CS205 Project Report - HCI

1) Contents

小组成员	**黄慧惠12010336**	**王思懿12010339**	**安钧文12012109**
组员分工	文档（1、4、6、7）、报告	文档（5）、demo视频剪辑、报告	文档（2、3）、报告
分工占比	33.33%	33.33%	33.33%

1. It supports to read images from a camera and then store the image, using APIs provided by OpenCV.
   libraries used

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui_c.h>
#include <iostream>
```

main core code

```
using namespace std;
using namespace cv;
string writePath = "tempt_img/";
int main(int argc, char **argv) {
    VideoCapture capture(0);//specify the camera device to open
    string name;
    namedWindow("hello", CV_WINDOW_AUTOSIZE); //window name and make the image appear the size of the original image
    int i = 0;
    while (1) {
        Mat frame;
        capture >> frame;
        if (32 == waitKey(20)) {            //space to photo
            name = writePath + to_string(i) + ".jpg"; //photo name
            imwrite(name, frame); // store photo
            cout << name << endl;
            i++;
        }
        imshow("hello", frame); //display the picture in the window
    }
}
```

2. We used haarcascade to identify right hand, left hand and fist.

```
Mat imgLines = Mat::zeros(img.size(), CV_8UC3); //used to draw lines
Mat imgShape = Mat::zeros(img.size(), CV_8UC3); //separate mat only contains drawn shape
Mat imgshapetempt= Mat::zeros(img.size(), CV_8UC3);
CascadeClassifier rhandCascade, lhandCascade, fistCascade;
rhandCascade.load("../haarcascade/rpalm.xml");
lhandCascade.load("../haarcascade/lpalm.xml");
fistCascade.load("../haarcascade/closed_frontal_palm.xml");
fistCascade.detectMultiScale(gray, fist, 1.1, 2, CASCADE_SCALE_IMAGE, Size(30,
30));
lhandCascade.detectMultiScale(gray, lhand, 1.1, 2, CASCADE_SCALE_IMAGE, Size(30,
30));
rhandCascade.detectMultiScale(gray, rhand, 1.1, 2, CASCADE_SCALE_IMAGE, Size(30,
30));
for (int i = 0; i < lhand.size(); i++) {
    rectangle(imgRect, lhand[i].tl(), lhand[i].br(), Scalar(0, 0, 255), 2);
}
for (int i = 0; i < fist.size(); i++) {
    rectangle(imgRect, fist[i].tl(), fist[i].br(), Scalar(0, 255, 0), 2);
}
for (int i = 0; i < rhand.size(); i++) {
    rectangle(imgRect, rhand[i].tl(), rhand[i].br(), Scalar(255, 0, 0), 2);
}
```

3. We used two sets of coordinates (current coordinate and last coordinates) to track the hands'. Current coordinate sets when a hand is detected, meanwhile updating last coordinate after a line is drawn. The drawn line is determined by current coordinate and last coordinate.

```
int curx, cury, lastx, lasty;
lastx = 1;
lasty = -1;
if (!rhand.empty()) {
    for (int i = 0; i < rhand.size(); i++) {
    rectangle(imgRect, rhand[i].tl(), rhand[i].br(), Scalar(255, 0, 0), 2);
    curx = (rhand[i].br().x + rhand[i].tl().x) / 2;
    cury = (rhand[i].br().y + rhand[i].tl().y) / 2;
    if (lastx >= 0 && lasty >= 0 && curx >= 0 && cury >= 0
    && abs(curx-lastx)<50 && abs(cury-lasty)<50 //ignore point too far
    && (abs(curx-lastx)>2 || abs(cury-lasty)>2)) { //ignore point too close
        line(imgLines, Point(lastx, lasty), Point(curx, cury), Scalar(255, 0,
255), 1);
}
if(abs(lastx-curx)>2){
    lastx = curx;
}
if(abs(lasty-cury)>2){
    lasty = cury;
}
}
}
else if (!lhand.empty()) {
    isFist= false;
    for (int i = 0; i < lhand.size(); i++) {
    rectangle(imgRect, lhand[i].tl(), lhand[i].br(), Scalar(0, 0, 255), 2);
    curx = (lhand[i].br().x + lhand[i].tl().x) / 2;
    cury = (lhand[i].br().y + lhand[i].tl().y) / 2;
    //cur=rhand[i].br();
    if (lastx >= 0 && lasty >= 0 && curx >= 0 && cury >= 0
    && abs(curx - lastx) < 100 && abs(cury - lasty) < 100 //ignore point too far
    && (abs(curx - lastx) > 10 || abs(cury - lasty) > 10)) { //ignore point too
close
    line(imgLines, Point(lastx, lasty), Point(curx, cury), Scalar(0, 0, 255),
1);
    }

        if (abs(lastx - curx) > 3) {
            lastx = curx;
        }
        if (abs(lasty - cury) > 3) {
            lasty = cury;
        }
    }
```

4. It supports to identify some specific meaning of trajectory, such as circle, rectangle or triangle etc
   libraries used

