Hand Gesture Recognition

Manish Shinde(56), Akshay Thite(65), Abhay Tiwari(66)

Computer Department Vivekanand Education Society Institute of Technology Mumbai,Maharashtra

1.Introduction:

Gesture recognition has been a very interesting problem in Computer Vision community for a long time. This is particularly due to the fact that segmentation of foreground object from a cluttered background is a challenging problem in real-time. The most obvious reason is because of the semantic gap involved when a human looks at an image and a computer looking at the same image. Humans can easily figure out what's in an image but for a computer, images are just 3-dimensional matrices. It is because of this, computer vision problems remains a challenge.

Input Segmentation [9]

keywords-contour, hand gesture, opencv

Problem statement

We are going to recognize hand gestures from a video sequence. To recognize these gestures from a live video sequence, we first need to take out the hand region alone removing all the unwanted portions in the video sequence. After segmenting the hand region, we then count the fingers shown in the video sequence to instruct a robot based on the finger count. Thus, the entire problem could be solved using 2 simple steps -

1. Find and segment the hand region from the video sequence.
2. Count the number of fingers from the segmented hand region in the video sequence.

To understand hand-gesture recognition in depth,

Segment the Hand region

The first step in hand gesture recognition is obviously to find the hand region by eliminating all the other unwanted portions in the video sequence.. It is easier using Python and OpenCV!

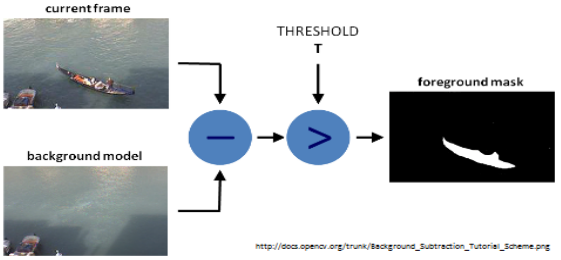
Video sequence is just a collection of frames or collection of images that runs with respect to time.

Background Subtraction

First, we need an efficient method to separate foreground from background. To do this, we use the concept of running averages. We make our system to look over a particular scene for 30 frames. During this period, we compute the running average over the current frame and the previous frames. By doing this

Ok robot! The video sequence that you stared at (running average of those 30 frames) is the **background**.

After figuring out the background, we bring in our hand and make the system understand that our hand is a new entry into the background



Motion Detection and Thresholding

To detect the hand region from this difference image, we need to threshold the difference image, so that only our hand region becomes visible and all the other unwanted regions are

painted as black. This is what Motion Detection is all about.

Thresholding is the assigment of pixel intensities to 0's and 1's based a particular threshold level so that our object of interest alone is captured from an image.

Contour Extraction

After thresholding the difference image, we find contours in the resulting image. The contour with the *largest area* is assumed to be our hand.

Contour is the outline or boundary of an object located in an image.

three simple steps.

1. Background Subtraction
2. Motion Detection and Thresholding
3. Contour Extraction

II. Histrory/Existing System

Gesture Sensing Technologies

GESTURE SENSING TECHNOLOGIES

Gesture recognition is the process by which gestures made by the user are made known to the system.. Various types of gesture recognition technologies in use currently are:

· **Contact type**

It involves touch based gestures using a touch pad or a touch screen. Touch pad or touch screen based gesture recognition is achieved by sensing physical contact on a conventional touch pad or touch screen. Touch pads & touch screens are primarily used for controlling cursors on a PC or mobile phones and are gaining user acceptance for point of sale terminals, PDAs, various industrial and automotive applications as well

· **Non-Contact**

· **Device Gesture Technologies**

Device-based techniques use a glove, stylus, or other position tracker, whose movements send signals that the system uses to identify the gesture.

One of the commonly employed techniques for gesture recognition is to instrument the hand with a glove; the glove is equipped with a variety of sensors to provide information about hand position, orientation, and flex of fingers. First commercial hand tracker

Styli are interfaced with display technologies to record and interpret gestures like the writing of text.

To reduce physical restriction due to the cables, an alternate technique used is to wear an ultrasonic emitter on the index finger and the receiver capable of tracking the position of the emitter is mounted on a head mounted device (HMD).

· **Vision-based Technologies**

There are two approaches to vision based gesture recognition;

Model based techniques:

They try to create a three dimensional model of the users hand and use this for recognition. Some systems track gesture movements through a set of critical positions. When a gesture moves through the same critical positions as does a stored gesture, the system recognizes it. Other systems track the body part being moved, compute the nature of the motion, and then determine the gesture. The systems generally do this by applying statistical modelling to a set of movements.

Image based methods:

Image-based techniques detect a gesture by capturing pictures of a user's motions during the course of a gesture. The system sends these images to computer-vision software, which tracks them and identities the gesture.

These methods typically extract flesh tones from the background images to find hands and then try and extract features such as fingertips, hand edges, or gross hand geometry for use in gesture recognition.

