IISRC International Conference on Industrial Scientific Research Engineering
International Industrial Scientific Research Corporation
www.worldconferences.org

Tracking the Fingertip Movements of the Users Index Finger Using Raspberry Pi

CHEEMALAMARRI REVATHI
Assistant Professor, Dept of ECE, Sagar Institute of Technology, Chevella, Hyderabad, India.

Abstract: Gesture Recognition is a technology which is used to identify human gestures with the help of mathematical algorithms. Gesture recognition recognizes the hand, tracks the hand movements & also provides information about hand position orientation and flux of the fingers. The color markers are placed at the tip of the user fingers. This helps the webcam to identify the movement of hand and the gesture recognition. The drawing application allows the user you to draw on any surface by tracking the fingertip movements of the user's index finger. The pictures that are drawn by the user can be stored and replaced on any other surface. The user can also shuffle through various pictures and drawing by using the hand gesture movements.

Keywords: Raspberry Pi, Python, OpenCV.

I. INTRODUCTION

With the rapid development of computer technology, contemporary human-computer interaction (HCI) devices/ techniques have become indispensable in individuals' daily lives. HCI devices/techniques have also dramatically altered our living habits with computers, consumer electronics, and mobile devices. The ease with which an HCI device or technique can be understood and operated by users has become one of the major considerations when selecting such a device. Therefore, it is necessary for researchers to develop advanced and user-friendly HCI technologies which are able to effortlessly translate users' intentions into corresponding commands without requiring users to learn or accommodate to commands without requiring users to learn or accommodate to the device.

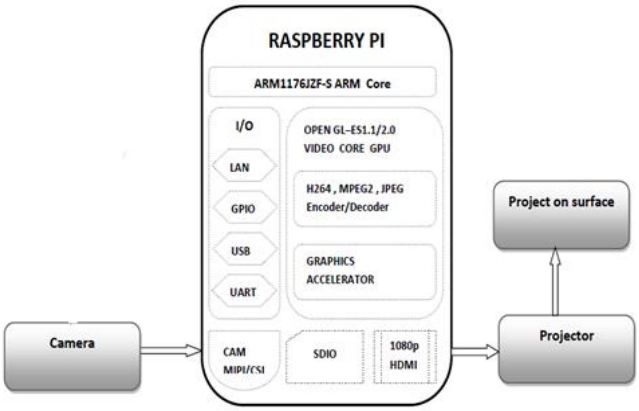

Figure1.

Technologies are being developed which are able to intuitively express users' intentions, such as handwriting, gestures, and human body language, to naturally control HCI devices. These technologies have many applications in the fields of remote control, virtual reality, sign language, signature authentication, sport science, health care, and medical rehabilitation. Current trends in human machine interface have developed quit rapid in consumer devices such as multi touch of iPhone, motion sensing devices of Wi-Fi, and etc. However, the hand tracking systems as user input have been limited due to the technical constraints and prices. In this paper, we introduce a hand motion capture system using a single camera that enable to track 2D hand pose and interact with applications in real-time. Our goal is aimed to design the system that uses a consumer grade webcam as a low-cost approach while having to deal with the robustness, precision, and real-time constraints. There are many research try to solve the same problem. A webcam generally sits on top of the computer screen looking down towards the marker of user's hands. Our related work is presented in the following sections: system overview, design, evaluation, and future work.

II. OVERVIEW

The project mainly focuses on the basis to implement the object detection and tracking based on its colour, which is a visual based project i.e., the input to the project will be the video/image data which is continuously captured with the help of a webcam which is interfaced to the Raspberry Pi. It will detect the object and it tracks that object by moving the camera in the direction of the detected object. The visual data captured by the webcam is processed in the Raspberry Pi and the object is detected based on the colour or shape and if the object is detected, the servo motor is rotated in such a way that wherever the object moves, the camera will be pointing to that object. Here, the servos are controlled by the help of a Microcontroller board called Aurdino board through its PWM pins. We can control the angle of servo rotations by the aurdino board i.e., by varying the pulse widths. The objective is to detect an object based on colour and the make use of open source hardware, hence Raspberry Pi processor board is the best option for an individual interested in low cost Arm processor. It has many inbuilt features and many ports which makes the used to experience the power of using a processor. The board comes with USB ports to which Camera, keyboard and mouse, Wi-Fi dongle can be connected which gives the feeling of working on a system.