```
vector<vector<Point>> contour;
```

Contours is defined as "vector<vector> contours", which is a double vector (each element in the vector holds a vector of a set of points composed of consecutive Points), and each set of points is a contour, Contours have as many elements as there are contours

```
vector<Vec4i> hierarchy;
```

Hierarchy is defined as "vector hierarchy", the definition of Vec4i: typedef Vec<int, 4> Vec4i; (each element in the vector contains 4 int variables), so by definition, hierarchy is a vector , each element in the vector is an array of 4 ints. The elements in the vector hierarchy and the elements in the contour vector contours are in one-to-one correspondence, and the vectors have the same capacity. The four int variables of each element in the hierarchy are hierarchy[i][0] ~ hierarchy[i][3], which respectively represent the numbers of the next contour, previous contour, parent contour and embedded contour of the current contour i index. If the current contour has no corresponding next contour, previous contour, parent contour and embedded contour, the corresponding hierarchy[i][*] is set to -1.

```
RETR_EXTERNAL
```

Only the outermost contour is detected, the inner contours contained within the outer contour are ignored

```
vector<vector<Point>> conPoly(contour.size());
```

Point offset, the offset of all contour information relative to the corresponding point of the original image, which is equivalent to adding the offset to each detected contour point, and Point can also be a negative value

```
vector<Rect> boundRect(contour.size());
```

Calculate the minimum rectangle of the vertical boundary of the contour, the rectangle is parallel to the upper and lower boundaries of the image

```
approxPolyDP(contour[i], conPoly[i], 0.02 * peri, true);
```

InputArray curve: generally a point set consisting of the contour points of the image
OutputArray approxCurve: Set of polygon points representing the output
double epsilon: mainly indicates the accuracy of the output, which is the maximum distance between the other contour points, 5,6,7,,8,,,,,
bool closed: Indicates whether the output polygon is closed

main core code

```
int main() {
    string path = "../test_q4.png";
    Mat img = imread(path);//read image
    Mat imgGray, imgBlur, imgCanny, imgDila, imgErode;
    Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3));
    //preprocessing
    cvtColor(img, imgGray,COLOR_BGR2GRAY, 0);
    GaussianBlur(imgGray, imgBlur, Size(65, 65), 1, 1);
    Canny(imgBlur, imgCanny, 40, 120);
    dilate(imgCanny, imgDila, kernel);
    //Grayscale -> Gaussian Filtering -> Canny Edge Algorithm -> Dilation
    judge_contour(imgDila,img);
    imshow("Image", img);
    waitKey(0);
}
```

5. Using the detected trajectories (sets of points) and the identified trajectory meaning, we implement mouse moving, click, double click and adjust the volume.
   Libraries used

```
#include <audioclient.h>
#include <endpointvolume.h>
#include <mmdeviceapi.h>
```

The block is to get and set volume when right hand is detected:

```
//设置系统音量
bool SetVolum(int volume)
{
    bool ret = false;
    HRESULT hr;
    IMMDeviceEnumerator* pDeviceEnumerator=0;
    IMMDevice* pDevice=0;
    IAudioEndpointVolume* pAudioEndpointVolume=0;
    IAudioClient* pAudioClient=0;

        hr = CoCreateInstance(__uuidof(MMDeviceEnumerator), NULL, CLSCTX_ALL,
__uuidof(IMMDeviceEnumerator), (void**)&pDeviceEnumerator);
        if (FAILED(hr)) throw "CoCreateInstance";
        hr = pDeviceEnumerator->GetDefaultAudioEndpoint(eRender, eMultimedia,
&pDevice);
        if (FAILED(hr)) throw "GetDefaultAudioEndpoint";
        hr = pDevice->Activate(__uuidof(IAudioEndpointVolume), CLSCTX_ALL, NULL,
(void**)&pAudioEndpointVolume);
        if (FAILED(hr)) throw "pDevice->Active";
        hr = pDevice->Activate(__uuidof(IAudioClient), CLSCTX_ALL, NULL,
(void**)&pAudioClient);
        if (FAILED(hr)) throw "pDevice->Active";

