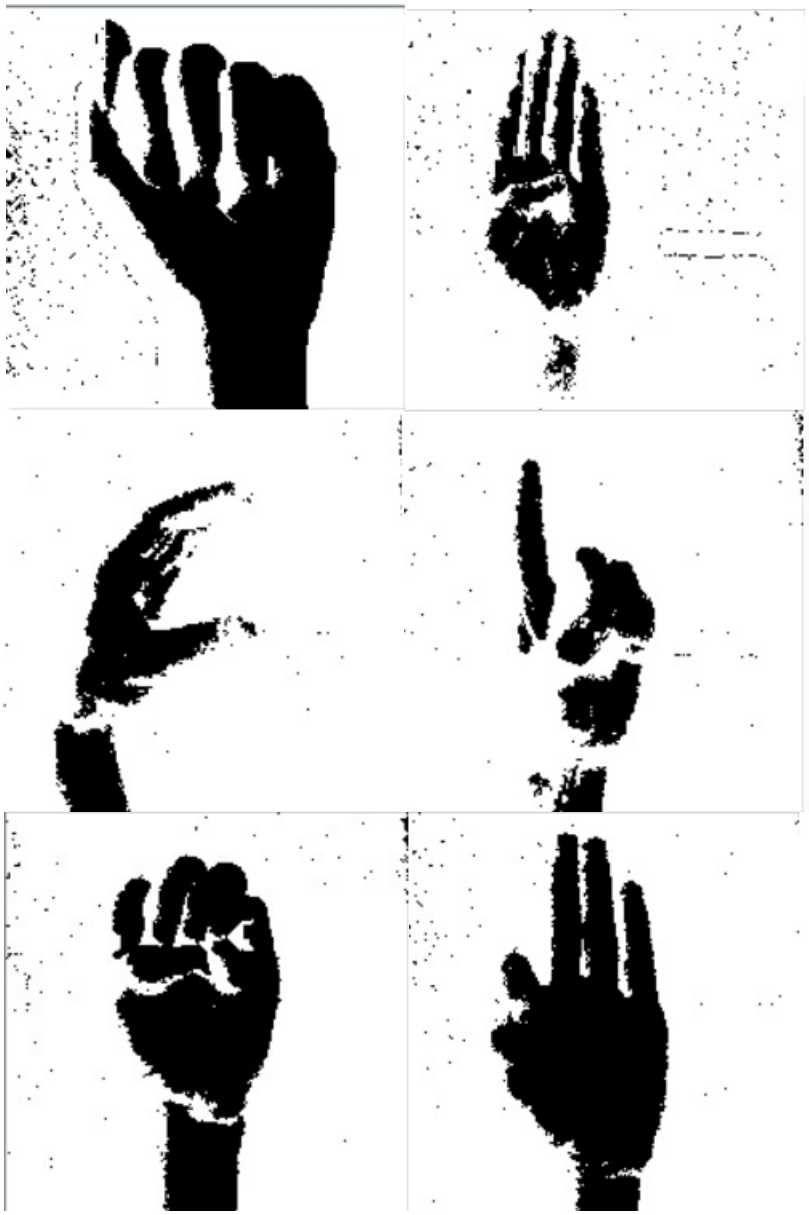
```
                break

            if cv.ContourArea(current_contour)> mincontour :

newseq=cv.ApproxPoly(current_contour,storage1,cv.CV_POLY_APPROX_DP,CV
CONTOUR_APPROX_LEVEL,0)
            #       cv.CvtColor(Image1,img1_8uc3,cv.CV_GRAY2BGR );
            #
cv.DrawContours(img1_8uc3,newseq,cv.CV_RGB(0,255,0),cv.CV_RGB(255,0,0),0,2
,8);
            #
cv.NamedWindow("ContourImage2",cv.CV_WINDOW_AUTOSIZE)
            #       cv.ShowImage("ContourImage2",img1_8uc3)


      current_contour=first_contour2

     # path of the second form of contours
      while 1:
            current_contour=current_contour.h_next()
            if (not(current_contour)):
                  break

            if cv.ContourArea(current_contour)> mincontour :

newseq2=cv.ApproxPoly(current_contour,storage2,cv.CV_POLY_APPROX_DP,C
VCONTOUR_APPROX_LEVEL,0)
            #       cv.CvtColor(Image2,img2_8uc3,cv.CV_GRAY2BGR);
            #
cv.DrawContours(img2_8uc3,newseq2,cv.CV_RGB(0,255,0),cv.CV_RGB(255,0,0),0,
2,8);
```

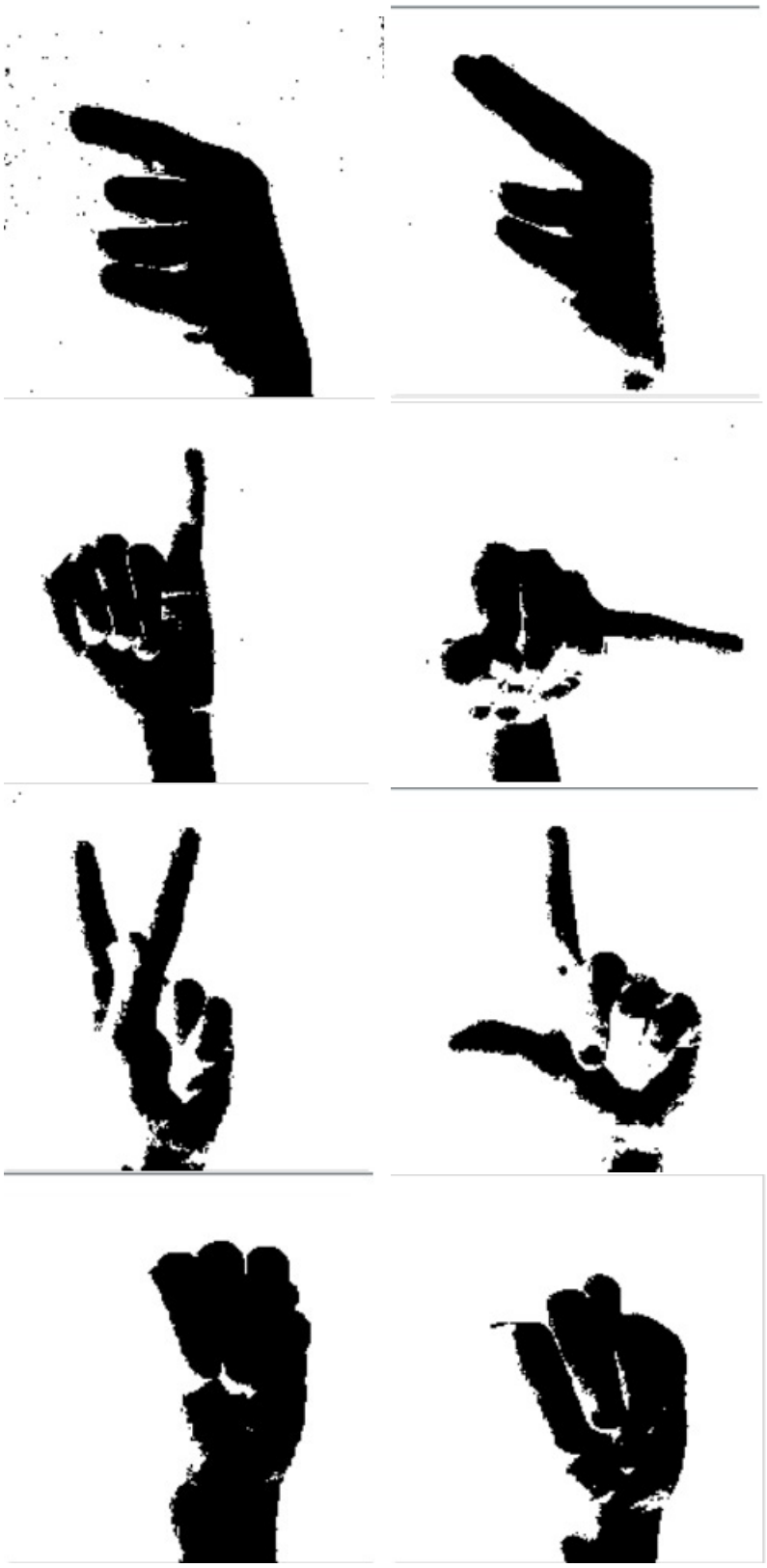
```
        #
cv.NamedWindow("ContourImage",cv.CV_WINDOW_AUTOSIZE)
        #       cv.ShowImage("ContourImage",img2_8uc3)


    matchresult=1;
    matchresult=cv.MatchShapes(newseq,newseq2,1,2)
    return matchresult
    #print("Match result :"+str(matchresult))

#main
font = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX,5,5, 0, 3, 8) #initialize
SignsList=["a.jpg","b.jpg","c.jpg","d.jpg","e.jpg","f.jpg"]  # list which contain all
images of signs
imagesList={"a.jpg":cv.LoadImage("signs/a.jpg",cv.CV_LOAD_IMAGE_GRAYS
CALE)}

for e in SignsList:

imagesList[e]=cv.LoadImage("signs/"+e,cv.CV_LOAD_IMAGE_GRAYSCALE)

#imagesList.append(cv.LoadImage("signs/"+e,cv.CV_LOAD_IMAGE_GRAYSCA
LE))
cv.NamedWindow("Input",cv.CV_WINDOW_AUTOSIZE)
cv.NamedWindow("Gesture Space",cv.CV_WINDOW_AUTOSIZE)
matchresult=1;
p_capWebcam=cv.CaptureFromCAM(0)
while 1 :
    p_imgOriginal =cv.QueryFrame(p_capWebcam)
    cv.Flip(p_imgOriginal,p_imgOriginal,1)
```

