Hand Gesture Recognition of Numbers in American Sign Language

Caleb Woo, cdw75
Clifford Chou,
Oct. 29, 2013
ECE 5470

Abstract

This project approaches the classification aspect of gesture recognition. It recognizes numbers 1 through 10 in American Sign Language (ASL). The general algorithm first thresholds the greyscale image. It then calculates center of intensity (COI) in order to predict locations for fingertips. At these locations it draws bounding boxes as a mask and counts how many hand pixels are in each. It can then determine the status of the fingers and thus recognize the ASL number. However, due to problems with lighting and the general dataset other considerations such as thresholding and limiting our region of interest wet taken. This gesture recognition has many applications such as an interactive command communication with robots and electronic devices, security passwords, and the obvious application for translating ASL.

1 Introduction

Gesture recognition is an important subject of computer vision that could potentially serve a variety of applications. Gesture recognition could be implemented to allow robots robots to perform commands given in gestures. Another important application is the ability to read simple gestures in American Sign Language (ASL). In conjunction with the application for robots and electronic devices, it allows people who are deaf and/or mute interactive capabilities with modern technology in an increasing interconnected world of advanced telecommunications.

Gesture recognition comprises of two main steps: detection/segmentation and classification. This project focuses on the classification of gestures. The detection of hands in a practical real-world environment is a challenging problem that is not addressed in this project. In this project, images of hands in a controlled environment depict the numbers 1 through 10 in ASL (see Fig. 5). Our project will take these hand images and determine the number. The general method of recognizing the ASL numbers uses a simple algorithm. First, the greyscale images are thresholded, using an iterative thresholding approach, to produce binary images. The center of intensity (COI) is computed and used as a reference point to predict bounding boxes around the fingertips. In each bounding box, the number of hand pixels is computed and compared against a threshold to determine the finger status. All finger status' are used to classify and recognize the ASL number.

The following section, Section 2, reviews the previous work in gesture recognition directly related to the classification aspect. Section 3 discusses the overall design of the moment-based algorithm for vessel detection. It will detail the main steps: thresholding, centroid calculation, bounding box construction, and classification. In Section 5, the design of the experiments, including training and testing sets, is detailed. It also details modifications made during training to the design of the experiment and the project due to new problems. In Section 6, the experimental results are discussed. Section 7 concludes the paper with a discussion on the overall effectiveness and efficiency of the program with suggestions for future applications.

2 Literature Review

The project is primarily based off the algorithm used by N. Dhruva and his team which is described in "Novel Algorithm for Image Processing Based Hand Gesture Recognition and Its Application in Security." Their concept of identifying American Sign Language through identifying the fingertips is the core of our algorithm. They applied this through a security concept while we expanded this to read the hand at anytime. They used a very controlled environment while we allowed for a slightly different algorithm to account for potential imaging issues. On top of that, we explored many other algorithms and papers to come up with ideas to add to this base idea. "A FAST ALGORITHM FOR VISION-BASED HAND GESTURE RECOGNITION FOR ROBOT CONTROL" also used gesture recognition for a greater purpose. They had a very simple algorithm to identify whether fingers were raised or not to use for robot control. We started our project with this concept but added more: from counting the number of fingers raised to being able to identify which fingers were raised, to being able to identify what the fingers represented. Both these papers were the starting points and the core of the algorithm we used. We added a bit more functionality for a less constrained world but for only the sake of classification.

3 Algorithm and Program Description

The algorithm presented here is largely similar to the work of Dhruva et al[1]. The central idea of the algorithm is it finds a reference point around the center of the palm of the hand to predict approximate fingertip locations. In order to check if each specific finger is up making a gesture, it uses these predicted locations to create bounding boxes and calculates the number of finger pixels inside. If the number of finger pixels satisfies a set threshold, then the specific finger is considered to be up. With the knowledge of which specific fingers are up the algorithm can easily classify which number gesture is made in ASL. The figures below show the process of a hand image through the algorithm.

