IMPLEMENTATION OF VIRTUAL MOUSE BASED ON MACHINE VISION

LI WENSHENG[1], DENG CHUNJIAN[2], LV YI[3]

(University of Electronic Science and Technology of China, Zhongshan Institute, Zhongshan 528402, China)
lws7166@126.com

Abstract:

A method to implement virtual mouse through machine vision are proposed. Firstly we present an efficient algorithm based on color to track the movement of fingertips in real time, then we present a set of messages (including elementary fingertip message and simulated mouse message) that are generated according to the result of fingertip tracking, These messages can be used to control the window based applications. We also presents a framework to help programmers realize human-computer interaction based on virtual mouse. The performance of vitual mouse shows a great promise that we will be able to use this methodology to control computers and achieve a more natural human-computer interaction.

Key words:

Machine vision; Fingertip tracking; Virtual Mouse; Human-Computer Interaction; Framework

1. Introduction

Hand gesture recognition based on machine vision can provide a more natural method of human-computer interaction compared with keyboard and mouse, and make it more convenient and intuitive for people to use the computer[1].

Rehg and Kanade presented "DigitEyes", a model-based hand tracking system that uses two cameras to recover the state of a hand[3]. Kjeldsen and Kender suggested a simple visual gesture system to manipulate graphic user interface of a computer[4]. Andrew et al. presented a method for tracking the 3D position of a finger, using a single camera placed several meters away from the user[5]. Most of these approaches are highly complicated and time-costing. Also, they are sensitive to hand variations, scales, rotations and illuminations.

Color-based target tracking methods are normally based on various target-color filters and region segmentation techniques[6-7]. These methods have many advantages. First, processing color is much more computationally inexpensive. Second, color models are scale/orientation/rotation invariant. These properties are particularly important for a real-time hand/face tracking system. Bradski's CamShift[2] is representative of a group of algorithms that exploit the color information to locate and subsequently track a human hand/face in a video sequence.

In this work, we put forward a method to implement virtual mouse based on fingertip tracking that simulate mouse operation through hand gesture and make it possible for user to control the window based applications by hand gesture. First, we present an efficient algorithm based on color to track the movement of multiple fingertips in real time. Second, we solve the problem due to the difference of resolution between the input image from the camera and the monitor in order to make the motion of the mouse pointer on the monitor smooth. Third we present a set of messages that are generated according to the result of fingertip tracking. These messages can be used to control window-based applications. We also present a framework to help programmers realize human-computer interaction based on the proposed virtual mouse.

2. the application framework based on virtual mouse

The application framework based on virtual mouse is shown as figure 1.

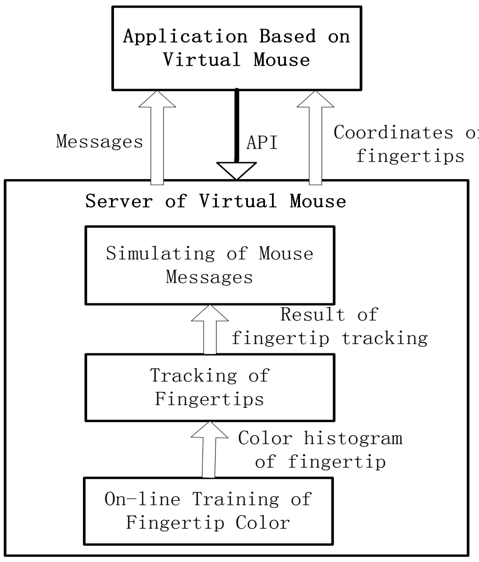
Application Based on Virtual Mouse
Messages
API
Coordinates of fingertips
Server of Virtual Mouse
Simulating of Mouse Messages
Result of fingertip tracking
Tracking of Fingertips
Color histogram of fingertip
On-line Training of Fingertip Color

Figure. 1. The application framework based on virtual mouse

The framework mainly consists of two parts: The first is the server of virtual mouse, the second is virtual mouse based applications (the client). The server is responsible for the detection and tracking of fingertips, constructing appropriate message and sending such messages to the client. The client responses to the messages and obtains the coordinates of multiple fingertips by calling the API function provided by the server, then does the appropriate message processing to achieve the control of the application.

