Hand Posture Recognition with Application to Robot Control

N. R. Raajan, S.Raghuraman, T.Vignesh
Department of ECE, SEEE, SASTRA University
Thanjavur, Tamil Nadu, India
nrraajan@ece.sastra.edu, raghuraman1291@gmail.com, vignesh.ykciv@gmail.com.

***Abstract*— In recent years, several researches are being done to improve the means by which humans interact with machines. Having developed a variety of input devices, we still are not completely comfortable with the present human–machine interaction processes. This stirred up the efforts undertaken to make the machines adapt to the human's natural means of communication which are speech and body language. The objective of this paper is to implement a real-time vision system which offers better comfort to humans while interacting with machines. In our paper, we showed a simple but efficient method to implement a hand posture recognition system and by means of which we control a bot wirelessly through Bluetooth. The simplicity of our method enables fast recognition of the hand postures shown and therefore achieves the real-time continuous control over the bot.**

Keyword – machine interaction, real-time vision system, hand posture, Bluetooth.

I. INTRODUCTION

Among humans, a significant way of communication is body language. It adds emphasis to speech and sometimes delivers the full information by itself. Therefore, hand posture recognition systems using image processing can be used for enhancing human-machine interaction. One of the important aims of Hand Posture Recognition is to distinguish different hand postures and correctly sort them out as precisely as possible. In our method, we identified hand postures by counting the number of fingers shown and based on the number of fingers, we controlled the movement of a bot. The bot is connected wirelessly to the system through Bluetooth. The sensor device used for capturing the image frames is a USB web cam. The frames from the camera are preprocessed using filters to remove noise and then background subtraction operation is performed to extract the hand region in binary form. We assume that the only moving foreground object present is hand. The hand contour is extracted and the number of fingers is obtained using convex hull and defect point features. We use OpenCV libraries in Dev C++ IDE for image processing. The following sections describe the processes used in our method in detail.

II. RELATED WORKS

A lot of methods have been proposed for hand gesture recognition using image processing and computer vision algorithms . There are algorithms which use markers on fingertips to detect gestures [1]. Use of markers is sometimes expensive and inconvenient to the users. [2] gives a simple algorithm for hand gesture recognition which takes less computational time. But its segmentation part is not robust leading to noise. [3] uses Hidden Markov Models to recognize the hand gestures. This requires intensive training. A robust approach is given in [4] which is scale and rotation invariant. This proposes a curvature space method which finds and uses the contours (outline) of the hand. Some of the other computer vision tools used for recognizing hand gestures are Haar-like features [5], Support Vector Machines [6] and particle filters.

III. HAND REGION EXTRACTION

A. Background Subtraction

Background subtraction is one of the most fundamental image processing operations. Before performing background subtraction, a background model needs to be prepared. The background scenes in an unconstrained environment often contain complicated moving objects such as curtains fluttering, fans turning, trees waving in the wind, e.t.c. Light intensity might also vary in such scenes depending on door-window positions and weather conditions. So the normal averaging background method would not be suitable. A good method to face this is to develop a time-series model for each pixel or a group of pixels to deal with the temporal fluctuations well.

To obtain a performance fairly close to that of adaptive background subtraction, we form a codebook to represent significant slowly varying states in the background. We compare the present value of a pixel with the previous values. If this value is near the previous value, it is considered as a small variation on that color. If the present value is not close to the previous value, then it starts a new code element and links it with the pixel. We choose HSV color space since it has a separate axis aligned with brightness. This separate axis helps us because background variation in most cases is not along the color axis, but along the brightness axis.

The following diagram clearly gives an example of the code method. As we can see, a codebook can be considered to be made up of boxes.

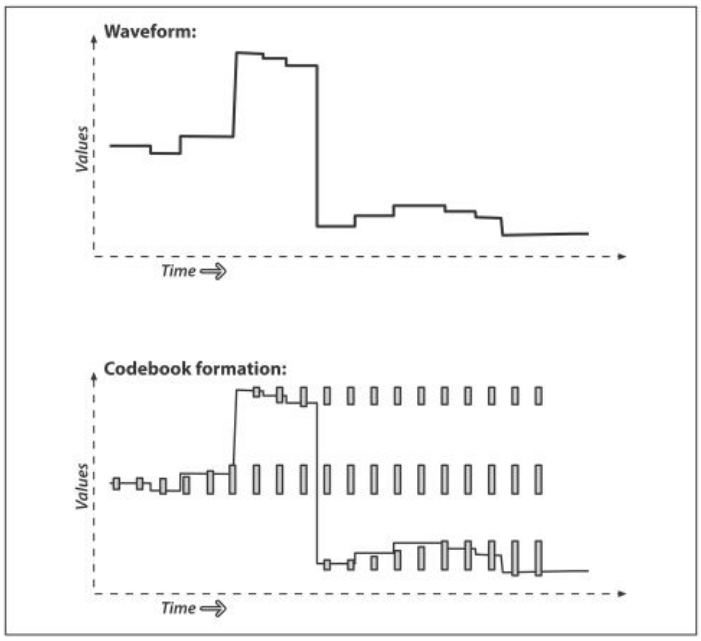


Fig. 1. Codebook Method example

Initially we take nearly 300 samples for the background model. Each pixel in the background is maintained with a codebook (structure) to make note of its variations. If the pixel changes during the sampling period, the codebook expands to include the value. In the above picture, each box is a code element. Thus each pixel will have codebook with different number of code elements.

