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

Angus Ding ad3180
Ayaka Kume ak3682

May 14, 2015

Contents

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-vertical 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. The user can simply touch the color blocks to choose the color he or she wants to use. In the mean 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 – a pointing finger. Touching the paper with the index finger means to draw, while moving the finger without touching the paper means to move the cursor without drawing. This gesture, when the user touches the palette instead of the empty area, means to select the color 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 erasing, because it is the movement one would 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 downward so that we can easily convert the coordinate from the input to the output. The canvas, which is the area on

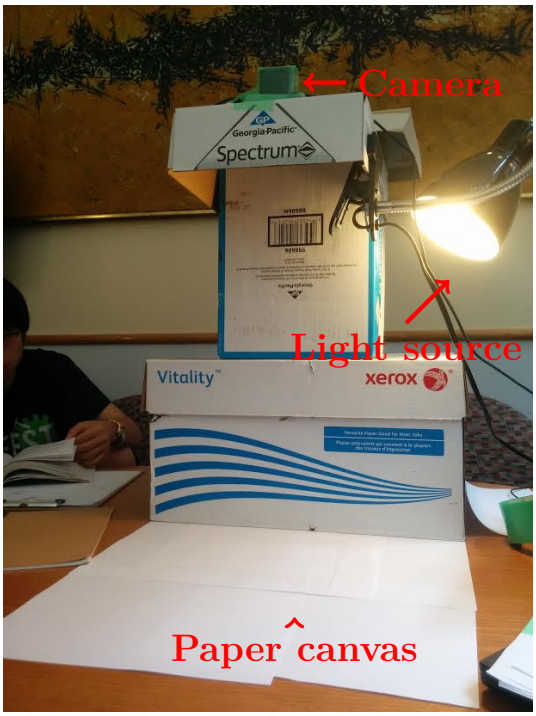


Figure 1: The input device

which the user can draw, is 19cm by 14cm. We set the background to white in order to detect the hand easily. The webcam model we use is logicool carl zeiss tessar. The system is implemented in python 2.7, and tested on a Windows 8 PC. As the light source, we use handy light.

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

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 used 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 white by default.

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 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 it.

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 wouldn'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 informations of each frame.

3 Hand detection

Because of our settings, there are only white background, shadow and the hand in a frame. 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

Since there are only white background, shadow and the hand in the screen, we can detect the hand with color. We use both RGB and HSV value to detect the skin. 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 ignore the pixels outside of the canvas.

Figure 2: Overview of hand detection

3.2 Orientation of the hand

In order to detect the hand region, it is useful to find the orientation of the hand. In our program, major axis.py implements this function. We define the orientation of the hand as the angle of the axis with the least second moment.

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

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

where $r(x,y)$ is the distance from the pixel (x,y) to the axis and b(x,y) = 1 if and only if the pixel belongs to the hand. Let the axis be $x\sin\theta - y\sin\theta + \rho = 0$. The distance of point (x,y) from axis is:

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

. Thus minimizing E means minimizing

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

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

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

where A is an area of the hand and (x_c, y_c) is center of the hand. 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 $x'\sin\theta - y'\cos\theta$ So,

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

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

Setting $\partial E/\partial\theta = 0$, 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 $\theta = atan2(b, a-c)/2$.

Fig.3 shows the results of orientation detection and the rotated image. The first column is the original image. The second column is the translated image. First, the center of the hand is moved to the center 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 stably. If the image of hand does not have enough amount of areas, for example, it can detect only a part of fingers, and not the the angle of the whole hand. However, because of our settings, we can always see enough amount of hand pixels in the target area.

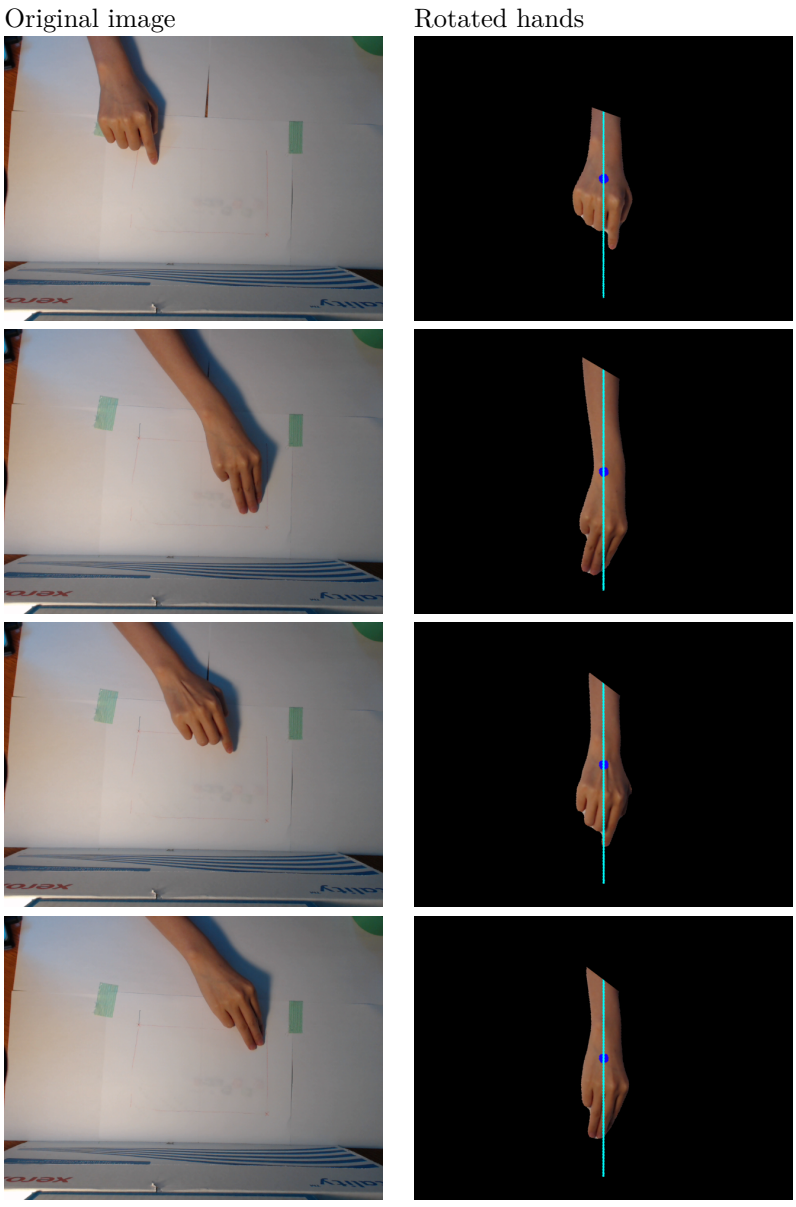


Figure 3: The results for orientation detection

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

3.3 Hand detection

To classify the posture of the hand, the arm is not necessary. Thus, we would like to extract the hand region. We modified the method introduced by [4]. The method is first to detect the skin region by color (and 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 the four borders of the bounding box of the skin region: 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).

Then we can assume the largest among these is the wrist side. This is because the hand is inside of the image, but human body itself is not.

Then, Raheja et al. detect the wrist end using intensity histogram. Intensity histogram is the sum of the number of the pixels on the row/cols. If wrist end is up/ down, it calculate along rows. Otherwise, it calculate along columns.