The server is divided into three parts: adaptive on-line training of fingertip color, tracking of fingertips, and mouse message simulating. The first part obtains the color histogram of fingertip through a small sample area of fingertip. The second part is fingertip detection and tracking. It first calculates the probability density of the video image through back-projection on the color histogram and gets the binary image by doing binarization, then detects the targets of fingertip through edge detection, and finally it locates the centroid of fingertip through Mean Shift Algorithm. The third part constructs the messages according to the results of fingertip tracking and sending them to the client.

3. Fingertip Tracking

In order to simulate the mouse operation, the method of fingertip tracking must be computationally inexpensive and can execute in real time. Our method to detect and track the fingertip is based on color and the user is required to wear fingertip-knots of given color.

3.1. Adaptive Online Training of Fingertip Color

We use an on-line training method to learn the color of fingertips. The user is asked to place his or her fingertip so that the red rectangular box can stay in the fingertip, and the system traverses all the pixels in the rectangular box, counts the number of occurrences of various colors, and get the color histogram by normalization. This color histogram shows directly the color distribution of fingertips in color space.

As we know, in the RGB color space, a triple of [R,G, B] represents not only color but also brightness, so RGB may be greatly affected by illumination and there is great correlativity between the three primary colors. What is more, the Euclidean distance between two colors is not linear to the color distance, so it is hard for RGB to measure the color difference accurately.

Compared with RGB, HSV separated pure colors from brightness. In order to reduce the effect of illumination and similar background, we have employed H (hue) and S (saturation) values of HSV for learning the probability distribution of fingertip colors. A 2D color histogram (2D look up table) is computed using H and S as the result of on-line training of fingertip colors. The 2D color histogram can be used as the basis of target fingertip detection and tracking.

3.2 Detection and Tracking of Fingertip Regions

We can detect and tack the fingertip regions as follows:

1) Transform the video image from RGB color space to HSV color space.

2) Build the probabilistic distribution image $I(x, y)$ through back-project calculation on the color histogram which is obtained in 3.1.

For each pixel in the input HSV image, we can get the probability of the pixel as object pixel by searching the color histogram. This process is called back-project

3) Obtain the binary image $B(x, y)$ by doing binarization with $I(x, y)$ as in equation (1) .

$$B(x, y) = \begin{cases} 0 & I(x, y) < p \\ 1 & I(x, y) \geq p \end{cases} \qquad (1)$$

where p is a given threshold value.

It is clear that $B(x, y)$ determines whether a pixel belongs to a fingertip and gives ROI (Region of Interest) of detecting fingertips. The computation time is saved a lot by confining necessary computation to ROI. What is more, thresholding can reduce the effect due to noise.

4) Remove noises and holes in the binary image with morphological operations

Sometimes, there are possibly a few noises in the binary image, which is caused by incorrectly detecting some colors in the background as target. We can use open operation of morphology as in equation (2) to deal with this problem.

$$B \circ S = (B \otimes S) \oplus S \qquad (2)$$

We also need to remove the holes in the binary image because it is impossible that there is a hole in a fingertip. We can use close operation of morphology as in equation (3) to remove holes.

$$B \bullet S = (B \oplus S) \otimes S \quad (3)$$

5) Determine the edge of target regions through edge detection, and label each connected region with a minimum bounding rectangle. These rectangles are called search windows of fingertips.

Since $B(x, y)$ is a binary image, we can determine whether a pixel is an edge point with its 8-neighber's value instead of the derivation of the image. If a pixel has value of 1 and has a neighbor whose value is 0, the pixel can be seen as an edge point. This will greatly enhance the speed of edge detection.

3.3 Tracking of Fingertip's centroid

Now we have got the search windows of fingertips. It is necessary to compute the cnetroid positions (mean locations) of fingertips so as to simulate mouse pointer.

Let W denotes the search window of a fingertip, we can calculate zero order moment M_{00}, first order moment M_{10}, M_{01} and centroid (x_c, y_c) in the search window as follows.

$$M_{00} = \sum_x \sum_y B(x,y), \quad M_{10} = \sum_x \sum_y xB(x,y), \quad M_{01} = \sum_x \sum_y yB(x,y) \quad (4)$$

$$x_c = M_{10} / M_{00}, \quad y_c = M_{01} / M_{00} \quad (5)$$

4. Display of a Mouse Pointer on the Monitor

Even if the fingertip is detected correctly, we might not get a natural motion of a mouse pointer due to the following problems.