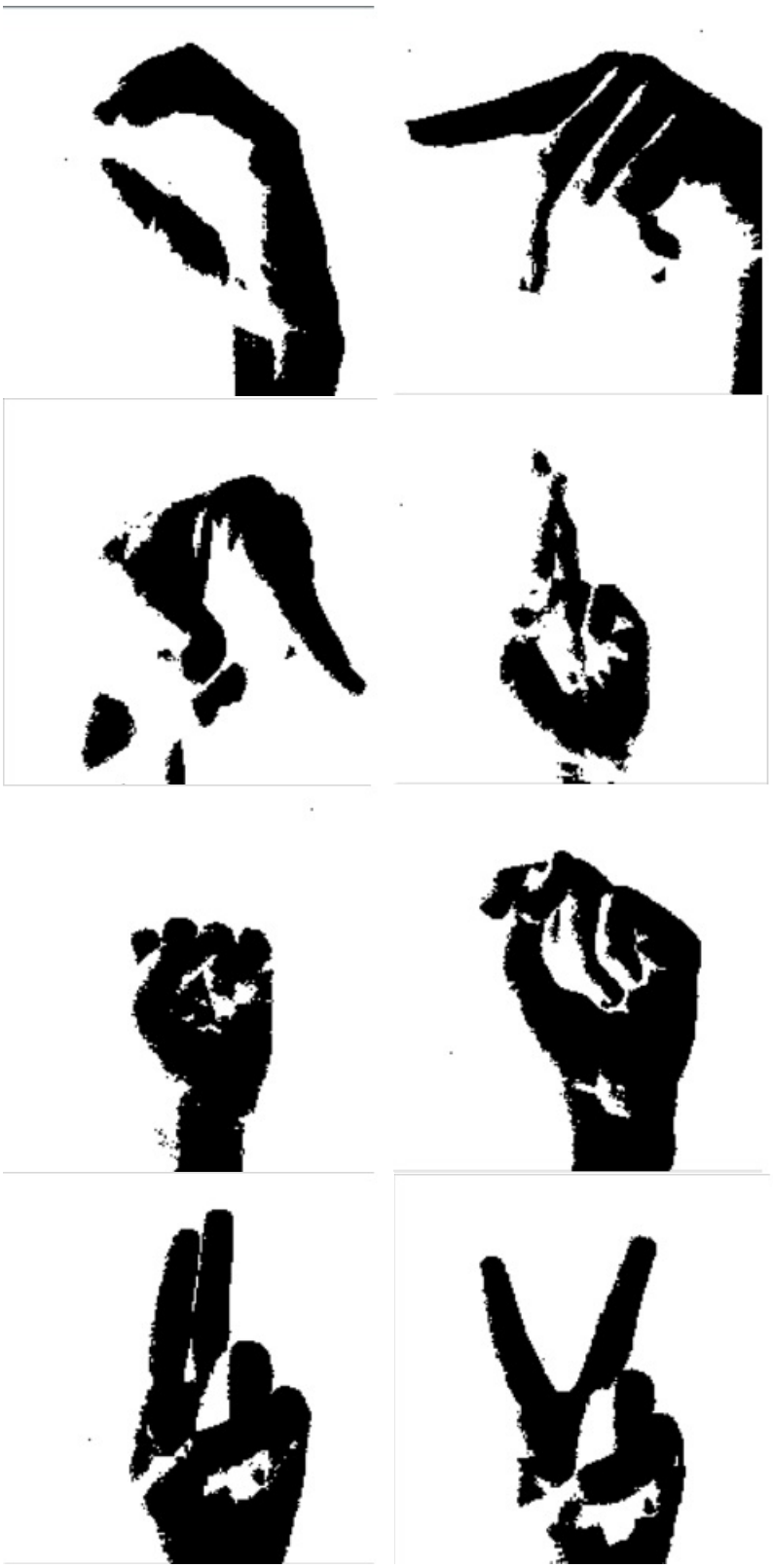
```
    # capture from webcam
     p_gray=cv.CreateImage(cv.GetSize(p_imgOriginal), 8, 1 )
     cv.CvtColor(p_imgOriginal,p_gray,cv.CV_BGR2GRAY)
     cv.SetImageROI(p_gray,(400,200,200,200))
    # Region setting of fixed interest
     cv.Threshold(p_gray,p_gray,100,255,cv.CV_THRESH_BINARY_INV)

     cv.Rectangle(p_imgOriginal,(400,200),(600,400),(255,0,0),4);
     j=0
     for imageI in imagesList :# path of the image list and test each image with the
ROI (region of interest)

#image_to_test=cv.LoadImage("signs/"+image_path,cv.CV_LOAD_IMAGE_GRA
YSCALE)
          matchresult=compare_2_formes(p_gray,imagesList[imageI])
#comparison
          #print("le match est "+str(matchresult))
          if matchresult < 0.13 and matchresult!=0 :
               sign_name=imageI.split('.')[0]
               print("letter :"+sign_name+",with a matching of :"+str(matchresult))
               cv.PutText(p_imgOriginal,sign_name,(5,120),font,255)

          cv.ShowImage("Input",p_imgOriginal)
          cv.ShowImage("Gesture Space",p_gray)
          j=j+1
     checkchar=cv.WaitKey(27)
     if checkchar==27 :
               cv.DestroyAllWindows("Input")
               cv.DestroyAllWindows("Gesture Space")
               break
```

