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

Angus Ding ad3180
Ayaka Kume ak3682

May 13, 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

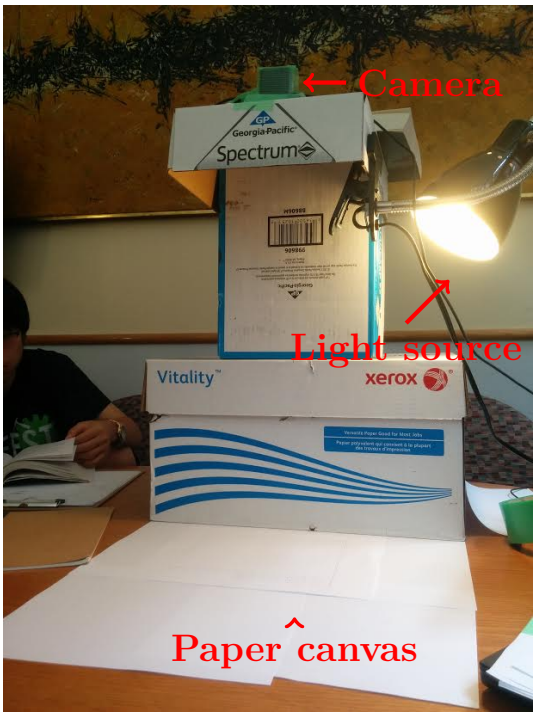


Figure 1: The input device

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.

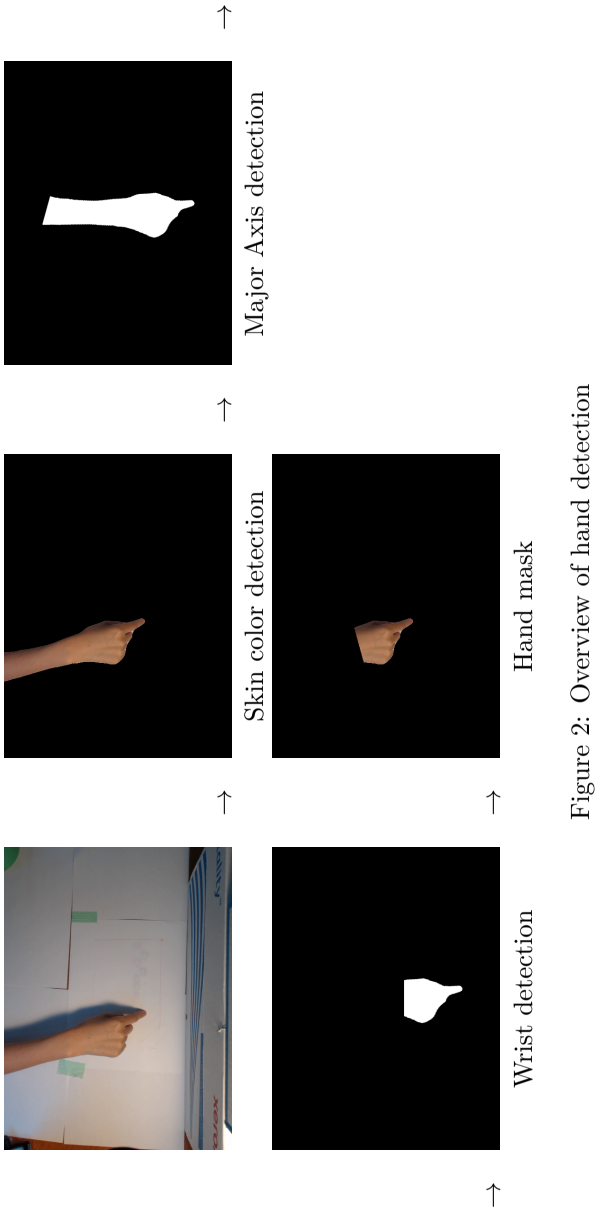


Figure 2: Overview of hand detection

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.