        float fVolume;
        fVolume = volume / 100.0f;
        hr = pAudioEndpointVolume->SetMasterVolumeLevelScalar(fVolume,
&GUID_NULL);
        if (FAILED(hr)) throw "SetMasterVolumeLevelScalar";

        pAudioClient->Release();
        pAudioEndpointVolume->Release();
        pDevice->Release();
        pDeviceEnumerator->Release();

        ret = true;

    return ret;
}
//获取系统音量
int GetVolume()
{
    int volumeValue = 0;
    HRESULT hr;
    IMMDeviceEnumerator* pDeviceEnumerator = 0;
    IMMDevice* pDevice = 0;
    IAudioEndpointVolume* pAudioEndpointVolume = 0;
    IAudioClient* pAudioClient = 0;

        hr = CoCreateInstance(__uuidof(MMDeviceEnumerator), NULL, CLSCTX_ALL,
__uuidof(IMMDeviceEnumerator), (void**)&pDeviceEnumerator);
        if (FAILED(hr)) throw "CoCreateInstance";
        hr = pDeviceEnumerator->GetDefaultAudioEndpoint(eRender, eMultimedia,
&pDevice);
        if (FAILED(hr)) throw "GetDefaultAudioEndpoint";
        hr = pDevice->Activate(__uuidof(IAudioEndpointVolume), CLSCTX_ALL, NULL,
(void**)&pAudioEndpointVolume);
        if (FAILED(hr)) throw "pDevice->Active";
        hr = pDevice->Activate(__uuidof(IAudioClient), CLSCTX_ALL, NULL,
(void**)&pAudioClient);
        if (FAILED(hr)) throw "pDevice->Active";

        float fVolume;
        hr = pAudioEndpointVolume->GetMasterVolumeLevelScalar(&fVolume);
        if (FAILED(hr)) throw "SetMasterVolumeLevelScalar";

        pAudioClient->Release();
        pAudioEndpointVolume->Release();
        pDevice->Release();
        pDeviceEnumerator->Release();

        volumeValue = fVolume * 100;

    return volumeValue;
}
```

The block is to set up the mouse movement within the specified rectangle when left hand is detected

```
POINT p;
int mouse_x=lastx+(curx-lastx)/smoothening;
int mouse_y=lasty+(cury-lasty)/smoothening;

if(mouse_x<=4.5*frameR&&mouse_x>=frameR/2&&mouse_y>=frameR/2&&mouse_y<=3.5*frame
R){
    SetCursorPos((mouse_x-frameR>0)?(mouse_x-frameR)*(windows_width/300):0,
                 mouse_y-frameR>0?((mouse_y-frameR)*(windows_height/200)):0);
    cv::circle(imgRect,Point(mouse_x, mouse_y) , 15, (255,255,255), cv::FILLED);
}
```

The block is to click when fist is detected and double click when fist is detected and move a distance:

```
  else if (!fist.empty()) {
            bool isDouble= false;

            for (int i = 0; i < fist.size(); i++) {
                rectangle(imgRect, fist[i].tl(), fist[i].br(), Scalar(0, 255,
0), 2);
                curx = (fist[i].br().x + fist[i].tl().x) / 2;
                cury = (fist[i].br().y + fist[i].tl().y) / 2;
                //cur=rhand[i].br();
                if (isFist&&(abs(cury - lasty) >10||abs(curx - lastx)>10)){
//ignore point too far
                    isDouble= true;
                }

                if(isDouble){
                    mouse_event(MOUSEEVENTF_LEFTDOWN,0,0,0,0);
                    Sleep(10);//要留给某些应用的反应时间
                    mouse_event(MOUSEEVENTF_LEFTUP,0,0,0,0);
                    mouse_event(MOUSEEVENTF_LEFTDOWN,0,0,0,0);
                    Sleep(10);//要留给某些应用的反应时间
                    mouse_event(MOUSEEVENTF_LEFTUP,0,0,0,0);
                    cout<<"Double click     curx:"<<curx<<"cury:"
<<cury<<"lastx:"<<lastx<<"lasty"<<lasty<<endl;
                }else{
                    mouse_event(MOUSEEVENTF_LEFTDOWN,0,0,0,0);
                    Sleep(10);//要留给某些应用的反应时间
                    mouse_event(MOUSEEVENTF_LEFTUP,0,0,0,0);
                    cout<<"single click     curx:"<<curx<<"cury:"
<<cury<<"lastx:"<<lastx<<"lasty"<<lasty<<endl;
                }

                if(abs(lastx-curx)>4){
                    lastx = curx;
                }
                if(abs(lasty-cury)>4){
                    lasty = cury;
                }
//                Sleep(1000);
                isFist= true;
            }
        }
```

6. Draw certain geometric shapes by hand, such as triangles, quadrilaterals, pentagons, etc. Display the drawn figures and type the names of these geometric figures on the computer screen
   Due to the indeterminate size of the drawn graphics, certain parameters have been changed.

