

Final Project
Gesture based paint system
Visual Interface COMSW 4735 Spring 2015

Angus Ding ad3180
Ayaka Kume ak3682

May 13, 2015

Contents

1	Introduction	2
1.1	Purpose of this project	2
1.2	Previous work	2
1.3	Program features	2
1.4	Domain engineering	2
1.5	Division	3
2	Overview of the method	4
3	Hand detection	5
3.1	Skin color detection	5
3.2	Orientation of the hand	7
3.3	Hand detection	9
3.4	Result	11
4	Posture recognition	13
5	Fingertip detection	15
6	Determine if the finger is touching the paper	18
7	Evaluation	19
7.1	The average error measurement	19
7.1.1	Points	19
7.1.2	Circle	19
7.1.3	Line	19
7.2	Results	21
8	Discussion	25

1 Introduction

1.1 Purpose of this project

In this project, we implemented a system that allows the user to paint on the screen using intuitive gestures instead of the mouse. Instead of using any specific draw pad or other device, we will implement the system using a simple white paper, a web cam, and a light source. Without any tactile input, we are going to rely on visual inputs to determine the gesture and the position of user's hand in real-time. The challenging part about this project is how to define the natural gestures that human use to indicate the drawing on a blank paper, and how to recognize them using pure visual signal processing. Because the system only rely on the visual input, this project is perfectly suitable as an example of a visual interface.

1.2 Previous work

EnhancedDesk[1] is a two handed drawing system using infrared camera. Left hand and right hand have the different role. Isard, Michael, and John MacCormick implemented a vision based drawing package to demonstrate the hand tracking method[3].

1.3 Program features

The user will be given a device that consists of a white paper, a webcam that looks down from above, and a light source that projects light onto the paper from a non-perpendicular angle. The distance between the webcam and the paper, the paper and the light source, and the angle of the light source are all fixed. On the bottom of the paper are some color blocks which represent a palette, and possibly some symbols which represent the drawing tools that the user can choose. The user can simply touch the color blocks to choose the color, and touch the tool symbols to choose the painting tool he or she wants to use. On the same time there will be a program on the computer screen which shows the canvas on which the user draws. To draw a picture, the user can use the most intuitive gesture – use the index finger like a pen. Touching the paper with the index finger means to draw, while moving the finger without touching the paper means to move the pen without drawing. This gesture, when the user touches the palette or the symbols instead of the empty area, means to select the color or the tool instead of drawing. For convenience, we will also define an erase gesture, which is a palm facing downwards with the four fingers stretching straight. This gesture is easy to use, and suitable for the semantic of erase, because it is the movement one will use to wipe something away from a surface.

1.4 Domain engineering

Fig.1 shows the setting of our device. The height from the camera to the paper canvas is 54 cm. The camera looks down so that we can convert the coordinate

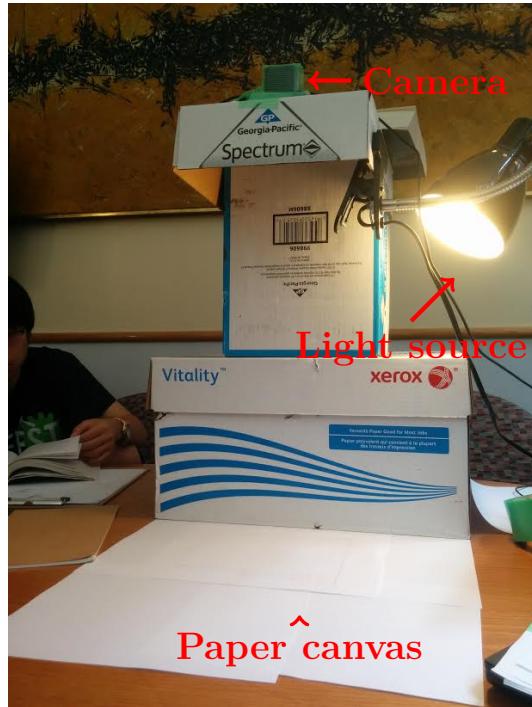


Figure 1: The input device

of the fingertip to the canvas coordinate easily. The canvas on the paper is 19cm by 14cm. We set all of back ground as white in order to detect hands easily. We use the web cam, logicoal carl zeiss tessar. We use Windows7 and python 2.7. As a light source, we use handy light. The user is assumed to be east Asian, because our training data contains only east Asian.

1.5 Division

Angus Ding (ad3180) implemented and wrote a report about posture detection, shadow detection and GUI part. Ayaka Kume (ak3682) implemented and wrote a report about hand detection, fingertip detection and evaluation part. We wrote introduction and discussion together.

2 Overview of the method

To complete our system, we need three informations for each frame: the posture of the hand, the location of the fingertip and whether the user's hand is touching the paper. We will dicuss how we get these informations in the following sections.

To implement the drawing function, the system has a drawing flag which is either true or false. The system is in the drawing state if and only if

- The posture of the hand is the "pointing finger"
- The fingertip is touching the paper

âAč If the system is in the drawing state in both the previous and the current frame, a line will be drawn from the previous location of the fingertip to that of the current one. We draw a line instead of drawing a point at each location because the movement of the user's hand is usually too fast for the frequency with which we process the frames, which makes a continuous movement of the finger draw many disconnected dots instead of a line, as might be the user's intention. If either the previous frame or the current frame is not a drawing frame, then we don't draw the line.

The similar process is defined for the erase function. The difference is that the "erasing" state is defined with the "palm" posture instead of the "pointing finger", and instead of drawing a line with the currently selected color, we simply draw a line with the background color, which is default to be white.

To allow the user to select different color to draw, we define a "hand down" event. A "hand down" event happens if the user is not touching the paper in the previous frame and is touching the paper in the curren frame. When this event happens and when the user's posture is "pointing finger", we check if the location of the finger point is in the predefined area of one of the colors. If it is, we change the current seleceted color to that color.

No matter what the events are, the location of the cursor is always set to the current location of the fingertip. This is important because even if the user is not currently drawing, without the cursor matching the movement of the hand the user won't know where he or she should put the hand down. If we can not detect the fingertip, then we assume the user is not currently using the system, and simply does nothing about this frame.

Now we discuss how we will reteive the three important information of each frame.

3 Hand detection

Because of our settings, there are only white background, shadow and the hand in an image. First, we detect hand using skin color. Then we detect wrist. Because there are only hand or wrist in the scene, we can get hand mask by erasing wrist region. Fig.2 shows the overview of the system. For wrist detection, we modify the method from [4]. All of the functions in this section is in majoraxis.py and hand_detection.py.

3.1 Skin color detection

Because of our settings, there are only white background, shadow and the hand in an image. So we detect the hand by color. We use both RGB and HSV value to detect our skin. Because both of our team mates are East Asian, we tried skin detection only for East Asian people. In particular, we define skin color pixel as:

- its Red value is larger than Blue value
- its Red value is larger than Green value
- its Value (HSV) is smaller than 73 %
- its Saturation is larger than 30 %

Also we mask where outside of the canvas.

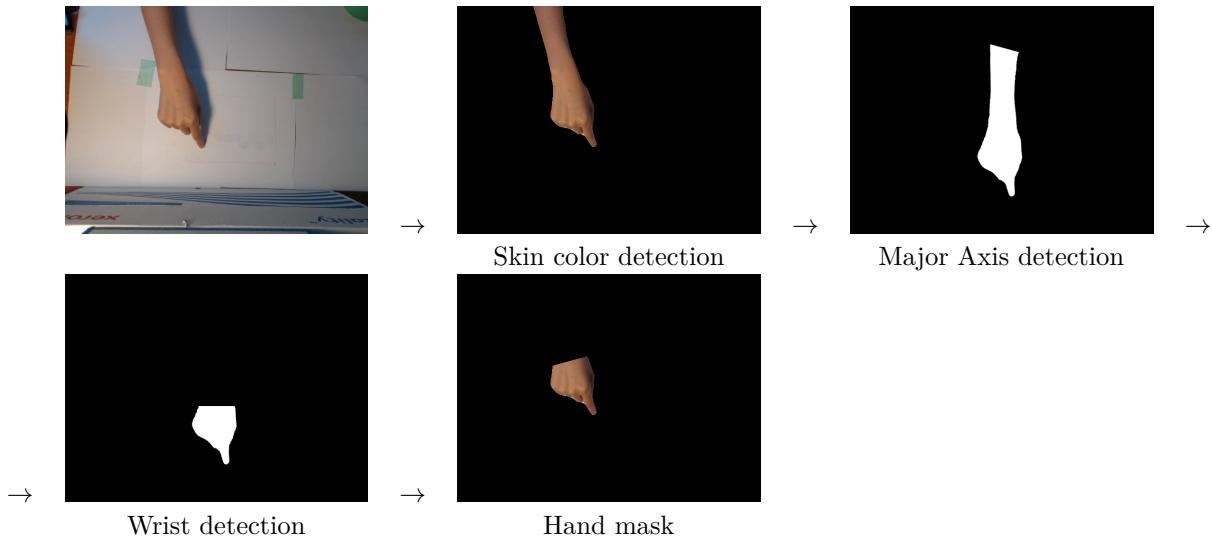


Figure 2: Overview of hand detection

3.2 Orientation of the hand

We want to detect the orientation of the hand in order to use training data, or detecting hand region. In our program, major axis.py has this function. We define the orientation of the hand as the angle of the axis of least second moment.

The input is the binary image of skin region. Axis of least second moment minimizes E, the sum of the distance from all points to the line. That is,

$$E = \int \int r^2 b(x, y) dx dy$$

where r is a distance from $b(x,y)$ to the axis and $b(x,y) = 1$ when a pixel (x,y) belongs to the object, otherwise 0. Let the axis be $x \sin \theta - y \cos \theta + \rho = 0$. Distance of point (x,y) from axis is:

$$r = |x \sin \theta - y \cos \theta + \rho|$$

. Thus minimizing E means minimizing

$$E = \int \int (x \sin \theta - y \cos \theta + \rho)^2 b(x, y) dx dy$$

Because $\partial E / \partial \rho = 0$, we get

$$A(x_c \sin \theta - y_c \cos \theta + \rho) = 0$$

where A is an area of the object and (x_c, y_c) is center of the object. This means, the axis should pass the center point of the object. Then, we shift the coordinate system in order to set the center point as origin. That is, $x' = x - x_c, y' = y - y_c$. Because this line should pass the origin, the line can be represented as $x' \sin \theta - y' \cos \theta = 0$. So,

$$E = a \sin^2 \theta - b \sin \theta \cos \theta + c \cos^2 \theta$$

. Where $a = \int \int (x')^2 b(x, y) dx' dy', b = 2 \int \int x' y' b(x, y) dx' dy', c = \int \int (y')^2 b(x, y) dx' dy'$.

Because $\partial E / \partial \theta = 0$, we get

$$(a - c) \sin 2\theta - b \cos 2\theta = 0$$

Also, minimizing E means the second derivative is larger than 0. Using these information, the orientation $\theta = \text{atan}2(b, a - c)/2$.