II. RELATED WORK

A. Trackers Using Color Information

As a fundamental problem in vision, visual tracking has been drawing research attention for decades. A comprehensive review of the topic can be found. Since our focus is on integrating color information in tracking, we review only previous color trackers due to space limitation. Table I lists the abbreviations of trackers discussed in this paper. A notable early work on color tracking is the color particle filter introduced, which calculates the likelihood of each particle by comparing its color histogram from the HSV color space with the reference color model. In the target model and target candidates are represented by smoothed color histograms quantized from the RGB color space, and mean shift is used to minimize the distance between the discrete distributions of the target model and target candidates. In RGB color distribution was used to describe the target model and candidates, and the target object was located by minimizing the Kullback Leibler distance between the color distributions of the target model and candidates with the help of a trust-region method.

VTD integrates basic trackers derived from the combination of different basic observation and motion models, and four basic observation models, which use hue, saturation, intensity and edge templates as features respectively, are adopted. LOT measures the similarity between a candidate and the target using locally orderless matching, and HSV color space is used to describe the appearance of each pixel. MEEM uses features extracted in the LAB color space. In the most recent work, CSK is extended with color names , and to speed up, the dimension of the original color names is reduced with an adaptive dimensionality reduction technique. There are also trackers that take color input, but do not explicitly exploit the use of color information. Despite previous arts, there is a lack of a systematic study and understanding of how color information can be used to improve visual tracking. Our work aims to fill the gap by thoroughly investigating the behavior of numerous state-of the-art visual trackers with various color representations.

B. Color Information in Other Vision Tasks:

Not surprisingly, the discriminative power of color information has been systematically investigated for various vision topics, such as object recognition, human action recognition, object detection, etc. While being highly motivated by these pioneering works and borrowing some ideas from them, our work however focuses on visual tracking. To the best of our knowledge, this is the first comprehensive study on encoding color information for visual tracking. In fact, as shown in our experiments, many modern grayscale trackers, when augmented with color information, outperform previously proposed color trackers.

III. WORKING/ DESIGN THEORY

OpenCV usually captures images and videos in 8-bit, unsigned integer, BGR format. In other words, captured images can be considered as 3 matrices; BLUE, GREEN and RED (hence the name BGR) with integer values ranges from 0 to 255. The following image shows how a color image is represented using 3 matrices.

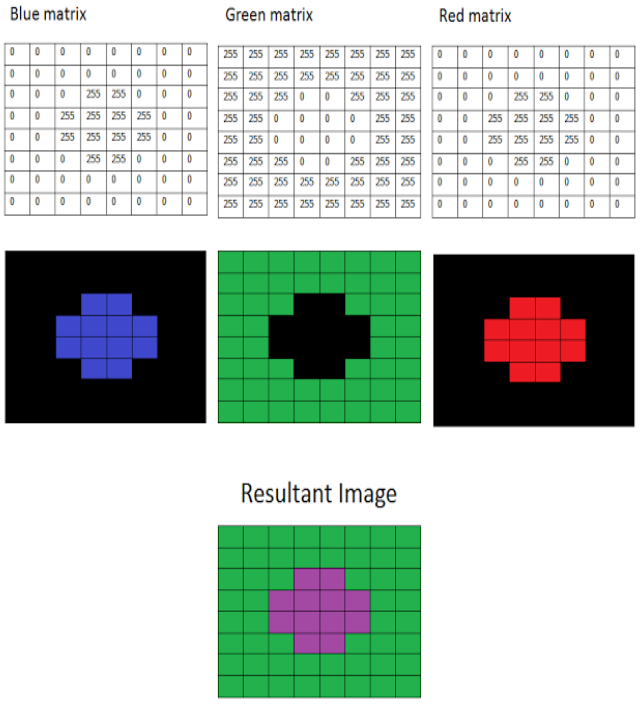

Figure2.

