CS4187 Computer Vision and Interactivity

Mini Project

WANG Yinuo 55670000

INTRODUCTION

After studying computer vision for one semester, I design and implement an interactive computer-vision application using OpenFrameworks in this project.

The first feature of the program is to get the real-time images from the webcam, then detect and track the keypoints within the face area using the method of optical flow.

Instead of showing all the keypoints or putting a bounding box round the head, this program will cover your face with a image.

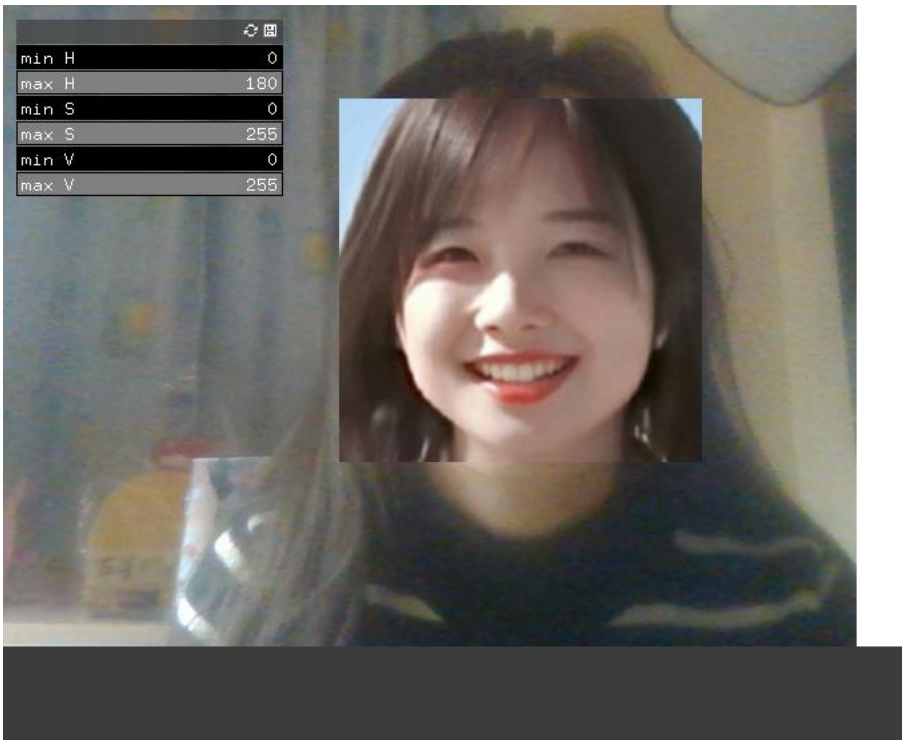


Figure 1 Capture of the real time face detection

The second feature is to draw interface using Color Tracking. It is an enhanced version of the finger-drawing application in assignment 3, supporting more functions, such as drawing in different colors and different stroke width, saving the drawing results, etc.