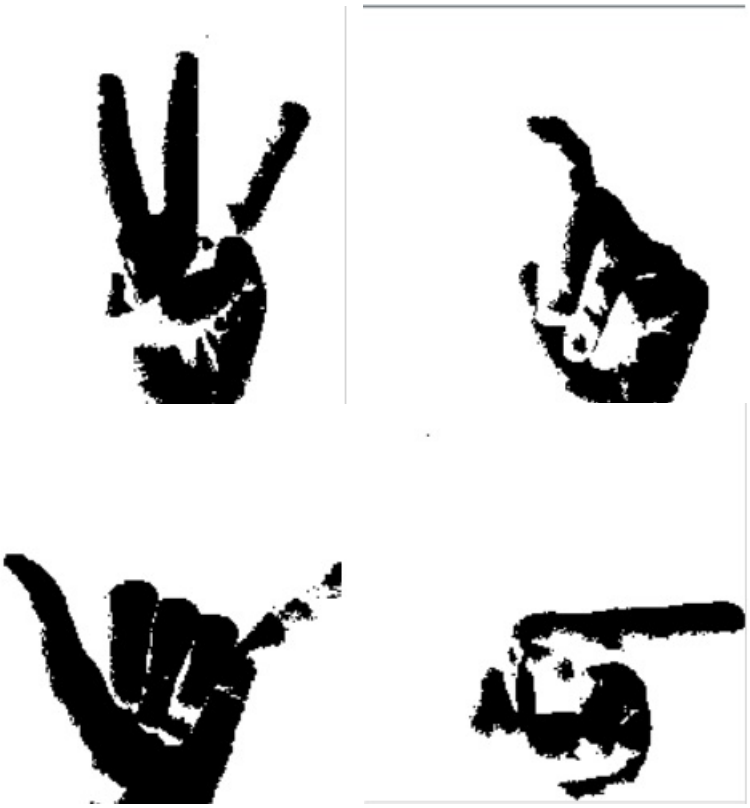
9. Example Contour Image Library

Note: .Jpg's will differ than those released on *asldetection.com*









10. Inconclusive Processing Source

Note: Excessive Dashes indicate file separation within build.

```
  ////////////////Final//////////////////////
import SimpleOpenNI.*;

import com.sun.jna.Pointer;
import cl.nui.CLNUI;
import processing.opengl.*;



Pointer motor, camera;
PImage depthData,handSmooth,xProjection,yProjection, handTrace, hand,handFat,traceSmooth;

//height and width
int w=640;
int h=480;

//lens for average filter size
int xLens=3;
int yLens=3;

//outlier filter sensitivity ie removes taret pixel if fewer than proxFilter pixels in lens area.
int proxFilter=20;




PFont ft;
PFont ft2;
// x projection variables
int xOfXProj=0;//x position of x projection pixels
int yOfXProj=0;//y position of x projection pixels
int smoothYAmount = 30;
int [] xPos = new int [smoothYAmount];//array to hold 10 values x values of x projecton
int maxXOfXProj; //maximum x value in x projection
int maxYOfXProj; //max y valu in x projeciton
int xProjXAve=0;//mean of values in array
int prevXProjXAve=0;//previous mean of values in array
int relMinXOfXProj=w;//x coord of relative minimum of x projection
int relMinYOfXProj=h;//y coord of relative minimum of x projection
int tempRelMinYOfXProj=h-1;
```

```
// y projection variables
int xOfYProj=0;//x position of y projection pixels
int yOfYProj=0;//y position of y projection pixels
int smoothXAmount = 30;
int [] yPos = new int [smoothXAmount];//array to hold 10 values x values of x projecton
int maxYOfYProj; //maximum x value in x projection
int maxXOfYProj; //max y valu in x projeciton
int yProjYAve=0;//mean of values in array
int prevYProjYAve=0;//previous mean of values in array

//finds mid point on hand y coord

int tempMaxXOfXProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify
midpoint of hands
int tempMaxYOfXProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify
midpoint of hands

//finds mid point on hand x coord

int tempMaxXOfYProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify
midpoint of hands
int tempMaxYOfYProj=0; // necessary becasue MaxXOfXProj is cleared and need this value to identify
midpoint of hands

//trace and edge detection
boolean boundary=false; // boundary test flag boundary = 0 means not boudanry
boolean boundaryBin=false; // boundary test flag boundary = 0 means not boudanry
int boundaryNeighbours; //test number of nieghbours that are also boundary n
int maxPoint=0; // fist point of edge array.
int edge [] = new int [w*4]; //edge array
int edgeSmooth [] = new int [w*4]; //edge array

//if true activates NN once
boolean analyse = false;




void setup()
```

```
{
  smooth();
  size( 800, 600);
  frameRate( 100 );

  motor = CLNUI.INSTANCE.CreateNUIMotor();
  camera = CLNUI.INSTANCE.CreateNUICamera();
  CLNUI.INSTANCE.StartNUICamera( camera );
  CLNUI.INSTANCE.SetNUIMotorLED( motor, (byte)7 );
  System.out.println( "Kinect Serial: " + CLNUI.INSTANCE.GetNUIMotorSerial(motor) );


  depthData = createImage( 640, 480, RGB );
//  handSmooth = createImage (640, 480, RGB);
  handTrace = createImage (640, 480, RGB);
//  xProjection = createImage (640, 480, RGB);
//  yProjection = createImage (640, 480, RGB);
//  hand = createImage (640, 480, RGB);
  handFat = createImage (640, 480, RGB);
  traceSmooth = createImage (640, 480, RGB);
  ft = loadFont("IrisUPCBold-28.vlw");
  ft2=loadFont("DialogInput.bolditalic-150.vlw");
}




public void draw()
{

  smooth();



  CLNUI.INSTANCE.GetNUICameraDepthFrameRGB32( camera, depthData.pixels, 0 );
            //reinterpret_cast<DWORD*>(uiDepthData) );
  //System.out.println( "Depth result: " + res );

  background( 0 );
  // image( handSmooth, 0, 0,400,300);
  // image( handTrace,0,0,800,600);
```

```
  //    image( hand, 400,0,400,300);
//  image( xProjection, 0,300,400,300);
//  image( yProjection, 0,0,400,300);



 for (int y = 0; y < h; y++)
 {
  for (int x = 0; x < w; x++)
  {

   int loc = x+y*w;
   float bl = blue (depthData.pixels[loc]);
   float re = red (depthData.pixels[loc]);

   //clearXProjection (loc); //clears x projection
   // clearHandSmooth  (x,y,loc); //clears handsmooth
   clearHandTrace (loc);//clears handTrace
   clearHandFat(loc);

   if(bl>250&&re>1&&y<h-10)
   {
    // maxPoint=loc-1-1*w;
    handFat(loc);
   }




  // xProjection(x,y,bl,re); //draws x projection
   //  hand(loc,y,bl,re);//draws hand
  }
 }

 if (maxPoint>10+10*w)
 {
  edge(maxPoint);
 }

//  for (int x = 0; x < w; x++)
//  {
//    for (int y = 0; y < h; y++)
```

```
//   {
//     int loc = x+y*w;
//     float bl = blue (depthData.pixels[loc]);
//     float re = red (depthData.pixels[loc]);
//     clearYProjection (loc); //clears y projection
//     yProjection(x,y,bl,re);
//
//   }
//  }




 // println(frameRate);



//     xProjectionMax();
//     xProjectionRelMin();
//     yProjectionMax();
//       midPoint(); //find mid point of hand
  depthData.updatePixels();
  //  handSmooth.updatePixels();
//  handTrace.updatePixels();
//  handFat.updatePixels();
  traceSmooth.updatePixels();
  //  hand.updatePixels();
//  xProjection.updatePixels();
//  yProjection.updatePixels();
}




void midPoint () //find mid poit on hand
{

  fill(0,0,255);
  ellipse(tempMaxXOfYProj*800/w,tempMaxYOfXProj*600/h,10,10);
}
```

```
void keyPressed()
{
  analyse = true;
}









void prnt (int loc,int i)
{

  //println("x"+loc%w+"y"+loc/w);
}



public void stop()
{
  CLNUI.INSTANCE.SetNUIMotorLED( motor, (byte)0 );
  CLNUI.INSTANCE.DestroyNUIMotor( motor );
  CLNUI.INSTANCE.DestroyNUICamera( camera );

  super.stop();
}
//////////////////////Clear////////////////////////////////////
void clearHandSmooth(int x, int y,int loc)
{
  if(x<w-(xLens-1)/2-1&&y<h-(yLens-1)/2-1) // clears handSmooth one x and y value in front of the smoothing
lens
  {
    handSmooth.pixels[loc+(xLens-1)/2+(yLens-1)/2*w]=color(0);
  }
}
```

```
void clearXProjection(int loc) // clears xProjection
{
  xProjection.pixels[loc] = color (0);
}

void clearYProjection(int loc) // clears yProjection
{
  yProjection.pixels[loc] = color (0);
}

void clearHandTrace(int loc) // clears handTrace
{
  handTrace.pixels[loc] = color (0);
}
void clearHand(int loc) // clears hand
{
  hand.pixels[loc] = color (0);
}
void clearHandFat(int loc) // clears hand
{
  handFat.pixels[loc]=color(0);
}


void hand(int loc,int y,float bl,float re) //draws hand
{
  if(bl>250&&re>1&&y<tempRelMinYOfXProj)
  {
    hand.pixels[loc]=color(0,255,0);
  }
}

//////////////////////////Neural_Network//////////////////////////////