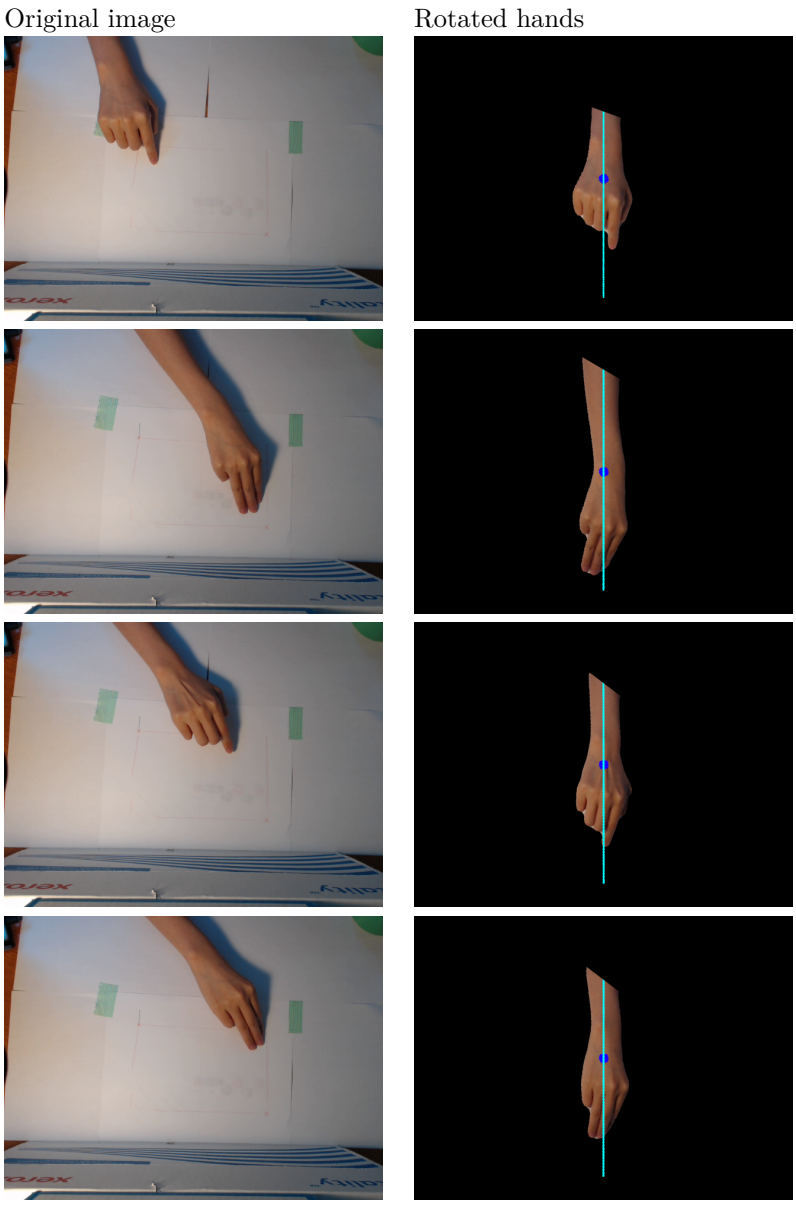


Figure 3: The results for orientation detection

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

3.3 Hand detection

If the arm is in the image, we may not classify the gesture correctly. Thus, we have 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 a point which is either on a left or right boundaries and the nearest from the bottom boundary. [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 is a spray hand and so the palm is always the widest. Like Fig.4 shows, we cannot find appropriate b because the leftmost and the rightmost points are not in the palm, but in the wrist and fingertip. This is because the arm is not pointing downward so it is difficult to detect the 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 tends to be in the palm region. In Fig.4 , the wrist can be detect by our method but not the previous method. This is because our method can detect the widest point as b. We assume b is not too near to the wrist end. That is, if a point is within 30 pixels from the wrist end, use another point. Fig.5 shows the example of the intensity histogram.

