AN EFFICIENT GESTURE RECOGNITION TOOLKIT

Abhijit Kane
Department of Computer Science
Birla Institute of Technology and Science, Pilani
Jawahar Nagar, Shamirpet (M)
Hyderabad, Andhra Pradesh 500078, India
abhijitkane@gmail.com
http://sites.google.com/site/abhijitkane

Abstract

The rapid growth of computing has made effective human-computer interaction essential. It is important for the growing number of computer users whose schedules will not allow the elaborate training and experience that was once necessary to take advantage of computing. As computing affects more aspects of our lives, the need for usable systems becomes even more important. An attempt has been made to develop a few basic tools for Human-Computer Interaction (HCI), and show how these can be used a basis for developing more advanced applications. The major component used in the project is OpenCV, which is an open-source image processing library developed by Intel. Basic interaction with the computer has been achieved using hand-held aids like colored thimbles. The ability to recognize the position of the fingers without these aids has also been explored. These simple techniques illustrate the steps required to achieve primitive interaction. These can be extended to achieve better interaction, both in terms of accuracy and scope.

Keywords: image-processing; computer-vision; hci

1. Introduction

In recent years, the paradigm has started to shift from the traditional mouse-and-keyboard interface to more natural ways of computer interaction. These include, but are not limited to: hand and body gestures, sound and facial expression. Apple's Siri is now widely-considered to be the cornerstone of speech recognition technology. Such a concept would have been seen only in science-fiction movies even five or ten years ago. The gaming consoles that exist today: Sony's PS3, Microsoft's Xbox 360 and Nintendo's Wii, all have some sort of visual recognition capabilities. These devices come with multiple cameras that are used to ascertain the movements of the user. The Kinect sensor that ships with the latest Xbox also has infrared sensors that can easily isolate the players' body. These are obviously very advanced implementations on very fast GPUs. But nonetheless, the concept is the same. The input image is the only input, and from it, the objective is to extract information about the user's position, gestures etc.

Of course, such processing power may not always be available. The aim of this project was to explore simple, innovative ways to provide a similar user-experience, while making sure that the algorithms run on less powerful platforms. These results can also be used for applications other than games. User interaction with UI elements like menus and icons is also possible.

This project made extensive use of OpenCV, an open-source image processing library developed by Intel. OpenCV provides a number of useful data structures, like matrices (to store image data) and contours. It also provides implementations of many common image processing algorithms, including edge-detection, thresholding and feature detection. OpenCV can also be run in an Android application.

The basic steps involved in the operation are - preprocessing (applying suitable filters on the image to remove noise, and improve chances of accurate detection), thresholding (converting the image to a monochrome image based on certain properties), position analysis (filtering out the regions where a particular object is present) and finally, using this information to interact with the computer.

2. Gesture Recognition

2.1. *Image acquisition*

The primary method for acquiring visual data is the computer's webcam. If deployed on a mobile device, the mobile phone's camera may be used. OpenCV supports direct access to the camera, so no other script is required to access the video feed. OpenCV has all the necessary structures to control the video feed. Once access to the video stream is established, we can capture individual frames as image data.

2.2. *Processing*

Once the image has been acquired, the general process is as follows: pre-processing, thresholding (making it a 2-bit image), feature extraction, and post-processing. The post-processing phase may include correlation with information extracted from previous frames.

2.2.1 *Pre-processing*

The first pre-processing step is to remove minor variations in pixel intensity by taking the weighted average of each pixel from the last two frames:

$$x'_t = (\alpha x_t) + ((1-\alpha)x_{t-1} \quad (1)$$

x'_t is the value of the pixel that is used in future calculations. x_t is the observed value of the pixel. x_{t-1} is the observed value of the same pixel in the previous frame. A value of $\alpha = 0.7$ was found to be suitable.

The resulting image is then put through a bank of filters: the median filter with kernel size = 3, and a Gaussian filter with kernel size 3. The median filter removes salt-and-pepper noise, and the Gaussian filter blurs the image, so that local intensity variations are mitigated to some extent.

Blurring can be a simple operation like taking the average value of pixels, or something more complicated like convolving the image with a Gaussian kernel. The following equation is used to generate the Gaussian kernel:

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2-y^2)}{2\sigma^2}} \quad (2)$$

Here, σ is the standard deviation of the Gaussian distribution for the kernel of size 3x3.