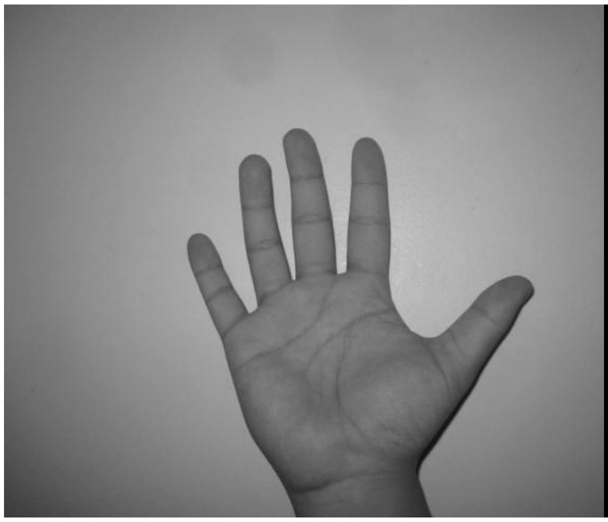


Figure 1: Greyscale Image

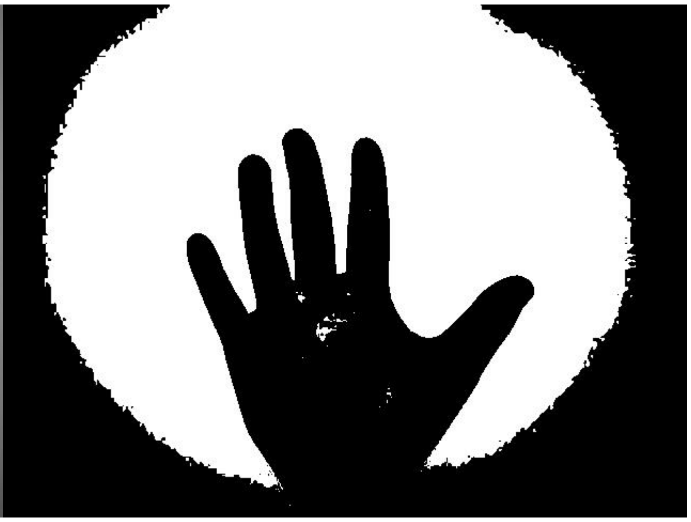


Figure 2: Thresholded Image

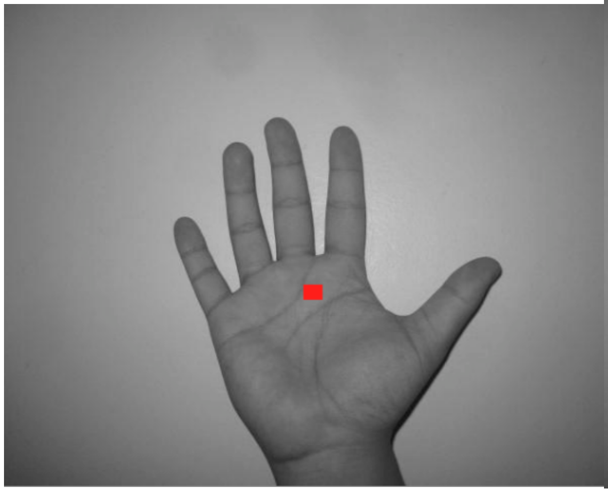


Figure 3: Center of Intensity (COI)

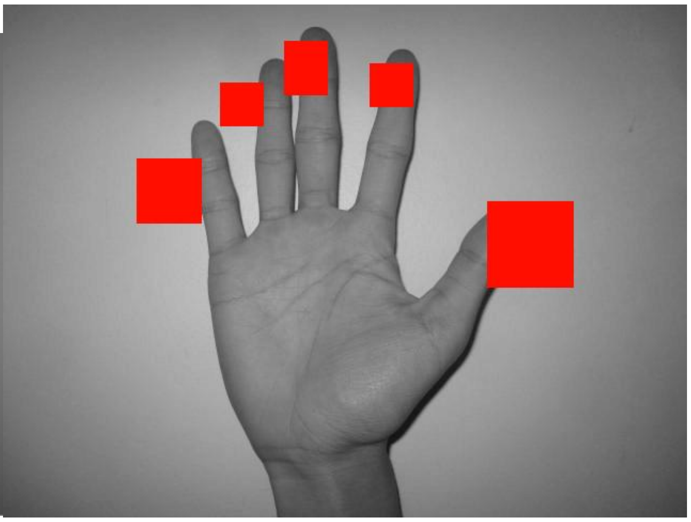


Figure 4: Bounding Boxes

3.1 Iterative Thresholding

The first step of the algorithm is to separate the hand from the background. Given that the background is consistently white and distinct from a hand, it is easy to separate hand pixels from background pixels by simple iterative thresholding. The method of iterative threshold selection determines a threshold by repeatedly calculating a threshold until it converges. It starts off with a guess for a threshold, separating the image pixel distribution into two regions. It then calculates the average pixel value for each region and selects a new threshold as the average of the two means. Again, the process of calculating averages for the two new regions is repeated. This whole process is repeated until the averages of the two regions converge and therefore the threshold converges.

A histogram is constructed for ease of calculation of the averages of pixel values. To construct it, the program loops through all pixels of the image and increments a bin representing that pixel value. Here is the complete method:

1. Construct histogram of pixel values for the image
2. Get an initial threshold from command line. Default is average pixel value.
3. The threshold splits the histogram into two regions R1 and R2. Calculate average pixel values in both regions. For a region R in the histogram extending from pixel value p1 to pixel value p2, the average pixel value is

$$\text{average} = \frac{\text{hist[p1]*p1} + \text{hist[p1+1]*(p1 + 1)} + ... + \text{hist[p2 - 1]*(p2 - 1)} + \text{hist[p2]*p2}}{\text{hist[p1]} + \text{hist[p1 + 1]} + ... + \text{hist[p2 - 1]} + \text{hist[p2]}} \tag{1}$$

4. With the averages for each region avg1 and avg2 calculate the new threshold.

$$\text{threshold} = \frac{\text{avg1 + avg2}}{2} \tag{2}$$

   If there is a relatively small change (below a threshold for difference in successive iterations) then the threshold is the final threshold. Otherwise, the whole process repeated starting from step 3.

