A Method for Controlling Mouse Movement using a Real-Time Camera

Hojoon Park
Department of Computer Science
Brown University, Providence, RI, USA
hojoon@cs.brown.edu

Abstract

This paper presents a new approach for controlling mouse movement using a real-time camera. Most existing approaches involve changing mouse parts such as adding more buttons or changing the position of the tracking ball. Instead, we propose to change the hardware design. Our method is to use a camera and computer vision technology, such as image segmentation and gesture recognition, to control mouse tasks (left and right clicking, double-clicking, and scrolling) and we show how it can perform everything current mouse devices can. This paper shows how to build this mouse control system.

1. Introduction

As computer technology continues to develop, people have smaller and smaller electronic devices and want to use them ubiquitously. There is a need for new interfaces designed specifically for use with these smaller devices. Increasingly we are recognizing the importance of human computing interaction (HCI), and in particular vision-based gesture and object recognition. Simple interfaces already exist, such as embedded-keyboard, folder-keyboard and mini-keyboard. However, these interfaces need some amount of space to use and cannot be used while moving. Touch screens are also a good control interface and nowadays it is used globally in many applications. However, touch screens cannot be applied to desktop systems because of cost and other hardware limitations. By applying vision technology and controlling the mouse by natural hand gestures, we can reduce the work space required. In this paper, we propose a novel approach that uses a video device to control the mouse system. This mouse system can control all mouse tasks, such as clicking (right and left), double-clicking and scrolling. We employ several image processing algorithms to implement this.

2. Related Work

Many researchers in the human computer interaction and robotics fields have tried to control mouse movement using video devices. However, all of them used different methods to make a clicking event. One approach, by Erdem et al, used finger tip tracking to control the motion of the mouse. A click of the mouse button was implemented by defining a screen such that a click occurred when a user's hand passed over the region [1, 3]. Another approach was developed by Chu-Feng Lien [4]. He used only the finger-tips to control the mouse cursor and click. His clicking method was based on image density, and required the user to hold the mouse cursor on the desired spot for a short period of time. Paul et al, used still another method to click. They used the motion of the thumb (from a 'thumbs-up' position to a fist) to mark a clicking event thumb. Movement of the hand while making a special hand sign moved the mouse pointer.

Our project was inspired by a paper of Asanterabi Malima et al. [8]. They developed a finger counting system to control behaviors of a robot. We used their algorithm to estimate the radius of hand region and other algorithms in our image segmentation part to improve our results. The segmentation is the most important part in this project. Our system used a color calibration method to segment the hand region and convex hull algorithm to find finger tip positions [7].

3. System Flow

The system order is shown in Figure 1. Initially, when we get an image from the camera, we convert the color space RGB to YCbCr. Then, we define a range of colors as 'skin color' and convert these pixels to white; all other pixels are converted to black. Then, we compute the centroid of the dorsal region of the hand. After we identify the hand, we find the circle that best fits this region and multiply the radius of this circle by some value to obtain the maximum extent of a 'non-finger region'. From the binary image of the hand, we get vertices of the convex hull of each finger. From the vertex and center distance, we obtain the positions of the active fingers. Then by extending any one vertex, we control the mouse movement.

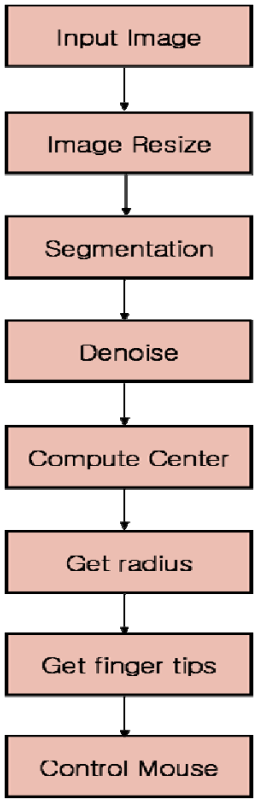


Figure 1. An overview of our hand gesture recognition and mouse control system. Input image converted to binary image to separate hand from background**.** Center of hand and computed calculated radius of hand found. Finger tip points using Convex Hull algorithm. Mouse controlled using hand gesture.