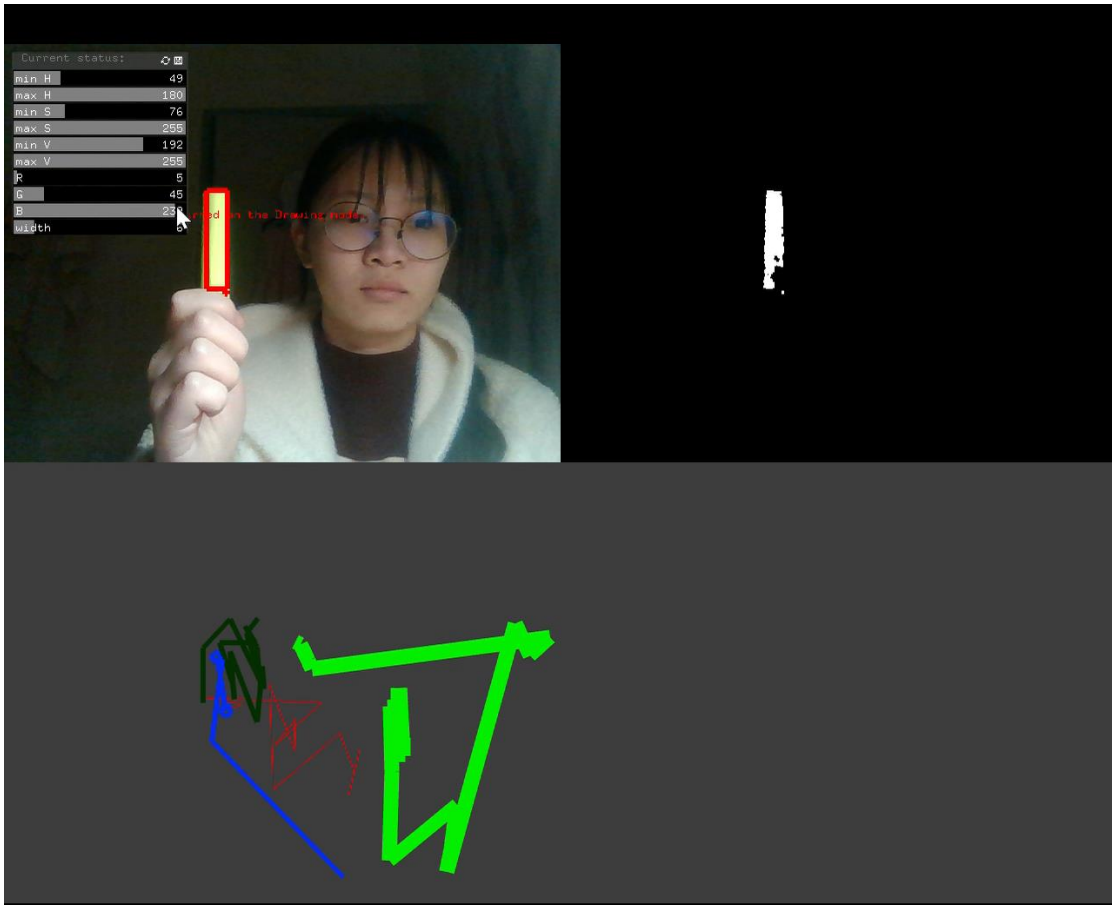


Figure 2 Capture of drawing application

The third function is music playing. You can interact with the keyboard to choose to play it or stop it. There's only one song in it, hopefully to put you in a good mood

RELATED WORK

1. OpenFramework and c++ learning
2. Basic knowledge of face detection
3. Basic knowledge of face/motion/color detection and tracking

APPROACH

1. **Face Detection and Real-time Tracking**

The program will get the real-time images from the webcam and allow user to start tracking only when the key ‘r’ is pressed.

Basically, there are six states that the program can capture:

- When the user shakes his/her head left/right/up/down, the left top of program will show the text “left/right/up/down”
- When the user doesn’t move his/her head, show the text “Still”
- When the user clears all the key points, show the text “Init”

The captured result is shown below.

To accurately detect the face, we set the position of key points based on two pretrained models (haarcascade_frontalface_default.xml and haarcascade_eye.xml).

The main code is as follows, where haarcascade_frontalface_default.xml and haarcascade_eye.xml are the pretrained face and eyes models respectively.

```
//need to use absolute path here, for the latest version of OpenFrameworks and its OpenCV addon
if (!face_cascade.load("C:/Users/w/Desktop/cs/2020-2021/cs4187cv/faceDetection_Image/bin/data/haarcascade_frontalface_default.xml")) {
    cout << "Error loading haarcascade_frontalface_default.xml" << endl;
}

if (!eye_cascade.load("C:/Users/w/Desktop/cs/2020-2021/cs4187cv/faceDetection_Image/bin/data/haarcascade_eye.xml")) {
    cout << "Error loading haarcascade_eye.xml" << endl;
}
```

These two pretrained files should be loaded with the absolute paths to avoid error loading.