The result is now a binary image of pixel values 0 and 255. Hand pixels, which are assumed to be below the threshold, are set to be 255 and background pixels, which are assumed to be above the threshold, are set to 0.

3.2 Centroid Calculation

Using the binary image, the center of mass (COM), or rather the center of intensity (COI), is calculated for the image. Zeroth and first order moments compute the COI. Moments are weighted averages of pixel intensities in an image. The generalized equation of an n $(p+q)$ order moment of a two-dimensional image is given below:

$$M_{pq} = \sum_x \sum_y x^p y^q f(x,y) \tag{3}$$

The zeroth order moment computes the total intensity of the image. It is a simple discrete integral (sum).

$$M_{00} = \sum_x \sum_y f(x,y) \tag{4}$$

The first order moment computes the distributions of the image in the x and y directions.

$$M_{10} = \sum_x \sum_y x f(x,y) \tag{5}$$

$$M_{01} = \sum_x \sum_y y f(x,y) \tag{6}$$

The coordinates of the COI are computed using these moments:

$$\bar{x} = \frac{M_{10}}{M_{00}},\ \bar{y} = \frac{M_{01}}{M_{00}} \tag{7}$$

The COI calculation is an essential prerequisite to the construction of bounding boxes according to predicted locations of fingertips. The coordinates of fingertips should always maintain the same distance relative to the COI as the reference point. The COI calculation helps to adjust the construction of bounding boxes given translations in the horizontal direction. Translations in the vertical direction are problematic because the reduction of hand pixels or introduction of wrist pixels of probable equal intensity to hand pixels would distort the COI position relative to the palm of the hand. The COI is used as a reference point for construction of bounding boxes around the fingertips.

3.3 Bounding Boxes Construction

Training provides standard coordinates for the fingertips relative to the reference point (COI). Bounding boxes with length and width specified by user input are constructed centered at each of the coordinates in a new temporary image structure. This temporary image serves as a mask for the original image. For finger detection, the algorithm will search for hand pixels in the original image and nonzero pixels in the mask image. Region growing, modified to move from right column to left column instead of bottom row to top row, labels each of these bounding boxes with an incrementer. The rightmost box corresponding to the thumb has pixel values of 1 and the leftmost box corresponding to the pinky has pixel values of 5.

Region growing technique uses a simple recursive method of labeling. First, a new temporary image file is created for the mask containing the bounding boxes. The program loops through each pixel of the binary image and checks if the pixel is an object (temporary image pixel not equal to zero) and if the pixel is unlabeled (mask image pixel equal to zero). If condition is met, the subprocedure setlabel is called. In this loop, the global variable l increments after setlabel is called so the next object has a new label.

The subprocedure setlabel has inputs of the pixel coordinates and an integer for labeling. Automatically, the pixel on the mask image at those coordinates is labeled with the integer. However, setlabel also checks its four neighbors (not diagonal neighbors) one at a time for the same condition above. If a neighbor pixel in the temporary image is nonzero (an object pixel) and unlabeled in the mask image, setlabel is called at those coordinates with the same label. This pixel is now labeled and its neighbors are checked one at a time. This continues until all pixels of an object are labeled.

3.4 Classification

Looping through each pixel of the image, the algorithm checks for hand pixels in the original image and labeled pixels in the mask image containing the bounding boxes. There are five separate counters to keep track of the number of hand pixels in each bounding box. Inside a bounding box (nonzero pixel in mask image), the counter corresponding to the box label is incremented for each hand pixel. If the count for a box meets a certain threshold, then the finger is considered to be up. Integer variables are used to represent the status of a finger. TRUE (1) corresponds to up for a finger and FALSE (0) corresponds to down for a finger.

The algorithm checks for each combination of finger gestures according to ASL for numbers. Figure 1 below shows the ASL chart for numbers.

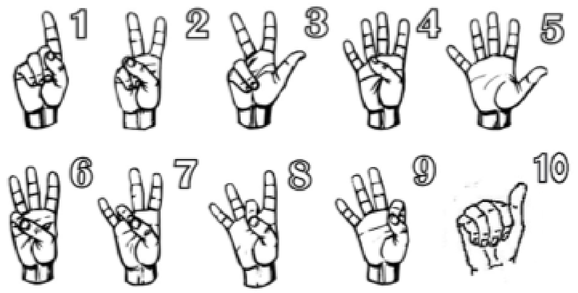


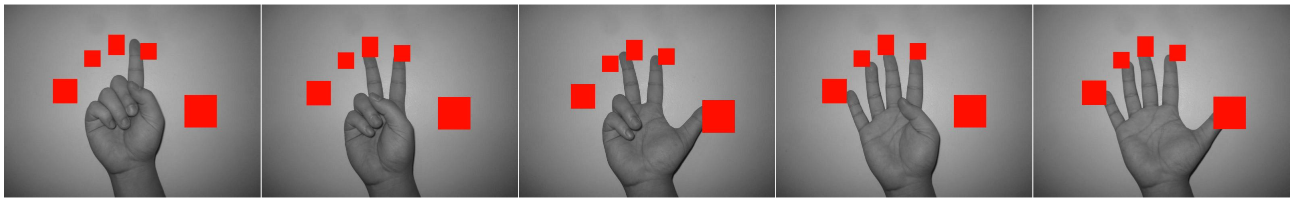
Figure 5: ASL chart of numbers 1-10 (from www.lifeprint.com)