Assume the wrist end is at the bottom. Let b be the skin pixel that is either on the left border or the right border of the minimum bounding box, and is nearest to the wrist end. Since b is either the leftmost or the rightmost pixel, the intensity histogram at b's row tend to be close to the maximum. On the other hand, the wrist is usually thinner than the arm and the hand, so the intensity histogram at the wrist tends to be close to the minimum. Let y_b be the y coordinate of b(or the x coordinate if the wrist end is on the left or right), and $hist(x)$ be the intensity histogram at row x (or the column x if the the wrist end is on the left or right). [4] found that, $hist(x_b) - hist(x)/(x_b - x)$ tends to be the maxium when x is at the wrist.

However, the wrist detection does not work well in the general case because the paper assumes that the hand gesture is a spray hand and so the palm is always the widest. Like Fig.4 shows, we cannot find the appropriate b because the leftmost and the rightmost points are not in the palm, but in the arm and fingertip. This is because the arm is not pointing downward and this makes it difficult to detect the hand region as it is. So, we rotate the image along the major axis so that we can assume the wrist is always on the 'up' side and

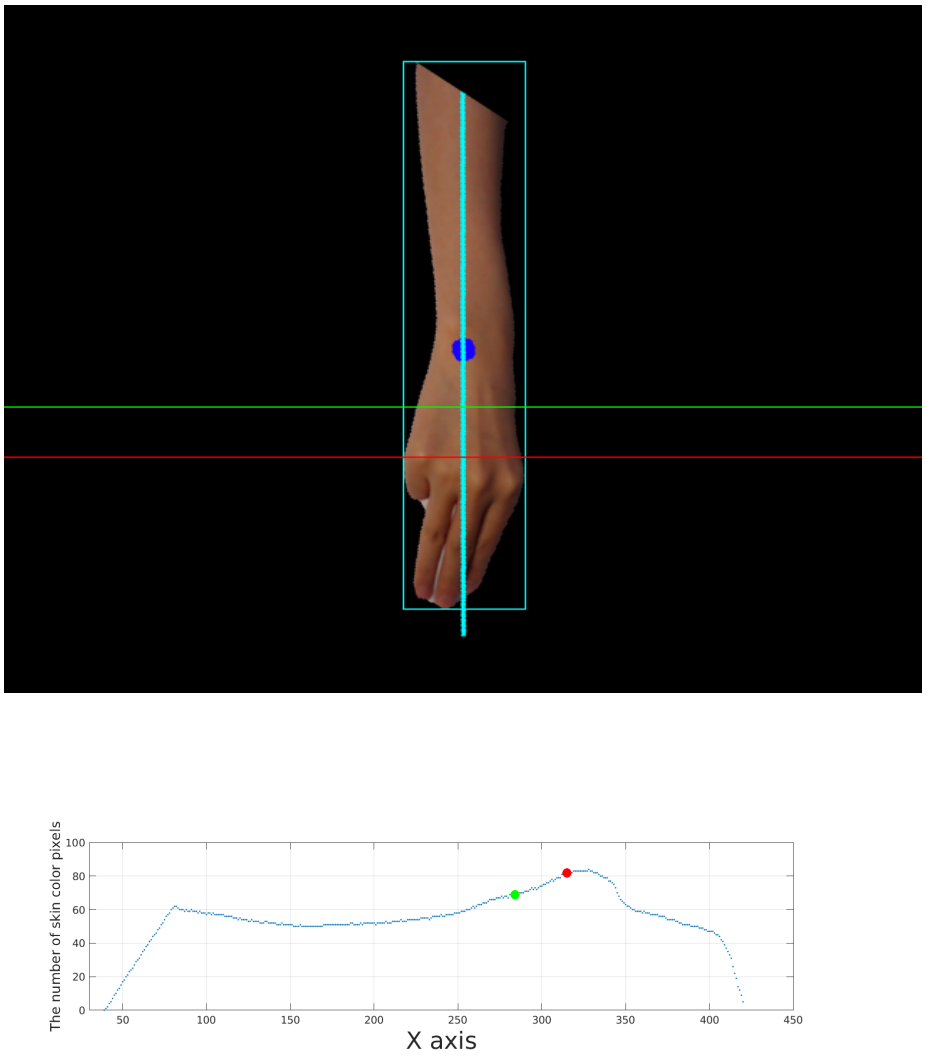


Figure 5: Hand image Top: Hand image. Green line is wrist position and red line is b. Bottom: histogram

the most left or right part tends to be in the palm region. In Fig.4 , the wrist can be detect by our method but not the previous one, because our method can detect the widest point as b correctly. We assume b is not too near to the wrist. That is, if a point is within 30 pixels from the wrist end, we use another point. Fig.5 shows an example of the intensity histogram. The red point in the histogram corresponds to b's coordinate. The green point in the histogram corresponds to the wrist's coordinate. The red line and the green line in the left image corresponds to b and the wrist detected.

Put together, our method is like 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 improved the method, in the case where the skin detection fails and the hand become smaller, the palm can be too small to be the widest part. In that case, we simply extract 1/4 of 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.

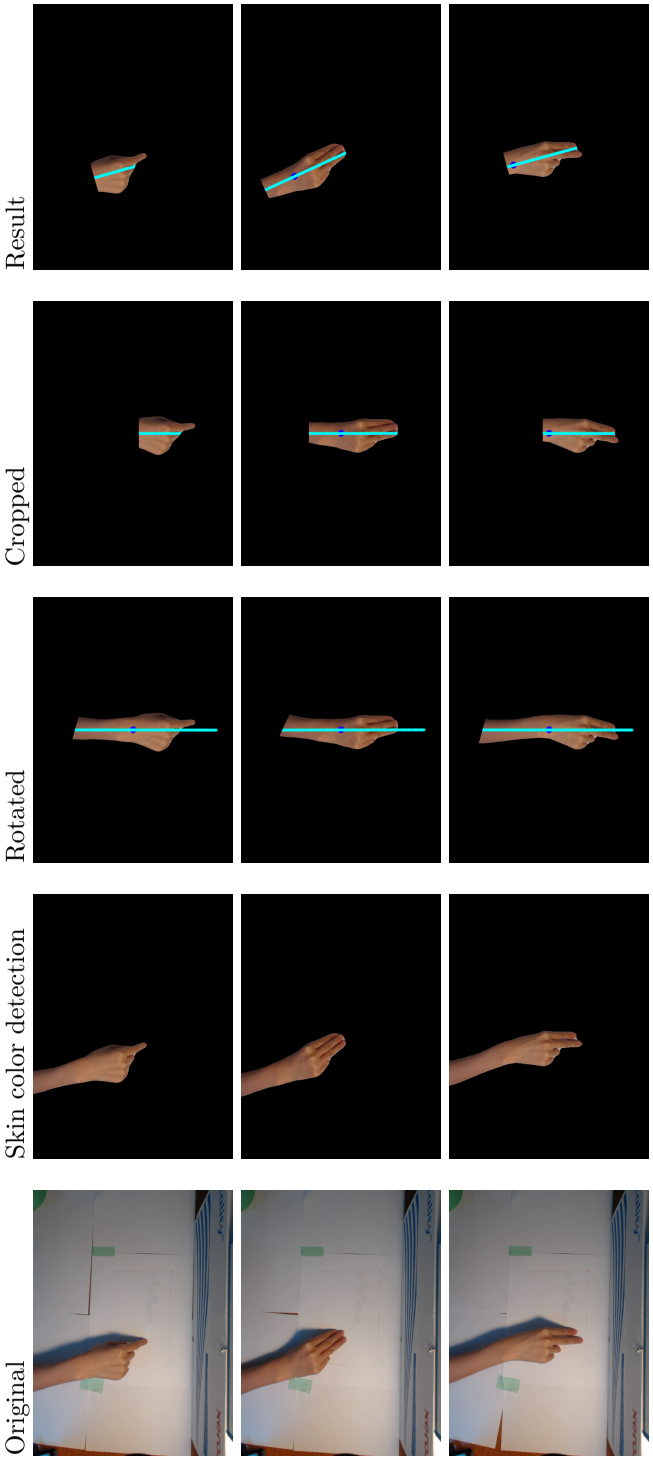


Figure 6: The results of hand detection

4 Posture recognition

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 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 difinition 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 informations in the image are 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