Fig.3 shows the results of orientation detection and the rotated image. The first column is an original image. The second column is a translated image. First, the center point of the hand moves to the center point of the image. The light blue line is the axis. The blue dot is the center point. For the mask, the image is rotated by the angle of $-\theta$ and translated to the original position. The figure shows that this axis does not depends on the small fingertip movement. If the binary image of hand has enough amount of areas, this system can detect the angle of the hand. If the image of hand does not have enough amount of areas, for example, it can detect only a part of fingers, it cannot detect the angle of the hand correctly. However, because of our settings, we always can see enough amount of hand in the target area.

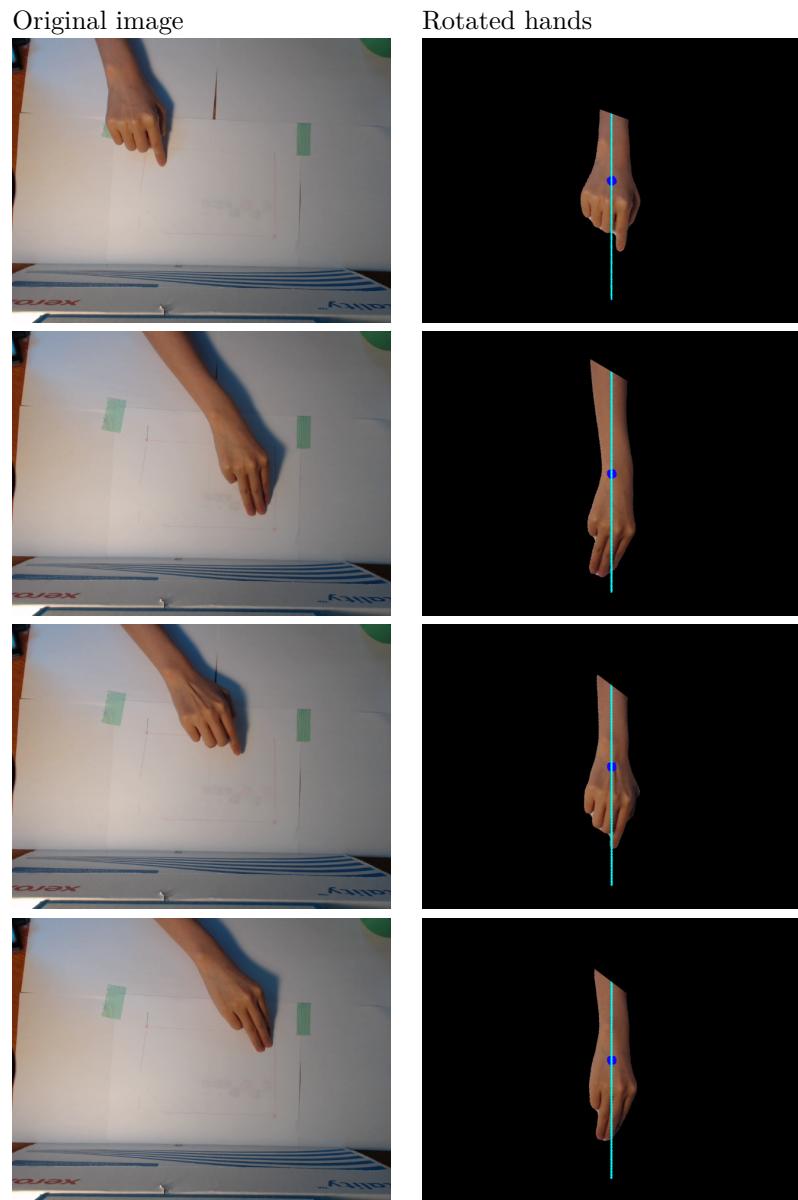


Figure 3: The results for orientation detection

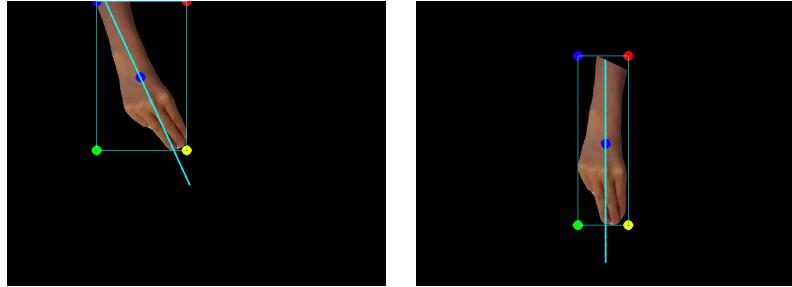


Figure 4: Hand image Left: original skin color image Right: rotated skin color image

3.3 Hand detection

Then we have to extract only hand region. If there are long arms, we may not classify the gesture correctly. We modified the method introduced by [4]. The method is first detect the skin region by color (HSV), and then detect the wrist end. Wrist end is detected by the simple method as follows: Fig.4 shows the image of the hand. First, we calculate the number of the pixels on boundaries, up (between blue point and red point), down (between green point and yellow point), left (between blue point and green point), right (between red point and yellow point). We can assume the most largest among up, down, left and right is the wrist side. This is because the author of [4] and we assume hand is inside of the image, but human itself is not. Then, they detect the wrist end using intensity histogram. Intensity histogram is the sum of the number of the pixels on the row/cols. If wrist is up/ down, it calculate along rows. Otherwise, it calculate along columns. Assume the wrist is down. Let p be a point which is either on a left or right boundaries and the nearest from the down boundary. The author of [4] found that the slope on the histogram between b and the wrist end is highest among the slopes between b and other points which is nearer to down than b . b is the point which is the most left or right point so it is always near the most widest region.

However, the wrist detection does not work well because the paper assumes the hand gesture as spray hand and so the palm is always the widest. Like Fig.4, we cannot find appropriate b because the most left or right points are not palm, but finger and wrist. This is because palms are not fully opened so it is difficult to detect hand region as it is. So we rotate the image along the axis so that we can assume the wrist is always 'up' side and the most left or right part tend to be the palm region. Fig.4 can detect wrist by our method but not the previous method. This is because our method can detect the most widest point as b . We assume b is not too near to the wrist end. That is, if left or right is within 30 pixel from up, use another point. Fig.5 shows the example of the histogram. Red point in the histogram corresponds to the b 's coordinate. Green point in the histogram corresponds to the wrist end's coordinate. The

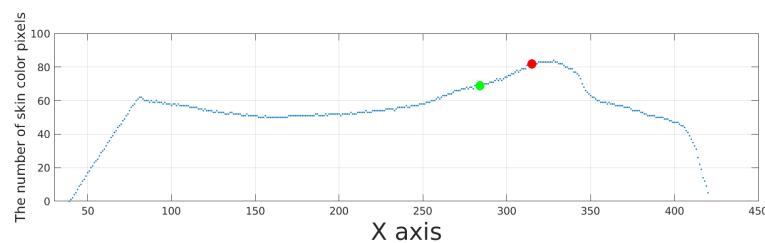
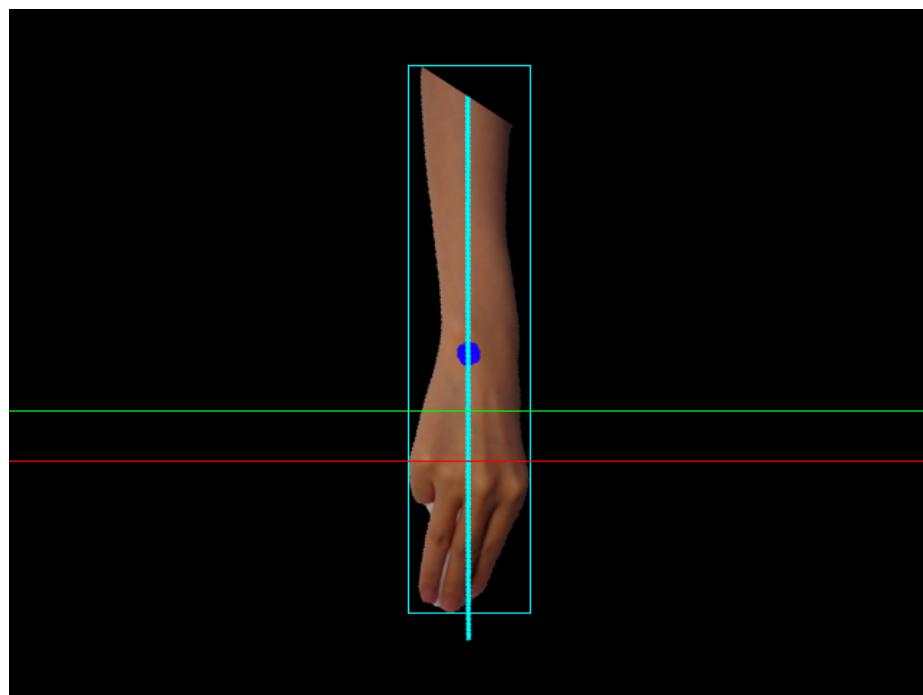


Figure 5: Hand image Top: Hand image. Green line is wrist position and red line is b. Bottom: histogram

red line and green line in the left image corresponds to the b and wrist end.

Taken together, our method is as follows.

1. Find skin color area
2. Find orientation of the skin area and rotate
3. Assume up side is wrist
4. Find wrist end and crop

Even though we improve the method, in case the skin detection fails and the hand become smaller or the palm is too small to be a widest length, in that case, we simply extract 1/4 of the all of the region.

3.4 Result

Fig.6 shows the results of the hand detection. As we can see, the hands are correctly detected.