4. Hand Gesture Recognition

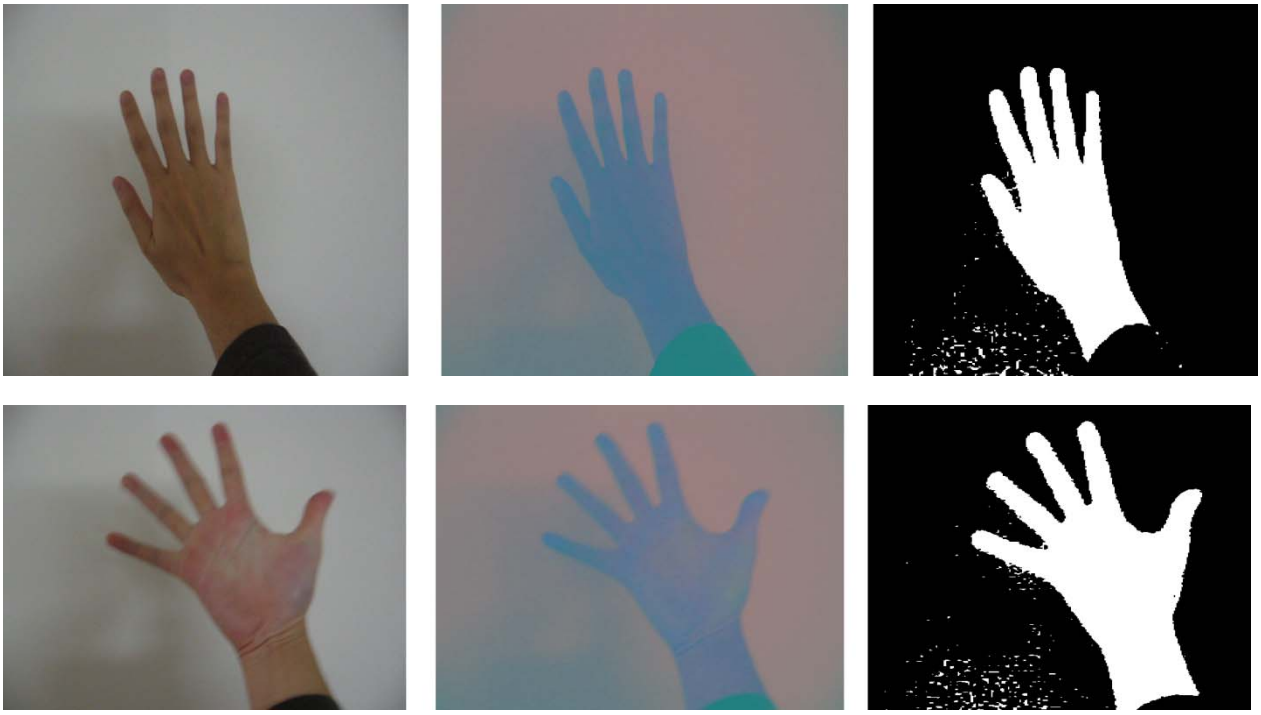
4.1 Image Resize

First to recognize a hand gesture, we need to resize the input image in order to map camera coordinates to screen coordinates. There are two ways to map from source image to destination image. The first way is to compute the ratio of screen resolution to camera resolution. To determine the *x* and *y* coordinates on the screen of a given camera pixel, we use the following equation:

$$x = \frac{x'}{640} \times cx, \quad y = \frac{y'}{480} \times cy$$

where (x' , y') is the camera position, (cx, cy) is the current screen resolution, and (x, y) is the corresponding screen position of camera position. The second way is to use the cvResize() function in OpenCV. This function maps a source image to a destination image as smoothly as possible. To accomplish this, the function uses interpolation. Interpolation is a method to add or subtract pixels to an image as necessary to expand or shrink its proportions while introducing minimum distortion. Using this function, we can easily map the position of each input image pixel to a screen position.

In this paper, we used the first method because it is easier to implement, more accurate and preserves more data.



(a) (b) (c)

Figure 2. Convert from original image to binary image. We converted RGB to YCrCb color space and segment hand region from original input image. (a) RGB image. (b) YCbCr image. (c) Binary image.