$$arctan(dx, dy) \tag{1}$$

âĂć 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 Sobel 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.

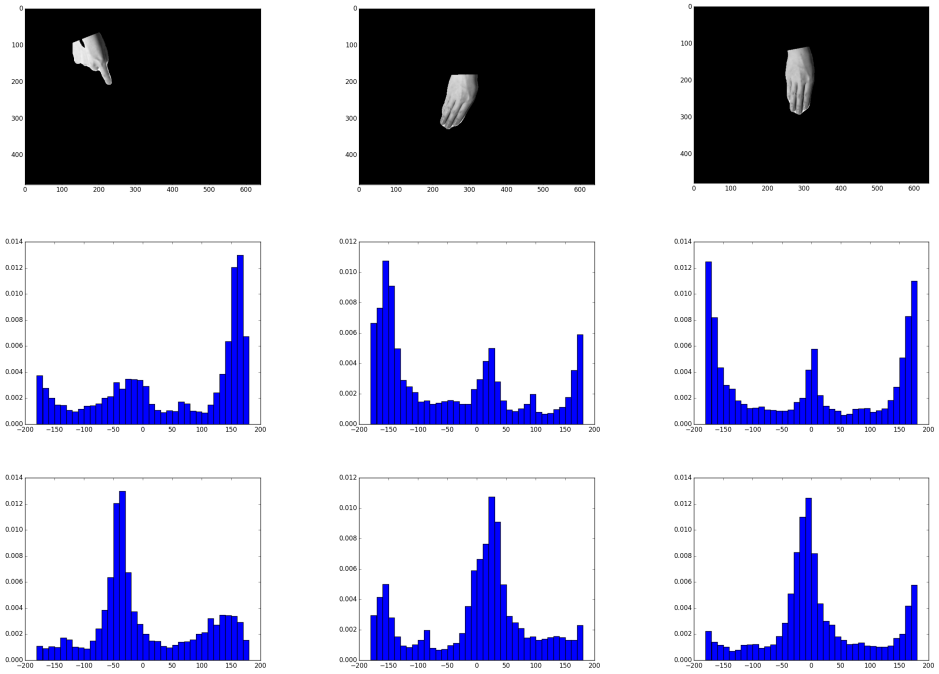


Figure 7: The orientation histograms. The first row is the input image, converted to gray scale and applied hand detection. The second row is the orientation histogram. The third row is the orientation histogram after the translation of the major axis.

Since the same hand posture can have different orientation, we can furthur 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 orientation histograms 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. âĂć

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 posture is the ' pointing finger', we detect the fingertip of the index finger. If the posture is the 'palm', we detect the fingertip of the middle finger.

We assume that the fingertip is the furthest point from the center of the mass that is on the opposite side of the wrist . Our environment allows us to assume the wrist side is always top of the image, so we can 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 fingertip detection. This method detects the finger tip correctly when the posture is one finger or two fingers. When the posture is a palm, the method sometimes detects the location between the middle finger and the other fingers. Table 1 shows the result of the detected fingertip when the posture is 'palm'. We tried 143 frames of this posture. The error rate 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 an error of about 10 pixels in the input image and 20 pixels on the output canvas, which is about 3% of the average length of the canvas.