And to eliminate false positive, we should tell whether there are eyes inside the detected face, and then draw a rectangle to indicate the human face.

```
face_cascade.detectMultiScale(matCam, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));

for (int i = 0; i < faces.size(); i++) {

    eye_cascade.detectMultiScale(matCam, eyes, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));
    //if no eye has been detected, we should skip
    if (eyes.size() < 2)
    {
        continue;
    }
    for (int j = 0; j < 2; j++) {

        if ((faces.at(i).x < eyes.at(j).x) &&
            ((faces.at(i).x + faces.at(i).width) > eyes.at(j).x) &&
            (faces.at(i).y < eyes.at(j).y) &&
            ((faces.at(i).y + faces.at(i).height) > eyes.at(j).y)) {

            Rect roi(faces.at(i).x, faces.at(i).y, faces.at(i).width, faces.at(i).height);
            mask = Mat::zeros(matCamGrey.size(), CV_8UC1);
            mask(roi).setTo(255);
        }
    }
}
```

We track these key points:

```
void ofApp::trackKeyPoints(cv::Size winSize) {

    vector<uchar> status;
    vector<float> err;
    if (matCamPreGrey.empty()) {
        matCamGrey.copyTo(matCamPreGrey);
    }
    calcOpticalFlowPyrLK(matCamPreGrey, matCamGrey, prePoints, curPoints, status, err, winSize);

}
```

And get their directions:

```
void ofApp::getDirections() {
    for (int i = 0; i < curPoints.size(); i++) {
        Point2f dir = Point2f(curPoints.at(i).x - prePoints.at(i).x, curPoints.at(i).y - prePoints.at(i).y);
        directions.push_back(dir);
    }
}

Point2f ofApp::getAverageDirection() {
    Point2f averageDir(0, 0);
    for (int i = 0; i < directions.size(); i++) {
        averageDir.x += directions.at(i).x;
        averageDir.y += directions.at(i).y;
    }
    if (directions.size() != 0) {
        averageDir.x /= directions.size();
        averageDir.y /= directions.size();
    }
    return averageDir;
}
```

Check whether the movement is larger than a threshold, which needs to be adjusted. If so, change the status.

```
//the threshold needs to be adjusted
if ((abs(averageDir.x) - abs(prevAverageDir.x)) / abs(prevAverageDir.x) > 0.3) {
    status = "left";
}
else if ((abs(averageDir.x) - abs(prevAverageDir.x)) / abs(prevAverageDir.x) < -0.3)
{
    status = "right";
}
else if ((abs(averageDir.y) - abs(prevAverageDir.y)) / abs(prevAverageDir.y) > 0.1) {
    status = "down";
}
else if ((abs(averageDir.y) - abs(prevAverageDir.y)) / abs(prevAverageDir.y) < -0.1)
{
    status = "up";
}
else {
    status = "Still";
}
prevAverageDir = averageDir;
```

Finally, if we want to clear all the key points, just press the key 'c', they will be all cleared and the status will be "Init".

Besides, in this program, due to my OpenCv version is less than 3.0, I use the flag CASCADE_SCALE_IMAGE instead of CV_HAAR_SCALE_IMAGE in the detectMultiScale() method.

```
// Because of my OpenCv version, I use CASCADE_SCALE_IMAGE instead of CV_HAAR_SCALE_IMAGE
// You can adjust for your own version of OpenCv.
// Here are the reference links: https://docs.opencv.org/3.1.0/d9/d31/group__objdetect__c.html#ga812f46d031349fa2ee78a5e7240f5016
//                               https://answers.opencv.org/question/20066/simple-face-recognition/
face_cascade.detectMultiScale(vidMat, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, cv::Size(30, 30));
```

You can adjust for your own version of OpenCv.