```
void getcontour(Mat imgDil,Mat img)
{
    vector<vector<Point>> contour;
    vector<Vec4i> hierarchy;
    findContours(imgDil, contour, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_TC89_KCOS);
    vector<vector<Point>> conPoly(contour.size());
    vector<Rect> boundRect(contour.size());
    for (int i = 0; i < contour.size(); i++)
    {
        int area = contourArea(contour[i]);
        cout << area << endl;
        string objectType;

        if (area > 5000)
        {
            float peri = arcLength(contour[i], true);
            approxPolyDP(contour[i], conPoly[i], 0.02 * peri, true);
            cout << conPoly[i].size() << endl;
            boundRect[i] = boundingRect(conPoly[i]);
            int objCor = (int)conPoly[i].size();
            if (objCor <= 6 && objCor>=6) { objectType = "Triangle"; }
            else if (objCor <= 10 && objCor>=7) { objectType = "Pentagon"; }
            else if (objCor == 4) {
                float aspRatio = (float)boundRect[i].width /
boundRect[i].height;
                cout << aspRatio << endl;
                if (aspRatio > 0.95 && aspRatio < 1.05) { objectType = "Square";
}
                else { objectType = "Rectangle"; }
            }
            else { objectType = "Circle"; }
            drawContours(img, conPoly, i, Scalar(255, 0, 255), 2);
            rectangle(img, boundRect[i].tl(), boundRect[i].br(), Scalar(0.255,
0), 5);
            putText(img, objectType, { boundRect[i].x,boundRect[i].y - 5
},1,FONT_HERSHEY_PLAIN, Scalar(0, 69, 255), 1);
        }
    }
}
```

7. Since the graphics drawn in the previous step are not standard, use the rules of these geometric shapes to modify the graphics in the previous step to meet these basic rules, and display the comparison before and after the correction on the computer screen.

   Pressing the w key will start to analyze the drawn graph, analyze the shape of the graph and correct the graph.

```
if(keyboard == 'w' ){
    Mat imgGray=Mat::zeros(img.size(), CV_8UC3);
    Mat imgBlur=Mat::zeros(img.size(), CV_8UC3);
    Mat imgCanny=Mat::zeros(img.size(), CV_8UC3);
    Mat imgDila=Mat::zeros(img.size(), CV_8UC3);
    Mat imgErode=Mat::zeros(img.size(), CV_8UC3);
    Mat kernel = getStructuringElement(MORPH_RECT, Size(50, 50));
    cvtColor(imgShape, imgGray,COLOR_BGR2GRAY, 0);
    GaussianBlur(imgGray, imgBlur, Size(65, 65), 1, 1);
    Canny(imgBlur, imgCanny, 40, 120);
    dilate(imgCanny, imgDila, kernel);
    getcontour(imgDila,imgshapetempt);
    imshow("Image dila", imgDila);
    imshow("Image tempt", imgshapetempt);
}
```