4.2 Segmentation

Next, we need to separate the hand area from a complex background. It is difficult to detect skin color in natural environments because of the variety of illuminations and skin colors. So, we need to carefully pick a color range. To get better results, we converted from RGB color space to YCbCr color space, since YCbCr is insensitive to color variation. The conversion equation is as follows.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.29900 & 0.587000 & 0.114000 \\ -0.168736 & -0.331264 & 0.500000 \\ 0.500000 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where Y is luminance, Cb is the blue value and Cr is the red value. From this information, we detect skin color by selecting a particular color range from the Cb and Cr values. In this paper, we choose Y, Cr and Cb values of 0 to 255, 77 to 127, and 133 to 173, as respectively, the skin color region. (It should be noted that these values were chosen for the convenience of the investigator.) Then we loop over all the image pixels, changing pixels within the skin color range to 0, and all others to 255. Hence, we obtain a binary image of the hand. Figure 2 shows the results.

4.3 Deleting noise

Using this approach, we cannot get a good estimate of the hand image because of background noise. To get a better estimate of the hand, we need to delete noisy pixels from the image. We use an image morphology algorithm that performs image erosion and image dilation to eliminate noise [1]. Erosion trims down the image area where the hand is not present and Dilation expands the area of the Image pixels which are not eroded. Mathematically, Erosion is given by,

$$A \Theta B = \{x \mid (B)_x \cap A^c = \phi\}$$

where A denotes input image and B denotes Structure elements. The Structure element is operated on the Image using a Sliding window and exact matches are marked. Figure 3 shows a graphical representation of the algorithm. Dilation is defined by,

$$A \oplus B = \{x \mid (\hat{B}) \cap A \neq \phi\}$$
$$= \{x \mid [(\hat{B})_x \cap A] \subseteq A\}$$

where A denotes the input image and B denotes the structure element. The same structure element is operated on the image and if the center pixel is matched, the whole area around that pixel is marked. Figure 4 shows the algorithm. Erosion and Dilation are changed by the shape of B. Thus, B should be selected in advance. We performed erode function with structure of 10x10 square pixels three times and dilate function with 6x6 square pixels structure three times to get a clearer hand. Figure 5 is the result of this algorithm.

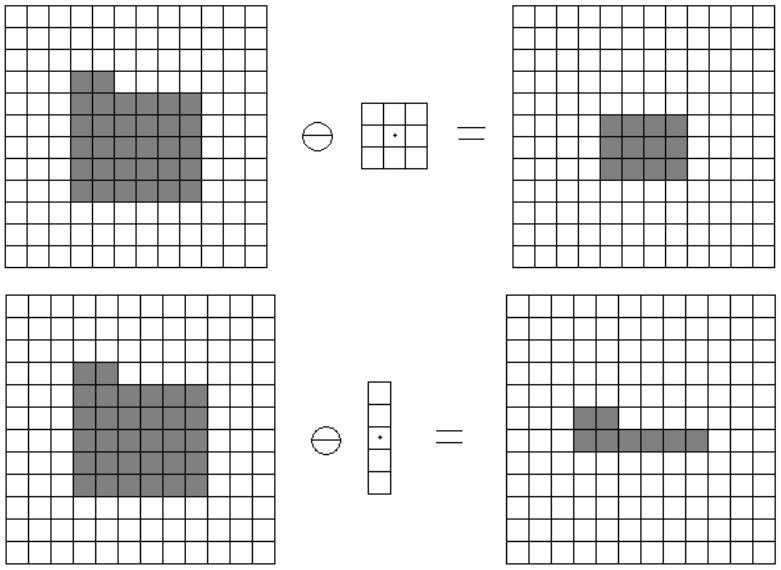
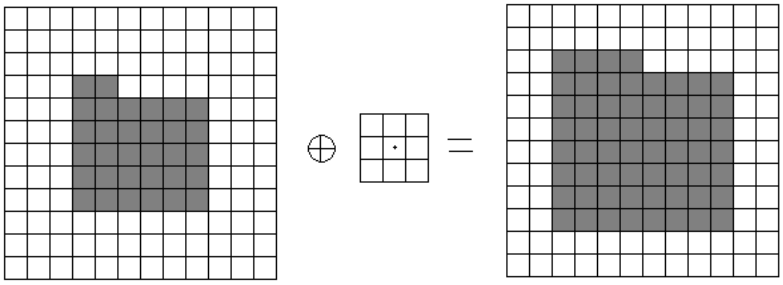


Figure 3. The result of image erosion. The structure element scans the input image. When the structure element matches, the central pixel is kept; when it does not match, all pixels are discarded.