2.2.2 *Thresholding*

A colour space is a way of representing colours using a set of numbers. A colour in the RGB colour space may be defined by three numbers (R, G and B) specifying the strength of the red, green and blue components. A fourth parameter, alpha, may also be added to specify the opacity of the image. A colour in the HSV space is defined by specifying Hue (perceived colour), Saturation (shade), and Value (brightness). The HSV and RGB models to specify colour are analogous to the Cartesian and Polar co-ordinate systems to specify the location of a point in 3D space.

Thresholding converts the colour image to a monochrome image. It was found that thresholding for this purpose gave better results when the HSV space was used. Only those pixels whose Hue value fell in a certain range were made white, and the rest of the pixels were made black. The results of the HSV thresholding operation were as follows:

Fig. 1. (L to R) The input frame, thresholded image for blue, thresholded image for green

HSV thresholding was used as it produced better results compared to the RGB space. The thresholding accuracies were (percentage of frames correctly thresholded):

Table 1. RGB and HSV thresholding results

Thresholding Method / Color	Blue	Green
RGB	74%	76%
HSV	89%	85%

2.2.3 *Feature extraction*

The thresholded images have some erroneous white areas. These need to be removed before the positions of the blue and green thimbles can be extracted. The erosion operation is useful for this purpose.

Erosion moves a window (5x5 in this project) across the image, and replaces the center pixel with the darkest pixel in the window. Thus, a white pixel can remain white if and only if its 24 closest neighbors are also white. This operation removes all small white areas.

After erosion, the resultant images are shown below:

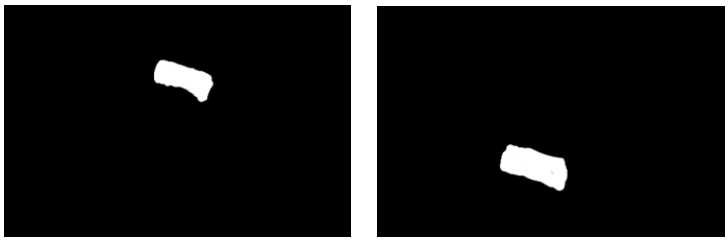

Fig. 2. The blue (left) and green (right) thresholds after the erosion operation

After this operation, only the thimbles remain white. The white areas can now be used to compute the positions of the blue and green thimbles.

To find the effective centers of the blue and green thimbles, the mean of the positions of all the white pixels in the thresholded image was taken.

2.2.4 *Post-processing*

To remove minor temporal variations in the positions of the green and blue regions, a weighted-average technique was used. This worked to mitigate it effects of the hand shaking.

$$\text{current position} = (0.8 * \text{observed current position}) + (0.2 * \text{previous position}) \quad (3)$$

This ensures that no sudden changes are observed in the positions of the thimbles.

These values are updated in real time, since the entire process of pre-processing, thresholding and feature extraction is done on each frame. The positions of the blue and green thimbles can be used to calculate if the virtual 'ball' should be picked up, moved, or dropped.

In this example, the ball was deemed to be 'picked up', when the two thimbles came relatively close to each other. Once the ball was picked up, and the proximity of the thimbles was maintained, the ball could be moved around the screen by moving the thimbles. The result is shown below:

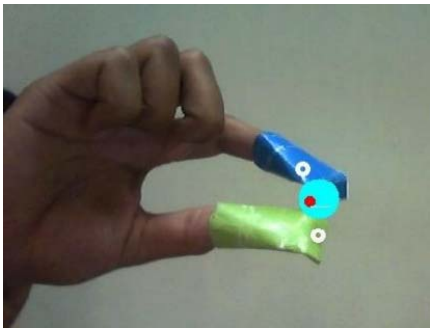

Fig. 3. The result of the entire process

In this way, intuitive human motion can be translated into movements of objects on the computer.

3. Conclusion

This paper highlights a few ways in which video from a camera can be processed in order to derive information about the gestures of the user. It emphasizes speed and efficiency of the algorithm, so that the entire process can run on mobile devices which have smaller computing capabilities.

The basic aim of the algorithm is to accurately identify the position of certain objects in the field of view. Once this has been done, the positional information can easily be used to create applications like two-player 'pong'-type games, recreating an experience like the Wii offers, without the need for a powerful processor

References

[1] Heijmans, Henk J. (1994): Morphological Image Operators. Advances in Electronics and Electron Physics.
[2] Oka K. (2002): Real-time fingertip tracking and gesture recognition. Computer Graphics and Applications.
[3] Soille, P. (2008). Morphological Image Analysis: Principles and Applications. Springer-Verlag New York INC, Syracuse.
[4] Wu Y, Huang, T. (1999): Vision-based gesture recognition: A Review. Lecture Notes in Computer Science, pp. 103-115.