In the above image, each small box represents a pixel of the image. In real images, these pixels are so small that human eye cannot differentiate. Usually, one can think that BGR color space is more suitable for color based segmentation. But HSV color space is the most suitable color space for color based image segmentation. So, in the above application, I have converted the color space of original image of the video from BGR to SV image. HSV color space is also consists of 3 matrices, HUE, SATURATION and VALUE. In OpenCV, value range for HUE, SATURATION and VALUE are respectively 0-179, 0-255 and 0-255. HUE represents the color, Saturation represents the amount to which that respective color is mixed with white and VALUE represents the amount to which that respective color is mixed with black. In the above application, I have considered that the red objecthas HUE,SATURATION and VALUE in between 17 0-180, 160-255, 60-255 respectively. Here the HUE is unique for that specific color distribution of that object. But SATURATION and VALUE may be vary according to the lighting condition of that environment. Hue values of basic colors:

- Orange 0-22
- Yellow 22- 38
- Green 38-75
- Blue 75-130
- Violet 130-160
- Red 160-179

These are approximate values. You have to find the exact range of HUE values according to the color of the object. I found that the range of 170-179 is perfect for the range of hue values of my object. The SATURATION and VALUE is depending on the lighting condition of the environment as well as the surface of the object. After thresholding the image, you'll see small white isolated objects here and there. It may be because of noises in the image or the actual small objects which have the same color as our main object. These unnecessary small white patches can be eliminated by applying morphological opening. Morphological opening can be achieved by a erosion, followed by the dilation with the same structuring element. Threshold image may also have small white holes in the main objects here and there. It may be because of noises in the image. These unnecessary small holes in the main object can be eliminated by applying morphological closing. Morphological closing can be achieved by a dilation, followed by the erosion with the same structuring element. Void in Range (InputArray src, InputArray lowerb, InputArray upperb, Output Array dst); Checks that each element of 'src' lies between 'lowerb' and 'upperb'. If so, that respective location of 'dst' is assigned '255', otherwise '0'. (Pixels with value 255 is shown as white whereas pixels with value 0 is shown as black)

Arguments:

- **Input Array src** - Source image
- **Input Array lowerb** - Inclusive lower boundary (If **lowerb=**Scalar(x, y, z), pixels which have values lower than x, y and z for HUE, SATURATION and VALUE respectively is considered as black pixels in **dst** image)
- **InputArray upperb** - Exclusive upper boundary (If it is **upperb=**Scalar(x, y, z), pixels which have values greater or equal than x, y and z for HUE, SATURATION and VALUE respectively is considered as black pixels in **dst** image)
- **OutputArray dst** - Destination image (should have the same size as the **src** image and should be 8-bit unsigned integer, CV_8U)

Void erode(InputArray src, OutputArray dst, Input Array kernel, Point anchor=Point(-1,-1), int iterations=1, int border Type=BORDER_CONSTANT, const Scalar& border Value=morphology Default Border Value())

This function erodes the source image and stores the result in the destination image. In-place processing is supported. (which means you can use the same variable for the source and destination image). If the source image is multi-channel, all channels are processed independently and the result is stored in the destination image as separate channels.

Arguments:

- **InputArray src -** Source image
- **OutputArray dst** - Destination image (should have the same size and type as the source image)
- **InputArray kernel** - Structuring element which is used to erode the source image
- **Point anchor** - Position of the anchor within the kernel. If it is Point(-1, -1), the center of the kernel is taken as the position of anchor
- **int iterations** - Number of times erosion is applied
- **int border Type** - Pixel extrapolation method in a boundary condition
- **const Scalar& border Value** – Value of the pixels in a boundary condition if **border Type = BORDER_CONSTANT**

Void dilate (InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, intborderType=BORDER_CONSTANT, co nst Scalar& border Value=morphology Default Border Value());

This function dilates the source image and stores the result in the destination image. In-place processing is supported. (which means you can use the same variable for the source and destination image). If the source image is multi-channel, all channels are processed independently and the result is stored in the destination image as separate channels.