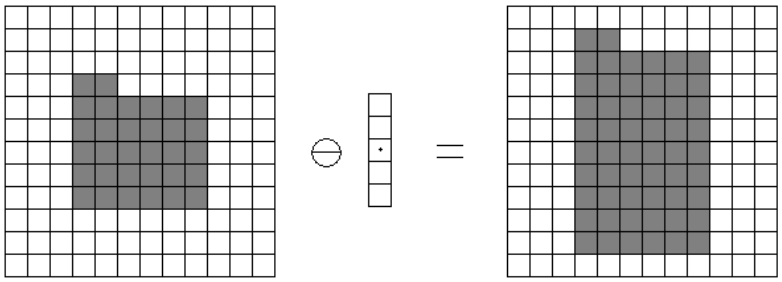


Figure 4. The result of image dilation. The structure element scans the input image. When the center of the structure matches any pixels, the bin of the structure element is filled.

(a) (b)

Figure 5. The result of image morphology. Perform erosion and dilation on binary image. (a) Original image. (b) The result of image morphology.

4.4 Finding center and the size of the hand

After segmenting hand region and background, we can calculate the center of the hand with the following equation:

$$\bar{x} = \frac{\sum_{i=0}^{k} x_i}{k} \quad , \quad \bar{y} = \frac{\sum_{i=0}^{k} y_i}{k}$$

where x_i and y_i are x and y coordinates of the i pixel in the hand region, and k denotes the number of pixels in the region. After we locate the center of the hand, we compute the radius of the palm region to get hand size. To obtain the size of the hand, we draw a circle increasing the radius of the circle from the center coordinate until the circle meets the first black pixel. When the algorithm finds the first black pixel then it returns to the current radius value. This algorithm assumes that when the circle meets the first black pixel, after drawing a larger and larger circle, then the length from the center is the radius of the back of the hand. Thus, the image segmentation is the most significant part because if some of the black pixels are made by shadows and illuminations near the center, then the tracking algorithm will meet earlier than the real background and the size of the hand region becomes smaller than the real hand. This problem breaks this system. Figure 6 shows the results.