Here are the reference links:
https://docs.opencv.org/3.1.0/d9/d31/group__objdetect__c.html#ga812f46d031349fa2ee78a5e7240f5016

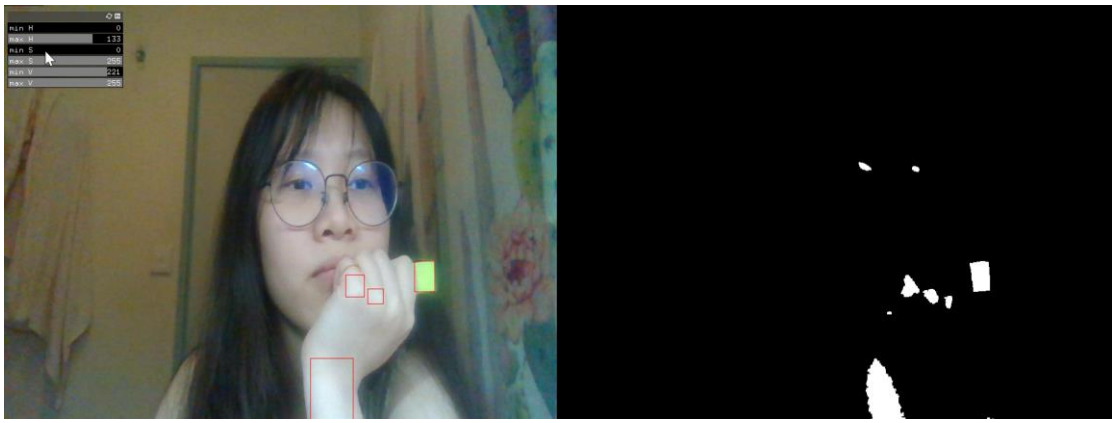
https://answers.opencv.org/question/20066/simple-face-recognition/

2. **Drawing Interface using Color Tracking**

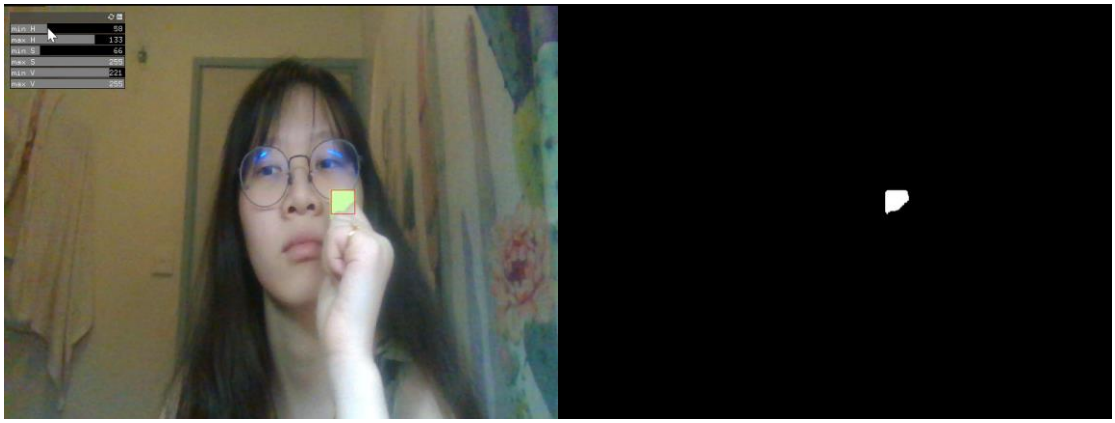
We used the method of color-based blob tracking, to fulfill the following interaction steps.

Step 1: Wear a finger cap on your finger.

Step 2: Adjust the sliding bars to track the finger cap the only recognizable object in the webcam image.



In this case, I used a green sticker instead of a finger cap, and then adjusted the slider to track my green sticker only in the webcam image.



Here is the main code.