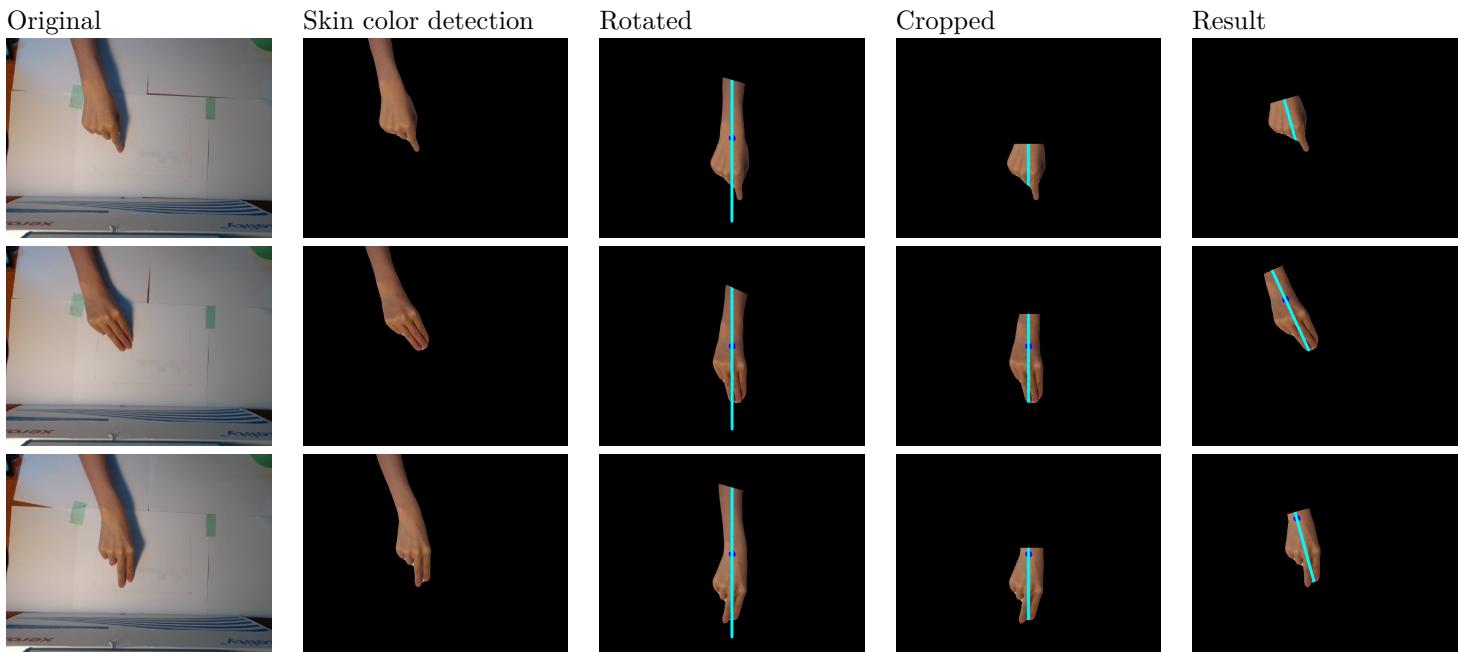


Figure 6: The results of hand detection

4 Posture recognition

To as suggested by [2], since we analyze the hand frame by frame instead of analyzing it over a time period, we would call the information posture instead of gesture, which suggests a movement. The problem we would like to solve here is, given the input image, we want to decide which posture does the hand in the image poses. As a matter of fact, this is a classic classification problem in the machine learning literature, so we choose to solve this with a machine learning approach. It is possible to come up with another way to deal with this problem, for example one could give a clear definition for each of the postures. However, one would have to do this for each postures used in the system. Thus, for generality, although our system only utilizes two postures, we will still solve it with a machine learning technique.

The algorithm we use is the k-nearest neighbors algorithm. Simply put, this algorithm finds the k images in a database that are the “nearest” to the image to be classified, then classifies it as the most common posture among the k neighbors. The “similarity” between the images should be properly defined to reflect the nature of the images and the postures. The problem then is how we should define the similarity.

Because we are only classifying the hand posture, the other information in the image is unnecessary. Thus for each image we first apply the hand detection algorithm to find the area of the hand, then we discard all the pixels that are deemed not to be hand. We can then calculate the orientation histogram of the hand. That is, we calculate the gradient orientation of each pixel, defined as

$$\arctan(dx, dy) \quad (1)$$

where dx, dy are the image gradient of the pixel. However, since the image $img(x, y)$ is a discrete function, instead of the gradient, we can only compute an approximate of it. The approximation we choose is the Sobel operator. The Solbel operator, in its simplest case, basically has two kernels: one for horizontal gradient and one for vertical gradient. The kernels are

$$G_x(Img) = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * Img$$

$$G_y(Img) = \begin{bmatrix} -1 & -2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & +1 \end{bmatrix} * Img$$

, where Img is the input image, and $*$ symbol is the 2-D convolution.

After we compute the orientation of each pixel, we put it into one of the 36 bins. The ‘ith bin is $[-\pi + (i - 1) * \pi/18, -\pi + i * \pi/18]$. Thus, we will have an orientation histogram of length 36. This vector can serve as our feature for the image.

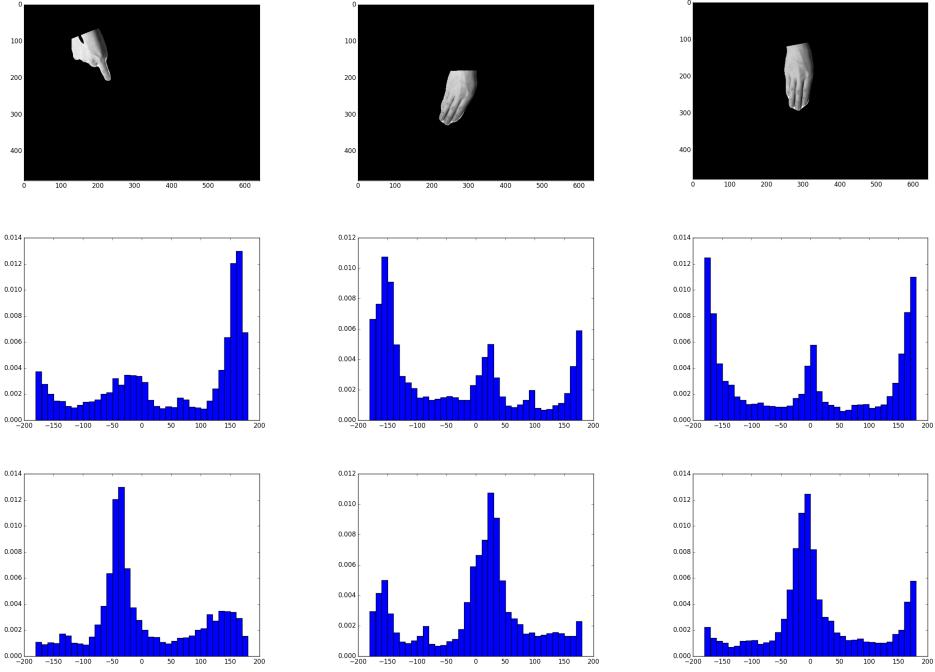


Figure 7: The orientation histograms. The first row is the input image, converted to gray scale and applied hand detection. The second row is the orientation histogram. The third row is the orientation histogram after the translation of the major axis.

Since the same hand posture can have different orientation, we can further reduce the intrinsic dimension of the feature by rotating the image. We assume that when the user makes the same posture with different orientations, the orientations histogram will have the same distribution, but will have a disposition. Thus, if we rotate the image so that the hand in the two image have the same orientation, their orientation histogram should be similar. This can be done by finding the major axis of the hand, then, instead of rotate the image before computing the orientation histogram, we can simply translate the orientation histogram by the angle of the major axis. As can be shown in Fig. 7, this method does give us a histogram that is distinctive with respect to the hand posture, and insensitive to the orientation of the hand. \square

Table 1: The result of three fingers' finger tip detection

The location of the result	
Index finger	0(0%)
Between index finger and middle finger	1(0%)
Middle finger	124 (87%)
Between middle finger and third finger	12(8%)
Third finger	6(4%)
Total	143

5 Fingertip detection

In order to draw points using finger information, we have to detect the location of the fingertip. If the gesture is 'draw (one finger)', we detect the fingertip of the index finger. If the gesture is 'erase (three fingers)' or 'move (two fingers)', we detect the fingertip of the middle finger.

We assume the fingertip is the further point from the center of the mass and also not the wrist side. Our environment allows us to assume the wrist side is always top of the image, we simply find the furthest point from the center of the mass among the bottom half of the mask.

This function is implemented in `fingertip.py`.

Fig. 8 shows the results of the finger tip detection. This method detects the finger tip correctly when the gesture is one finger or two fingers. When the gesture is three fingers, the method sometimes detect the location between middle finger and other fingers. Table 1 shows the result of detected finger tip when the gesture is 'three fingers'. We tried 143 frames of three fingers. The error ratio is 12 %. Fig. 9 shows the examples of failure. As we can see, it fails when the hand is not on the canvas. This would cause about 10 pixels error in video image and cause about 20 pixels error in canvas. This is about 3% of the average length of the canvas.