void neuralNetwork(float f1,float f2,float f3,float f4,float f5,float v1,float v2,float v3,float v4)
{
  //inputs
  float x1A = f1+0.1;
  float x2A = f2+0.1;
  float x3A = f3+0.1;
  float x4A = f4+0.1;
```

```
float x5A = f5+0.1;
float x6A = v1+0.1;
float x7A = v2+0.1;
float x8A = v3+0.1;
float x9A = v4+0.1;



//initial weights
//input layer weights
float w1A = -0.7730226;
float w1B = 0.7183541;
float w1C = -0.6928675;
float w1D = 0.5296034;
float w1E = 0.8375236;
float w2A = -0.51678956;
float w2B = -0.31572336;
float w2C = -0.8320275;
float w2D = -0.24141584;
float w2E = -0.95364016;
float w3A = -0.5566789;
float w3B = 0.94730735;
float w3C = 0.26510268;
float w3D = 0.8865323;
float w3E = -0.15867522;
float w4A = -0.49440315;
float w4B = 0.7721455;
float w4C = -0.262894;
float w4D = 0.94908196;
float w4E = -0.39958116;
float w5A = 0.08508696;
float w5B = 0.27202654;
float w5C = -1.0197405;
float w5D = -0.008199202;
float w5E = -1.0913138;
float w6A = -0.49093857;
float w6B = 0.8819983;
float w6C = 0.10008627;
float w6D = 1.0063748;
float w6E = -0.21145786;
float w7A = 0.44217157;
float w7B = 0.0989018;
```

```
  float w7C = 0.3772578;
  float w7D = 0.3345932;
  float w7E = 0.987472;
  float w8A = 0.89786;
  float w8B = 1.0219779;
  float w8C = -0.3381234;
  float w8D = -0.8775543;
  float w8E = 0.581145;
  float w9A = -0.62561995;
  float w9B = 0.002124509;
  float w9C = -0.54920745;
  float w9D = 0.5327601;
  float w9E = -0.86570156;

  //output layer weights
  float w1 = -0.1945392;
  float w2 = 0.79378915;
  float w3 = -0.32651442;
  float w4 = 0.55492806;
  float w5 = -0.9274519;

  //node inputs and outputs
  float y;
  float y1_in;
  float y1_out;
  float y2_in;
  float y2_out;
  float y3_in;
  float y3_out;
  float y4_in;
  float y4_out;
  float y5_in;
  float y5_out;

String result;

  //euler's number
  float e = (float) Math.E;
```

```
  //forward pass and y output

    //calculate hidden layer inputs
    //calculate hidden layer inputs
    y1_in = x1A*w1A + x2A*w2A + x3A*w3A + x4A*w4A + x5A*w5A + x6A*w6A + x7A*w7A + x8A*w8A +
x9A*w9A;
    y2_in = x1A*w1B + x2A*w2B + x3A*w3B + x4A*w4B + x5A*w5B + x6A*w6B + x7A*w7B + x8A*w8B +
x9A*w9B;
    y3_in = x1A*w1C + x2A*w2C + x3A*w3C + x4A*w4C + x5A*w5C + x6A*w6C + x7A*w7C + x8A*w8C +
x9A*w9C;
    y4_in = x1A*w1D + x2A*w2D + x3A*w3D + x4A*w4D + x5A*w5D + x6A*w6D + x7A*w7D + x8A*w8D +
x9A*w9D;
    y5_in = x1A*w1E + x2A*w2E + x3A*w3E + x4A*w4E + x5A*w5E + x6A*w6E + x7A*w7E + x8A*w8E +
x9A*w9E;

    //calclute hidden layer outputs rounded to 4 decs
    y1_out = 1/(1+pow(e,-y1_in));
    y2_out = 1/(1+pow(e,-y2_in));
    y3_out = 1/(1+pow(e,-y3_in));
    y4_out = 1/(1+pow(e,-y4_in));
    y5_out = 1/(1+pow(e,-y5_in));



    // calculate y output to 4 decs
    y = y1_out*w1 + y2_out*w2 + y3_out*w3 + y4_out*w4 +y5_out*w5;

//displays result
fill(255);
textFont(ft2,100);
if(y<0.3)
{
 text ("U",w-40,h-50);
}
else if (y>=0.3&&y<0.5)
{
  text ("I",w-40,h-50);
```

```
}
if(y>=0.5&&y<0.7)
{
  text ("4",w-40,h-50);
}
else if (y>=0.7)
{
  text ("5",w-40,h-50);
}



  //prints the input values, outputs value, output target, pass number and learning rate

    println();
    print("_INPUTS");
    println();
    print("x1A = " + x1A);
    println();
    print("x2A = " + x2A);
    println();
    print("x3A = " + x3A);
    println();
    print("x4A = " + x4A);
    println();
    print("x5A = " + x5A);
    println();
    print("x6A = " + x6A);
    println();
    print("x7A = " + x7A);
    println();
    print("x8A = " + x8A);
    println();
    print("x9A = " + x9A);
    println();


    print("Output");
    println();
    print("y_out = " + y);
    println();
```

```
}
//////////////////////BoundaryBinTest////////////////////////////////
void boundaryBin(int loc)
{
 if (loc%w>0&&loc/w>0&&loc%w<w-1&&loc/w<h-1)
 {
   int neighbours =0; //clears value of number of pixels in a 3*3 square around the pixel at loc
   int xLens=3;
   int yLens=3;
   int  x3=loc%w;
   int y3=loc/w;
   float bl=blue (handFat.pixels[loc]);
   float re=red (handFat.pixels[loc]);

   if (x3>(xLens-1)/2&&y3>(yLens-1)/2&&x3<w-(xLens-1)/2-1&&y3<h-(yLens-1)/2-1)//ensures averaging lense
stays within boundary of image
   {

    for (int x=0;x<xLens;x++)
    {
     for (int y=0;y<yLens;y++)
     {
       int locTemp = loc+x+y*w-(xLens-1)/2-((yLens-1)/2)*w; //temp position of pixel in square to be tested
       float blTemp=blue (handFat.pixels[locTemp]);
       float reTemp=red (handFat.pixels[locTemp]);
       if (blTemp>250&&reTemp>1)
       {
         neighbours++;
       }
     }
    }
    if(neighbours<xLens*yLens&&bl>250&&re>1&&loc/w<h-3)
    {
     boundaryBin=true;
      //handFat.pixels[loc]=color(255,0,0);

    }

    else
    {
```

```
      boundaryBin=false;

    }
   }
  }
  else
  {
   boundaryBin=false;

  }
}
//////////////////////BoundryNeighbours//////////////////////
void boundaryNeighbours (int loc) // test nnumber of neighbours that are boundary pixels
{
  boundaryNeighbours = 0;
  boundaryBin(loc-1);
  if(boundaryBin==true)
  {
   boundaryNeighbours++;
  }
  boundaryBin(loc-w);
  if(boundaryBin==true)
  {
   boundaryNeighbours++;
  }
  boundaryBin(loc+1);
  if(boundaryBin==true)
  {
   boundaryNeighbours++;
  }
  boundaryBin(loc+w);
  if(boundaryBin==true)
  {
   boundaryNeighbours++;
  }
}
//////////////////////////BoundaryTest/////////////////////
void boundary(int loc)
{
  if (loc%w>0&&loc/w>0&&loc%w<w-1&&loc/w<h-1)
  {
   int neighbours =0; //clears value of number of pixels in a 3*3 square around the pixel at loc
```

```
   int xLens=3; //chagnes size of permiter removal
   int yLens=3;
   int  x3=loc%w;
   int y3=loc/w;
   float bl=blue (depthData.pixels[loc]);
   float re=red (depthData.pixels[loc]);

   if (x3>(xLens-1)/2&&y3>(yLens-1)/2&&x3<w-(xLens-1)/2-1&&y3<h-(yLens-1)/2-1)//ensures averaging lense
stays within boundary of image
   {

    for (int x=0;x<xLens;x++)
    {
     for (int y=0;y<yLens;y++)
     {
      int locTemp = loc+x+y*w-(xLens-1)/2-((yLens-1)/2)*w; //temp position of pixel in square to be tested
      float blTemp=blue (depthData.pixels[locTemp]);
      float reTemp=red (depthData.pixels[locTemp]);
      if (blTemp>250&&reTemp>1)
      {
       neighbours++;
      }
     }
    }
    if(neighbours<xLens*yLens&&bl>250&&re>1&&loc/w<h-3)
    {
     boundary=true;
      //handTrace.pixels[loc]=color(255);
    }

    else
    {
     boundary=false;

    }
   }
  }
  else
  {
   boundary=false;

  }
```

```
}
//////////////////////EdgeTrace//////////////////////////
void edge(int maxPoint)
{

  boolean change=false;