```
Canny(imgBlur, imgCanny, 40, 120);
```

The processing flow of the Canny edge detection algorithm
The Canny edge detection algorithm can be divided into the following 5 steps:

1. Use a Gaussian filter to smooth the image and filter out noise.
2. Calculate the gradient strength and direction of each pixel in the image.
3. Apply Non-Maximum Suppression to eliminate spurious responses from edge detection.
4. Apply Double-Threshold detection to determine true and potential edges.
5. The edge detection is finally completed by suppressing the isolated weak edges.
   The implementation ideas of each step are described in detail below.

3.1 Gaussian smoothing filterIn order to minimize the influence of noise on edge detection results, it is necessary to filter out noise to prevent false detections caused by noise. To smooth the image, a Gaussian filter is used to convolve the image, a step that smoothes the image to reduce the noticeable noise effect on the edge detector. The generating equation for a Gaussian filter kernel of size (2k+1)x(2k+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right);1 \le i, j \le (2k+1) \qquad (3\text{-}1)$$

3.2 Calculate gradient strength and direction
Edges in an image can point in all directions, so the Canny algorithm uses four operators to detect horizontal, vertical, and diagonal edges in an image. Edge detection operators (such as Roberts, Prewitt, Sobel, etc.) return the first derivative values of the horizontal Gx and vertical Gy directions, from which the gradient G and direction theta of the pixel can be determined. where G is the gradient strength, theta is the gradient direction, and arctan is the arc tangent function.

$$\begin{aligned} G &= \sqrt{G_x^2 + G_y^2} \\ \theta &= arc\tan\left(G_y / G_x\right) \end{aligned} \qquad (3\text{-}2)$$

3.3 Non-maximum suppression

Non-maximum suppression is an edge sparse technique, and the role of non-maximum suppression is to "thin" edges. After the gradient calculation is performed on the image, the edges extracted based only on the gradient values are still blurred. For criterion 3, there is and should be only one exact response to the edge. Non-maximum suppression can help suppress all gradient values except the local maximum to 0. The algorithm for non-maximum suppression of each pixel in the gradient image is:

1. Compare the gradient strength of the current pixel with two pixels along the positive and negative gradient directions.
2. If the gradient strength of the current pixel is the largest compared with the other two pixels, the pixel is reserved as an edge point, otherwise the pixel will be suppressed.
   Usually for a more accurate calculation, linear interpolation is used between two adjacent pixels across the gradient direction to obtain the pixel gradient to be compared.

3.4 Double Threshold Detection

After applying non-maximum suppression, the remaining pixels can more accurately represent the actual edges in the image. However, there are still some edge pixels due to noise and color changes. To address these spurious responses, edge pixels must be filtered with weak gradient values, and edge pixels with high gradient values must be preserved, which can be achieved by choosing high and low thresholds. If the gradient value of the edge pixel is higher than the high threshold, it is marked as a strong edge pixel; if the gradient value of the edge pixel is less than the high threshold and greater than the low threshold, it is marked as a weak edge pixel; if the gradient value of the edge pixel is less than A low threshold is suppressed. The choice of threshold depends on the content of the given input image.
The pseudocode for double threshold detection is described as follows:

```
$if \ \ G_p \geq HighThreshold$
    $G_p \ is \ an \ strong \ edge$
$else \ \ if \ \ G_p \geq LowThreshold$
    $G_p \ is \ an \ weak \ edge$
$else$
    $G_p \ \ should \ \ be \ \ \sup pressed$
```

3.5 Suppression of isolated low threshold points

So far, pixels classified as strong edges have been identified as edges because they are extracted from real edges in the image. However, for weak edge pixels, there will be some debate, as these pixels can be extracted from true edges or caused by noise or color changes. For accurate results, weak edges caused by the latter should be suppressed. Typically, weak edge pixels caused by true edges will be connected to strong edge pixels, while noise responses are not. To track edge connections, by looking at a weak edge pixel and its 8 neighbor pixels, as long as one of them is a strong edge pixel, the weak edge point can be preserved as a true edge.

The pseudocode description for suppressing isolated edge points is as follows:

```
if $G_p == LowThreshold$ and $G_p$ connected to a strong edge pixel
    $G_p$ is an strong edge
else
    $G_p$ should be suppressed
```

2) Highlights

- Supports one-key actions.

```
c //clears the whole image
w //detect & adjust the user's drawn shape
esc //exit the program
```

- Different colors are used to distinguish various actions.

```
purple: user's hand-drawn trajectories
blue: user's left palm
red: user's right palm
green: user's fist
```

- A scaled window is provided to move the mouse more comfortably, as the user easily move the mouse around the whole screen in a window instead of the entire screen.
- Fine-tuned parameters for drawing tracjectories and hand detection, achieving an improved detection accuracy.