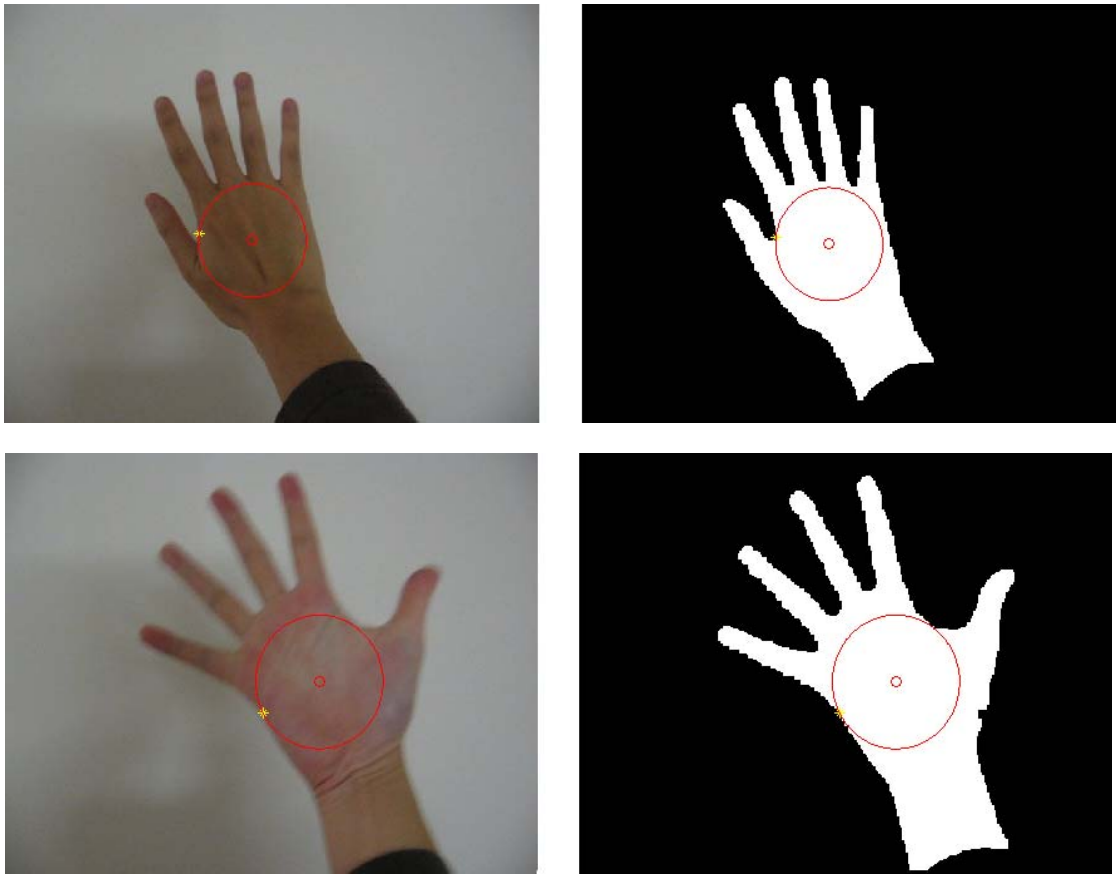


Figure 6. The result of the computed hand size. Drawing a larger and larger circle, we obtained the size of the back of the hand.

4.5 Finding finger tip

To recognize that a finger is inside of the palm area or not, we used a convex hull algorithm. The convex hull algorithm is used to solve the problem of finding the biggest polygon including all vertices. Using this feature of this algorithm, we can detect finger tips on the hand. We used this algorithm to recognize if a finger is folded or not. To recognize those states, we multiplied 2 times (we got this number through multiple trials) to the hand radius value and check the distance between the center and a pixel which is in convex hull set. If the distance is longer than the radius of the hand, then a finger is spread. In addition, if two or more interesting points existed in the result, then we regarded the longest vertex as the index finger and the hand gesture is clicked when the number of the result vertex is two or more.

The result of convex hull algorithm has a set of vertexes which includes all vertexes. Thus sometimes a vertex is placed near other vertexes. This case occurs on the corner of the finger tip. To solve this problem, we deleted a vertex whose distance is less than 10 pixels when comparing with the next vertex. Finally, we can get one interesting point on each finger. Figure 7 shows the results.

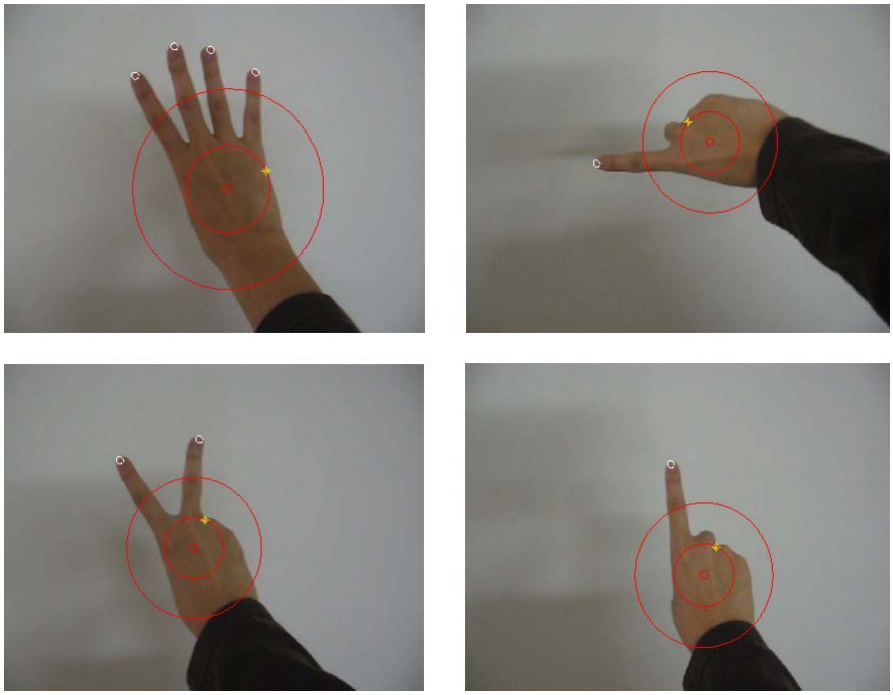


Figure 7. The result of finger tip tracking algorithm. If a tip of a finger is on the outside of the larger red circle, then it can be found by the convex hull algorithm.