  boolean complete=false;
  boolean firstSearch=true;
  int loc=maxPoint; //location
  int up=-w;
  int down=w;
  int left=-1;
  int right=1;
  int r = (int) random (0,255); //red
  int g = (int) random (0,255); //green
  int b = (int) random (0,255); //blue
  int locP1 =0;// 1st previous loc positon
  int locP2 =0;// 2nd previous loc
  int locP3 =0;// 3rd previous loc
  int locP4 =0;// 4th previous loc
  boolean junction =false; // true if there is a junction
  int junction1=1; //1st previous junction positon
  int junction1Prev=2;//position before junction position
  int junction1Post=3;//position after most recent junction

  int  junction2=3;// 2nd previous junciton posiiton
  int  junction2Prev=4; //2dn previous position before junction position
  int  junction2Post=5; //position after 2nd last junction junction position

  int  junction3=6;// 3rd previous junciton posiiton
  int  junction3Prev=7; //3rd previous position before junction position
  int  junction3Post=8; //position after 3rd last junction position

  int k=0; //edge array index
  int xPoint=0; //x point of vertex on smoothed hand
  int yPoint=0; //y point of vertex on smoothed hand
  int counter =1; //counts numberof indexes in edgeSmooth
  int   centroidX=0;
  int  centroidY=0;

  int aveX=0;
  int aveY=0;
```

```
//to calculate angles between points

int pass =0; //corresponds to previous m for identified point/valley
int xPointP4=0;  //2 previous x and y positions to find anles
int   yPointP4=0;

int  xPointP3=0;
int  yPointP3=0;
int xPointP2=0;  //2 previous x and y positions to find anles
int   yPointP2=0;

int  xPointP1=0;
int  yPointP1=0;
float angleB=0; //angle abc between last three points

int [] fingertip = new int  [7]; // array and indexes to store fingertip and valley points
int fin =0;
int [] valley = new int  [7];
int val =0;
int [] fingerDir=new int [7];

float [] valleyNN = new float  [7]; //distances for neural network
float [] fingertipNN=new float [7];

int flagTip=0; //test if a fingertip has been detected

int xTipAve=0; //average of previous x fingertip coords for points in a small space
int tipX =0; //last x coord of fingertip
int yTipAve=0;
int tipY =0;

int    xPointP12=0;   //previous tip coordiantes for proximate tips
int    xPointP11=0;
int  yPointP12=0;
int  yPointP11=0;
int avePass=1;


int testXAve=0; //average loc of test for tips
int testYAve=0;
int testY1=0; //test coordiantes stored in the first instance before averaging
```

```
 int testX1=0;
 int   testXP2=0; //previous tip test for proximate tips
 int       testXP1=0;
  int      testYP2=0;
  int      testYP1=0;




 for(int i=1;complete==false&&i<w*3;i++)
 {



   //boundary neighbour check
   boundaryNeighbours(loc);// checks how many neighbour pixels are also edge pixels above,below left and right (max4)
   if (boundaryNeighbours>2)
   {
    junction3=junction1; //3rd previous junction
    junction3Prev=junction1Prev;//pixel before 3rd previous junction
    junction2=junction1; //2nd previous junction
    junction2Prev=junction1Prev;//pixel before 2nd previous junction
    junction1=loc; //pixel is a juncton
    junction1Prev=locP1; //edge pixel before first junction pixels
    //       r= (int) random(0,255);
    //       g= (int) random(0,255);
    //       b= (int) random(0,255);
    junction = true;
   }
   //left
   boundaryNeighbours(loc+left);//tests for dead end (if<2)
   boundaryBin(loc+left); //check if left pixel is part of edge
   if (boundaryBin==true&&boundaryNeighbours>1&&loc+left!=locP1&&loc+left!=junction1Post) // if so colour it
   {
    locP4=locP3;
    locP3=locP2;
    locP2=locP1;
    locP1=loc;
    loc=loc+left;
```

```
    // handTrace.pixels[loc]=color(r,g,b);
    edge[k] = loc;
    k++;
    change=true;
    prnt(loc,i);
    if (junction==true) //test if previos position was junction and remembers position after last 3 junctions
    {
      junction3Post=junction2Post;
      junction2Post=junction1Post;
      junction1Post=loc;
      junction=false;
    }
   }
   if (firstSearch==false&&loc==maxPoint)
   {
    complete=true;
   }


   // up

   //boundary neighbour check
   boundaryNeighbours(loc);// checks how many neighbour pixels are also edge pixels above,below left and right
(max4)
   if (boundaryNeighbours>2)
   {
    junction3=junction1; //3rd previous junction
    junction3Prev=junction1Prev;//pixel before 3rd previous junction
    junction2=junction1; //2nd previous junction
    junction2Prev=junction1Prev;//pixel before 2nd previous junction
    junction1=loc; //pixel is a juncton
    junction1Prev=locP1; //edge pixel before first junction pixels
    //      r= (int) random(0,255);
    //      g= (int) random(0,255);
    //      b= (int) random(0,255);
    junction = true;
   }
   //up
   boundaryNeighbours(loc+up); //tests for dead end (if<2)
   boundaryBin(loc+up); //check if up pixel is part of edge
   if (boundaryBin==true&&boundaryNeighbours>1&&loc+up!=locP1&&loc+up!=junction1Post) // if so
colour it
```

```
  {
   locP4=locP3;
   locP3=locP2;
   locP2=locP1;
   locP1=loc;
   loc=loc+up;
   // handTrace.pixels[loc]=color(r,g,b);
   edge[k] = loc;
   k++;

   if (junction==true) //test if previos position was junction and remembers position after last 3 junctions
   {
    junction3Post=junction2Post;
    junction2Post=junction1Post;
    junction1Post=loc;
    junction=false;
   }
  }
  if (firstSearch==false&&loc==maxPoint)
  {
   complete=true;
  }


  //right

  //boundary neighbour check
  boundaryNeighbours(loc);// checks how many neighbour pixels are also edge pixels above,below left and right
(max4)
  if (boundaryNeighbours>2)
  {
   junction3=junction1; //3rd previous junction
   junction3Prev=junction1Prev;//pixel before 3rd previous junction
   junction2=junction1; //2nd previous junction
   junction2Prev=junction1Prev;//pixel before 2nd previous junction
   junction1=loc; //pixel is a juncton
   junction1Prev=locP1; //edge pixel before first junction pixels
   //      r= (int) random(0,255);
   //      g= (int) random(0,255);
   //      b= (int) random(0,255);
   junction = true;
  }
```

```
  //right
  boundaryNeighbours(loc+right);//tests for dead end (if<2)
  boundaryBin(loc+right); //check if right pixel is part of edge
  if (boundaryBin==true&&boundaryNeighbours>1&&loc+right!=locP1&&loc+right!=junction1Post) // if so
colour it
  {
   locP4=locP3;
   locP3=locP2;
   locP2=locP1;
   locP1=loc;
   loc=loc+right;
   // handTrace.pixels[loc]=color(r,g,b);
   edge[k] = loc;
   k++;
   change=true;
   prnt(loc,i);
   if (junction==true) //test if previos position was junction and remembers position after last 3 junctions