Each code elememt has two thresholds (max and min) for each axis in the HSV color space. These thresholds will enlarge (max increasing, min decreasing) if the new samples from the background fall within the thresholds used for learning (learnHigh and learnLow). Otherwise if the samples fall outside the thresholds (max, min) and (learnHigh, learnLow), then a new code element will be created. This is how we create the background model. In the background difference mode, we define acceptance thresholds maxMod and minMod. By use of these threshold values, we can prevent creating a new code element if a pixel value is close enough to a max or a min boundary.

The pictures portray the background and the extracted foreground using codebook method

Fig. 2. Foreground with background

Fig. 3. Foreground extracted separately

B. Finding largest contour area

We have assumed that the only foreground moving object is hand. Therefore after background subtraction, we will get a noisy binary image with the extracted hand region. To remove the noises, we find the contours in the noisy binary image and then take the only contour with the largest area. Contours are the outlines of the Binary Linked Objects present in the image. This gives us a clear binary image with the hand region only and with less noise. The picture with noiseless hand region is given below.

Fig. 4 and Fig. 5. Unwanted portion in hand region

However as we can see in the picture above, there are some objects attached with the hand due to sudden unpredictable change in camera position or illumination. To reduce this, we use the above binary image as mask in the original frame. In the masked original frame, we extract the hand region using HSV color space. The Hue, Saturation and Brightness (V) values for skin color are found to be from 0 to 28, 8 to 140 and 0 to 255 respectively.

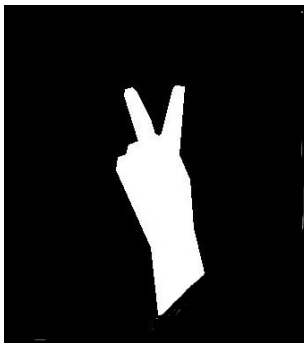


Fig. 6. Masked hand region

Fig. 7. Final binary hand image

IV. GESTURE RECOGNITION

In our method, the number of fingers shown in the frame is considered as a gesture. Given the binary image of the hand region, the number of fingers shown is calculated by means of convex hull. A convex hull of a set of M points in space is the smallest convex set that includes all these M points. The convex hull is found by means of Sklansky's algorithm [7]. Then we find the defects inside the convex hull and take note of the defects' start and end points. The start points approximately give the position of the tip of the fingers. The following figure gives an explanation of convex hull and defects.

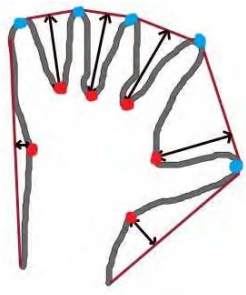


Fig. 8. Convex hull

The red line represents the convex hull. The black lines with arrow marks represent the defects. The blue points are the start of the defects and the red points are the end of the defects or depth point.

Hence the number of start points gives the number of fingers. However due to some unavoidable noise in the image, the start points and end points tend to vary. If this varies frequently, it cannot be used to control a bot.

So we define a rectangular area in the frame. The number of start points inside the rectangle is taken as the number of fingers. Care is taken to define the rectangular area in the portion of the background which is noiseless. The following picture shows our output which gives the number of fingers.

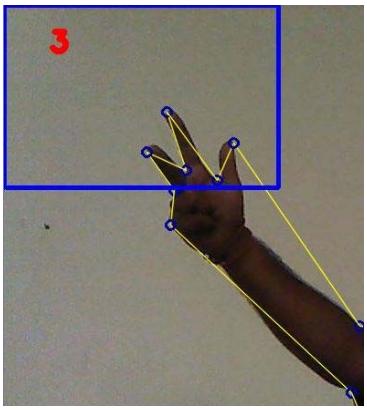


Fig. 9. Output displaying the number of fingers.

The blue rectangle is the defined rectangular area.

V. GESTURE CONTROLLED BOT

After getting the number of fingers shown, we send suitable command to bot through Bluetooth to control its movements. We used Simple Labs' BTBee as Bluetooth receiver in the bot. A Bluetooth USB Dongle is taken as the transmitter. The processor used for image processing is Intel's Core i5 at 2.4GHz. The image resolution of the frames is 640x480. The bot uses Simple Labs' Induino board for processing. Based on the command received, the Induino board is programmed to give suitable output for the four movements of the bot.