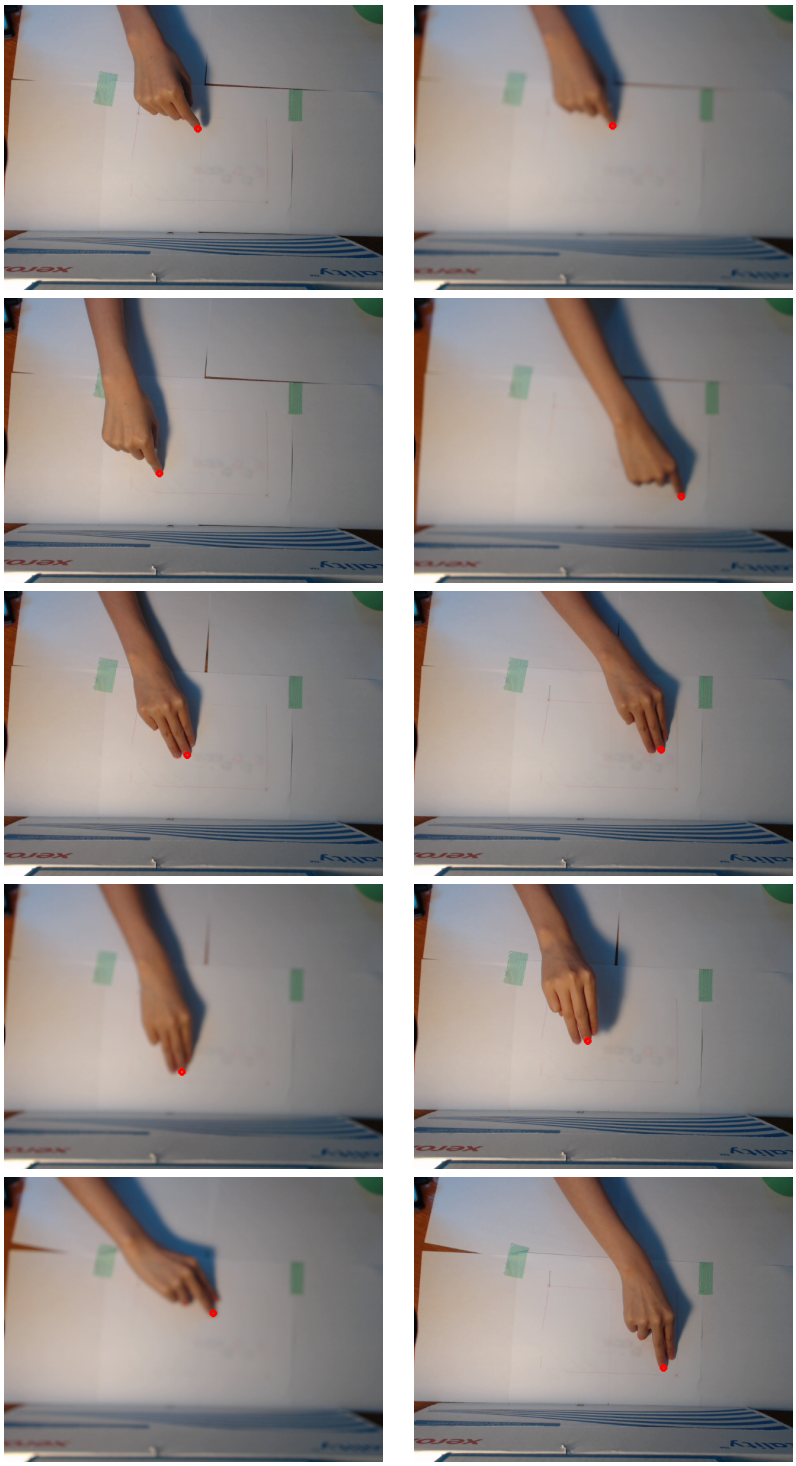


Figure 8: The results of finger detection

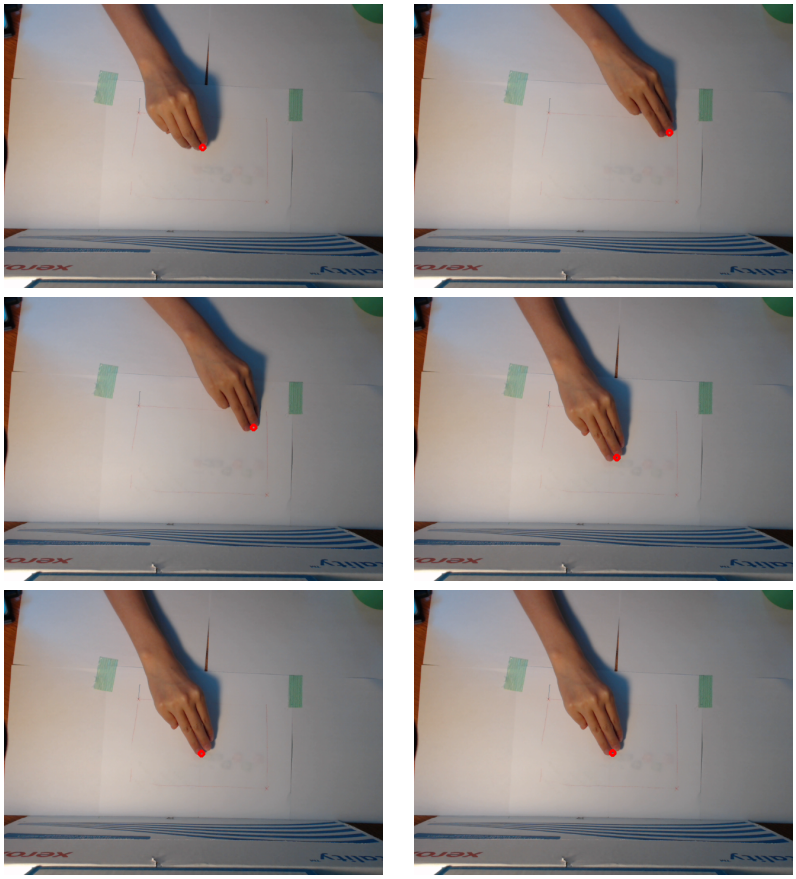


Figure 9: The errors

6 Determine if the finger is touching the paper

It is difficult to determine whether the finger is touching 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 an 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 fingertip 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 actual conditions we use are as follows:

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

, 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 on 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 three users. Two of them are 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 mouspaint.py make the gui environment. evaluation.py and evaluation_m.py is the actual execution files.

7.1 The error measurement

7.1.1 Points

We measure the distances between what the user draws and the nearest point of the sample. To make the measurement easy, we divide the canvas into four parts. If a point drawn by the user is on the upperleft part, we compare the distance with the upperleft point of the sample. Same with the other three parts. The error is the average distance between the points drawn by the user and their corresponding example point.

7.1.2 Circle

A circle is the collection of the points whose distances from the center point equals to the radius. We know the center point and the radius of the sample circle, so we can measure the absolute difference between the radius and the actual distance from the center point to each points drawn by the user. We then take the average of them as the error.

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.

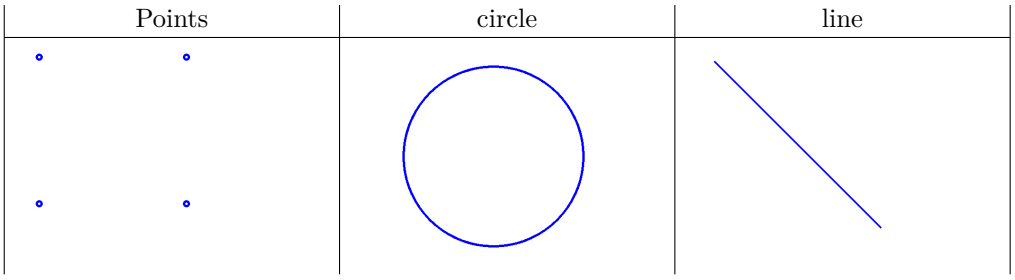


Figure 10: Three tasks

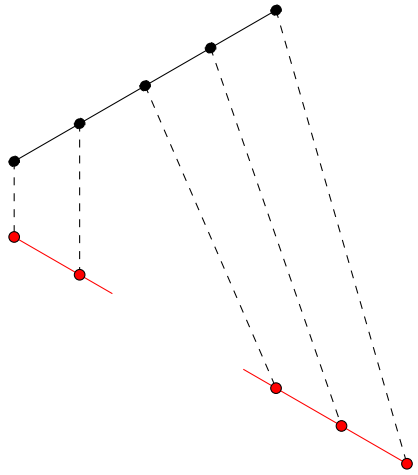


Figure 11: Line error measurement

7.2 Results

Table 3 and 2 shows the measurement results and the canvas images of the points task. Table 4 and Table 5 show the results of the circle task and the actual canvas image. Table 6 and Table 7 show the results of the line task and the actual canvas image. Table 8 shows the standard deviation of them. VI2 means the second trial for our input device. In all tasks, mouse device results are better than our input device. However, in circle and line task, the practice make better results.

Among three tasks, for our device performance is better in circle task. Points task is difficult for our device. One reason is our device is not suitable for drawing a very short line. Other reason is our device tends fail to detect touching correctly on the left side of the canvas.

We can see the time deviation is better than the error deviation. This is because altough the task does not complete perfectly, they gave up and finish the task. For exapmle, Table 4's user B's Trial 2 of VI.