   {
    junction3Post=junction2Post;
    junction2Post=junction1Post;
    junction1Post=loc;
    junction=false;
   }
  }
  if (firstSearch==false&&loc==maxPoint)
  {
   complete=true;
  }

  //down

  // boundary neighbour check
  boundaryNeighbours(loc);// checks how many neighbour pixels are also edge pixels above,below left and right
(max4)
  if (boundaryNeighbours>2)
  {
   junction3=junction1; //3rd previous junction
   junction3Prev=junction1Prev;//pixel before 3rd previous junction
   junction2=junction1; //2nd previous junction
   junction2Prev=junction1Prev;//pixel before 2nd previous junction
   junction1=loc; //pixel is a juncton
   junction1Prev=locP1; //edge pixel before first junction pixels
```

```
    //      r= (int) random(0,255);
    //      g= (int) random(0,255);
    //      b= (int) random(0,255);
     junction = true;
   }


   //down
   boundaryNeighbours(loc+down);//tests for dead end (if<2)
   boundaryBin(loc+down); //check if down pixel is part of edge
   if (boundaryBin==true&&boundaryNeighbours>1&&loc+down!=locP1&&loc+down!=junction1Post) // if so
colour it
   {
    locP4=locP3;
    locP3=locP2;
    locP2=locP1;
    locP1=loc;
    loc=loc+down;
    //handTrace.pixels[loc]=color(r,g,b);
    edge[k] = loc;
    k++;
    change=true;
    prnt(loc,1);
    if (junction==true) //test if previos position was junction and remembers position after last 3 junctions
    {
     junction3Post=junction2Post;
     junction2Post=junction1Post;
     junction1Post=loc;
     junction=false;
    }
   }
   if (loc==maxPoint)//search is complete
   {
    complete=true;
   }

   if (loc==locP4||loc==junction1||loc==junction2) //search is stuck so start again
   {
    complete=true;
   }
  }
```

```
//displays averaged trace array
int f=25; //filter size
int freq=15; //frequency of samples from fitlered array (edgeSmooth)


beginShape();

for (int m=(f-1)/2;m<k-(f+1)/2;m=m+freq)
{
 aveX=0;
 aveY=0;

 for (int n=-1*(f-1)/2;n<(f+1)/2;n++)
 {
  aveX=edge[m+n]%w+aveX;
  aveY=edge[m+n]/w+aveY;
 }
 aveX=aveX/f;
 aveY=aveY/f;
 edgeSmooth[m]=aveX+aveY*w;


 //prevSmooth=edgeSmooth[m]
 stroke (255);
 fill (255);




 xPointP2=xPointP1;   //2 previous x and y positions to find angles
 yPointP2=yPointP1;
 xPointP1=xPoint;
 yPointP1=yPoint;
 xPoint=edgeSmooth[m]%w;
 yPoint=edgeSmooth[m]/w;

 float  ab = dist (xPoint,yPoint,xPointP1,yPointP1);   //finds angle between three previous points
 float  bc = dist (xPointP1,yPointP1,xPointP2,yPointP2);
 float ac = dist (xPoint,yPoint,xPointP2,yPointP2);
 angleB = acos((ab*ab+bc*bc-ac*ac)/(2*ab*bc));
```

```
   fill(10);
   curveVertex (xPoint*800/w,yPoint*600/h); //draws line between every x points in array



   centroidX=centroidX+xPoint;   //finds average x value of vertices for centroid
   centroidY=centroidY+yPoint;  // finds ave y""""
   counter++;

   //finds fingertips
   if(angleB<2.2&&yPointP1<maxPoint/w-20)
   {

     int acAveX=(xPoint+xPointP2)/2;   // average of x coords of point next to identified angle point
     int acAveY= (yPoint+yPointP2)/2;// average of y coords of point next to identified angle point


     int P1DiffX = 5*(xPointP1-acAveX)/2; //find the difference between the middle point and the points on either
side
     int P1DiffY = 5*(yPointP1-acAveY)/2;

     int testX=xPointP1+P1DiffX; //find pixel to test for fingertip or valley
     int testY=yPointP1+P1DiffY;

     float bl = blue (handFat.pixels[testX+testY*w]);// test the pixel a differences length past the P1 ie further in
to the hand for valleys and further out from the finger for fingertips


     if (m<pass+40&&flagTip==1&&fin>0) // remembers history of fingertip points that are close together
     {

       xPointP12=xPointP11;
       xPointP11=xPointP1;
       yPointP12=yPointP11;
       yPointP11=yPointP1;

       testXP2=testXP1;
       testXP1=testX;
       testYP2=testYP1;
       testYP1=testY;
```

```
 avePass++;
}



if (m>pass+40) //prevents more than one point being identified every x coordinates along edge
{
 if(avePass>1) //find average of fingertip pixels that are close together
 {
  xTipAve= (tipX+xPointP11+xPointP12)/avePass;
  yTipAve= (tipY+yPointP11+yPointP12)/avePass;
  testXAve= (testX1+testXP1+testXP2)/avePass;
  testYAve= (testY1+testYP1+testYP2)/avePass;

  fingertip[fin-1]=xTipAve+yTipAve*w;
  fingerDir[fin-1]=testXAve+testYAve*w;
  avePass=1;
  xPointP12=0;
  xPointP11=0;
  yPointP12=0;
  yPointP11=0;
   testXP2=0;
 testXP1=0;
testYP2=0;
testYP1=0;



 }

 if (bl==0&&fin<5) //finds fingertips
 {
  tipX=xPointP1;
  tipY= yPointP1;
  fingertip[fin] = xPointP1+yPointP1*w;
  testX1=testX;
  testY1=testY;
  fingerDir[fin]= testX+testY*w;
  fin++;
  flagTip=1;
 }
```

```
      if(bl==255&&val<4&&flagTip==1) //finds valleys
      {
        valley[val] = xPointP1+yPointP1*w;
        val++;
        flagTip=0;
      }
      pass=m;
    }
  }
 }

  curveVertex(edgeSmooth[(f-1)/2]%w*800/640,edgeSmooth[(f-1)/2]/w*600/480); //draws line between first and
last point
  endShape(CLOSE);
  centroidX=centroidX/counter*800/640;
  centroidY=centroidY/counter*600/480;
  fill(255,0,0);
  ellipse(centroidX,centroidY,10,10);

  for(int i=0; i<fin+1;i++) //displays fingertips and direction
  {
    noStroke();
    fill(255,0,0);
    ellipse (fingertip[i]%w*800/w,fingertip[i]/w*600/h,10,10);
   fill(0);
    ellipse (fingertip[i]%w*800/w,fingertip[i]/w*600/h,6,6);

    if(fingertip[i]>0)
    {
     stroke(170,10,200);
    line(fingertip[i]%w*800/w,fingertip[i]/w*600/h,fingerDir[i]%w*800/w,fingerDir[i]/w*600/h);
    float distanceF = (dist(centroidX,centroidY,fingertip[i]%w*800/w,fingertip[i]/w*600/h)/230)*0.8; //normalised
between 0.1 and 0.9 for NN
    fingertipNN[i]=distanceF;

    stroke(0,255,255);
    line(centroidX,centroidY,fingertip[i]%w*800/w,fingertip[i]/w*600/h);
    }
    fill(0,255,255);
     textFont(ft,18);
    text (i+1,(fingerDir[i]%w+10)*800/w,(fingerDir[i]/w)*600/h);
```

```
  fill (100,100,180);
  ellipse (fingerDir[i]%w*800/w,fingerDir[i]/w*600/h,2,2);
 }

 for(int i=0; i<val+1;i++) //displays valleys
 {
  if(valley[i]>0)
  {
  fill(230,180,40);
  noStroke();
  ellipse (valley[i]%w*800/w,valley[i]/w*600/h,5,5);
  fill(0);
  ellipse (valley[i]%w*800/w,valley[i]/w*600/h,2,2);
  fill(0,255,0);
  text (i+1,(valley[i]%w+10)*800/w,(valley[i]/w)*600/h);
  stroke(0,255,0);
  line(centroidX,centroidY,valley[i]%w*800/w,valley[i]/w*600/h);
  float distanceV = (dist(centroidX,centroidY,valley[i]%w*800/w,valley[i]/w*600/h)/230)*0.8; //normalised
between 0.1 and 0.9 for NN
  valleyNN[i]=distanceV;
  }
 }