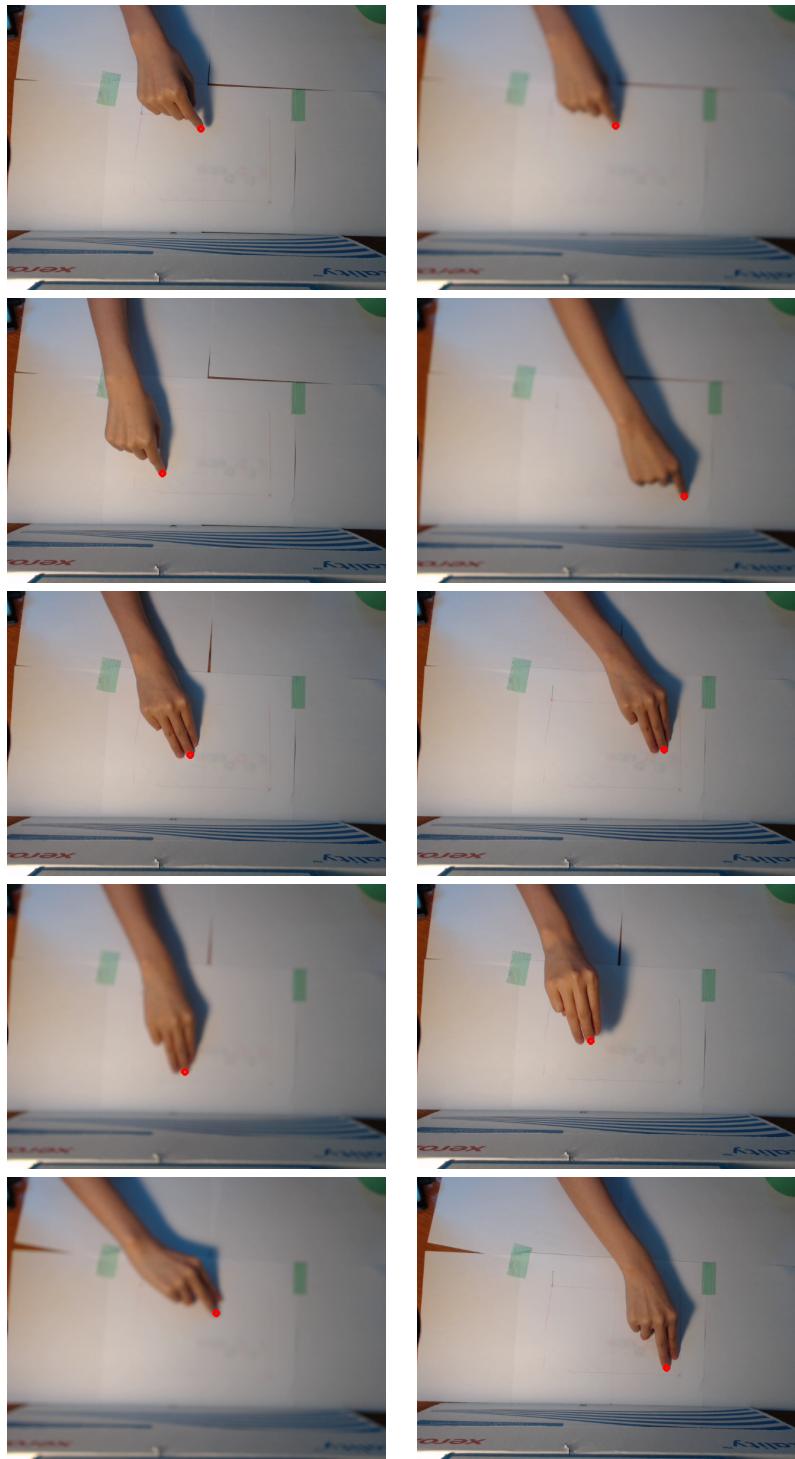


Figure 8: The results of finger detection
Page 16 of 55

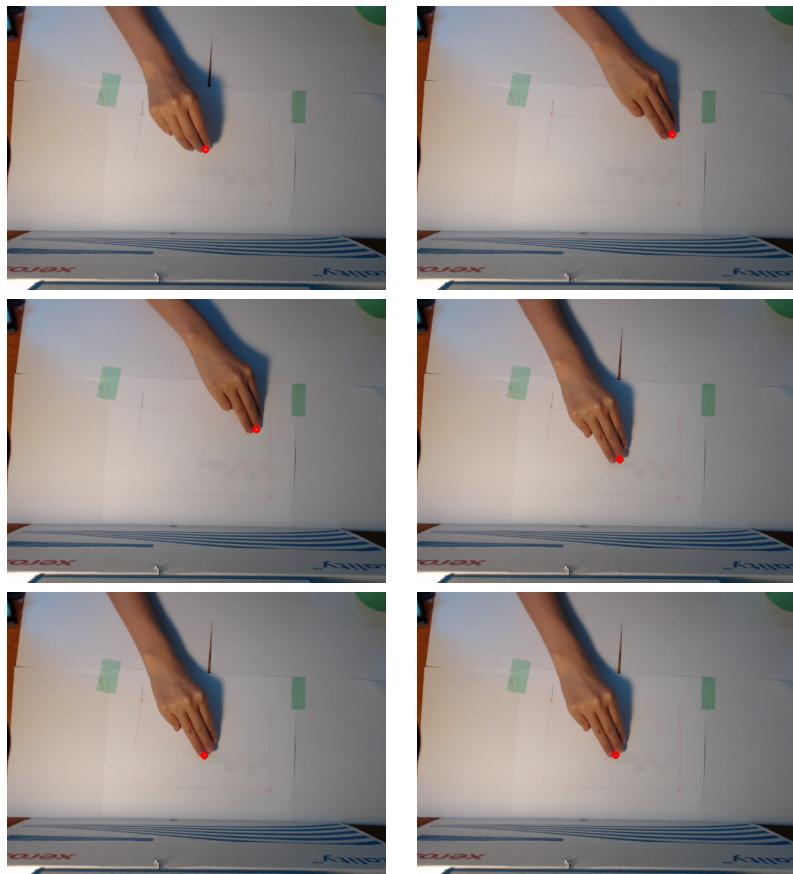


Figure 9: The errors

6 Determine if the finger is touching the paper

It is difficult to determine whether the finger touches the paper solely with the image taken from the direct above. As such, we use a light source to project the shadow of the hand onto the side of the hand. Suppose the light source is set to have angle θ from the vertical line, and the fingertip is at height h , then the distance between the fingertip of the shadow and the x-coordinate of the real fingership is $d = h \tan(\theta)$. So, if θ is not 0, we can find h with $h = d \cot(\theta)$. Or, since we only care if the finger is touching the paper or not, we can set a threshold d_{th} such that we can assume the finger is touching the paper if and only if $d < d_{th}$.

This method thus requires the detection of the shadow and its fingertip. We detect the shadow with the similar method for hand detection: using color filters and HSV filters. The actually condition we use are as follows:

- $R < 107.1$
- $G < 86.7$
- $B < 120.6$
- $V < 100.8$

â€¢ , where R, G, B are the red, green and blue values respectively, and V is the value in HSV representation.

To find the fingertip of the shadow, we can not use the same method proposed for finding the real fingertip because a large portion of the shadow is blocked by the hand when the hand is close to the paper, thus the center of mass of the shadow can be greatly affected. Instead, we simply assume that the fingertip is always pointing down, and find the pixel of the shadow that has the lowest y-coordinate. Because we set the light source to be at the lower left corner of the screen, the shadow of the hand will always be at the upper-right side of it. Under the assumption that the finger is always pointing down, we ignore all the shadow pixels that are to the left and lower than the fingertip to reduce noise. Further more, since we only care if the fingertip is touching the paper, we can ignore all the shadow pixels that are too far away from the real fingertip. So, we only find the shadow fingertip in the region of $\{(x, y) : x_{finger} < x < x_{finger} + 40, y_{finger} < y\}$.

Finally, we have to decide the threshold for the distance between the real fingertip and the shadow fingertip. This threshold is affected by the posture. When the user is posing the palm posture and is touching the paper, a larger portion of the shadow will be blocked than when the user is posing the pointing finger. So, the threshold we decided was 30 for the pointing finger and 50 for the palm.

7 Evaluation

For evaluation, we measure time, error rate and the consistency of each trials. If time is fast, it means the system is easy to operate. If the average error is low, it means the system is accurate. If time and average error does not change for every trial, it means the system is stable.

We only evaluate about 'drawing' function because 'drawing' function is the most essential function of the system. We evaluate this system using three tasks. Fig. 10 shows three examples. The flow of the evaluation is as follows. At first, there is a white canvas and pointer. When the tester press 'r', it shows the sample to trace and starts timer. User starts drawing and when he finish drawing, the tester press 'q' and timer ends. The user do same task three times. Time is measured by this timer. The average error is measured by the difference between the example and what he draw. The consistency is measured by the standard deviation of time and average error.

To compare the results, we also tested the mouse as the input device. When we point a point in the canvas using a mouse, the pointer moves. When we drag with left click, it draws a line.

We used user as both of our team member. For this evaluation, I used evaluation.py, evaluation_m.py, eval_ui.py, mousepaint.py and evala.py. evala.py is the main function for measurement. eval_ui.py and mousepaint.py make gui environment. evaluation.py and evaluation_m.py is actual execution files.

7.1 The average error measurement

7.1.1 Points

We measure the distance between what the user draw and the nearest point of the sample. To make the measurement easy, we divide the canvas into 4 section. If a point the user draw is on upper left, we compare the distance with upper left point of the sample. Same as upper right, lower left and lower right.

7.1.2 Circle

Circle is the points which distance from the center point are radius. We know the center point and the radius of the sample circle. Thus we measure the absolute difference between radius and the actual difference from the center point for each points. Then we average them.

7.1.3 Line

We measure the distance between the actual line and sample line by comparing only 20 points of the lines. We equally sample 20 points in the lines along the vertical axis. And from the most highest points to the most lowest Points, we measure the difference of them. Fig.11 shows the example. Blue line is a sample line and red line is user's line. The points are sample points. Each points are measured with the points which are connected by the black line.

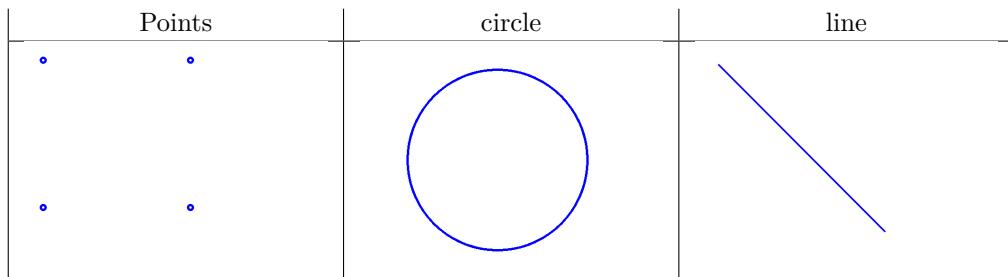


Figure 10: Three tasks

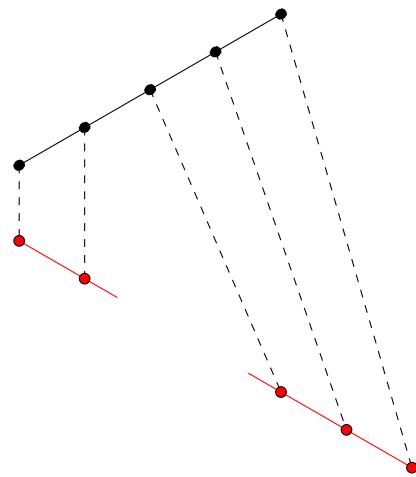


Figure 11: Line error measurement

7.2 Results

Table 2 shows the measurement results of the points task. Table 3 and Table 4 show the results of the circle task and the actual canvas image. Table 5 and Table 6 show the results of the line task and the actual canvas image. Table 7 shows the standard deviation of them. In all tasks, mouse device results are better than our input device. Among three tasks, for our device performance is better in circle task. We can see the time deviation is better than the error deviation.

Table 2: The result of the user study (points)

User	Device	Trial	Time[sec]	Error Pixels
A	VI	1	19.74	43.59
A	VI	2	16.64	58.73
A	VI	3	14.38	67.47
B	VI	1	23.06	49.28
B	VI	2	19.85	42.29
B	VI	3	22.29	66.24
A	Mouse	1	8.18	4.65
A	Mouse	2	6.28	7.80
A	Mouse	3	6.60	6.47
B	Mouse	1	6.35	3.78
B	Mouse	2	5.48	4.18
B	Mouse	3	5.29	4.55

Table 3: The result of the user study (circle)

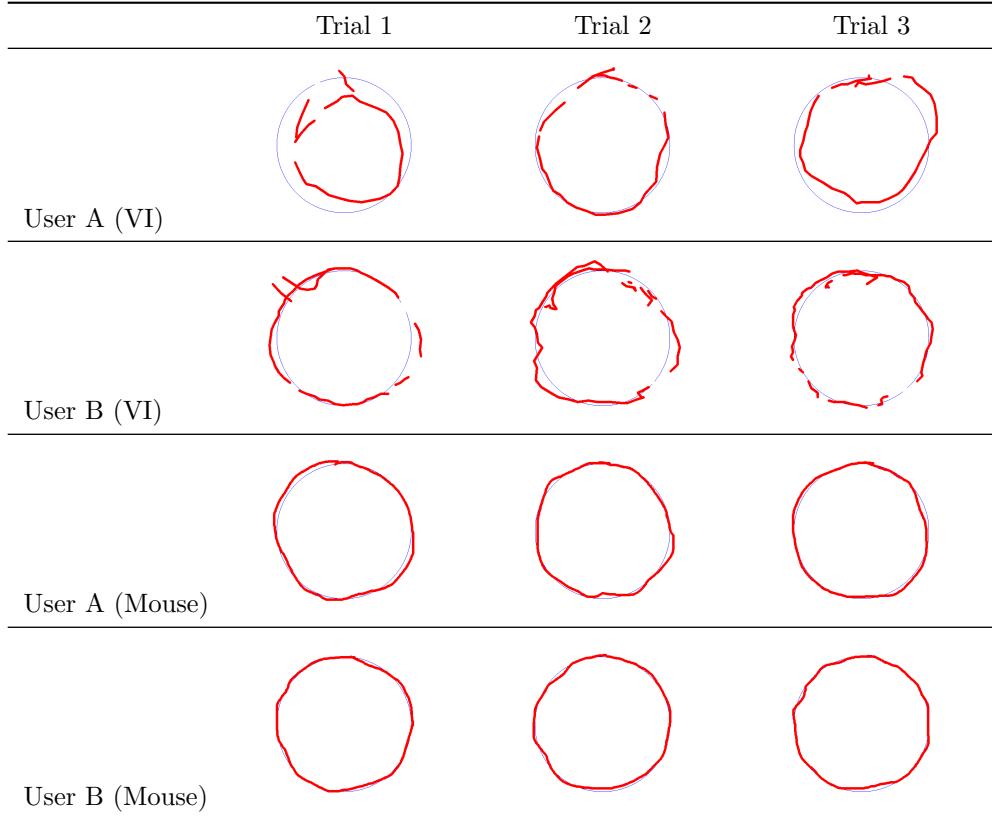


Table 4: The result of the user study (circle)