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

	Trial 1	Trial 2	Trial 3
User A (VI)			
User B (VI)			
User C (VI)			
User A (VI2)			
User B (VI2)			
User C (VI2)			
User A (Mouse)			
User B (Mouse)			
User C (Mouse)			

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

User	Device	Trial	Time[sec]	Error [pixels]
A	VI	1	13.32	68.558
A	VI	2	14.522	88.314
A	VI	3	15.282	68.406
B	VI	1	17.87	72.838
B	VI	2	12.132	68.172
B	VI	3	6.599	71.913
C	VI	1	13.127	74.432
C	VI	2	14.004	51.564
C	VI	3	13.183	67.076
A	VI2	1	11.856	63.8
A	VI2	2	12.086	58.081
A	VI2	3	18.693	89.614
B	VI2	1	10.93	88.821
B	VI2	2	13.809	104.269
B	VI2	3	8.009	67.539
C	VI2	1	18.943	31.226
C	VI2	2	17.128	66.85
C	VI2	3	16.892	87.412
A	Mouse	1	6.716	4.125
A	Mouse	2	7.444	10.296
A	Mouse	3	7	4.634
B	Mouse	1	8.113	4.746
B	Mouse	2	6.293	8.115
B	Mouse	3	6.428	7.388
C	Mouse	1	5.4	4.989
C	Mouse	2	5.481	4.768
C	Mouse	3	5.199	6.603

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

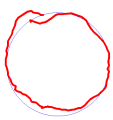
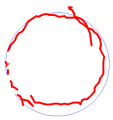
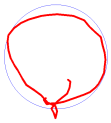
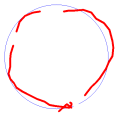
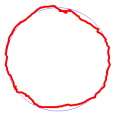
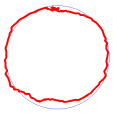
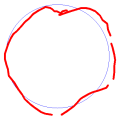
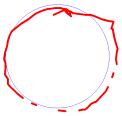
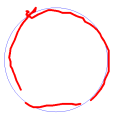
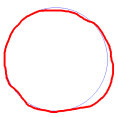
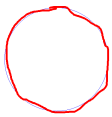
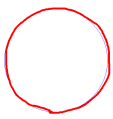
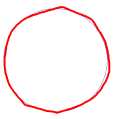
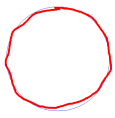
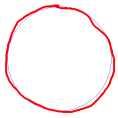
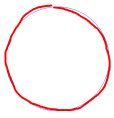
	Trial 1	Trial 2	Trial 3
User A (VI)			
User B (VI)			
User C (VI)			
User A (VI2)			
User B (VI2)			
User C (VI2)			
User A (Mouse)			
User B (Mouse)			
User C (Mouse)			

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

User	Device	Trial	Time[sec]	Error [pixels]
A	VI	1	14.507	12.745
A	VI	2	17.756	18.283
A	VI	3	20.314	17.653
B	VI	1	8.131	24.242
B	VI	2	7.84	14.619
B	VI	3	7.962	13.559
C	VI	1	8.433	46.652
C	VI	2	10.818	17.856
C	VI	3	11.376	10.398
A	VI2	1	20.229	6.966
A	VI2	2	17.148	9.991
A	VI2	3	17.836	7.156
B	VI2	1	10.819	12.892
B	VI2	2	11.19	13.977
B	VI2	3	12.79	17.879
C	VI2	1	10.963	12.154
C	VI2	2	11.597	7.066
C	VI2	3	12.712	8.156
A	Mouse	1	8.187	8.815
A	Mouse	2	10.502	4.736
A	Mouse	3	9.929	5.995
B	Mouse	1	8.801	4.548
B	Mouse	2	7.999	4.067
B	Mouse	3	8.039	4.741
C	Mouse	1	7.474	6.215
C	Mouse	2	7.187	6.929
C	Mouse	3	7.519	5.348

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

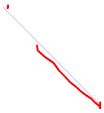
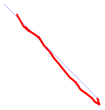
	Trial 1	Trial 2	Trial 3
User A (VI)			
User B (VI)			
User C (VI)			
User A (VI2)			
User B (VI2)			
User C (VI2)			
User A (Mouse)			
User B (Mouse)			
User C (Mouse)			

Table 7: The result of the user study (line)

User	Device	Trial	Time[sec]	Error [pixels]
A	VI	1	13.77	27.164
A	VI	2	7.046	19.362
A	VI	3	5.343	15.372
B	VI	1	4.458	37.579
B	VI	2	3.722	95.408
B	VI	3	4.02	45.112
C	VI	1	5.747	22.694
C	VI	2	6.295	10.122
C	VI	3	4.84	27.196
A	VI2	1	7.57	24.115
A	VI2	2	7.232	21.615
A	VI2	3	6.307	15.858
B	VI2	1	6.521	73.189
B	VI2	2	5.203	16.805
B	VI2	3	5.734	15.275
C	VI2	1	5.735	17.494
C	VI2	2	6.322	13.375
C	VI2	3	7.169	23.3
A	Mouse	1	3.511	15.208
A	Mouse	2	3.817	7.954
A	Mouse	3	3.525	10.621
B	Mouse	1	4.219	10.441
B	Mouse	2	4.342	10.821
B	Mouse	3	4.703	7.558
C	Mouse	1	4.059	7.018
C	Mouse	2	4.304	2.958
C	Mouse	3	4.134	4.7

Table 8: Standard Deviation of User study

Task	User	Device	StdDev of Time	StdDev of Error
points	A	VI	0.808	9.349
points	B	VI	4.602	2.018
points	C	VI	0.401	9.532
points	A	VI2	3.17	13.717
points	B	VI2	2.368	15.058
points	C	VI2	0.916	23.211
points	A	mouse	0.3	2.797
points	B	mouse	0.828	1.447
points	C	mouse	0.119	0.818
circle	A	VI	2.376	2.476
circle	B	VI	0.119	4.806
circle	C	VI	1.276	15.632
circle	A	VI2	1.32	1.383
circle	B	VI2	0.855	2.141
circle	C	VI2	0.723	2.188
circle	A	mouse	0.984	1.705
circle	B	mouse	0.369	0.284
circle	C	mouse	0.147	0.646
line	A	VI	3.638	4.897
line	B	VI	0.302	25.67
line	C	VI	0.6	7.225
line	A	VI2	0.534	3.457
line	B	VI2	0.541	26.948
line	C	VI2	0.589	4.071
line	A	mouse	0.141	2.996
line	B	mouse	0.205	1.457
line	C	mouse	0.102	1.663

8 Discussion

Apparently the performance of the system is worse than the mouse input, as expected. However, part of the reason of this is because the users are all very familiar with mouse, but only uses this system for the first time. Note that for the circle experiment, the average error for user B is not too much worse than that of the mouse, probably because user B has practiced with the system for a longer time. This shows that, with enough practice, the precision of the system can achieve the same level of the mouse.

From the experience of the user, the system has a hard time correctly classifying the postures. The user needs to find a posture that the system recognizes the best, and try to maintain that posture during the task. This interferes with the speed the user can draw, and also makes the user fatigue. This problem can possibly be fixed with more complete training dataset.

The processing time for each frame is too long for a practical real time application. This could be because the method we use for classification is k nearst neighbors and we have too many training data, resulting in the system needing too much time to search for the nearest k neighbors. A possible solution is to use SVM which is not effected by the number of the training data and is effective once the training is done. Another possible improvement is to utilize the information from the previous frame. For example, one can search for the fingertip only around the fingertip in the previous frame, instead of the whole image.

The environmental conditions are also crutial. The light from the environment can affect the intensity of the shadow, thus the system can not properly determine if the user is touching the paper. One way to solve this problem is to put the system in a box where the interference from the outside is the minimum.

We conclude that, it is possible to implement a system for the task of drawing solely relying on the visual input signal, although much more work has to be done.