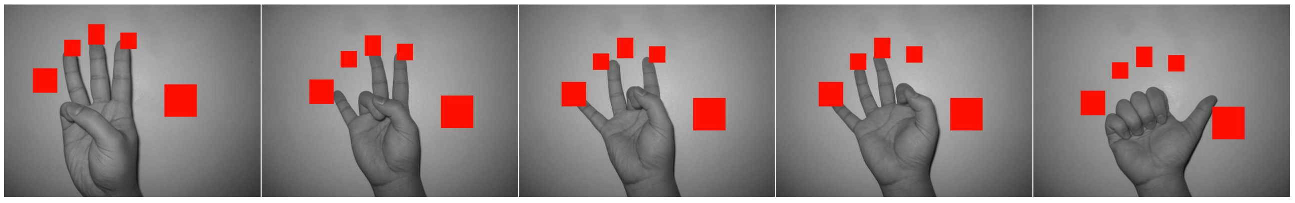
Table 1 below shows the values of each boolean variable for each number.

Table 1: ASL table of numbers 1-10

Number (1-10)	Thumb	Index	Middle	Ring	Pinky
1	FALSE	TRUE	FALSE	FALSE	FALSE
2	FALSE	TRUE	TRUE	FALSE	FALSE
3	TRUE	TRUE	TRUE	FALSE	FALSE
4	FALSE	TRUE	TRUE	TRUE	TRUE
5	TRUE	TRUE	TRUE	TRUE	TRUE
6	FALSE	TRUE	TRUE	TRUE	FALSE
7	FALSE	TRUE	TRUE	FALSE	TRUE
8	FALSE	TRUE	FALSE	TRUE	TRUE
9	FALSE	FALSE	TRUE	TRUE	TRUE
10	TRUE	FALSE	FALSE	FALSE	FALSE

Here is an example of the results of the program on a training set.





4 Experimental Design and Training

The datasets vary but maintain some important constraints. All original jpeg pictures were taken with the camera fixed on a table and the hand against a white/beige wall to maintain distance and angle orientation. This allows hand detection to become a simple thresholding problem and the main focus of the project is classification. Hands had slight variation in rotating towards or away from the camera. However, variation of angle orientation along the plane of the wall (rotating left or right), is not allowed. This problem is too difficult especially given that several palm orientation algorithms depend on finger detection and known status of fingers to calculate orientation. This defeats the purpose of our algorithm because our focus is to classify a hand without knowledge of the status of each finger. The program was tested on 10 sets for training and 10 sets for testing. A set contains all numbers 1 through 10 in ASL.

The algorithm has three main parameters to vary: length of bounding box, width of bounding box, and threshold for number of hand pixels in a bounding box. The locations of the bounding boxes relative to the reference point (COI coordinates) are fixed after training briefly with the datasets. The variation of length and width of the bounding boxes will be sufficient to cover variation among different hands assuming that fingertip positions have similar angle orientations from the COI. In addition, each finger has a personalized bounding box that may differ in length and width. Training is conducted on 6 sets of data, each dataset containing all 10 numbers represented in ASL.

Training Procedure:

1. For each training set, find the positions of the fingertips relative to the calculated COI.
2. Calculate the average relative position of the fingertips to each COI. The average relative positions will be used as the predicted coordinates for fingertips for every new image and new calculated COI.
3. The largest possible difference, below a threshold, between the average relative x position for a fingertip and the relative x positions for that finger in the training set becomes the new half width of the bounding box for that finger. There is a threshold to prevent overlap between neighboring fingers.
4. Similarly, the largest possible difference, below a threshold, between the average y position for a fingertip and the relative y positions for that finger in the training set becomes the new half length of the bounding box for that finger.
5. Various thresholds are tested to train the set and find the optimum threshold that correctly classifies most of the images.

During the training the original algorithm was modified according to problems with initial assumptions of the dataset. Certain standards of the datasets were also modified in order to avoid problems that would require other algorithms to rectify them.

4.1 Iterative Thresholding

Overhead lighting created fundamental problems with shadowing that severely disrupts the iterative thresholding process and incompletely separates the hand from the background. To account for this, the flash on the camera was used to prevent shadows from fingers. However, this created an almost Gaussian distribution of intensity surrounding the hand. Background pixels around the hand were much lower in intensity than background pixels at the center. This meant the result of iterative thresholding successfully separated the hand from the background but also created two backgrounds. In the resulting binary image, background pixels at the corners of the image were also considered to be binary high and equal in intensity to the hand pixels. However, these pixels are nowhere in the proximity of the bounding boxes and not a problem. It does cause problems with the centroid calculation of COI. The method modification will be elaborated in the next section.