User	Device	Trial	Time[sec]	Error Pixels
A	VI	1	13.31	38.25
A	VI	2	12.65	9.47
A	VI	3	13.16	24.21
B	VI	1	18.53	11.54
B	VI	2	22.82	12.97
B	VI	3	22.10	7.69
A	Mouse	1	7.90	6.73
A	Mouse	2	7.97	5.12
A	Mouse	3	7.79	4.95
B	Mouse	1	9.38	3.84
B	Mouse	2	9.58	4.27
B	Mouse	3	9.98	3.64

Table 5: The result of the user study (line)

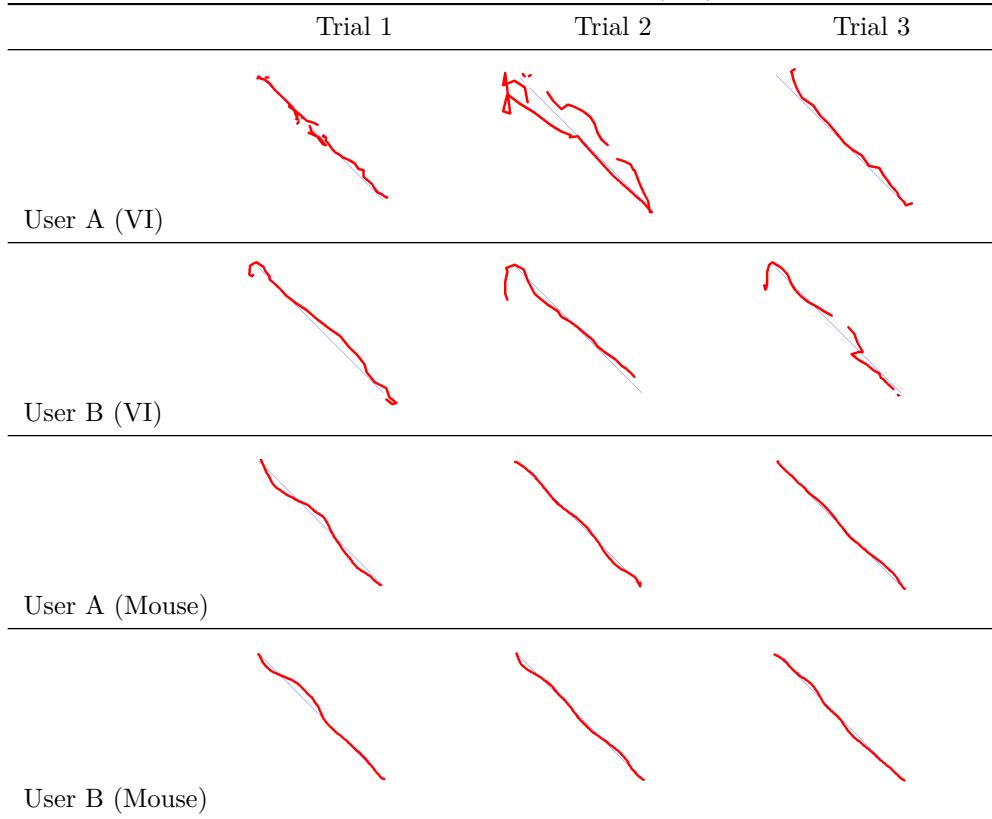


Table 6: The result of the user study (line)

User	Device	Trial	Time[sec]	Error Pixels
A	VI	1	20.16	22.25
A	VI	2	14.75	76.46
A	VI	3	5.73	25.77
B	VI	1	7.76	39.92
B	VI	2	8.03	88.50
B	VI	3	10.03	58.27
A	Mouse	1	4.07	13.87
A	Mouse	2	5.19	6.53
A	Mouse	3	4.07	8.01
B	Mouse	1	3.98	8.99
B	Mouse	2	4.83	7.24
B	Mouse	3	4.61	7.71

Table 7: Standard Deviation of User study

Task	User	Device	StdDev of Time	StdDev of Error
Points	A	VI	2.20	9.87
Points	B	VI	1.37	10.06
Points	A	Mouse	0.83	1.29
Points	B	Mouse	0.46	0.31
Circle	A	VI	0.28	11.75
Circle	B	VI	1.88	2.23
Circle	A	Mouse	0.07	0.80
Circle	B	Mouse	0.25	0.26
Line	A	VI	5.95	24.77
Line	B	VI	1.01	20.03
Line	A	Mouse	0.53	3.17
Line	B	Mouse	0.36	0.74

8 Discussion

References

- [1] Xinlei Chen et al. “Two-handed drawing on augmented desk system”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM. 2002, pp. 219–222.
- [2] William T Freeman and Michal Roth. “Orientation histograms for hand gesture recognition”. In: *International workshop on automatic face and gesture recognition*. Vol. 12. 1995, pp. 296–301.
- [3] Michael Isard, John MacCormick, et al. *Hand tracking for vision-based drawing*. Tech. rep. Technical report, Visual Dynamics Group, Department of Engineering Science, University of Oxford, 2000.
- [4] Jagdish Lal Raheja, Karen Das, and Ankit Chaudhary. “An efficient real time method of fingertip detection”. In: *arXiv preprint arXiv:1108.0502* (2011).

Appendix:Code main

main.py

```
1 import sys
2 import os.path
3 includespath = os.path.abspath('../includes')
4 sys.path.insert(0, includespath)
5 import posture
6 import cv2
7 import gui
8 import hand_detection
9 import fingertip
10 import hand_detection as hd
11 import numpy as np
12
13 def mouse_callback(event,x,y,flags,param):
14     i,j = (y-1,x-1)
15     if event == cv2.EVENT_LBUTTONUP:
16         print i,j
17
18 def getinput(cap, pos_recognizer):
19     ret, frame = cap.read()
20     # Display the input stream only for debug purposes
21     cv2.imshow('input',frame)
22     # print "check point 1"
23     label, hand_mask, theta, skin_mask = pos_recognizer.classify(frame)
24     # print "label = ", label
25
26
27     #cv2.imshow('debug', frame[])
28     # frame_tmp = np.copy(frame)
29     # frame_tmp[hand_mask==False] = 0
30     # cv2.namedWindow('debug')
31     # cv2.setMouseCallback('debug',mouse_callback)
32     # cv2.imshow('debug', frame_tmp)
33
34     if(label == posture.poses["UNKNOWN"]):
35         print "posture = UNKNOWN"
36         # print "check point 2"
37     location, wrist_end = fingertip.find_fingertip(label, skin_mask)
38     wrist_end = 'up'
39     if(not location):
40         return label, location, False
41     # print "location= ", location
```

```

42     # print "wrist_end = ", wrist_end
43     # print "check point 3"
44     touching = posture.isTouching(frame, label, location, wrist_end, hand_mask)
45     return label, location, touching
46
47 def main():
48     modelfilename = sys.argv[1]
49     pos_recognizer = posture.PostureRecognizer.load(modelfilename)
50     ui = gui.GUI()
51     #Get the image and do the classification here
52     cap = cv2.VideoCapture(1)
53
54     while(True):
55         # Capture frame-by-frame
56         label, location, touching = getinput(cap, pos_recognizer)
57         if(not location):
58             if cv2.waitKey(20) == 27:
59                 break
60             continue
61         # print "touching=", touching
62         #####The grammar goes here#####
63         print "label = ", label, "location", location, "touching", touching
64         ui.handle_input(label, location, touching)
65         cv2.imshow('Canvas', ui.get_screen())
66
67
68         pressedKey = cv2.waitKey(60)
69         if pressedKey == 27:
70             break
71
72 if __name__ == '__main__':
73     main()

```

gui.py

```

1 import cv2
2 import numpy as np
3 import posture
4
5 palette_ub = (163, 181)
6
7 palette = {
8     (215, 237): "RED",
9     (249, 271): "BLACK",
10    (282, 304): "GREEN",

```

```
11     (315, 338): "BLUE",
12 }
13 #BGR
14 color_map = {
15     "WHITE":(255,255,255),
16     "BLACK":(0,0,0),
17     "RED":(0,0,255),
18     "BLUE":(255,0,0),
19     "GREEN":(0,255,0),
20 }
21
22 toolmap = {
23     1:'pen',
24     2:'drag',
25     3:'eraser',
26 }
27
28
29 class GUI(object):
30     """docstring for GUI"""
31     def __init__(self):
32         self.size = (480,640)
33         self.canvas = np.ones((480,640, 3L),)*255
34         self.cursor = Cursor(self.size[0]/2, self.size[1]/2)
35         self.color = color_map["BLACK"]
36         self.bgcolor = color_map["WHITE"]
37
38         self.drawing = False
39         self.erasing = False
40
41         self.screen = np.copy(self.canvas)
42         self.update_screen()
43
44
45     def conv_coord(self, input_coord):
46         """Convert the coordinate from the input to the output"""
47         i,j = input_coord
48         center1 = (261.5,322)
49         center2 = (self.size[0]/2.0, self.size[1]/2.0)
50         di1 = i-center1[0]
51         dj1 = j-center1[1]
52
53         scalei = center2[0]*2/151.0
54         scalej = center2[1]*2/222.0
55
56         di2 = - di1*scalei #Up side down
```

```
57         dj2 = - dj1*scalej
58
59     return (int(center2[0]+di2), int(center2[1]+dj2))
60
61     def drawline(self, loc1, loc2):
62         cv2.line(self.canvas, (loc1[1], loc1[0]), (loc2[1], loc2[0]), color=self.color, thickness=2)
63
64     def eraseline(self, loc1, loc2):
65         cv2.line(self.canvas, (loc1[1], loc1[0]), (loc2[1], loc2[0]), color=self.bgcolor, thickness=2)
66
67     def setcolor(self, color_name):
68         """color_name is a string"""
69         self.color = color_map[color_name]
70
71     def settool(self, tool_number):
72         """Tool is a string indicating which tool to change to """
73         self.cursor.tool = toolmap[tool_number]
74
75     def setcursor(self, location):
76         self.cursor.location = location
77
78     def update_screen(self):
79         """update the screen with canvas and cursor"""
80         self.screen = np.copy(self.canvas)
81         cursor_loc_i = self.cursor.location[0]
82         cursor_loc_j = self.cursor.location[1]
83         cv2.line(self.screen, (cursor_loc_j-5, cursor_loc_i), (cursor_loc_j+5, cursor_loc_i), self.color)
84         cv2.line(self.screen, (cursor_loc_j, cursor_loc_i-5), (cursor_loc_j, cursor_loc_i+5), self.color)
85
86     def get_screen(self):
87         """update the screen with canvas and cursor, then return the screen"""
88         self.update_screen()
89         return self.screen
90
91     def handle_input(self, label, location, isTouching):
92         """The method to handle the signals from the device"""
93         cvt_coord = self.conv_coord(location)
94         #paint first
95         if self.drawing and label == posture.poses['POINTING'] and isTouching:
96             self.drawline(self.cursor.location, cvt_coord)
97         if self.erasing and label == posture.poses['PALM'] and isTouching:
98             self.eraseline(self.cursor.location, cvt_coord)
99
100        #Color selection
101        if not self.drawing and isTouching and label == posture.poses['POINTING']: #Finger down
102            if location[0] > palette_ub[0] and location[0] < palette_ub[1]:
```

```
103         for r in palette:
104             if location[1]> r[0] and location[1] < r[1]:
105                 self.setcolor(palette[r])
106
107     # if not self.erasing and isTouching and label == posture.poses['PALM']:#Finger down
108
109
110     #Finally, update state
111     self.drawing = label == posture.poses['POINTING'] and isTouching
112     self.erasing = label == posture.poses['PALM'] and isTouching
113     self.setcursor(cvt_coord)
114
115     self.update_screen()
116
117 def save_canvas(self,filename):
118     """ save canvas """
119     print 'save images'
120     cv2.imwrite(filename,self.canvas)
121
122 def draw_sample(self,image):
123     """ draw sample image on canvas """
124     self.canvas = image
125
126 def handle_input_m(self, label, location, isTouching):
127     """The method to handle the signals from the mouse"""
128     cvt_coord = location
129     #paint first
130     if self.drawing and label == posture.poses['POINTING'] and isTouching:
131         self.drawline(self.cursor.location, cvt_coord)
132     if self.erasing and label == posture.poses['PALM'] and isTouching:
133         self.eraseline(self.cursor.location, cvt_coord)
134     #Finally, update state
135     self.drawing = label == posture.poses['POINTING'] and isTouching
136     self.erasing = label == posture.poses['PALM'] and isTouching
137     self.setcursor_m(cvt_coord)
138
139     self.update_screen()
140
141 def setcursor_m(self, location):
142     self.cursor.location = location
143
144
145 class Cursor(object):
146     """docstring for Cursor"""
147     def __init__(self, init_i, init_j):
148         self.location = (init_i, init_j)
```