```
fistCascade.detectMultiScale(gray, fist, 1.1, 2, CASCADE_SCALE_IMAGE,
Size(30, 30));
lhandCascade.detectMultiScale(gray, lhand, 1.1, 2, CASCADE_SCALE_IMAGE,
Size(30, 30));
rhandCascade.detectMultiScale(gray, rhand, 1.1, 2, CASCADE_SCALE_IMAGE,
Size(30, 30));
```

3) Novelties

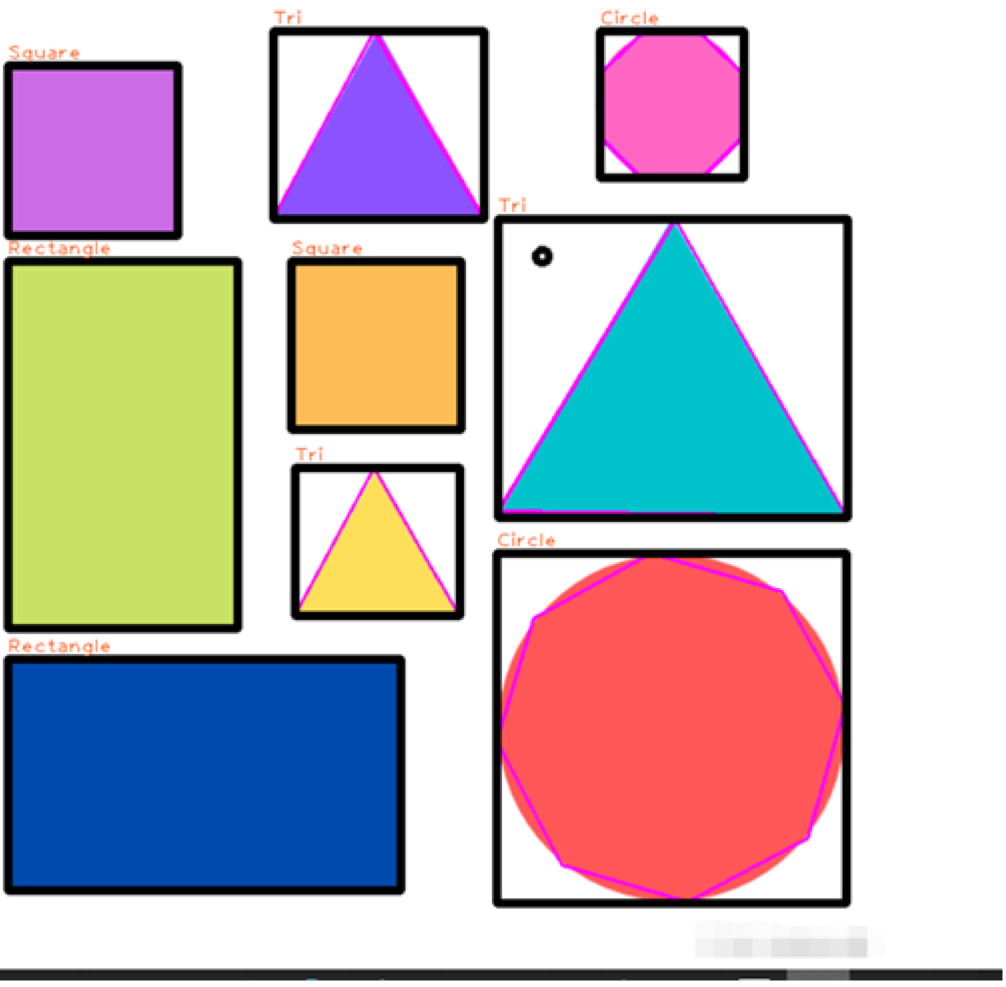
- Implemented an improved logic for drawing shapes. Here we only show the core code.

```
if(!lhand.empty()){
    if (lastx >= 0 && lasty >= 0 && curx >= 0 && cury >= 0
        && abs(curx-lastx)<50 && abs(cury-lasty)<50 //ignore point too far
        && (abs(curx-lastx)>1 || abs(cury-lasty)>1)) { //ignore point too
close
        line(imgLines, Point(lastx, lasty), Point(curx, cury), Scalar(255,
0, 255), 1);
    }
    //only update coordinates if they aren't too close
    if(abs(lastx-curx)>2){
        lastx = curx;
    }
    if(abs(lasty-cury)>2){
    lasty = cury;
    }                                           }
    //if user's hand is not present, reset coordinates to -1
else {
    lastx = -1;
    lasty = -1;
}
```

- Implemented functions to manipulate volume using hand gesture. Several OS libraries are used to get the system variables.

```
#include <mmdeviceapi.h>
#include <endpointvolume.h>
#include <audioclient.h>
#include <windows.h>
```

- Implemented logic to distinguish rectangle and square. They are determined by ratio of height and width.



4) Difficulties

- The program may not run at optimal efficiency, as the camera image stutters from time to time.
- Limited by our resources, we were not able to train our own haarcascade, so we used pre-trained models for hand detection.

- Shape detection's accuracy still have room for improvement, since the user's hand drawn figure varies in each practice.