4.2 Centroid Calculation

As mentioned, the "hand" pixels at the corners of the image distort the coordinates of the COI. Given that the camera is fixed, the distribution of light is consistent. Therefore, in the centroid calculation, all pixel values a certain horizontal distance away from the right and left hand sides are simply excluded from the calculation. The bounds of the for loop that goes through the image are modified to loop based on these new starting and end bounds.

4.3 Scaling

The distance from the camera to the hand is fixed. Therefore changes in hand size are due to inherent differences in hand sizes. In order to adjust our program for different hand sizes, scaling is introduced. For a new hand size, scaling would need to only scale the distance from the fingertip to the reference point assuming all hands have the same natural angle orientation between its fingers and the center of the hand. This means simply scaling the relative x and y coordinates of the fingertips. In addition, the size of the bounding boxes would need to be scaled. This simply requires scaling length and width.

To determine the necessary scaling factor to adjust our trained fingertip positions and bounding boxes, we use the ratio of width of the palm to the average palm width of our training sets. To approximate the width of the palm, we use the y coordinate of the COI. We move from the right and left hand sides of the image towards the COI at a fixed height above the bottom of the image to detect the first nonzero pixels which would be the two edge pixels of our palm. However, the problem with the hand pixels in the background due to camera flash again affect our results so we again use modified bounds to detect the first palm pixels to calculate width.

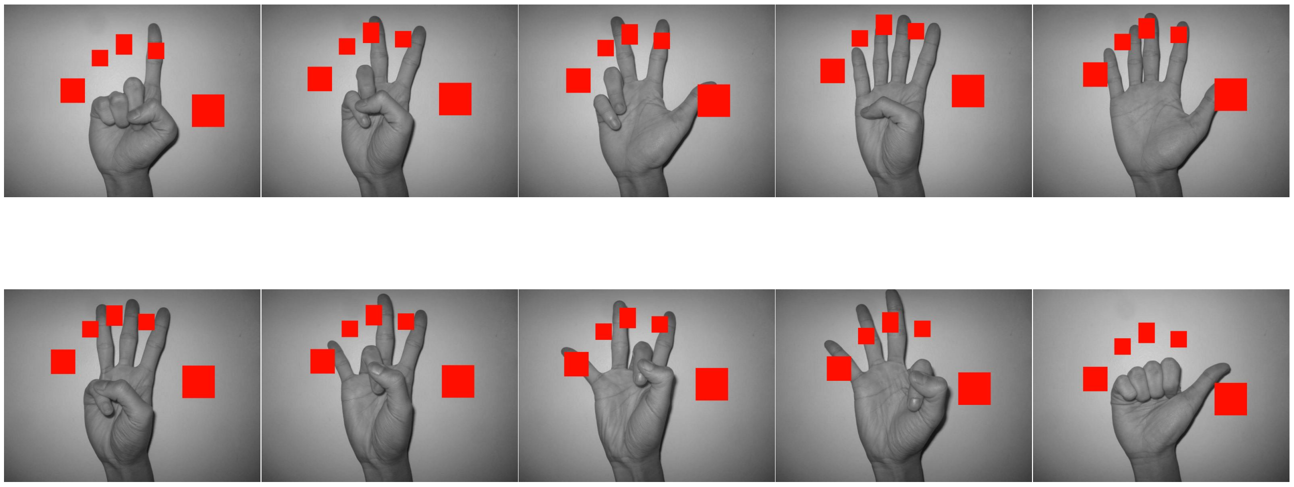
5 Experimental Results

The table below shows the results of the testing. The program was tested on 10 additional sets of data of our hands in the same background. There was allowed some variation in rotation.