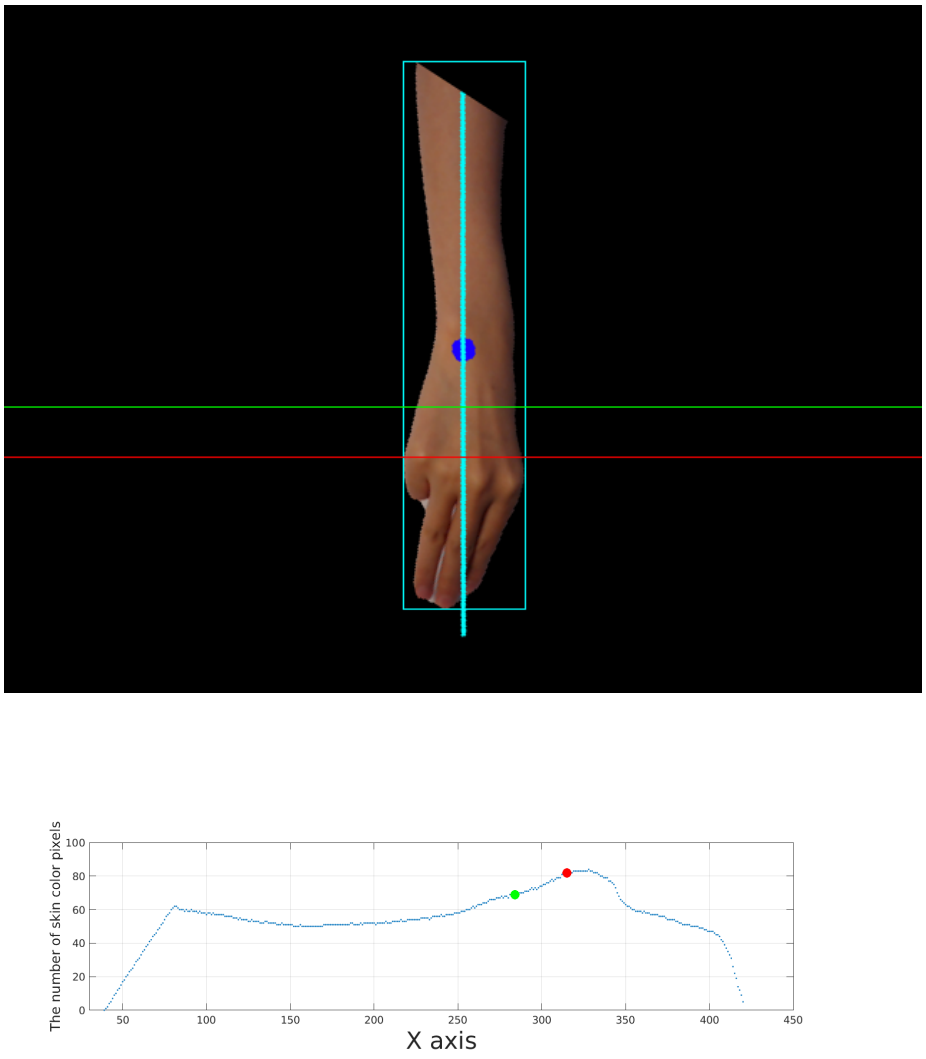


Figure 5: Hand image Top: Hand image. Green line is wrist position and red line is b. Bottom: histogram

Red point in the histogram corresponds to b's coordinate. Green point in the histogram corresponds to the wrist end's coordinate. The red line and green line in the left image corresponds to the b and wrist end.

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 improve 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 length. 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.