149 **self.tool = toolmap[1]**
150

Appendix:Code hand detection

hand_detection.py

```

1  """Package for hand detection"""
2  import cv2
3  import numpy as np
4  import pdb
5  import time
6  import math
7  import majoraxis
8
9  # wrist detection part
10 def detectwrist(mask,theta,c_xo,c_yo):
11     (x,y) = mask.nonzero()
12     if len(x) == 0:
13         return mask
14     # find the minimum / maximum x / y
15     x_min = min(x)
16     y_min = min(y)
17     x_max = max(x)
18     y_max = max(y)
19     # find the center
20     c_x = x.mean(axis=0)
21     c_y = y.mean(axis=0)
22     l = np.where(mask[:,y_min])[0][0]
23     r = np.where(mask[:,y_max])[0][0]
24     slopes = []
25     if l > r and r > x_min + 30:
26         x_2 = r
27         y_2 = sum(mask[r,:])
28     elif l > x_min + 30:
29         x_2 = l
30         y_2 = sum(mask[l,:])
31     elif r > x_min + 30:
32         x_2 = r
33         y_2 = sum(mask[r,:])
34     else:
35         # if the condition is not good, simply 1/4 it
36         x_2 = -1
37         mask[x_min:x_min+int(1*(x_max-x_min)/4),:] = 0
38     # use the method from paper[5].
39     if x_2 > 0:
40         y2mat = np.ones((x_2-30 - x_min,))* y_2
41         x2mat = np.ones((x_2-30 - x_min,))* x_2

```

```
42         index = np.arange(x_min, x_2-30)
43         ss = np.sum(mask, axis=1)/255
44         slope = (y2mat - ss[x_min:x_2-30])/(x2mat - index)
45         wrist = np.argmax(slope)
46         mask[x_min:x_min+wrist,:] = 0
47         rows,cols = mask.shape[:2]
48         M = cv2.getRotationMatrix2D((cols/2,rows/2),(theta*180/math.pi),1)
49         dst = cv2.warpAffine(mask,M,(cols,rows))
50         M = np.float32([[1,0,-cols/2+c_yo],[0,1,-rows/2 +c_xo]])
51         dst = cv2.warpAffine(dst,M,(cols,rows))
52     return dst
53
54 # in order to detect wrist correctly, rotate image before detecting wrist
55 def rotateim(im,theta,c_x,c_y):
56     c_x = int(c_x)
57     c_y = int(c_y)
58     rows,cols = im.shape[:2]
59     dst = im
60     M = np.float32([[1,0,cols/2-c_y],[0,1,rows/2 -c_x]])
61     dst = cv2.warpAffine(im,M,(cols,rows))
62     M = cv2.getRotationMatrix2D((cols/2,rows/2),-(theta*180/math.pi),1)
63     dst = cv2.warpAffine(dst,M,(cols,rows))
64     return dst
65
66 # skin color detection using hsv,rgb
67 def skin_color(image):
68     # detect hand region
69     hsv_im = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
70     HSV = cv2.split(hsv_im)
71     S = HSV[1]
72     V = HSV[2]
73     RGB = cv2.split(image)
74     Blue = RGB[0]
75     Green = RGB[1]
76     Red = RGB[2]
77     mask = Red - Blue
78     mask[mask > 0] = 255
79     mask[Blue > Red] = 0
80     mask[Green > Red] = 0
81     mask[V > 0.73*255] = 0
82     mask[S < 0.3*255] = 0
83     # erase desks
84     mask[350:,:] = 0
85     mask[:,130] = 0
86     mask[:,430:] = 0
87     # erase spot noise
```

```

88     mask = cv2.medianBlur(mask,15)
89     return mask
90
91 # main function -- detecting hand region
92 def hand_detection(image):
93     # skin color detection from the original image
94     mask = skin_color(image)
95     # detect the major axis angle
96     [theta,c_x,c_y] = majoraxis.majoraxis(mask)
97     # rotate image and detect wrist
98     if not math.isnan(c_x):
99         rmask = rotateim(mask,theta,c_x,c_y)
100        handmask = detectwrist(rmask,theta,c_x,c_y)
101    else:
102        handmask = mask
103        theta = 0
104    handmask = np.bool_(handmask)
105    return handmask, theta, mask

```

majoraxis.py

```

1 import numpy as np
2 import math
3 def majoraxis(mask):
4     """
5         Find the major axis of the hand. The input image is
6         a binary image of the hand mask.
7         Return the angle and the y-intercept of the major axis.
8         Major axis is compute by minimizing the integral of the
9         distances from all hand pixels to the
10        major axis.
11
12
13 @param np.array img
14 The input image A numpy.array of size (480,640), dtype=np._bool
15 return (theta, c_x, c_y)
16 The majoraxis described with the angle theta and the center point of the object.
17 Theta is measured in radius. """
18 # non zero area
19 (x,y) = mask.nonzero()
20 if len(x) == 0:
21     return 0,0,0
22 # area
23 area = len(x)
24 # center position

```

```
25     c_x = x.mean(axis=0)
26     c_y = y.mean(axis=0)
27     # second moment
28     a_o = (np.multiply(x,x)).sum(axis=0)
29     b_o = (np.multiply(x,y)).sum(axis=0)
30     c_o = (np.multiply(y,y)).sum(axis=0)
31     # the minimum moment of inertia
32     # convert the second moment for the origin to the second moment for the center
33     a = a_o - area*(c_x**2)
34     b = 2*b_o - 2 * area * c_x * c_y
35     c = c_o - area*(c_y**2)
36     # find theta for the major axis
37     theta = math.atan2(b,a-c)/2.0
38     #print i,",",theta, ",","",c_x,"","",c_y
39     return theta, c_x, c_y
```

Appendix:Code Posture recognition

```
1 import numpy as np
2 import cPickle
3 import cv2
4 import os
5 import sys
6 includespath = os.path.abspath('../includes')
7 sys.path.insert(0, includespath)
8 import re
9 import posture
10
11 # Load the data set file.
12 #data and labels are lists of numpy.ndarray
13
14
15
16 print "Loading Data"
17 inputfolder = sys.argv[1]
18 modelfilename = sys.argv[2]
19 datalist, labellist = [], []
20
21
22 for f in os.listdir(inputfolder):
23     s = re.search(r'train_data_[0-9]_[0-9]+\.png', f)
24     label = s.group(1)
25     n = s.group(2)
26     print label, n
27     if label == posture.poses["DBFINGER"]:
28         continue
29     img = cv2.imread(inputfolder+'/'+f)
30     # cv2.imshow(f, img)
31     # cv2.waitKey(0)
32     # label = raw_input("Enter the gesture:")
33     # touch = raw_input("Does the finger touch the paper?")
34     datalist.append(img)
35     labellist.append(label)
36     # touchlist.append(touch)
37 print "Done\n"
38 data = datalist
39 labels = labellist
40
41 indices = np.random.permutation(len(datalist)).tolist()
42 traindata, trainlabels = [], []
43 testdata, testlabels = [], []
```

```

44
45 counter = 0
46 for ind in indices:
47
48     if counter < len(datalist)/10*5:
49         traindata.append(datalist[ind])
50         trainlabels.append(labellist[ind])
51         print ind, "label = ", labellist[ind], "train"
52     else:
53        testdata.append(datalist[ind])
54         testlabels.append(labellist[ind])
55         print ind, "label = ", labellist[ind], "test"
56     counter = counter+1
57
58 if os.path.isfile(modelfilename):
59     print "Loading: ", modelfilename
60     pos_recognizer = posture.PostureRecognizer.load(modelfilename)
61 else:
62     print "Creating: ", modelfilename
63     pos_recognizer = posture.PostureRecognizer()
64 #Do the training here
65 print "Start training"
66 pos_recognizer.train(traindata, trainlabels)
67 print "saving data"
68 pos_recognizer.save(modelfilename)
69
70
71 #####validation#####
72 score = 0.0
73 total = 0.0
74 for i in xrange(len(testlabels)):
75     pred = pos_recognizer.classify(testdata[i])[0]
76     total+=1
77     print i, 'pred = ', pred, 'label=' ,testlabels[i]
78     if str(pred) == testlabels[i]:
79         score +=1
80
81 print "=====\\n"
82 print "Score = ", score, "/", total, "=", score/total, "\\n"

```

```

1 from sklearn import svm, neighbors
2 import numpy as np
3 import pickle
4 import os