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

```
import sys
import os.path
includespath = os.path.abspath('../includes')
sys.path.insert(0, includespath)
import posture
import cv2
import gui
import hand_detection
import fingertip
import hand_detection as hd
import numpy as np

def mouse_callback(event,x,y,flags,param):
    i,j = (y-1,x-1)
    if event == cv2.EVENT_LBUTTONUP:
        print i,j

def getinput(cap, pos_recognizer):
    ret, frame = cap.read()
    # Display the input stream only for debug purposes
    cv2.imshow('input',frame)
    # print "check point 1"
    label, hand_mask, theta, skin_mask = pos_recognizer.classify(frame)
    # print "label = ", label


    #cv2.imshow('debug', frame[])
    # frame_tmp = np.copy(frame)
    # frame_tmp[hand_mask==False] = 0
    # cv2.namedWindow('debug')
    # cv2.setMouseCallback('debug',mouse_callback)
    # cv2.imshow('debug', frame_tmp)

    if(label == posture.poses["UNKNOWN"]):
        print "posture = UNKNOWN"
    # print "check point 2"
    location, wrist_end = fingertip.find_fingertip(label, skin_mask)
    wrist_end = 'up'
    if(not location):
        return label, location, False
    # print "location= ", location
```

```
    # print "wrist_end = ", wrist_end
    # print "check point 3"
    touching = posture.isTouching(frame, label, location, wrist_end, hand_mask)
    return label, location, touching

def main():
    modelfilename = sys.argv[1]
    pos_recognizer = posture.PostureRecognizer.load(modelfilename)
    ui = gui.GUI()
    #Get the image and do the classification here
    cap = cv2.VideoCapture(1)

    while(True):
        # Capture frame-by-frame
        label, location, touching = getinput(cap, pos_recognizer)
        if(not location):
            if cv2.waitKey(20) == 27:
                break
            continue
        # print "touching=" ,touching
        #=======The grammar goes here==============
        print "label = ", label, "location", location, "touching", touching
        ui.handle_input(label, location, touching)
        cv2.imshow('Canvas', ui.get_screen())


        pressedKey = cv2.waitKey(60)
        if pressedKey == 27:
            break

if __name__ == '__main__':
    main()
```

gui.py

```
import cv2
import numpy as np
import posture

palette_ub = (163, 181)

palette = {
    (215, 237): "RED",
    (249, 271): "BLACK",
    (282, 304): "GREEN",
```

```
    (315, 338): "BLUE",
}
#BGR
color_map = {
    "WHITE":(255,255,255),
    "BLACK":(0,0,0),
    "RED":(0,0,255),
    "BLUE":(255,0,0),
    "GREEN":(0,255,0),
}

toolmap = {
    1:'pen',
    2:'drag',
    3:'eraser',
}


class GUI(object):
    """docstring for GUI"""
    def __init__(self):
        self.size = (480,640)
        self.canvas = np.ones((480,640, 3L),)*255
        self.cursor = Cursor(self.size[0]/2, self.size[1]/2)
        self.color = color_map["BLACK"]
        self.bgcolor = color_map["WHITE"]

        self.drawing = False
        self.erasing = False

        self.screen = np.copy(self.canvas)
        self.update_screen()


    def conv_coord(self, input_coord):
        """Convert the coordinate from the input to the output"""
        i,j = input_coord
        center1 = (261.5,322)
        center2 = (self.size[0]/2.0, self.size[1]/2.0)
        di1 = i-center1[0]
        dj1 = j-center1[1]

        scalei = center2[0]*2/151.0
        scalej = center2[1]*2/222.0

        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, thic
63
64      def eraseline(self, loc1, loc2):
65          cv2.line(self.canvas, (loc1[1], loc1[0]), (loc2[1], loc2[0]), color=self.bgcolor, th
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 secreen 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)
84          cv2.line(self.screen, (cursor_loc_j, cursor_loc_i-5), (cursor_loc_j, cursor_loc_i+5)
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 d
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

```
"""Package for hand detection"""
import cv2
import numpy as np
import pdb
import time
import math
import majoraxis

# wrist detection part
def detectwrist(mask,theta,c_xo,c_yo):
    (x,y) = mask.nonzero()
    if len(x) == 0:
        return mask
    # find the minimum / maximum x / y
    x_min = min(x)
    y_min = min(y)
    x_max = max(x)
    y_max = max(y)
    # find the center
    c_x = x.mean(axis=0)
    c_y = y.mean(axis=0)
    l = np.where(mask[:,y_min])[0][0]
    r = np.where(mask[:,y_max])[0][0]
    slopes = []
    if l > r and r > x_min + 30:
        x_2 = r
        y_2 = sum(mask[r,:])
    elif l > x_min + 30:
        x_2 = l
        y_2 = sum(mask[l,:])
    elif r > x_min + 30:
        x_2 = r
        y_2 = sum(mask[r,:])
    else:
    # if the condition is not good, simply 1/4 it
        x_2 = -1
        mask[x_min:x_min+int(1*(x_max-x_min)/4),:] = 0
    # use the method from paper[5].
    if x_2 > 0:
        y2mat = np.ones((x_2-30 - x_min,))* y_2
        x2mat = np.ones((x_2-30 - x_min,))* x_2
```

```
        index = np.arange(x_min, x_2-30)
        ss = np.sum(mask, axis=1)/255
        slope =  (y2mat - ss[x_min:x_2-30])/(x2mat - index)
        wrist = np.argmax(slope)
        mask[x_min:x_min+wrist,:] = 0
    rows,cols = mask.shape[:2]
    M = cv2.getRotationMatrix2D((cols/2,rows/2),(theta*180/math.pi),1)
    dst = cv2.warpAffine(mask,M,(cols,rows))
    M = np.float32([[1,0,-cols/2+c_yo],[0,1,-rows/2 +c_xo]])
    dst = cv2.warpAffine(dst,M,(cols,rows))
    return dst

# in order to detect wrist correctly, rotate image before detecting wrist
def rotateim(im,theta,c_x,c_y):
    c_x = int(c_x)
    c_y = int(c_y)
    rows,cols = im.shape[:2]
    dst = im
    M = np.float32([[1,0,cols/2-c_y],[0,1,rows/2 -c_x]])
    dst = cv2.warpAffine(im,M,(cols,rows))
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-(theta*180/math.pi),1)
    dst = cv2.warpAffine(dst,M,(cols,rows))
    return dst