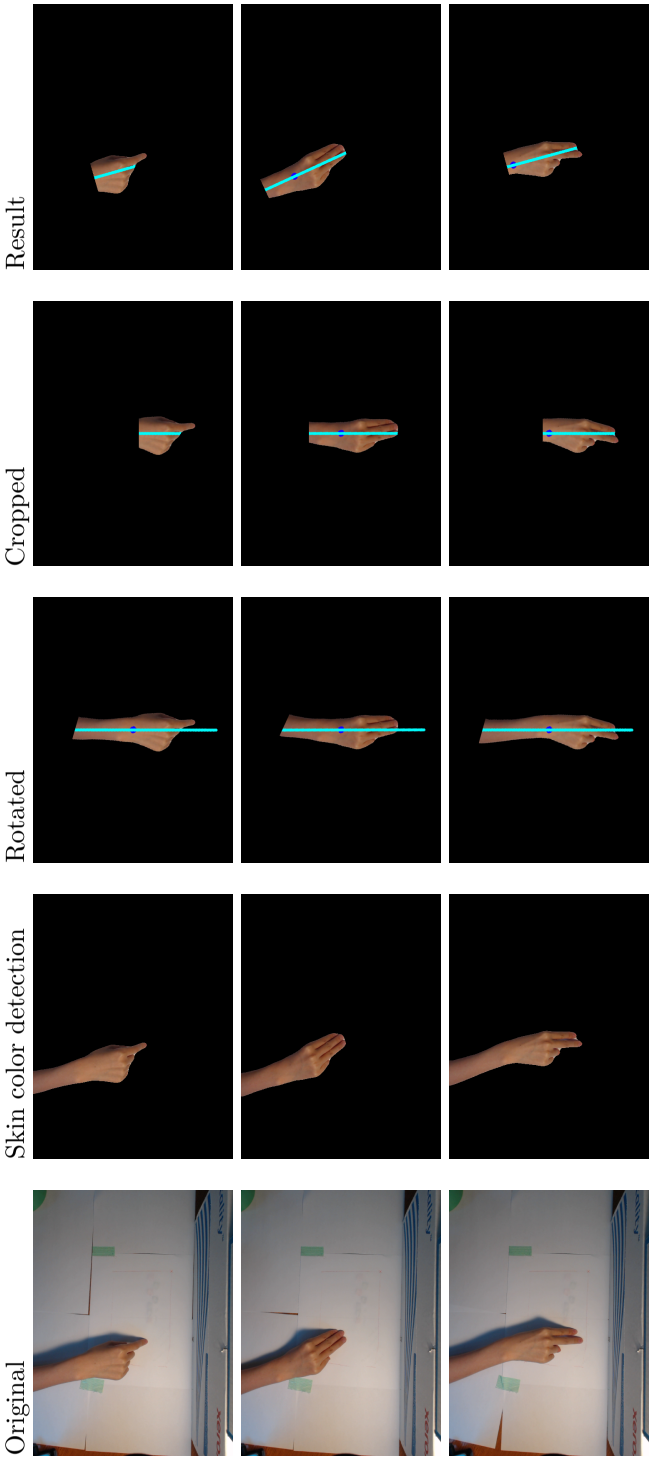


Figure 6: The results of hand detection

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.

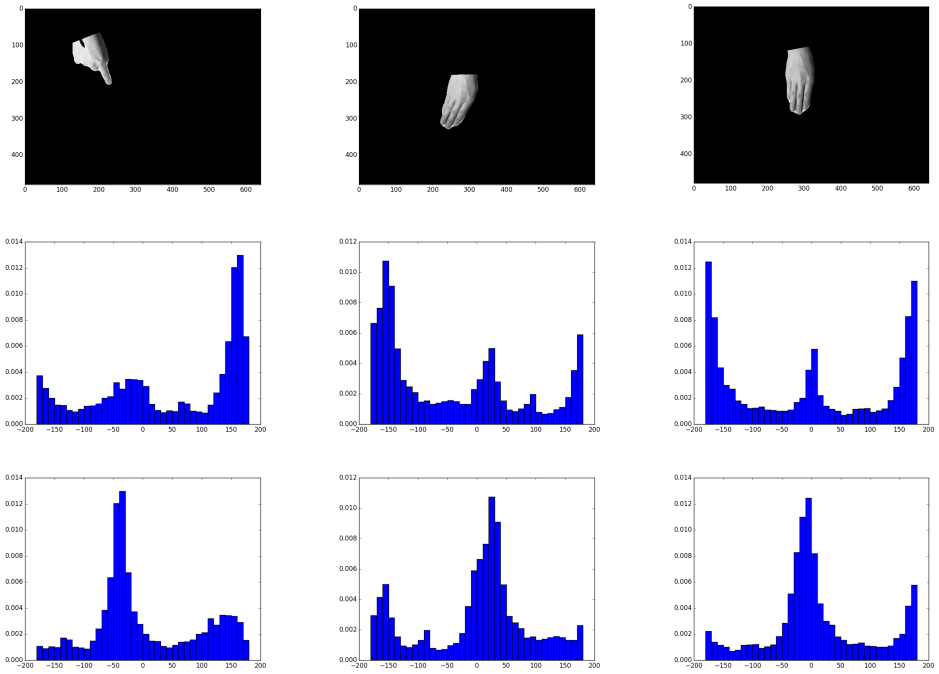


Figure 7: The orientation histograms. The first row is the input image, converted to gray scale and applied hand detection. The second row is the orientation histogram. The third row is the orientation histogram after the translation of the major axis.

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 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.