```

```
5 import hand_detection
6 import feature_extraction as fe
7 import cv2
8 from collections import deque
9
10 poses = {
11     'POINTING': '1',
12     'DBFINGER': '2',
13     'PALM': '3',
14     'UNKNOWN': '-1',
15 }
16 ## 1 = pointing finger
17 ## 2 = piece sign
18 ## 3 = palm
19 ## -1 = unknown
20
21
22
23 class PostureRecognizer(object):
24     """A class for hand posture recognition"""
25     def __init__(self):
26         super(PostureRecognizer, self).__init__()
27         self.feature_extractor = fe.OrientationHistogramFeature()
28         self.classifier = neighbors.KNeighborsClassifier()
29         #self.classifier = svm.SVC()
30         self.model = None
31
32
33     def save(self, filename):
34         """Save the current trained model to a pickle file.
35
36         params filename: The output filename.
37         """
38         if os.path.isfile(filename):
39             os.remove(filename)
40         outputfile = open(filename, 'w')
41         pickle.dump(self, outputfile)
42         outputfile.close()
43
44     @staticmethod
45     def load(filename):
46         """load a trained model from the filename.
47
48         params filename: The filename of the trained model
49         return model: A PostureRecognizer instance. """
50         modelfile = open(filename, 'r')
```

```
51         self = pickle.load(modelfile)
52     modelfile.close()
53     return self
54
55     def extract_features(self, image):
56         """Extract the features from the image"""
57         return self.feature_extractor.extract_features(image)
58
59
60     def train(self, train_data, train_label):
61         """train on the training data and the labels. The train_data is a matrix of n x m x
62         where each row is the features of one example. The train_label is a vector of n x 1
63
64         params train_data: the training data, a list .
65         params train_label: the training labels, a list. """
66         #Get the dimension of the feature first
67
68         img = train_data[0]
69         feature= self.extract_features(img)[0]
70         features = np.zeros((len(train_data), feature.size))
71         labels = np.zeros((len(train_label)))
72         for i in xrange(len(train_data)):
73             label = train_label[i]
74             print i, ": label = ", label
75             img = train_data[i]
76             feature= self.extract_features(img)[0]
77             features[i,:] = feature
78             labels[i] = label
79
80         print "Start fitting"
81         self.classifier.fit(features,labels)
82
83
84     def hand_detection(self, image):
85         """transfer the input image into a binary matrix which marks where the hand is. Each
86         of the output is Ture if and only if the corresponding pixel in the original image
87         hand area.
88
89         params image: A numpy.ndarray representing the input image taken with cv2.imread()
90         return mask: A numpy.ndarray with the same shape with the input image.
91         """
92         return hand_detection.hand_detection(self, image)
93
94     def classify(self, image):
95         """Classify the image into one of the defined postures.
96
```

```

97     params image: A numpy.ndarray representing the input image taken with cv2.imread()
98     params hand_mask: A numpy.ndarray with the same shape with the input image which indicates
99     return posture :A string which corresponds to one of the defined postures, or -1 if
100    feature, hand_mask, theta, skin_mask = self.extract_features(image)
101    pred = self.classifier.predict(feature)
102    pred = str(int(pred))
103    # print "pred = ", str(int(pred))
104    #TODO return -1 if dist is too large
105    if pred not in poses.values():
106        return poses["UNKNOWN"], hand_mask, theta, skin_mask
107    return pred[0], hand_mask, theta, skin_mask
108
109 def find_shadow_of(img, location, hand_mask):
110     img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
111     mask_B = (img[:, :, 0] > 0 * 180) * (img[:, :, 0] < 0.67*180)
112     mask_G = (img[:, :, 1] > 0.1 * 255) * (img[:, :, 1] < 0.34*255)
113     mask_R = (img[:, :, 2] > 0 * 255) * (img[:, :, 2] < 0.42*255)
114
115     # mask_H = (img_hsv[:, :, 0] > 0.21 * 180) * (img_hsv[:, :, 0] < gr*180)
116     # mask_S = (img_hsv[:, :, 1] > gray_val * 180) * (img_hsv[:, :, 1] < 1*180)
117     mask_V = (img_hsv[:, :, 2] > 0.0 * 180) * (img_hsv[:, :, 2] < 0.56*180)
118     mask_nothand = hand_mask == False
119
120     mask = mask_R*mask_G*mask_B*mask_V*mask_nothand
121     mask[location[0]:, :] = False
122     mask[:, location[1]+40:] = False
123     mask[:, :location[1]] = False
124
125     # for i in xrange(binimg.shape[0]):
126     #     for j in xrange(binimg.shape[1]):
127     #         if i>location[0] or j < location[1] or j > location[1] + 40:
128     #             binimg[i, j]= False
129     return mask
130
131 def shadow_fingertip(shadow_mask, wrist_end):
132     shadow_finger = None
133     shadow_indices_i, shadow_indices_j = np.where(shadow_mask)
134     if len(shadow_indices_i) ==0:
135         return shadow_finger
136     if wrist_end == 'up':
137         finger_ind = shadow_indices_i.argmax()
138     elif wrist_end == 'down':
139         finger_ind = shadow_indices_i.argmin()
140     elif wrist_end == 'left':
141         finger_ind = shadow_indices_j.argmax()
142     elif wrist_end == 'right':

```

```

143         finger_ind = shadow_indices_j.argmin()
144     else:
145         raise ValueError('Unknown wrist_end')
146     shadow_finger = (shadow_indices_i[finger_ind], shadow_indices_j[finger_ind])
147     return shadow_finger
148
149 def isTouching(frame, label, location, wrist_end, hand_mask):
150     """
151     Determin if the finger is touching the paper.
152     """
153     shadow_mask = find_shadow_of(frame, location, hand_mask)
154     shadow_ft = shadow_fingertip(shadow_mask, wrist_end)
155     # frame_tmp = np.copy(frame)
156     # frame_tmp[shadow_mask==False]=0
157     # cv2.circle(frame_tmp,shadow_ft ,3,(0,0,255),-1)
158     # cv2.imshow('shadow_fingertip', np.multiply(frame, shadow_mask))
159     # frame_tmp = np.copy(frame)
160     # frame_tmp[shadow_mask==False]=0
161     # cv2.circle(frame_tmp,(shadow_ft[1], shadow_ft[0]) ,3,(0,0,255),-1)
162     # cv2.circle(frame_tmp,(location[1], location[0]) ,3,(255,0,0),-1)
163     # # cv2.imshow('shadow', frame_tmp)
164     if shadow_ft is None:
165         return False
166
167     dist = ((location[0]-shadow_ft[0])**2 + (location[1]-shadow_ft[1])**2) **0.5
168     # print "dist = ", dist, "wrist=", wrist_end, "touching =", dist<30
169     t = False
170     if label == poses["PALM"]:
171         t = dist < 50
172     else:
173         t = dist < 30
174     return t
175
176
177
178
179
180
181

```

```

1 import numpy as np
2 import hand_detection as hd
3 import majoraxis
4 import cv2
5 from numpy import pi

```

```
6 import matplotlib.pyplot as plt
7 import matplotlib
8
9 class FeatureExtractor(object):
10     """docstring for FeatureExtractor"""
11     def __init__(self):
12         super(FeatureExtractor, self).__init__()
13
14     def extract_feature(self, image):
15         """returns the feature vector of the image as a 1-D numpy array"""
16
17 class OrientationHistogramFeature(FeatureExtractor):
18     """docstring for OrientationHistogramFeature"""
19     def __init__(self):
20         super(OrientationHistogramFeature, self).__init__()
21
22
23     def extract_features(self, image):
24         hand_mask, ang, skin_mask= hd.hand_detection(image)
25         # while cv2.waitKey(20) != ord('a'):
26         #     pass
27         # hand_mask_tmp = np.reshape(hand_mask, hand_mask.shape+(1,) )
28         # hand_mask_tmp = np.concatenate((hand_mask_tmp,hand_mask_tmp,hand_mask_tmp),axis=2)
29
30         image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
31         #masked = np.multiply(image, hand_mask_tmp)
32         masked = np.multiply(image_gray, hand_mask)
33         plt.figure()
34         plt.imshow(masked, cmap = matplotlib.cm.Greys_r)
35         # plt.figure()
36         # plt.plot(masked)
37         # cv2.imshow('masked', masked)
38         # while cv2.waitKey(20) != ord('a'):
39         #     pass
40
41
42         #masked = image[hand_mask]
43         sobelx = cv2.Sobel(masked,cv2.CV_64F,1,0,ksize=5)
44         sobely = cv2.Sobel(masked,cv2.CV_64F,0,1,ksize=5)
45
46         angles = np.angle(sobelx + 1j*sobely,deg=True)
47         # amplitudes = np.absolute(sobelx + 1j*sobely)
48         # angles = angles.flatten()
49
50         angles = angles.flatten()
51         angles = angles[np.nonzero(angles)]
```

```
52
53     # hist = cv2.calcHist(np.array([angles,amplitudes]).astype('float32'), \
54     #     channels=[0,1], \
55     #     mask=None, \
56     #     histSize=[36,5], \
57     #     # ranges=[[-180.0,180.0],[int(np.min(amplitudes)), int(np.max(amplitudes))]] \
58     #     ranges=[[-180.0,180.0],[0,256]])
59     n_bins = 36
60     # ang_hist, bins = np.histogram(angles, bins=n_bins,range=(-180.0,180.0))
61     # ang_hist[18]=0
62     # #print type(ang_hist)
63     # amp_hist = np.histogram(amplitudes, bins=5)[0]
64     # plt.figure()
65     ang_hist,bins,patches = plt.hist(angles.flatten(), bins=n_bins, range=(-180, 180), r
66     # print sum(ang_hist)

67
68
69
70     #####Angular disposition#####
71     # ang, cx, cy = majoraxis.majoraxis(hand_mask)
72     ang = ang/pi*180
73     try:
74         majorangle = np.where(np.histogram([ang], bins=n_bins,range=(-180,180))[0] != 0)
75         ang_hist = np.roll(ang_hist, (n_bins - majorangle)%n_bins )
76     except(IndexError):
77         pass
78     # plt.figure()
79     # plt.hist(range(-180,180, 10), bins=n_bins, range=(-180,180), weights=ang_hist)
80     # plt.show()
81     return ang_hist, hand_mask, ang, skin_mask
82
83
84
85
```

Appendix:Code Fingertip detection

fingertip.py

```
1 import cv2
2 import numpy as np
3 import pdb
4 import time
5 import math
6 def find_fingertip(label, mask):
7     """
8         Find the location of the fingertips. Only have to return one pixel.
9         @param np.array          hand_mask
10            a numpy array of size(480,640,1)
11        @label string            label
12            A string represting the label of the frame.
13
14
15        Look at posture.py for more informations.
16        return (int, int)        loc
17            The location of the fingertip of the longest finger.
18        return string           wrist_end
19            "up/down/left/right" representing the side of the wrist.
20        """
21        mask[mask != 0] = 255
22        (x,y) = mask.nonzero()
23        # find the minimum / maximum x / y
24        if len(x) == 0:
25            return None,None
26        x_min = min(x)
27        y_min = min(y)
28        x_max = max(x)
29        y_max = max(y)
30        # find the center
31        c_x = x.mean(axis=0)
32        c_y = y.mean(axis=0)
33        # find the most largest edge (= wrist)
34        up = (sum(mask[x_min,:])/255)
35        bottom = (sum(mask[x_max,:])/255)
36        left = (sum(mask[:,y_min])/255)
37        right = (sum(mask[:,y_max])/255)
38        wrist = max(up,bottom,left,right)
39        wrist_ch = ''
40        # mask wrist side
41        if wrist == up:
```

```
42     wrist_ch = 'up'
43 elif wrist == bottom:
44     wrist_ch = 'down'
45 elif wrist == left:
46     wrist_ch = 'left'
47 elif wrist == right:
48     wrist_ch = 'right'
49 # find the most furthest point (= finger)
50 mask[:c_x,:] = 0
51 (x,y) = mask.nonzero()
52 z = (x - c_x)**2 + (y - c_y)**2
53 if len(z) > 0:
54     return (x[z.argmax(0)],y[z.argmax(0)]),wrist_ch
55 else:
56     return None, None
```