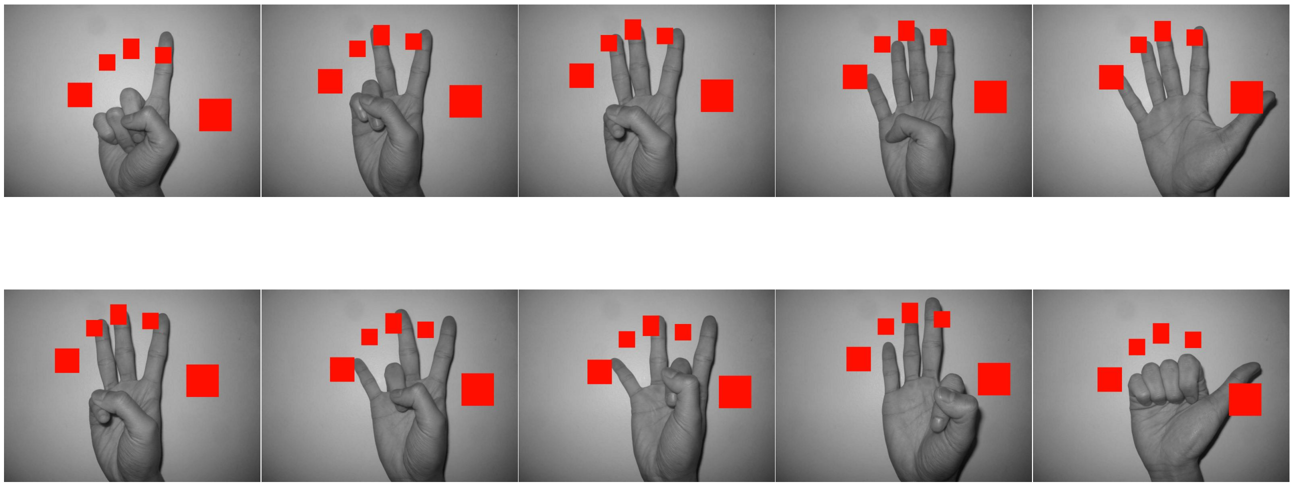
Table 2: Experimental Results

Gesture	Hits	Percentage Accurate
One	7	70%
Two	8	80%
Three	6	60%
Four	5	50%
Five	10	100%
Six	9	90%
Seven	8	80%
Eight	7	70%
Nine	7	70%
Ten	5	50%

Below is an example of one the test sets.



Below is another example of one the test sets.



The overall accuracy is 72%. Of the 28 errors, 13 were mismatches (incorrect classification) and 15 were just errors (unable to classify).
The most outstanding results was for the number 5 in which all fingers are up. This is not surprising given the large size of the bounding boxes and all fingers are up. There could be potential overlap of fingers into bounding boxes belonging to other fingers but we did not have the time to look. The program had the greatest difficulty with recognizing 3 and 4. This is a notable problem because the bounding boxes were very limited in width due to the close proximity of the neighboring fingers. Most of the errors for these numbers were not mismatches but rather unrecognizable patterns of fingers. The greatest source of error was definitely rotation along the plane of the wall (in the left and right directions). Although we did try to limit this, without the use of a structure like a stand to constrict our hand to a certain position, some variation in rotation is expected.

6 Conclusion

Program worked well considering the angle orientation variation. Given time, thresholding would have also included a better scheme based on edge detection to find the hand pixels without the problem of extra pixels in the background. We would have also tried to address the problem of angle orientation with an approach using iterative predictions of the hand but adjusting angle orientation. In addition, a better conclusion would have been written given more time.

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System.* Addison Wesley, Massachusetts, 2nd Edition, 1994.