5. Controlling Mouse

5.1 Moving Mouse Cursor

We used the index finger as a cursor controller to control mouse cursor. We used two different approaches for moving the mouse cursor. The first method is mapping cursor control. It means that the index finger on a camera screen can position maps to a desktop screen position. In other words, the mouse cursor is placed on the desktop window along with the index finger tips position displayed on the camera screen position. This method has a problem. If the resolution of the desktop window has a higher resolution than the camera resolution, then the cursor position cannot be accurate because when the camera resolution converts to the desktop window resolution we lose intermediate value. We expect the ratio of jumping pixel is up to 4 pixels. The second method is weighted speed cursor control. We get a difference of the finger of the current image and the previous image and compute the distance between the two. Next, we move the mouse cursor if the gap between the two finger images (current and previous frame) is far then the mouse cursor moves fast or, if the gap is close then the cursor moves slow. This algorithm also has a problem. Since we are working with real-time image processing, the client machine should be fast. The image input speed is 15 frames per second and we need image processing time on CPU. It means that some machines which cannot achieve image processing 15 images per sec do not work smoothly because computing the image center and the hand shape takes time. Thus, this algorithm does not work properly. In this paper, we used the first method which uses absolute position of finger tips because it is more accurate than the second method.

5.2 Left Clicking and Double-Clicking

To call system event for left clicking, at least two convex hull vertexes have to be off the palm area which was computed in the previous part. In addition, the x position of one of two vertexes should be lower than the other to restrict detection of other finger tips. When the degree of the index finger and thumb is 70 to 90 degree then we can recognize that the gesture is left clicking. Actually, if the thumb is placed outside the circle of the hand region, then the gesture is left clicking. The double-clicking occurs when the thumb moves 0 to 90 degree and back two times fast.

5.3 Right Clicking

We simply implemented this part using previous gestures. If we make the hand pose left clicking for 3 seconds, then the system calls the right clicking event.

5.4 Scrolling

Scrolling is a very useful task with hand gestures. We also implemented this feature in our system. As figure 5 shows, we assigned a 10% margin as a scroll area on the right side of the desktop screen region and divided by half. If the index finger placed the upper area of the scroll area with clicking pose then the system called the scroll up function or else the system called the scroll down function.

6. Experiments and Results

We tested all mouse tasks such that left click, right click, double-click, dragging, and scrolling on windows. The tested system is that Core2-Duo T8300, 2GB memory, GeForce 8300, Microsoft Windows XP, Microsoft LifeCam VX-1000 (640x480 resolution, 15fps). We could not compare with the mouse device because this hand gesture system always shows lower performance than real mouse device. Instead of comparing with a real mouse, we allowed to use this system to four testers to know how it can be adapted easily.

Our method for evaluating performance is that we checked the time that a task, such as clicking, double clicking and scrolling, is done. We designed four experiments to get performance. In the first experiment we placed an icon on the center of desktop window and put the cursor in the top-left corner. We then measured the time in how long it takes to select the icon. In the second experiment the position of the icons is same and we only measured the time to show the drop down menu on the icon. In the third experiment the position of the icons is same with the second experiment and we measured the time to open the icon. In the final experiment we opened a web site (http://news.yahoo.com) on the center of the desktop window and measured the time until the scroll bar moves from top to bottom. The results are shown below: (time = sec)

	Trial 1	**Trial 2**	**Trial 3**	**Average**
User1	1.2	1.0	1.1	1.10
User2	1.5	1.3	1.0	1.26
User3	1.2	1.1	1.1	1.33
User4	1.0	0.9	1.3	1.06

Table 1. The result of Left-Clicking

	Trial 1	**Trial 2**	**Trial 3**	**Average**
User1	4.2	4.1	4.2	4.16
User2	3.8	4.0	4.2	4.00
User3	4.2	4.1	4.1	4.13
User4	4.0	4.0	4.2	4.06

Table 2. The result of Right-Clicking

	Trial 1	**Trial 2**	**Trial 3**	**Average**
User1	2.5	3.1	2.2	2.6
User2	2.6	1.9	1.9	2.13

User3	1.7	1.9	2.0	1.86
User4	1.6	2.3	2.1	2.00

Table 3. The result of Double-Clicking

	Trial 1	**Trial 2**	**Trial 3**	**Average**
User1	3.7	2.5	7.4	4.5
User2	2.3	4.9	2.6	3.2
User3	3.2	5.2	4.1	4.1
User4	3.4	2.1	4.8	3.4

Table 4. The result of Scrolling

	Left-Clicking	**Right-Clicking**	**Double-Clicking**	**Scrolling**
User1	1.10	4.16	2.6	4.5
User2	1.26	4.00	2.13	3.2
User3	1.33	4.13	1.86	4.1
User4	1.06	4.06	2.00	3.4

Table 5. The average time of all experiment results. Four users tested clicking, double-clicking, and scrolling. Measuring the time until finishing a given task, we estimated the system performance.