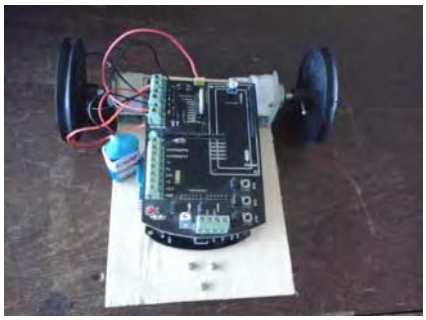
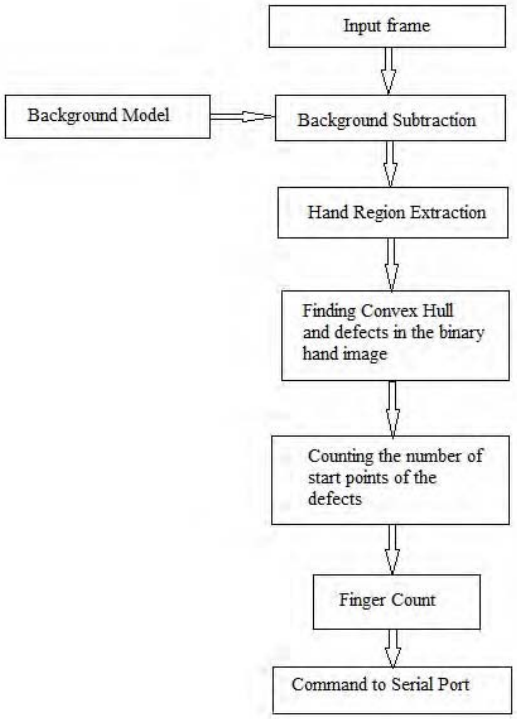


Fig. 10. Our Bot

VI. FLOWCHART



VII. CONCLUSION

We have proposed an algorithm which is simple and takes less computational time for the hand gesture recognition problem. Our algorithm segments the hand region by means of background subtraction and counts the number of fingers shown and eventually controls a bot through Bluetooth. With our algorithm, we have achieved good accuracy and efficiency. In our method, we have considered only a fixed number of gestures which are atmost five. Our method can be expanded to recognize a large number of gestures. The hand segmentation section in our algorithm still gives some noise in a cluttered background. So it has to be improved if the system is used in cluttered environments. However we should take into account that the perfect foreground segmentation in an arbitrary background is still open and under research.

ACKNOWLEDGMENT

We include our special thanks to all the researchers working in field of image processing and computer vision who in some way helped us to attain our goal. Also, We wish to express our gratefulness to the faculties in our university who shared their views and gave suggestions to make our project a success.

REFERENCES

[1] J. Davis and M. Shah "Visual Gesture Recognition", IEEE Proc.-Vis. Image Signal Process., Vol. 141, No.2, April 1994.
[2] Asanterabi Malima, Erol Ozgur, and Miijdat Cetin “A Fast Algorithm for Vision-Based Hand Gesture Recognition for Robot Control”, IEEE International conference on Signal Processing and Communications Applications, 2000.
[3] Sebastian Marcel, Oliver Bernier, Jean Emmanuel Viallet and Daniel Collobert, “ Hand Gesture Recognition using Input – Output Hidden Markov Models”, IEEE Proc. on Automatic Face and Gesture Recognition, pp. 456 - 461 2000.
[4] C.-C. Chang, I.-Y Chen, and Y.-S. Huang, "Hand Pose Recognition Using Curvature Scale Space", IEEE International Conference on Pattern Recognition, 2002..
[5] Qing Chen, Nicolas D. Georganas and Emil M. Petriu, “Hand Gesture Recognition Using Haar-Like Features and a Stochastic Context-Free Grammar”, IEEE Transactions On Instrumentation And Measurement, Vol. 57, No. 8, August 2008.
[6] Nasser H. Dardas and Nicolas D. Georganas, Real-Time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques, IEEE Transactions On Instrumentation And Measurement, Vol. 60, No. 11, November 2011.
[7] J. Sklansky, “Measuring concavity on a rectangular mosaic”, IEEE Transactions On Computers, Vol. C-21, No. 12, December 1972
[8] T.S. Huang and V.I. Pavlovic, “Hand Gesture Modeling,Analysis and Synthesis,” in International Workshop on Automatic Face and Gesture Recognition, pp. 73-79, 1995.
[9] Gary Rost Bradski and Adrian Kaehler, *Learning openCV*, 1st ed., O'Reilly Media Inc., September 2008.