Source Code Listing

```
/*******************************************************************/
/* Final Project: ASL Numerical Counting Gesture Recognition        */
/*******************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */
VisXfile_t *VXin,            /* input file structure                 */
           *VXout;           /* output file structure                */
VisXelem_t *VXlist,*VXptr;   /* VisionX data structure               */
VXparam_t par[] =            /* command line structure               */
{
{  "if=",   0,   " input file, vtpeak: threshold between hgram peaks"},
{  "of=",   0,   " output file "},
{  "d=",    0,   " min dist between hgram peaks (default 10)"},
{  "th=",   0,   " threshold of number of pixels inside bounding box"},
{  "l=",    0,   " length of bounding box"},
{  "w=",    0,   " width of bounding box"},
{  "-v",    0,   " (verbose) print threshold information"},
{   0,      0,   0} /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  DVAL   par[2].val
#define  TVAL   par[3].val
#define  LVAL   par[4].val
#define  WVAL   par[5].val
#define  VFLAG  par[6].val

/* create booleans */
#define true 1
#define false 0

#define trainedHandWidth 180

/* pulling these from within main method (to use in method setlabel) */
VisXimage_t im; // i/o image structure
VisXimage_t tm; // temporary image structure
VisXimage_t tm2; // temporary image structure
VisXimage_t mm; //temporary mask image to hold bounding boxes

int l = 1; // index counters
int range; // range value
int first = 0;
double handWidth = 0;

void setlabel(int i, int j, int n); // initialize function

main(argc, argv)
int argc;
char *argv[];
{
    int        i,j,k;                /* index counters                 */
    int hist[256];                 /* histogram bins                 */
    int thresh;                    /* threshold                      */
    int dist;                   /* minimum distance between maxima   */
    int avg1,avg2; /* averages of Region1 and Region2   */
    int old1,old2;         /* saved averages from previous iteration */
    int total;
    int total1=0,total2=0;    /* for computing average     */
    int number1=0,number2=0;


    VXparse(&argc, &argv, par);    /* parse the command line         */
    VXin  = VXopen(IVAL, 0);       /* open input file                */
    VXout = VXopen(OVAL, 1);       /* open the output file           */

/************ Bounding Box Location Specification ********************/
    int xpos[5] = {185, 55, -25, -85, -153}; /*x positions of bounding boxes relative to COM*/
    int ypos[5] = {28, 177, 193, 159, 78}; /*y positions of bounding boxes relative to COM*/

/************ End of Parameter and initialization section ************/

    while((VXlist = VXptr = VXreadframe(VXin)) != VXNIL){ /* every frame */
        VXfupdate(VXout, VXin); /* update global constants */
/* find next byte image */
        while (VXNIL != (VXptr = VXfind(VXptr, VX_PBYTE)))  {
            VXsetimage(&im, VXptr, VXin); /* initialize input structure */

    /* initialize the length and width vectors for bounding boxes */
    int length[5] = {80, 40, 50, 40, 60};
    int width[5] = {80, 40, 40, 40, 60};

    dist = 10; /* default dist */
         if (DVAL) dist = atoi(DVAL); /* if d= was specified, get value */
    if (dist < 0 || dist > 255) {
fprintf(stderr, "d= must be between 0 and 255\nUsing d=10\n");
dist = 10;
    }

         int length1 = 30; /*threshold in bounding box*/
    if (LVAL) {
length1 = atoi(LVAL); /* if d= was specified, get value */
for (k = 0; k < 5; k++) {
    length[k] = length1;
}
    }
         if (length1 < im.ylo || length1 > im.yhi) {
         fprintf(stderr, "l= must be between %d and %d\nUsing l=10\n",im.ylo,im.yhi);
         for (k = 0; k < 5; k++) {
    length[k] = 30;
}
         }
         int width1 = 30; /*threshold in bounding box*/
    if (WVAL) {
width1 = atoi(WVAL); /* if d= was specified, get value */
for (k = 0; k < 5; k++) {
    width[k] = width1;
}
    }
         if (dist < im.xlo || dist > im.xhi) {
         fprintf(stderr, "w= must be between %d and %d\nUsing w=10\n",im.xlo,im.xhi);
         for (k = 0; k < 5; k++) {
    width[k] = 30;
}
         }
         int fthresh = 9; /*threshold of count in bounding box*/
    if (TVAL) fthresh = atoi(TVAL); /* if d= was specified, get value */
         if (dist < 0 || dist > length1*width1) {
         fprintf(stderr, "th= must be between 0 and %d\nUsing th=20\n",length1*width1);
         fthresh = 9;
         }

/***************** Application specific section *********************/

            /* clear the histogram */
            for (i = 0; i < 256; i++) hist[i] = 0;

            /* compute the histogram */
            for (i = im.ylo; i <= im.yhi; i++){
                for (j = im.xlo; j <= im.xhi; j++){
                    hist[im.u[i][j]]++;
}
    }

            /* compute the threshold */
    for (i=0; i <256; i++) {
    number1=number1+hist[i];
total1=total1+i*hist[i];
            }
    thresh=total1/number1;
    fprintf(stderr, "thresh = %d\n",thresh);
    total1=0;
    number1=0;
    for (j=0; j<=10000; j++){
   for (i=0; i<thresh; i++) {
total1 = total1+i*hist[i];
   number1=number1+hist[i];
   }
   for (i=thresh; i<256; i++){
total2=total2+i*hist[i];
number2=number2+hist[i];
   }
   if (number1 !=0){
       avg1=total1/number1;
   }else {
avg1=thresh/2;
   }
   if (number2 !=0){
    avg2=total2/number2;
   }else {
avg2=(255-thresh)/2;
   }
           thresh=(avg1+avg2)/2;
   if (avg1==old1 && avg2==old2){
fprintf(stderr, "avg1 = %d, avg2 = %d\n", avg1, avg2);
break;
   }else{
old1=avg1;
old2=avg2;
   }
            }

    if (VFLAG)
fprintf(stderr, "thresh = %d\n",thresh);
```

```
            /* apply the threshold */
            for (i = im.ylo; i <= im.yhi; i++) {
                for (j = im.xlo; j <= im.xhi; j++) {
                    if (im.u[i][j] >= thresh) im.u[i][j] = 0;
                    else                      im.u[i][j] = 1;
                }
            }

     /* elminiate flash brightness area outside of the hand */