 if(analyse==true) //runs nerual network once
 {

neuralNetwork(fingertipNN[0],fingertipNN[1],fingertipNN[2],fingertipNN[3],fingertipNN[4],valleyNN[0],valley
NN[1],valleyNN[2],valleyNN[3]);
 analyse =false;
 }
 //clears arrays storing tips and valleys
  for(int i=0; i<fin+1;i++)
 {

  fingertip[i]=0;
  fingertipNN[i]=0;
 }
 for(int i=0; i<val+1;i++)
 {
  valley[i]=0;
  valleyNN[i]=0;
 }
```

```
}
////////////////////handFat/////////////////////

void handFat(int loc)
{
 boundary(loc);
 if(boundary==true)
 {
  handFat.pixels[loc]=color(0);
 }


  int expansion =7 ;
 if(boundary==false&&loc%w>(expansion-1)/2&&loc/w>(expansion-1)/2) //fattens hand
 {
  for (int x=-1*(expansion-1)/2;x<(expansion+1)/2;x++)
  {
   for (int y=-1*(expansion-1)/2;y<(expansion+1)/2;y++)
   {

    handFat.pixels[loc+x+y*w]=color(255,255,255);
    maxPoint=loc; //this should be loc+x+y*w not sure why it works like this
   }
  }
 }
}
//////////////////////////Smooth/////////////////


void smoothPixels(int x3, int y3,int loc,float bl, float re)
{

 if (x3>(xLens-1)/2&&y3>(yLens-1)/2&&x3<w-(xLens-1)/2-1&&y3<h-(yLens-1)/2-
1&&bl>250&&re>1)//ensures averaging lense stays within boundary of image
 {


  //clears value for active neighbours
  int pixelQuant=0; // number of active pixels to use as divisor

  //test for active pixels in a 5*5 squaree
  int totX = 0;
```

```
int totY =0;
int smoothedLoc=0;
int xAvg=0;
int yAvg=0;


for (int x=0;x<xLens;x++)
{
 for (int y=0;y<yLens;y++)
 {

   int locTemp = loc+x+y*w-(xLens-1)/2-((yLens-1)/2)*w; //temp position of pixel in square to be tested

   float slTemp=blue (depthData.pixels[locTemp]);
   float reTemp=red (depthData.pixels[locTemp]);
   if (slTemp>250&&reTemp>1&&(abs ((loc%w)-(locTemp%w))+ abs((loc/w)-(locTemp/w))  )<(xLens-
1)/2)//sets lens shape

   {

    totX=totX+x;
    totY=totY+y;
    pixelQuant++;
    //      if (loc==w/2+h/2*w)
    //      {
    //        handSmooth.pixels[locTemp]=color(0,255,0);//shows lens shape
    //      }
   }
 }
}

xAvg=totX/pixelQuant;
yAvg=totY/pixelQuant;
smoothedLoc = loc+xAvg+yAvg*w-(xLens-1)/2-((yLens-1)/2)*w;
if(pixelQuant<=proxFilter)
{
 handSmooth.pixels[smoothedLoc]=color(0); // clears pixels with fewer than 36 neighbours
}
else
{
 handSmooth.pixels[smoothedLoc]=color(0,255,0);//draws smoothed hand
 // handSmooth.pixels[smoothedLoc-w]=color(0,255,0);
```

```
    //handSmooth.pixels[smoothedLoc+w]=color(0,255,0);
    //handSmooth.pixels[smoothedLoc-1]=color(0,255,0);
    //handSmooth.pixels[smoothedLoc+1]=color(0,255,0);
    //  handSmooth.pixels[smoothedLoc-w-1]=color(0,255,0);
    //  handSmooth.pixels[smoothedLoc+w+1]=color(0,255,0);
    //  handSmooth.pixels[smoothedLoc+w-1]=color(0,255,0);
    //  handSmooth.pixels[smoothedLoc-w+1]=color(0,255,0);
   }
  }
}
//////////////////////xProjection//////////////////////
void xProjection (int x,int y,float bl,float re)
{
  if (y==h-1&&x==w-1) // clears array caontiang x values of projection
  {
    for (int i=0; i<xPos.length;i++)
    {
      xPos[i]=0;
    }
  }
  int sumXPosArray=0;//sum of values in array


  if(bl>250&&re>1)
  {

    xOfXProj++;
  }


  if(x==w-1)
  {
    for (int i=0; i<xPos.length-1;i++) // stores new x value in array and shift previous x values down one place
    {
      xPos[i] = xPos[i+1];
    }

    xPos[xPos.length-1] = xOfXProj;

    for (int i=0; i<xPos.length;i++) // sum array
    {
      sumXPosArray+=xPos[i];
```

```
   }
   xProjXAve=sumXPosArray/xPos.length; //mean of x values in projection over xPos.length

    if (maxXOfXProj<xProjXAve) // finds maximum x value of x projection
   {
    maxXOfXProj=xProjXAve;
    maxYOfXProj=yOfXProj-(xPos.length-1)/2;
   }
   if (relMinXOfXProj>xProjXAve&&y>tempMaxYOfXProj&&y<h-(xPos.length-1)/2)//finds relative minimum
x value of x projection to isolate hand from arm
   {
    relMinXOfXProj=xProjXAve;
    relMinYOfXProj=yOfXProj-(xPos.length-1)/2;
   }




   if(y>(xPos.length-1)/2) //keeps array in bounds
   {
    xProjection.pixels[xProjXAve+yOfXProj*w-(xPos.length-1)/2*w] = color(0,255,0); //displays average of x
positions of pixels contained in array
   }

   xOfXProj=0;//reset x value
   yOfXProj=y; //next y position
  }
}


void xProjectionMax()
{

  fill(255,0,0);
  ellipse(maxXOfXProj*400/w,maxYOfXProj*300/h+300,10,10);
  tempMaxXOfXProj=maxXOfXProj; //stores previus x value of maximum for use in midPoint of hand finder
  maxXOfXProj=0; //clears y coord of maximum x
  tempMaxYOfXProj=maxYOfXProj; //stores previus x value of maximum for use in midPoint of hand finder
  maxYOfXProj=0; //clears y coord of maximum x
}

void  xProjectionRelMin()
```

```
{
 fill (0,0,255);
 ellipse(relMinXOfXProj*400/w,relMinYOfXProj*300/h+300,10,10);
 tempRelMinYOfXProj=relMinYOfXProj;
 relMinXOfXProj=w; //clears y coord of maximum x
 relMinYOfXProj=h; //clears y coord of maximum x
}
//////////////////////yProjection//////////////////////
void yProjection (int x,int y,float bl,float re)
  {

    if (y==tempRelMinYOfXProj-1&&x==w-1) // clears array caontiang y values of projection
    {
      for (int i=0; i<yPos.length;i++)
      {
        yPos[i]=0;
      }
    }
    int sumYPosArray=0;//sum of values in array


    if(bl>250&&re>1&&y<tempRelMinYOfXProj)
    {

      yOfYProj++;
    }


    if(tempRelMinYOfXProj>(xPos.length-1)/2&&tempRelMinYOfXProj<h-(xPos.length-
1)/2&&y==tempRelMinYOfXProj-1)
    {

      for (int i=0; i<yPos.length-1;i++) // stores new y value in array and shift previous y values down one place
      {
        yPos[i] = yPos[i+1];
      }

      yPos[yPos.length-1] = yOfYProj;
      println(yOfYProj);

      for (int i=0; i<yPos.length;i++) // sum array
      {
```

```
      sumYPosArray= sumYPosArray+ yPos[i];
    }



    yProjYAve=sumYPosArray/yPos.length; //mean of y values in projection over yPos.length

     if (maxYOfYProj<yProjYAve)
    {
     maxYOfYProj=yProjYAve;
     maxXOfYProj=xOfYProj-(yPos.length-1)/2;
    }



    if(x>(yPos.length-1)/2) //keeps array in bounds
    {

     yProjection.pixels[xOfYProj+yProjYAve*w-(yPos.length-1)/2] = color(0,255,0); //displays average of y
positions of pixels contained in array
    }


    yOfYProj=0;//reset x value
    xOfYProj=x; //next y position
   }

  }

 void yProjectionMax()
  {

   fill(255,0,0);
   ellipse(maxXOfYProj*400/w,maxYOfYProj*300/h,10,10);
   tempMaxYOfYProj=maxYOfYProj; //stores previus y value of maximum for use in midPoint of hand finder
   maxYOfYProj=0; //clears y coord of maximum Y
   tempMaxXOfYProj=maxXOfYProj; //stores previus x value of maximum for use in midPoint of hand finder
   maxXOfYProj=0; //clears X coord of maximum Y
  }
```