Firstly, set up the webcam and the slider bar for adjustment.

```
gui.setup();
gui.add(minH.setup("min H", 0, 0, 180));
gui.add(maxH.setup("max H", 180, 0, 180));
gui.add(minS.setup("min S", 0, 0, 255));
gui.add(maxS.setup("max S", 255, 0, 255));
gui.add(minV.setup("min V", 0, 0, 255));
gui.add(maxV.setup("max V", 255, 0, 255));
```

Then we get images from the webcam and set the image within the range and eliminate the noise by erode() and dilate() functions.

```
cvtColor(matCam, mat_HSV, CV_BGR2HSV);
inRange(mat_HSV, Scalar(minH, minS, minV), Scalar(maxH, maxS, maxV), mat_HSV_Threshold);
erode(mat_HSV_Threshold, mat_HSV_Threshold, Mat());
dilate(mat_HSV_Threshold, mat_HSV_Threshold, Mat());

ofImage im_temp;
ofxCvGrayscaleImage im_temp_gray;

toOf(mat_HSV_Threshold, im_temp);

im_temp_gray.setFromPixels(im_temp.getPixels());

contourFinder.findContours(im_temp_gray, 5, (640 * 480) / 4, 4, false, true);
```

Then we use findContours() function to detect the objects in particular color.

If we find something, we draw the results.

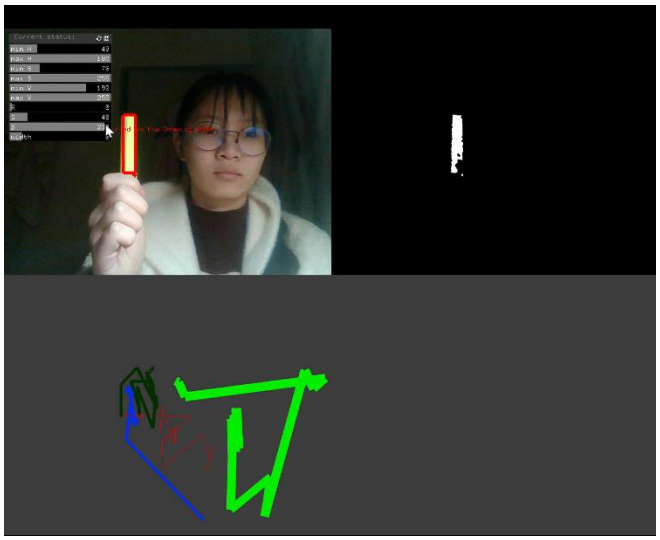
```
for (int i = 0; i < contourFinder.nBlobs; i++) {
    ofRectangle r = contourFinder.blobs.at(i).boundingRect;

    centerX = r.x + (r.width / 2);
    centerY = r.y + (r.height / 2);

    if (drawMode) {
        ofDrawBitmapString("Turned on the Drawing mode.", 200, 200);
        if (usingObjColor) {
            curColor[lineNo] = myVideoGrabber.getPixelsRef().getColor(centerX, centerY);
        }
        else
        {
            curColor[lineNo] = ofColor(R, G, B);
        }
        ofSetColor(curColor[lineNo]);
        drawLine[lineNo].addVertex(centerX, centerY+matCam.rows);
        curWidth[lineNo] = width;
        ofDrawCircle(centerX, centerY+matCam.rows, curWidth[lineNo]);
        ofSetLineWidth(curWidth[lineNo]);
        drawLine[lineNo].draw();
    }

    ofSetColor(255, 0, 0);
    ofNoFill();
    ofDrawRectangle(r);
}
```

Step 3: Press key ‘d’ to start/stop the drawing mode.