Arguments:

- **InputArray src** - Source image
- **OutputArray dst** - Destination image (should have the same size and the type as the source image)
- **InputArray kernel** - Structuring element which is used to dilate the source image
- **Point anchor** - Position of the anchor within the kernel. If it is Point(-1, -1), the center of the kernel is taken as the position of anchor
- **int iterations** - Number of times dilation is applied
- **int border Type** - Pixel extrapolation method in a boundary condition
- **const Scalar& border Value** - Value of the pixels in a boundary condition if **border Type = BORDER_ CONSTANT**

void cvt Color(InputArray src, OutputArray dst, int code, int dstCn=0)

This function converts a source image from one color space to another. In-place processing is supported. (which means you can use the same variable for the source and destination image)

- **InputArray src** - Source image
- **OutputArray dst** - Destination image (should have the same size and the depth as the source image)
- **int code** - Color space conversion code (e.g - COLOR_BGR2HSV, COLOR_RGB2HSV, COLOR_B GR2GRAY, COLOR_BGR2YCrCb, COLOR_BGR2BG RA, etc)
- **Int dstCn** - Number of channels in the destination image. If it is 0, number of channels is derived automatically from the source image and the color conversion code.

IV. CONCLUSION

We designed and build a model that can detect the object of specified color and that works on the basis of visual data captured from a typical webcam which has a fair clarity. The algorithm is tested in the laboratory live and the success rate is 100%. The algorithm works well under all conditions and the time taken to detect and track the object.

V. REFERENCES

[1] K. Altun, B. Barshan, and O. Tunçel, "Comparative study on classifying human activities with miniature inertial and magnetic sensors," Pattern Recognit., vol. 43, no. 10, pp. 3605–3620, 2010.
[2] A. Akl, C. Feng, and S. Valaee, "A novel accelerometer-based gesture recognition system," IEEE Trans. Signal Process., vol. 59, no. 12, pp. 6197–6205, Dec. 2011.
[3] W. C. Bang, W. Chang, K. H. Kang, E. S. Choi, A. Potanin, and D. Y. Kim, "Self-contained spatial input device for wearable computers," in Proc. IEEE Int. Conf. Wearable Comput., Oct. 2003, pp. 26–34.
[4] G. Bailador, C. Sanchez-Avila, J. Guerra-Casanova, and A. de Santos Sierra, "Analysis of pattern recognition techniques for in-air signature biometrics," Pattern Recognit., vol. 44, nos. 10–11, pp. 2468–2478, 2011.
[5] C. M. N. Brigante, N. Abbate, A. Basile, A. C. Faulisi, and S. Sessa, "Towards miniaturization of a MEMS-based wearable motion capture system," IEEE Trans. Ind. Electron., vol. 58, no. 8, pp. 3234–3241, Aug. 2011.
[6] M. J. Caruso, "Application of magnetoresistive sensors in navigation systems," in Proc. SAE, 1997, pp. 15–21.
[7] R. Xu, S. Zhou, and W. J. Li, "MEMS accelerometer based nonspecificuser hand gesture recognition," IEEE Sensors J., vol. 12, no. 5, pp. 1166–1173, May 2012.
[8] Z. Dong, C. Wejinya, and W. J. Li, "An optical-tracking calibration method for MEMS-based digital writing instrument," IEEE Sensors J., vol. 10, no. 10, pp. 1543–1551, Oct. 2010.
[9] M. H. Ko, G. West, S. Venkatesh, and M. Kumar, "Using dynamic time warping for online temporal fusion in multisensor systems," Inform. Fusion, vol. 9, no. 3, pp. 370–388, 2010.
[10] Y. S. Kim, B. S. Soh, and S. G. Lee, "A new wearable input device: SCURRY," IEEE Trans. Ind. Electron., vol. 52, no. 6, pp. 1490–1499, Dec. 2005.
[11] S. Kallio, J. Kela, P. Korpipää, and J. Mäntyjärvi, "User independent gesture interaction for small handheld devices," Int. J. Pattern Recognit Artif. Intell., vol. 20, no. 4, pp. 505–524, 2006.
[12] S. Katsura and K. Ohishi, "Acquisition and analysis of finger motions by skill preservation system," IEEE Trans. Ind. Electron., vol. 54, no. 6, pp. 3353–3361, Dec. 2007.
[13] S. Kim, G. Park, S. Yim, S. Choi, and S. Choi, "Gesture-recognizing hand-held interface with vibrotactile feedback for 3D interaction," IEEE Trans. Consum. Electron., vol. 55, no. 3, pp. 1169–1177, Aug. 2009.

Author Profile:

Cheemalamarri Revathi, Assistant Professor, ECE department, Sagar Institute of technology, Chevella, Hyderabad, India.