11. Works Noted

Davison, Dr. Andrew. "Kinect Chapters 1 and 2. Kinect Imaging." *Five Dots*. N.p., n.d. Web. 1 Feb. 2014. <http://fivedots.coe.psu.ac.th/~ad/jg/nui13/KinectImaging.pdf>.

"3D Sensing Technology Solutions - PrimeSense." *PrimeSense*. N.p., n.d. Web. 22 Mar. 2014. <http://www.primesense.com/>.

Ballard, Dana Harry, and Christopher M. Brown. *Computer vision*. Englewood Cliffs, N.J.: Prentice-Hall, 1982. Print.

Barry, Paul. *Head first Python*. Farnham: O'Reilly, 2010. Print.

"CLNUI 4 Java (Kinect) - Processing Forum." *Processing Forums*. N.p., n.d. Web. 22 Mar. 2014. <http://forum.processing.org/one/topic/clnui-4-java-kinect.html>.

Earnshaw, Rae A.. *Fundamental algorithms for computer graphics*. Berlin: Springer-Verlag, 1985. Print.

Eckel, Bruce. *Thinking in Java*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2000. Print.

Group, Jiayang Liu, Zhen Wang, Jehan Wickramasuriya and Venu Vasudevan . "uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications ." *Rice.EDU*. N.p., n.d. Web. 23 Sept. 2013. <http://www.ruf.rice.edu/~mobile/publications/liu09percom.pdf>.

Judson, Tjohm . "OpenNI to Max/MSP via OSC." *tohm judson*. N.p., n.d. Web. 22 Mar. 2014. <http://tohmjudson.com/?p=30.html>.

"Kinect for Windows features." *Product Features*. N.p., n.d. Web. 22 Mar. 2014. <http://www.microsoft.com/en-us/kinectforwindows/discover/features.aspx>.

Lutz, Mark. *Learning Python*. 4th ed. Sebastopol, Calif.: O'Reilly Media, 2009. Print.

Richert, Willi, and Luis Pedro Coelho. *Building Machine Learning Systems with Python*. Birmingham: Packt Publishing, 2013. Print.

"Sign Language Interpreter app." - *Google+ Help*. N.p., n.d. Web. 22 Mar. 2014. <https://support.google.com/plus/answer/2990988?hl=en>.

University, Singularity. "Smart Gloves Turn Sign Language Gestures Into Vocalized Speech." *Forbes*. Forbes Magazine, 20 Sept. 2012. Web. 22 Mar. 2014. <http://www.forbes.com/sites/singularity/2012/09/20/smart-gloves-turn-sign-language-gestures-into-vocalized-speech/>.

Zinza, Jason E., Xiaohong Fang, and James Sbarra. *Master ASL!: level one*. Burtonsville, Md.: Sign Media, Inc., 2006. Print.
Zukowski, John. *Java 6 platform revealed*. Berkeley, CA: Apress ;, 2006. Print.
Campbell-Swinton, A. A. (1908-06-18). "Distant Electric Vision (_rst paragraph)". Nature 78 (2016): 151
Ekaterini Stergiopoulou, Nikos Papamarkos: Hand gesture recognition using a neural network shape _tting technique. Eng. Appl. of AI 22(8): 1141-1158 (2009)
Overview of Binomial Filters, Konstantinos G. Derpani, Department of Computer Science and Engineering York University March 5, 200
Stathakis, D. International Journal of Remote Sensing Vol. 30, No. 8, 20 April 2009, 2133_2147 How many hidden layers and nodes?
[5] Neural Networks for Pattern Recognition Christopher M. Bishop Oxford University Press (1995)
Arthur Earl Bryson, Yu-Chi Ho (1969). Applied optimal control: optimization, estimation, and control. Blaisdell Publishing Company or Xerox College Publishing. pp. 481.
The Back Propogation Algorithm. Robert Gordon University online course materials http://www4.rgu.ac.uk/_les/chapter3%20-%20bp.pdf.
Muang, Tin. Real-Time Hand Tracking and Gestures Recognition System Using Neural Networks. World Academy of Science and Technology 50 2009.
Elmezain, Ayoub, Appendrtodt and Michaelis. A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition. World Academy of Science and Technology 41 2008.
Segan, J, Controlling computers with gloveless gestures in Virtual Reality Systems. 1993
Hunter, E. Posture estimation in reduced model gesture imput systems, Proceedings of International Workshop on Automated Face

and Gestures Recognition, June 1995.
Yoon, Soh, Bae, Yang. 2001. Hand Gesture Recognition using combined features of location, angle and velocity. Pattern Recognition 34 (7) 1491 - 1501.
[13] R. Jain, R. Kasturi, B.G. Schunck. Machine Vision. McGraw-Hill, 1995
Search Algorithm and Edge Detector Combination for Outdoor Trajectory Planning,Kamil .idek,Transfer inovácií 18/2010
Emmory, Karen. "Language, Gesture, And Space" (1995): 82.
"Histogram Comparison¶." *Histogram Comparison — OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 1 Apr. 2014. <http://docs.opencv.org/doc/tutorials/imgproc
"Basic Structures¶." *Basic Structures — OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 1 Apr. 2014. <http://docs.opencv.org/modules/core/doc/

"About Pythonâ„¢ | Python.org." *Python.org*. N.p., n.d. Web. 2 Apr. 2014. <https://www.python.org/about/>.

"Basic StructuresÂ¶." *Basic Structures â€" OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 1 Apr. 2014. <http://docs.opencv.org/modules/core/doc/basic_structures.html>.

"Basic StructuresÂ¶." *Basic Structures â€" OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 2 Apr. 2014. <http://docs.opencv.org/modules/core/doc/basic_structures.html>.

Butterworth, Rod, and Mickey Flodin. "history of sign language." *berkley publishing group*. berkley publishing group, n.d. Web. 2 Mar. 2014. <https://www2.uic.edu/stud_orgs/cultures/daa/ASLHistory.html>.

"Histogram ComparisonÂ¶." *Histogram Comparison â€" OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 1 Apr. 2014. <http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html>.

"Histogram ComparisonÂ¶." *Histogram Comparison â€" OpenCV 2.4.8.0 documentation*. N.p., n.d. Web. 2 Apr. 2014. <http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html>.

Michaell, Ross. "How many deaf people are there in the United States." *How many deaf people are there in the United States*. N.p., n.d. Web. 2 Apr. 2014. <http://research.gallaudet.edu/Demographics/deaf-US.php>.

"OpenCV." *Wikipedia*. Wikimedia Foundation, 31 Mar. 2014. Web. 2 Apr. 2014. <http://en.wikipedia.org/wiki/OpenCV>.

"The Five Senses - Lesson 3: Hearing." *Paso Partners*. N.p., n.d. Web. 2 Apr. 2014. <http://www.sedl.org/scimath/pasopartners/senses/lesson3.html>.

"Willow Garage." *Willow Garage*. N.p., n.d. Web. 2 Apr. 2014. <http://www.willowgarage.com/>.