    int upY = 410; // parameters determined by testing
    int lowY = 60;
    int upX = 440;
    int lowX = 160;

    int leftWidth = 0;
    int rightWidth = 0;

    int height = (im.yhi - im.ylo) / 2;
    for (i = lowX; i <= upX; i++) {
leftWidth++;
if (im.u[height][i] != 0) { break; }
    }
    for (i = upX; i >= lowX; i--) {
rightWidth++;
if (im.u[height][i] != 0) { break; }
    }
    handWidth = im.xhi - im.xlo - upX - lowX - rightWidth - leftWidth;
    if (handWidth >= 400 || handWidth <= 50) { handWidth = trainedHandWidth; } // default width of hand

/************* Find center of mass using moments ***********************/
    VXembedimage(&tm,&im,1,1,1,1); /* image structure with border */
    VXembedimage(&tm2,&im,1,1,1,1);
            total = 0;
            int xtotal = 0;
            int ytotal = 0;
            int xcom;
            int ycom;
            for (i = im.ylo + lowY; i <= upY; i++) {
                for (j = im.xlo + lowX; j <= upX; j++) {
                    total = total + tm.u[i][j];
                    xtotal = xtotal + j*tm.u[i][j];
                    ytotal = ytotal + i*tm.u[i][j];
                }
            }
            xcom = xtotal/total;
            ycom = ytotal/total;

    fprintf(stderr, "xcom = %d, ycom = %d\n", xcom, ycom);

/************ Bounding Box Mask Construction ***************************/
    VXembedimage(&mm,&im,1,1,1,1); /* image structure with border */
            for (i = im.ylo; i <= im.yhi; i++) {
                for (j = im.xlo; j <= im.xhi; j++) {
                    mm.u[i][j] = 0;
    tm2.u[i][j] = 0;
                }
            }

    double scale = handWidth / trainedHandWidth;

    for (k = 0; k < 5; k++) {
fprintf(stderr, "%d %d %d %d\n", length[k], width[k], ypos[k], xpos[k]);
length[k] = (double)length[k] * scale;
width[k] = (double)width[k] * scale;
     ypos[k] = (double)ypos[k] * scale;
     xpos[k] = (double)xpos[k] * scale;
    }

            for (k = 0; k < 5; k++) {
                for (i = -length[k]/2; i <= length[k]/2; i++){
                    for (j = -width[k]/2; j <= width[k]/2; j++){
                        tm2.u[ycom+ypos[k]+i][xcom+xpos[k]+j] = 255;
                    }
                }
            }
            l = 1;
            for (j = im.xhi; j >= im.xlo; j--) {
                for (i = im.ylo; i <= im.yhi; i++) {
                    if (tm2.u[i][j] != 0 && mm.u[i][j] == 0) {
        setlabel(i,j,l);
                        if (l == 255) {
                            l = 1;
                        }
                        else {
                            l = l + 1;
                        }
                    }
                }
            }

/************ Finger Detection *********************************************/
            int f1 = 0, f2 = 0, f3 = 0, f4 = 0, f5 = 0; /* counts for each finger in order thumb, pointer, mi
            for (i = im.ylo; i <= im.yhi; i++) {
                for (j = im.xlo; j <= im.xhi; j++) {
                    if (tm.u[i][j] != 0 && mm.u[i][j] != 0) {
                        switch (mm.u[i][j]){
                            case 1:
                                f1++;
                                break;
                            case 2:
                                f2++;
                                break;
                            case 3:
                                f3++;
                                break;
                            case 4:
                                f4++;
                                break;
                            case 5:
                                f5++;
                                break;
                        }
                    }
                }
            }
    fprintf(stderr, "f1 = %d, f2=%d, f3=%d, f4=%d, f5=%d\n", f1, f2, f3, f4,f5);
            int thumb = false, pointer = false, middle = false, ring = false, pinky = false;
            if (f1 > fthresh) thumb = true;
            if (f2 > fthresh) pointer = true;
            if (f3 > fthresh) middle = true;
            if (f4 > fthresh) ring = true;
            if (f5 > fthresh) pinky = true;

        /******** ASL NUMBER DEFINITION **********/
            int ASLNUM = 0;
            if (!thumb && pointer && !middle && !ring && !pinky) ASLNUM = 1;
            else if (!thumb && pointer && middle && !ring && !pinky) ASLNUM = 2;
            else if (thumb && pointer && middle && !ring && !pinky) ASLNUM = 3;
            else if (!thumb && pointer && middle && ring && pinky) ASLNUM = 4;
            else if (thumb && pointer && middle && ring && pinky) ASLNUM = 5;
            else if (!thumb && pointer && middle && ring && !pinky) ASLNUM = 6;
            else if (!thumb && pointer && middle && !ring && pinky) ASLNUM = 7;
            else if (!thumb && pointer && !middle && ring && pinky) ASLNUM = 8;
            else if (!thumb && !pointer && middle && ring && pinky) ASLNUM = 9;
            else if (thumb && !pointer && !middle && !ring && !pinky) ASLNUM = 10;
    else fprintf(stderr, "Not an ASL number\n");

    fprintf(stderr, "ASLNUM = %d\n", ASLNUM);

    for (i = im.ylo; i <= im.yhi; i++) {
for (j = im.xlo; j <= im.xhi; j++) {
    im.u[i][j] = tm2.u[i][j];
}
    }

/************** End of the Application specific section **************/

            VXresetimage(&im); /* free the im image structure  */
            VXptr = VXptr->next; /* move to the next image */
        } /* end of every image section */
        VXwriteframe(VXout,VXlist); /* write frame */
        VXdellist(VXlist);          /* delete the frame */
    } /* end of every frame section */
    VXclose(VXin);  /* close files */
    VXclose(VXout);
    exit(0);
}

/* setlabel: recursively sets label based on requirements documented in lab writeup */
void setlabel(int i, int j, int l) {
mm.u[i][j] = l;
if (tm2.u[i][j+1] != 0 && mm.u[i][j+1] == 0) { setlabel(i,j+1,l); }
if (tm2.u[i][j-1] != 0 && mm.u[i][j-1] == 0) { setlabel(i,j-1,l); }
if (tm2.u[i+1][j] != 0 && mm.u[i+1][j] == 0) { setlabel(i+1,j,l); }
if (tm2.u[i-1][j] != 0 && mm.u[i-1][j] == 0) { setlabel(i-1,j,l); }
}
```