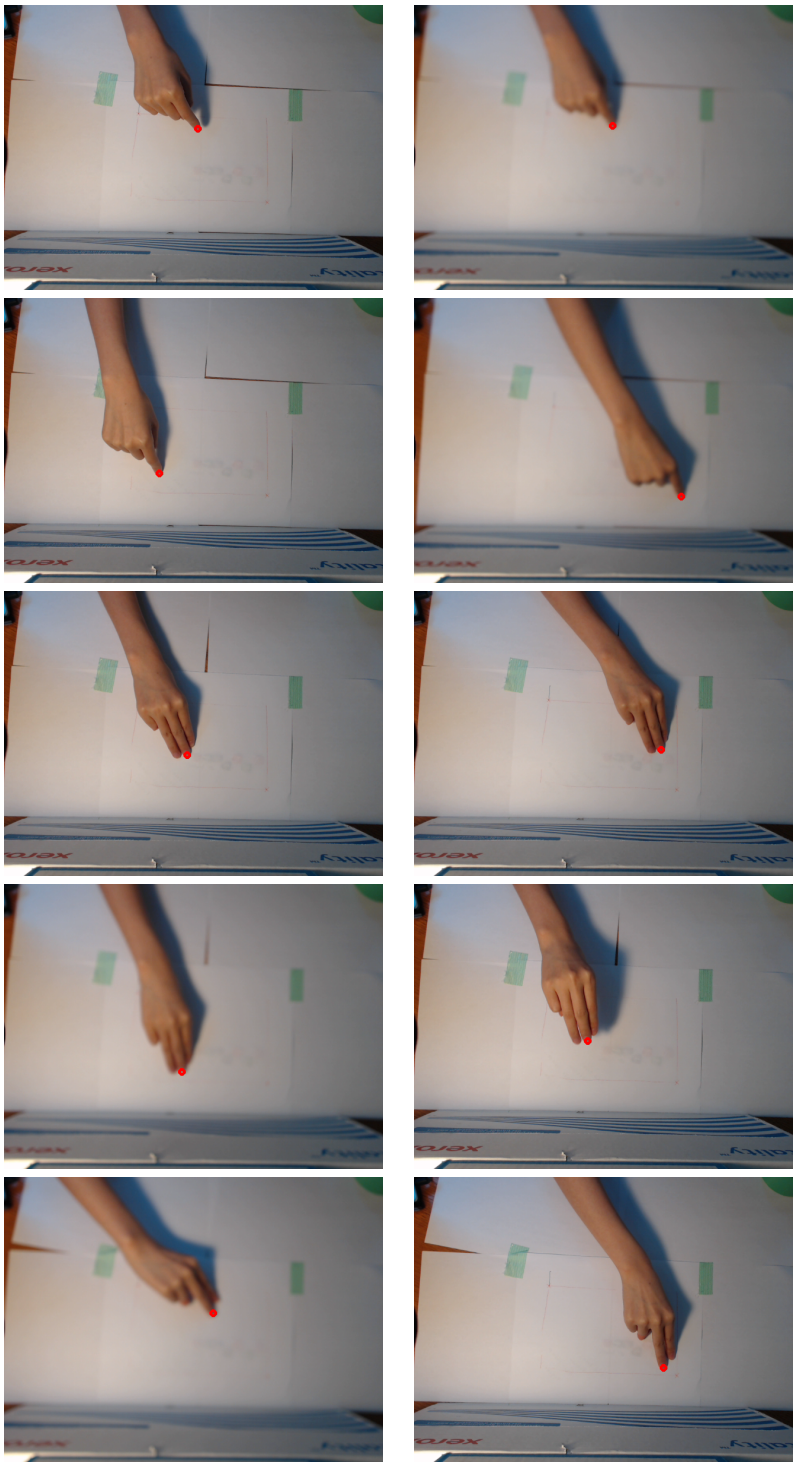


Figure 8: The results of finger detection

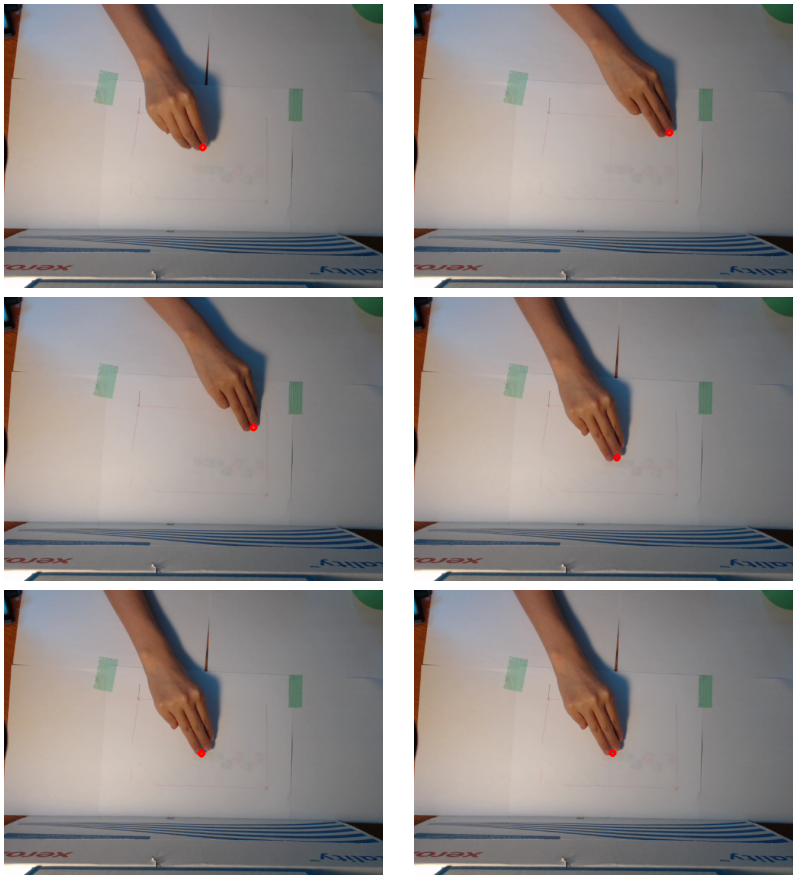


Figure 9: The errors

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:

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

âĂć , 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 ligth 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 mouspaint.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.

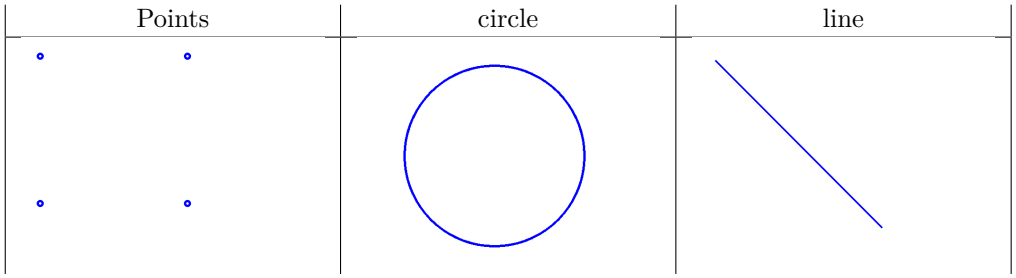


Figure 10: Three tasks

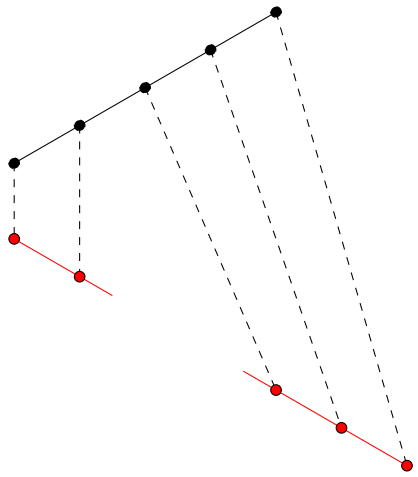


Figure 11: Line error measurement

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)

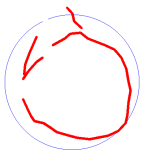
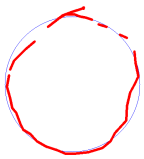
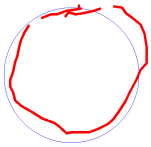
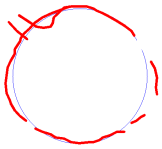