Every tester was new in this system. The clicking task showed similar times with all testers. However, the scrolling result showed the time as unstable. The reason is that the focus of the camera lens works automatically so when the index finger went to the border on the screen then part of the hand colors becomes dim. Eventually, hand segmentation failed because colors could not be on the color range which is in the segmentation part. Thus, we could not get a clear shape of the hand from the video streaming.

7. Discussion

In this project, the problem was when the finger shook at lot. Since we used real-time video, the illumination changes every frame. Hence the position of the hand changes every frame. Thus, the finger tip position detected by convex hull algorithm is also changed. Then the mouse cursor pointer shakes fast. To fix this problem, we added a code that the cursor does not move if the difference of the previous and the current finger tips position is within 5 pixels. This constraint worked well but it makes it difficult to control the mouse cursor sensitively. Another problem by illumination issue is segmentation of the background for extracting the hand shape. Since the hand reflects all light sources, the hand color is changed according to the place. If hand shape is not good then our algorithm cannot work well because our algorithm assumes the hand shape is well segmented. If the hand shape is not good then we cannot estimate the length of radius of the hand.

For finding the center of hand, it has a problem to find the center accurately. If the camera showed a hand with wrist, then the center will move a little towards the wrist because the color of the wrist is the same as hand color. Therefore, it causes that algorithm system to fail because if the center is moved down, then the radius of the hand can be smaller than the actual size. Furthermore, the circle will move down and become smaller so when a user crooks his or her hand then the finding finger tips algorithm can also fail because a thumb can be placed outside of the circle every time.

8. Conclusion

We developed a system to control the mouse cursor using a real-time camera. We implemented all mouse tasks such as left and right clicking, double clicking, and scrolling. This system is based on computer vision algorithms and can do all mouse tasks. However, it is difficult to get stable results because of the variety of lighting and skin colors of human races. Most vision algorithms have illumination issues. From the results, we can expect that if the vision algorithms can work in all environments then our system will work more efficiently. This system could be useful in presentations and to reduce work space. In the future, we plan to add more features such as enlarging and shrinking windows, closing window, etc. by using the palm and multiple fingers.

9. References

[1] *Computer vision based mouse*, A. Erdem, E. Yardimci, Y. Atalay, V. Cetin, A. E. Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASS). IEEE International Conference

[2] *Virtual mouse vision based interface, Robertson* P., Laddaga R., Van Kleek M. 2004.

[3] Vision based Men-Machine Interaction http://www.ceng.metu.edu.tr/~vbi/

[4] Chu-Feng Lien, *Portable Vision-Based HCI - A Real-time Hand Mouse System on Handheld Devices*.

[5] Hailing Zhou, Lijun Xie, Xuliang Fang, *Visual Mouse: SIFT Detection and PCA Recognition,* cisw, pp.263-266, 2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007), 2007

[6] James M. Rehg, Takeo Kanade, *DigitEyes: Vision-Based Hand Tracking for Human-Computer Interactionin*, Proc. of the IEEE Workshop on Motion of Non-Rigid and Articulated Objects, Austin, Texas, November 1994, pages 16-22.

[7] Gary Bradski, Adrian Kaehler, *Learning OpenCV*, O'Reilly Media, 2008.

[8] Asanterabi Malima, Erol Ozgur, and Mujdat Cetin, *A Fast Algorithm for Vision-Based Hand Gesture Recognition for Robot Control*

[9] J. Serra, *Image Analysis and Mathematical Morphology*, New York: Academic Press, 1983.

[10] K. Homma and E.-I. Takenaka, "An image processing method for feature extraction of space-occupying lesions," *journal of Nuclear Medicine* 26 (1985): 1472-1477