First, if the location of the fingertip in each frame is converted to the monitor coordinates directly, the difference of the resolution between the input image from the camera and the monitor makes it very difficult to position the mouse pointer accurately. We give a map as in equation (6) to map the fingertip location in the input image to the location on the monitor:

$$x_s = x_c / w_c * w_s, \quad y_s = y_c / h_c * h_s \quad (6)$$

where (x_c, y_c) denotes the fingertip location in the input image, (x_s, y_s) denotes the mouse pointer position in the monitor, $w_c \times h_c$ denotes the resolution of the input image and $w_s \times h_s$ denotes the resolution of the monitor.

Second, because of the limitation on the tracking rate, we can merely get discontinuous coordinates of fingertip locations. Accordingly, if these coordinates of the fingertips are used for the display on the monitor without any smoothing, the mouse pointer is going to jump around in a very unnatural fashion. Moreover, the jitter can occur even though the fingertip remains still. A small movement of the fingertip in the input image can cause a significantly large movement of the mouse pointer on the monitor. To circumvent this problem, it is necessary to smooth the trajectory of the mouse pointer.

Our observations shows that we need to achieve the tracking rate of more than 20 fps so that a user can perceive a natural movement of the mouse pointer. If a system cannot support this rate, we have to generate locations of the mouse pointer on the monitor during the time between two image frames so that the resulting path of the mouse pointer may be smooth. This can be achieved through interpolation method. It is shown in figure 2 that the trajectory of the mouse pointer is smoothed by interpolation.

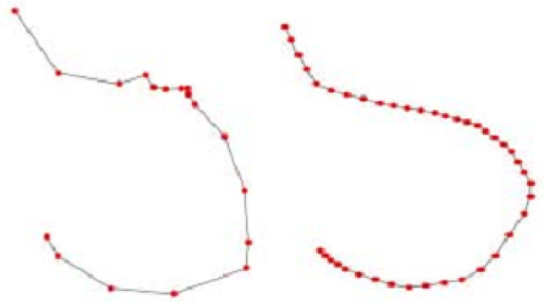

Figure 2. (a) Raw tracking results, (b) Smoothed version of the raw tracking results

5. Control the Application Through Virtual Mouse

Now we have got the result of fingertip tracking and mapped the position of fingertip to the position of virtual mouse pointer in the monitor. We still need construct appropriate messages on the basis of tracking results and send them to the application so as to control it.

5.1 Definition of Message

Message of virtual mouse can be divided into two categories: elementary fingertip messages and simulated mouse messages. Elementary fingertip message is mainly used to notice an application that a fingertip is entering, or leaving the shooting area, or a fingertip is moving in the shooting area. Table 1 lists all of the elementary fingertip messages.

Table 1 List of elementary fingertip messages

Name	Trigger
SSSM_ENTER	Triggered when a fingertip enters the shooting area
SSSM_MOVE	Triggered when detecting the movement of a fingertip
SSSM_LEAVE	Triggered when a fingertip leave the shooting area

Traditional windows applications is generally operated and controlled by mouse, so it is need to simulate mouse operation by hand gesture. We simulate the mouse operation by using two fingertips and recognize mouse action by BP neural network. Table 2 lists all of the simulated mouse messages.

Table 2 List of simulated mouse messages

Mouse Message	Trigger
Left Button Down	trigger when two fingertips are approaching, and the distance is less than a certain threshold
Left Button Up	trigger when two fingertips are further away, and the distance is more than a certain threshold
Move	trigger When two fingertips move in parallel, we use the midpoint of two fingertips as mouse pointer
Drag and Drop	Combination of Left button Down, Move, Left Button Up
Double-Click	Combination of two consecutive operations of Left Button Down and Left Button Up

5.2The Communication between the Server and the Client Application

Since the server and the client are different processes, so we have to consider the communication between them. In windows, a process can not access the address space of another, so the client application can not obtain directly the information of fingertips by accessing the server's memory. In order to solve the problem of data sharing, Windows provides a method called “memory mirror file”. However, it is not a good way to share data through memory mirror file. Since it interference the logic of the client and expose the internal format of memory mirror file, it may cause safety problems. To make the process of communication between them transparent, we use "stub" which is a dynamic link library (dll) to deal with the data exchange and communications between the server and the client application.

Tthe client registers its window handle by calling SSS_RegistWindow (HWND hWnd) in the stub, which will add its window handle into the List of Window Handles in memory mirror file. After registration, the window of the client can receive messages from the server. Client can call SSS_UnregistWindow (HWND hWnd) in the stub to cancel registration, which will remove the window handle of the client from the List of Window Handles. After unregistration, the window of the client will not receive the messages from the server.