# skin color detection using hsv,rgb
def skin_color(image):
    # detect hand region
    hsv_im = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
    HSV = cv2.split(hsv_im)
    S = HSV[1]
    V = HSV[2]
    RGB = cv2.split(image)
    Blue = RGB[0]
    Green = RGB[1]
    Red = RGB[2]
    mask = Red - Blue
    mask[mask > 0] = 255
    mask[Blue > Red] = 0
    mask[Green > Red] = 0
    mask[V > 0.73*255] = 0
    mask[S < 0.3*255] = 0
    # erase desks
    mask[350:,:] = 0
    mask[:,:130] = 0
    mask[:,430:] = 0
    # 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 inerita
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

```
import numpy as np
import cPickle
import cv2
import os
import sys
includespath = os.path.abspath('../includes')
sys.path.insert(0, includespath)
import re
import posture

# Load the data set file.
#data and labels are lists of numpy.ndarray



print "Loading Data"
inputfolder = sys.argv[1]
modelfilename = sys.argv[2]
datalist, labellist = [],[]


for f in os.listdir(inputfolder):
    s = re.search(r'train_data_([0-9])_([0-9]+)\.png', f)
    label = s.group(1)
    n = s.group(2)
    print label, n
    if label == posture.poses["DBFINGER"]:
        continue
    img = cv2.imread(inputfolder+'/'+f)
    # cv2.imshow(f, img)
    # cv2.waitKey(0)
    # label = raw_input("Enter the gesture:")
    # touch = raw_input("Does the finger touch the paper?")
    datalist.append(img)
    labellist.append(label)
    # touchlist.append(touch)
print "Done\n"
data = datalist
labels = labellist

indices = np.random.permutation(len(datalist) ).tolist()
traindata, trainlabels= [], []
testdata, testlabels = [], []
```

```

counter = 0
for ind in indices:

    if counter < len(datalist)/10*5:
        traindata.append(datalist[ind])
        trainlabels.append(labellist[ind])
        print ind, "label = ", labellist[ind], "train"
    else:
        testdata.append(datalist[ind])
        testlabels.append(labellist[ind])
        print ind, "label = ", labellist[ind], "test"
    counter = counter+1

if os.path.isfile(modelfilename):
    print "Loading: ", modelfilename
    pos_recognizer = posture.PostureRecognizer.load(modelfilename)
else:
    print "Creating: ", modelfilename
    pos_recognizer = posture.PostureRecognizer()
#Do the training here
print "Start training"
pos_recognizer.train(traindata, trainlabels)
print "saving data"
pos_recognizer.save(modelfilename)


#==========validation===============
score = 0.0
total = 0.0
for i in xrange(len(testlabels)):
    pred = pos_recognizer.classify(testdata[i])[0]
    total+=1
    print i, 'pred = ', pred, 'label=' ,testlabels[i]
    if str(pred) == testlabels[i]:
        score +=1

print "================\n"
print "Score = ", score, "/", total, "=", score/total, "\n"
```

```
from sklearn import svm, neighbors
import numpy as np
import pickle
import os
```

```
import hand_detection
import feature_extraction as fe
import cv2
from collections import deque

poses = {
'POINTING':'1',
'DBFINGER':'2',
'PALM':'3',
'UNKNOWN':'-1',
}
## 1 = pointing finger
## 2 = piece sign
## 3 = palm
## -1 = unknown



class PostureRecognizer(object):
    """A class for hand posture recognition"""
    def __init__(self):
        super(PostureRecognizer, self).__init__()
        self.feature_extractor = fe.OrientationHistogramFeature()
        self.classifier = neighbors.KNeighborsClassifier()
        #self.classifier = svm.SVC()
        self.model = None


    def save(self, filename):
        """Save the current trained model to a pickle file.

        params filename: The output filename.
        """
        if os.path.isfile(filename):
            os.remove(filename)
        outputfile = open(filename,'w')
        pickle.dump(self,outputfile)
        outputfile.close()

    @staticmethod
    def load(filename):
        """load a trained model from the filename.

        parmas  filename: The filename of the trained model
        return  model: A PostureRecognizer instance. """
        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. Eac
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 in
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
```

```
import matplotlib.pyplot as plt
import matplotlib

class FeatureExtractor(object):
    """docstring for FeatureExtractor"""
    def __init__(self):
        super(FeatureExtractor, self).__init__()

    def extract_feature(self, image):
        """returns the feature vector of the image as a 1-D numpy array"""

class OrientationHistogramFeature(FeatureExtractor):
    """docstring for OrientationHistogramFeature"""
    def __init__(self):
        super(OrientationHistogramFeature, self).__init__()


    def extract_features(self, image):
        hand_mask, ang, skin_mask= hd.hand_detection(image)
        # while cv2.waitKey(20) != ord('a'):
        #     pass
        # hand_mask_tmp = np.reshape(hand_mask, hand_mask.shape+(1,) )
        # hand_mask_tmp = np.concatenate((hand_mask_tmp,hand_mask_tmp,hand_mask_tmp),axis=2)

        image_gray =  cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        #masked = np.multiply(image, hand_mask_tmp)
        masked = np.multiply(image_gray, hand_mask)
        # plt.figure()
        # plt.imshow(masked, cmap = matplotlib.cm.Greys_r)
        # plt.figure()
        # plt.plot(masked)
        # cv2.imshow('masked', masked)
        # while cv2.waitKey(20) != ord('a'):
        #     pass


        #masked = image[hand_mask]
        sobelx = cv2.Sobel(masked,cv2.CV_64F,1,0,ksize=5)
        sobely = cv2.Sobel(masked,cv2.CV_64F,0,1,ksize=5)

        angles = np.angle(sobelx + 1j*sobely,deg=True)
        # amplitudes = np.absolute(sobelx + 1j*sobely)
        # angles = angles.flatten()

        angles = angles.flatten()
        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          if len(angles) == 0:
61              return np.zeros((n_bins,)), hand_mask, ang, skin_mask
62          ang_hist, bins = np.histogram(angles, bins=n_bins,range=(-180.0,180.0))
63          # ang_hist[18]=0
64          # #print type(ang_hist)
65          # amp_hist = np.histogram(amplitudes, bins=5)[0]
66          # plt.figure()
67          #ang_hist,bins,patches = plt.hist(angles.flatten(), bins=n_bins, range=(-180, 180),
68          # print sum(ang_hist)
69
70
71
72          #====Angular disposition=====
73          # ang, cx, cy = majoraxis.majoraxis(hand_mask)
74          ang = ang/pi*180
75          try:
76              majorangle = np.where(np.histogram([ang], bins=n_bins,range=(-180,180))[0] != 0)
77              ang_hist = np.roll(ang_hist, (n_bins - majorangle)%n_bins )
78          except(IndexError):
79              pass
80          # plt.figure()
81          # plt.hist(range(-180,180, 10), bins=n_bins, range=(-180,180), weights=ang_hist)
82          # plt.show()
83          return ang_hist, hand_mask, ang, skin_mask
84
85
86
87
```

Appendix:Code Fingertip detection

fingertip.py

```
import cv2
import numpy as np
import pdb
import time
import math
def find_fingertip(label, mask):
    """
    Find the location of the fingertips. Only have to return one pixel.
    @param np.array         hand_mask
    a numpy array of size(480,640,1)
    @label string           label
    A string represting the label of the frame.


    Look at posture.py for more informations.
    return (int, int)       loc
    The location of the fingertip of the longest finger.
    return string           wrist_end
    "up/down/left/right" representing the side of the wrist.
    """
    mask[mask != 0] = 255
    (x,y) = mask.nonzero()
    # find the minimum / maximum x / y
    if len(x) == 0:
        return None,None
    x_min = min(x)
    y_min = min(y)
    x_max = max(x)
    y_max = max(y)
    # find the center
    c_x = x.mean(axis=0)
    c_y = y.mean(axis=0)
    # find the most largest edge (= wrist)
    up = (sum(mask[x_min,:])/255)
    bottom = (sum(mask[x_max,:])/255)
    left = (sum(mask[:,y_min])/255)
    right = (sum(mask[:,y_max])/255)
    wrist = max(up,bottom,left,right)
    wrist_ch = ''
    # mask wrist side
    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

```
import sys
import os
includespath = os.path.abspath('../includes')
sys.path.insert(0,includespath)
import evala as e
if __name__ == '__main__':
    username = sys.argv[1]
    e.exp_points(username)
    e.exp_line(username)
    e.exp_circle(username)
```

evaluation_m.py