Appendix:Code to determine if the hand is touching the paper

Appendix:Code Evaluation

evaluation.py

```
 1 import sys
 2 import os
 3 includespath = os.path.abspath('../includes')
 4 sys.path.insert(0,includespath)
 5 import evala as e
 6 if __name__ == '__main__':
 7     username = sys.argv[1]
 8     e.exp_points(username)
 9     e.exp_line(username)
10     e.exp_circle(username)
```

evaluation_m.py

```
 1 import sys
 2 import os
 3 includespath = os.path.abspath('../includes')
 4 sys.path.insert(0,includespath)
 5 import evala as e
 6 if __name__ == '__main__':
 7     username = sys.argv[1]
 8     e.exp_points_m(username)
 9     e.exp_line_m(username)
10     e.exp_circle_m(username)
```

evala.py

```
 1 import cv2
 2 import numpy as np
 3 import pdb
 4 import time
 5 import math
 6 import os
 7 import csv
 8 import eval_ui as eu
 9 import mousepaint as mo
10 def drawing(im):
11     rgb = cv2.split(im)
12     # use only B (because we use Red color to write)
13     im2 = rgb[0]
```

```
14     im2[rgb[0] == 0] = 50
15     im2[rgb[0] == 255] = 0
16     return im2
17
18 def evalcircle(im,center,r):
19     im2 = drawing(im)
20     (x,y) = im2.nonzero()
21     sump = 0.0
22     for i in range(len(x)):
23         sump += math.fabs((x[i]-center[0])**2 + (y[i]-center[1])**2)**(0.5) - r
24     sump /= len(x)
25     return sump
26
27 def evalline(im,start,end):
28     # this evaluation cannot be used for lines parallel to x axis
29     im2 = drawing(im)
30     (x,y) = im2.nonzero()
31     x = np.sort(x)
32     min_y = min(y)
33     max_y = max(y)
34
35     print min_y,max_y
36     sump = 0
37     for i in range(20):
38         point = [0,0]
39         epoint = [0,0]
40         point[0] = start[0] + i * (end[0] - start[0]) / 20
41         point[1] = start[1] + i * (end[1] - start[1]) / 20
42         epoint[0] = x[int(len(x) * i / 20)]
43         ys = np.where(im2[epoint[0],:])[0]
44         if (len(ys) == 0):
45             raise IndexError("Out of bound")
46         else:
47             epoint[1] = ys[0]
48         print point, epoint
49         sump += evalpoint(epoint,point)
50     sump /= 20
51     return sump
52
53 def evalpoint(a,b):
54     return ((a[0] - b[0])**2 + (a[1] - b[1])**2)**0.5
55
56 def evalpoints(im,points):
57     im2 = drawing(im)
58     rows,cols = im2.shape
59     (x,y) = im2.nonzero()
```

```
60     sump = 0
61     print points
62     for i in range(len(x)):
63         print x[i],y[i]
64         if x[i] < rows/2 and y[i] < cols/2:
65             sump += evalpoint([x[i],y[i]],points[0])
66         elif x[i] < rows/2 and y[i] >= cols/2:
67             sump += evalpoint([x[i],y[i]],points[1])
68         elif y[i] < cols/2:
69             sump += evalpoint([x[i],y[i]],points[2])
70         else:
71             sump += evalpoint([x[i],y[i]],points[3])
72     return sump/len(x)
73
74 def exp_circle(username):
75     if(not os.path.exists('../experiments')):
76         os.mkdir('../experiments')
77     f = open('../experiments/circle.csv','ab')
78     csvWriter = csv.writer(f)
79     # do experiment
80     center = [250,290]
81     r = 190
82     for i in range(3):
83         ## for debuging
84         im2 = cv2.imread('../test.png')
85         cv2.circle(im2,(290,250),190,(255,0,0),1)
86         filename = '../experiments/circle_'+username+'_'+str(i) + '.png'
87         im,time = eu.evaluation(filename,im2)
88         c = evalcircle(im,center,r)
89         csvWriter.writerow([username,i,time,c])
90         print 'wirte'
91
92 def exp_line(username):
93     if(not os.path.exists('../experiments')):
94         os.mkdir('../experiments')
95     f = open('../experiments/line.csv','ab')
96     csvWriter = csv.writer(f)
97     # do experiment
98     start = (50,50)
99     end = (400,400)
100    for i in range(3):
101        ## for debuging
102        im2 = cv2.imread('../test.png')
103        cv2.line(im2,(50,50),(400,400),(255,0,0),1)
104        filename = '../experiments/line_'+username+'_'+str(i) + '.png'
105        im,time = eu.evaluation(filename,im2)
```

```
106         ## for debugging
107         c = evalline(im,start,end)
108         csvWriter.writerow([username,i,time,c])
109
110     def exp_points(username):
111         if(not os.path.exists('../experiments')):
112             os.mkdir('../experiments')
113         f = open('../experiments/points.csv','ab')
114         csvWriter = csv.writer(f)
115         # do experiment
116         points = [(40,40),(40,350),(350,40),(350,350)]
117         for i in range(3):
118             im2 = cv2.imread('../test.png')
119             cv2.circle(im2,(40,40),3,(255,0,0),1)
120             cv2.circle(im2,(350,350),3,(255,0,0),1)
121             cv2.circle(im2,(40,350),3,(255,0,0),1)
122             cv2.circle(im2,(350,40),3,(255,0,0),1)
123             filename = '../experiments/points_'+username+'_'+str(i) + '.png'
124             im,time = eu.evaluation(filename,im2)
125             c = evalpoints(im,points)
126             csvWriter.writerow([username,i,time,c])
127
128     def exp_circle_m(username):
129         if(not os.path.exists('../experiments')):
130             os.mkdir('../experiments')
131         f = open('../experiments/circle.csv','ab')
132         csvWriter = csv.writer(f)
133         # do experiment
134         center = [250,290]
135         r = 190
136         for i in range(3):
137             ## for debugging
138             im2 = cv2.imread('../test.png')
139             cv2.circle(im2,(290,250),190,(255,0,0),1)
140             filename = '../experiments/circle_'+username+'_'+str(i) + '_m.png'
141             im,time = mo.evaluation(filename,im2)
142             c = evalcircle(im,center,r)
143             csvWriter.writerow([username,i,time,c,'mouse'])
144             print 'wirte'
145
146     def exp_line_m(username):
147         if(not os.path.exists('../experiments')):
148             os.mkdir('../experiments')
149         f = open('../experiments/line.csv','ab')
150         csvWriter = csv.writer(f)
151         # do experiment
```

```

152     start = (50,50)
153     end = (400,400)
154     for i in range(3):
155         ## for debugging
156         im2 = cv2.imread('../test.png')
157         cv2.line(im2,(50,50),(400,400),(255,0,0),1)
158         filename = '../experiments/line_'+username+'_'+str(i) + '_m.png'
159         im,time = mo.evaluation(filename,im2)
160         ## for debugging
161         c = evalalline(im,start,end)
162         csvWriter.writerow([username,i,time,c,'mouse'])
163
164 def exp_points_m(username):
165     if(not os.path.exists('../experiments')):
166         os.mkdir('../experiments')
167     f = open('../experiments/points.csv', 'ab')
168     csvWriter = csv.writer(f)
169     # do experiment
170     points = [(40,40),(40,350),(350,40),(350,350)]
171     for i in range(3):
172         im2 = cv2.imread('../test.png')
173         cv2.circle(im2,(40,40),3,(255,0,0),1)
174         cv2.circle(im2,(350,350),3,(255,0,0),1)
175         cv2.circle(im2,(40,350),3,(255,0,0),1)
176         cv2.circle(im2,(350,40),3,(255,0,0),1)
177         filename = '../experiments/points_'+username+'_'+str(i) + '_m.png'
178         im,time = mo.evaluation(filename,im2)
179         c = evalpoints(im,points)
180         csvWriter.writerow([username,i,time,c,'mouse'])
181
182 if __name__ == '__main__':
183     exp_points('aya')

```

eval_ui.py

```

1 import sys
2 import os.path
3 includespath = os.path.abspath('../includes')
4 sys.path.insert(0, includespath)
5 includespath = os.path.abspath('../bin')
6 sys.path.insert(0,includespath)
7 import posture
8 import cv2
9 import gui
10 import hand_detection

```

```
11 import fingertip
12 import time
13 import numpy as np
14 import main
15 global ui
16 ui = gui.GUI()
17
18 def cutframe(frame):
19     return frame[:380]
20
21 modelfilename = '../models/modelKNN2.mdl'
22
23 #Get the image and do the classification here
24 cap = cv2.VideoCapture(1)
25 def evaluation(filename = 'test.png', image =None):
26     start = 0
27     end = 0
28     ui.canvas = np.ones((480,640,3L),)*255
29     ui.color = (0,0,255)
30     pos_recognizer = posture.PostureRecognizer.load(modelfilename)
31
32
33 while(True):
34     label, location, touching = main.getinput(cap, pos_recognizer)
35     if(not location):
36         if cv2.waitKey(20) == 27:
37             break
38         continue
39     #####The grammar goes here=====
40     ui.handle_input(label, location, touching)
41
42     cv2.imshow('Canvas', ui.get_screen())
43     pressedKey = cv2.waitKey(60)
44     if pressedKey == ord('q'):
45         end = time.time()
46         ui.save_canvas(filename)
47         cv2.destroyAllWindows()
48         return ui.canvas, end - start
49     if pressedKey == ord('r'):
50         # add something to add image
51         ui.draw_sample(image)
52         start = time.time()
53     if pressedKey & 0xFF == 27:
54         break
55     cv2.destroyAllWindows()
```

mousepaint.py

```
1 import os
2 import sys
3 import gui
4 import cv2
5 import numpy as np
6 import time
7 global ui
8 ui = gui.GUI()
9 global palette
10 palette = {"RED":(0,0,255)}
11 ui.color = palette["RED"]
12 global mode
13 global isTouching
14 isTouching = False
15 mode = '1'
16
17 def mouse_cb(event,x,y,flags,param):
18     global ui
19     global mode
20     global isTouching
21     if(event == cv2.EVENT_LBUTTONDOWN):
22         isTouching = True
23     if(event == cv2.EVENT_LBUTTONUP):
24         isTouching = False
25     ui.handle_input_m(mode, (y,x), isTouching)
26
27 def evaluation(filename = 'test.png',image = None):
28     cv2.namedWindow('gui')
29     cv2.setMouseCallback('gui', mouse_cb)
30     start = 0
31     end = 0
32     ui.canvas = np.ones((480,640,3L),)*255
33     while(True):
34         cv2.imshow('gui',ui.get_screen())
35         pressedKey = cv2.waitKey(60)
36
37         if pressedKey == ord('r'):
38             ui.draw_sample(image)
39             start = time.time()
40             print 'r'
41         if pressedKey == ord('q'):
42             end = time.time()
43             ui.save_canvas(filename)
```

```
44     print 'q'
45     cv2.destroyAllWindows()
46     return ui.canvas, end - start
47 if pressedKey & 0xFF == 27:
48     break
49     cv2.destroyAllWindows()
```