When the server detects a fingertips entering or moveing in the shooting area, it will get the information of fingertip (including fingertip coordinates, index and time) and save it in the List of Fingertip Information in the memory mirror file. Next, it will construct the appropriate message and call PostMessage to send the message to the window which handle is in the List of Window Handles.

When the client receives the message from the server, it will deal as follows: For simulated mouse message, it will respond directly; For elementary fingertip messages, it will call the SSS_GetPointsInfo in the stub to get information of all fingertips and conduct appropriate treatment. Figure 3 is the sequence diagram of message processing.

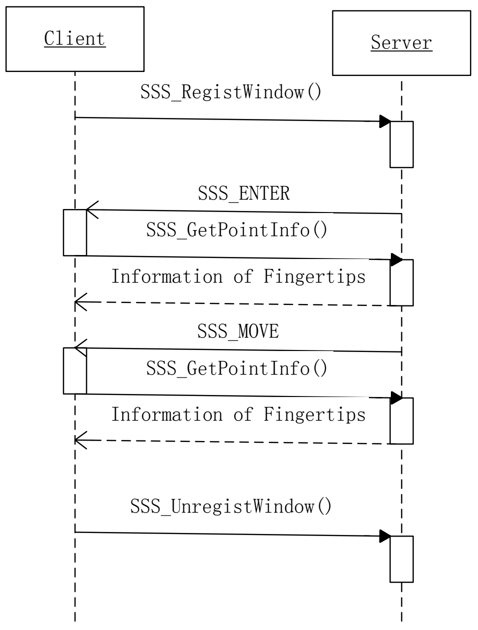

Figure 3. Sequence diagram of message processing

6. Conclusions

In this paper we have proposed a method to implement virtual mouse and a framework of application based on virtual mouse. The proposed method for fingertip tracking shows a good performance under cluttered background and variations of local illumination by online training of fingertip colors and optimal determination of ROI. In addition, a natural motion of the mouse pointer is achieved by smoothing the path of the fingertip location using physical model of the mouse pointer. We have developed a prototype of virtual mouse through DirectShow+OpenCV. The experimental results show that the proposed method can be applied to manipulate window based applications.

Acknowledgment

The Project was funded by Natural Science Foundation of Guangdong Province, China

(8152840301000009), and supported by Science and Technology Planning Project of Guangdong Province, China (2009B030803031)

References

[1] Dong Shihai, "Progress and Challenge of Human-Computer Interaction" [J], JOURNAL OF COMPUTER-AIDED DESIGN & COMPUTER GRAPHICS, 2004,16(1)

[2] Gary R. Bradski, "Computer Vision Face Tracking For Use in a Perceptual User Interface", Intel Technology Journal Q2, 1998.

[3] J. Rehg and T. Kanade, "Visual Analysis of High DOF Articulated Object with Application to Hand Tracking", CMU Tech. Report CMU-CS-95-138, Carnegie Mellon University, April, 1993.

[4] R. Kjeldsen and J. Kender, "Towards the use of Gesture in Traditional User Interfaces", International Conference on Automatic Face and Gesture Recognition, pp. 151–156, 1996.

[5] Andrew Wu, Mubarak Shah and N. da Vitoria Lobo, " A Virtual 3D Blackboard: 3D Finger Tracking using a Single Camera", Proceedings of the Fourth International Conference on Atomatic Face and Gesture Reconition, pp. 536–543, 2000.

[6] ZHANG Hongzhi，ZHANG Jinhuan，YUE Hui，HUANG Shilin, Object tracking algorithm based on CamShift, Computer Engineering and Design, 2006，27(11)： 2012-2014.

[7] XU Kun,HE Yuyao,WANG Weiya, Object tracking algorithm with adaptive color space based on CamShift, Journal of Computer Applications, 2009，29(3):757-760.

[8] Wang Xiuhui，Bao Hujun, Gesture Recognition Based on Adaptive Genetic Algorithm, JOURNAL OF COMPUTER-AIDED DESIGN & COMPUTER GRAPHICS, 2007，19(8)：1056-1062.

[9] SUN Lijuan，ZHANG Licai，GUO Cailong, Technologies of Hand Gesture Recognition Based on Vision, COMPUTER TECHNOLOGY AND DEVELOPMENT, 2008,18(10):214-221.

[10] JIANPING LI,Robust face recognition with partial distortion and occlusion from small number of samples per class, School of Computer Science and Engineering, University of Electronic Science and Technology of China,ICACIA 2008