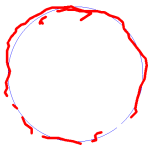
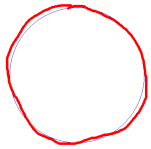
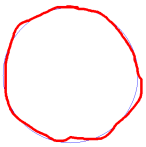
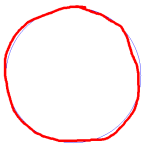
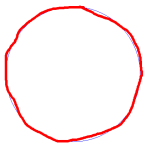
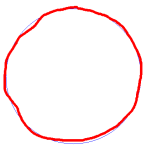
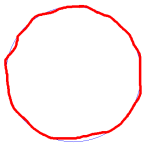
	Trial 1	Trial 2	Trial 3
User A (VI)			
User B (VI)			
User A (Mouse)			
User B (Mouse)			

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)

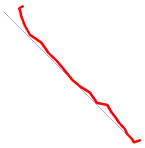
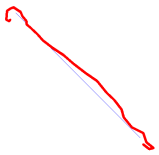
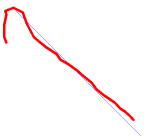
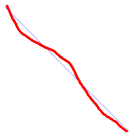
	Trial 1	Trial 2	Trial 3
User A (VI)			
User B (VI)			
User A (Mouse)			
User B (Mouse)			

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

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).