```
import sys
import os
includespath = os.path.abspath('../includes')
sys.path.insert(0,includespath)
import evala as e
if __name__ == '__main__':
    username = sys.argv[1]
    e.exp_points_m(username)
    e.exp_line_m(username)
    e.exp_circle_m(username)
```

evala.py

```
import cv2
import numpy as np
import pdb
import time
import math
import os
import csv
import eval_ui as eu
import mousepaint as mo
def drawing(im):
    rgb = cv2.split(im)
    # use only B (because we use Red color to write)
    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      ff = open('../experiments/std.csv','ab')
80      csvWriter2 = csv.writer(ff)
81      # do experiment
82      center = [250,290]
83      r = 190
84      ts = []
85      cs = []
86      for i in range(3):
87          ## for debuging
88          im2 = cv2.imread('../test.png')
89          cv2.circle(im2,(290,250),190,(255,0,0),1)
90          filename = '../experiments/circle_'+username+'_'+str(i) + '.png'
91          im,time = eu.evaluation(filename,im2)
92          c = evalcircle(im,center,r)
93          csvWriter.writerow([username,'VI', i , round(time,3),round(c,3)])
94          ts.append(time)
95          cs.append(c)
96          if i == 2:
97              csvWriter2.writerow(['circle',username,'VI',round(np.std(ts),3),round(np.std(cs)
98          print 'wirte'
99
100 def exp_line(username):
101     if(not os.path.exists('../experiments')):
102         os.mkdir('../experiments')
103     f  = open('../experiments/line.csv','ab')
104     csvWriter = csv.writer(f)
105     ff = open('../experiments/std.csv','ab')
```

```
106     csvWriter2 = csv.writer(ff)
107     ts = []
108     cs = []
109     # do experiment
110     start = (50,50)
111     end = (400,400)
112     for i in range(3):
113         ## for debuging
114         im2 = cv2.imread('../test.png')
115         cv2.line(im2,(50,50),(400,400),(255,0,0),1)
116         filename = '../experiments/line_'+username+'_'+str(i) + '.png'
117         im,time = eu.evaluation(filename,im2)
118         ## for debugging
119         c = evalline(im,start,end)
120         csvWriter.writerow([username,'VI', i , round(time,3),round(c,3)])
121         ts.append(time)
122         cs.append(c)
123         if i == 2:
124             csvWriter2.writerow(['line',username,'VI',round(np.std(ts),3),round(np.std(cs),3
125 
126 def exp_points(username):
127     if(not os.path.exists('../experiments')):
128         os.mkdir('../experiments')
129     f  = open('../experiments/points.csv','ab')
130     csvWriter = csv.writer(f)
131     ff = open('../experiments/std.csv','ab')
132     csvWriter2 = csv.writer(ff)
133     # do experiment
134     ts = []
135     cs = []
136     points = [(40,40),(40,350),(350,40),(350,350)]
137     for i in range(3):
138         im2 = cv2.imread('../test.png')
139         cv2.circle(im2,(40,40),3,(255,0,0),1)
140         cv2.circle(im2,(350,350),3,(255,0,0),1)
141         cv2.circle(im2,(40,350),3,(255,0,0),1)
142         cv2.circle(im2,(350,40),3,(255,0,0),1)
143         filename = '../experiments/points_'+username+'_'+str(i) + '.png'
144         im,time = eu.evaluation(filename,im2)
145         c = evalpoints(im,points)
146         csvWriter.writerow([username,'VI', i , round(time,3),round(c,3)])
147         ts.append(time)
148         cs.append(c)
149         if i == 2:
150             csvWriter2.writerow(['points',username,'VI',round(np.std(ts),3),round(np.std(cs)
151 
```

```
152 def exp_circle_m(username):
153     if(not os.path.exists('../experiments')):
154         os.mkdir('../experiments')
155     f  = open('../experiments/circle.csv','ab')
156     csvWriter = csv.writer(f)
157     ff = open('../experiments/std.csv','ab')
158     csvWriter2 = csv.writer(ff)
159    # do experiment
160     center = [250,290]
161     r = 190
162     ts = []
163     cs = []
164     for i in range(3):
165         ## for debuging
166         im2 = cv2.imread('../test.png')
167         cv2.circle(im2,(290,250),190,(255,0,0),1)
168         filename = '../experiments/circle_'+username+'_'+str(i) + '_m.png'
169         im,time = mo.evaluation(filename,im2)
170         c = evalcircle(im,center,r)
171         csvWriter.writerow([username,'mouse',i,round(time,3),round(c,3)])
172         print 'wirte'
173         ts.append(time)
174         cs.append(c)
175         if i == 2:
176             csvWriter2.writerow(['circle',username,'mouse',round(np.std(ts),3),round(np.std(
177
178 def exp_line_m(username):
179     if(not os.path.exists('../experiments')):
180         os.mkdir('../experiments')
181     f  = open('../experiments/line.csv','ab')
182     csvWriter = csv.writer(f)
183     ff = open('../experiments/std.csv','ab')
184     csvWriter2 = csv.writer(ff)
185     ts = []
186     cs = []
187
188     # do experiment
189     start = (50,50)
190     end = (400,400)
191     for i in range(3):
192         ## for debuging
193         im2 = cv2.imread('../test.png')
194         cv2.line(im2,(50,50),(400,400),(255,0,0),1)
195         filename = '../experiments/line_'+username+'_'+str(i) + '_m.png'
196         im,time = mo.evaluation(filename,im2)
197         ## for debugging
```

```
        c = evalline(im,start,end)
        csvWriter.writerow([username,'mouse',i,round(time,3),round(c,3)])
        ts.append(time)
        cs.append(c)
        if i == 2:
            csvWriter2.writerow(['line',username,'mouse',round(np.std(ts),3),round(np.std(cs

def exp_points_m(username):
    if(not os.path.exists('../experiments')):
        os.mkdir('../experiments')
    f  = open('../experiments/points.csv','ab')
    csvWriter = csv.writer(f)
    ff = open('../experiments/std.csv','ab')
    csvWriter2 = csv.writer(ff)
    # do experiment
    points = [(40,40),(40,350),(350,40),(350,350)]
    ts = []
    cs = []
    for i in range(3):
        im2 = cv2.imread('../test.png')
        cv2.circle(im2,(40,40),3,(255,0,0),1)
        cv2.circle(im2,(350,350),3,(255,0,0),1)
        cv2.circle(im2,(40,350),3,(255,0,0),1)
        cv2.circle(im2,(350,40),3,(255,0,0),1)
        filename = '../experiments/points_'+username+'_'+str(i) + '_m.png'
        im,time = mo.evaluation(filename,im2)
        c = evalpoints(im,points)
        csvWriter.writerow([username,'mouse',i,round(time,3),round(c,3)])
        ts.append(time)
        cs.append(c)
        if i == 2:
            csvWriter2.writerow(['points',username,'mouse',round(np.std(ts),3),round(np.std(

if __name__ == '__main__':
    exp_points('aya')
```

eval_ui.py

```
import sys
import os.path
includespath = os.path.abspath('../includes')
sys.path.insert(0, includespath)
includespath = os.path.abspath('../bin')
sys.path.insert(0,includespath)
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/modelKNN5.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()
```