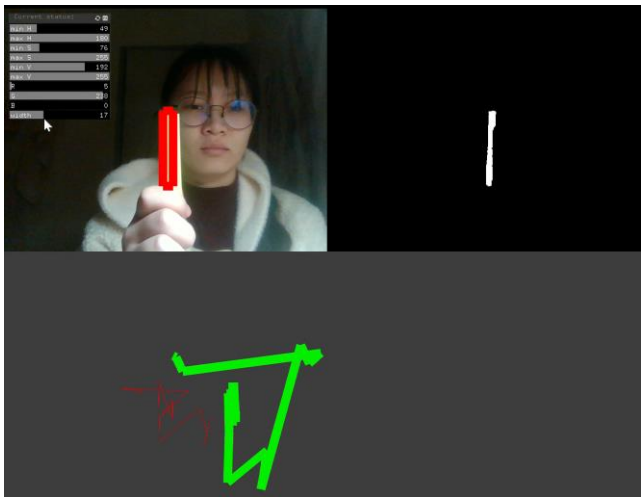
Under the drawing mode, you can move your finger in front of webcam and draw the line sketch with the color you like based on RGB value in the slider bar, as well as the width of the line.

If you don’t want to choose color by yourself, you can press ‘o’, it will set the color of the finger cap on top of the webcam image.

Drawing Mode:



Stop Drawing:



When the user stops the drawing mode, and presses ‘d’ to start again, a new line sketch will be created.

Here is the main code.

```
if (drawMode) {
    ofDrawBitmapString("Turned on the Drawing mode.", 200, 200);
    if (usingObjColor) {
        curColor[lineNo] = myVideoGrabber.getPixelsRef().getColor(centerX, centerY);
    }
    else
    {
        curColor[lineNo] = ofColor(R, G, B);
    }
    ofSetColor(curColor[lineNo]);
    drawLine[lineNo].addVertex(centerX, centerY+matCam.rows);
    curWidth[lineNo] = width;
    ofDrawCircle(centerX, centerY+matCam.rows, curWidth[lineNo]);
    ofSetLineWidth(curWidth[lineNo]);
    drawLine[lineNo].draw();
}
```

In addition, We allow users to create different line sketches with the color of the object held multiple times, so I created a ofPolyline variable “drawLine[10]” with ofColor variable “curColor[10]” and float curWidth[10] to represent different drafts, which means users can draw up to 10 different line sketches.

```
for (int i = 1; i <= lineNo; i++)
{
    ofSetColor(curColor[i]);
    ofSetLineWidth(curWidth[i]);
    drawLine[i].draw();
}
```

You can press ‘s’ to save the drawing reault. The image will be stored in the program’s data file, and with a file name “mySketch_0x.png” where x is the number of the images you have saved.

You can also press ‘i’ to clear the graffiti.

3. **Music player**

You can press ‘m’ to play the music and press ‘n’ to stop it.

All the interaction with the keyboard is shown below.

```
//--------------------------------------------------------------
void ofApp::keyPressed(int key) {

    switch (key) {
    case 'c':
        cout << "clear points" << endl;
        prePoints.clear();
        curPoints.clear();
        status = "Init";
        break;
    case 'r':
        cout << "re-initialize" << endl;
        needToInit = true;
        break;
    case 'd':
        if (!drawMode) {
            drawMode = true;
            lineNo++;
        }
        else {
            drawMode = false;
        }
        break;
    case 'i':
        lineNo = 0;
        break;
    case 'm':
        mySound.play();
        break;
    case 'n':
        mySound.stop();
        break;
    case 'o':
        usingObjColor = !usingObjColor;
        break;
    case 's':
        outputDraw.grabScreen(0, matCam.rows, matCam.cols, matCam.rows);
        outputDraw.save(filename + ofToString(countSketch, 2, '0') + ".png");
        countSketch++;
        break;
    }

}
```

DEMO LINK

Link of an online video showcasing my application: https://drive.google.com/file/d/1d-FfxnH91zLJ62TCsA5BcDM_p92H75YH/view?usp=sharing

REFERENCE

1. Gary Bradski and Adrian Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library.
2. OpenFrameworks Community. OpenFrameworks. https://openframeworks.cc/documentation/