Electrical Field Sensing

Proximity of a human body or body part can be measured by sensing electric fields; the term used to refer to a family of non- contact measurements of the human body that may be made with slowly varying electric fields.

III.Algorithm used

1. Find the convex hull of the segmented hand region (which is a contour) and compute the most extreme points in the convex hull (Extreme Top, Extreme Bottom, Extreme Left, Extreme Right).
2. Find the center of palm using these extremes points in the convex hull.
3. Using the palm's center, construct a circle with the maximum Euclidean distance (between the palm's center and the extreme points) as radius.
4. Perform bitwise AND operation between the thresholded hand image (frame) and the circular ROI (mask). This reveals the finger slices, which could further be used to calcualate the number of fingers .

4.Software and Hardware requirements

Pycharm,Camera ,OpenCV,Jetbrains PyCharm

IV.Applications

a) Virtual Reality: Gestures for virtual and augmented reality applications have experienced one
of the greatest levels of uptake in computing. Virtual reality interactions use gestures to enable
realistic manipulations of virtual objects using ones hands, for 3D display interactions or 2D
displays that simulate 3D interactions

b) Games: When, we look at gestures for computer games. Freeman tracked a player's hand or
body position to control movement and orientation of interactive game objects such as cars. Konrad et al. [10] used gestures to control the movement of avatars in a virtual world, and Play Station 2 has introduced the Eye Toy, a camera that tracks hand movements for interactive games

c) Sign Language: Sign language is an important case of communicative gestures. Since sign languages are highly structural, they are very suitable as test beds for vision algorithms [12]. At the same time, they can also be a good way to help the disabled to interact with computers. Sign language for the deaf (e.g. American Sign Language) is an example that has received significant attention in the gesture literature

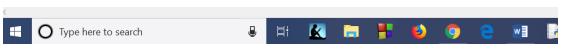
V .Conclusion

In this project, we have learnt about Background Subtraction, Motion Detection, Thresholding and Contour Extraction to nicely segment hand region from a real-time video sequence using OpenCV and Python.

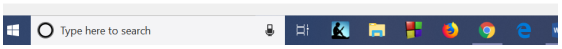
7.Code and Output screenshots

Code Screenshots

```
# Imports
import numpy as np
import cv2
import math

# Open Camera
capture = cv2.VideoCapture(0)

while capture.isOpened():

    # Capture frames from the camera
    ret, frame = capture.read()

    # Get hand data from the rectangle sub window
    cv2.rectangle(frame, (100, 100), (300, 300), (0, 255, 0), 0)
    crop_image = frame[100:300, 100:300]

    # Apply Gaussian blur
    blur = cv2.GaussianBlur(crop_image, (3, 3), 0)

    # Change color-space from BGR -> HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    # Create a binary image with where white will be skin colors and rest is black
    mask2 = cv2.inRange(hsv, np.array([2, 0, 0]), np.array([20, 255, 255]))

    # Kernel for morphological transformation
    kernel = np.ones((5, 5))

    # Apply morphological transformations to filter out the background noise
    dilation = cv2.dilate(mask2, kernel, iterations=1)
    erosion = cv2.erode(dilation, kernel, iterations=1)

    # Apply Gaussian Blur and Threshold
    filtered = cv2.GaussianBlur(erosion, (3, 3), 0)
    ret, thresh = cv2.threshold(filtered, 127, 255, 0)

    # Show threshold image
    cv2.imshow("Thresholded", thresh)

    # Find contours
    image, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```



```
# Find contours
image, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

try:
    # Find contour with maximum area
    contour = max(contours, key=lambda x: cv2.contourArea(x))

    # Create bounding rectangle around the contour
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(crop_image, (x, y), (x + w, y + h), (0, 0, 255), 0)

    # Find convex hull
    hull = cv2.convexHull(contour)

    # Draw contour
    drawing = np.zeros(crop_image.shape, np.uint8)
    cv2.drawContours(drawing, [contour], -1, (0, 255, 0), 0)
    cv2.drawContours(drawing, [hull], -1, (0, 0, 255), 0)

    # Find convexity defects
    hull = cv2.convexHull(contour, returnPoints=False)
    defects = cv2.convexityDefects(contour, hull)

    # Use cosine rule to find angle of the far point from the start and end point i.e. the convex
    # tips) for all defects
    count_defects = 0

    for i in range(defects.shape[0]):
        s, e, f, d = defects[i, 0]
        start = tuple(contour[s][0])
        end = tuple(contour[e][0])
        far = tuple(contour[f][0])

        a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
        b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
        c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
        angle = (math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 180) / 3.14

        # if angle > 90 draw a circle at the far point
        if angle <= 90:
            count defects += 1




          a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
          b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
          c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
          angle = (math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 180) / 3.14

          # if angle > 90 draw a circle at the far point
          if angle <= 90:
              count_defects += 1
              cv2.circle(crop_image, far, 1, [0, 0, 255], -1)

          cv2.line(crop_image, start, end, [0, 255, 0], 2)

      # Print number of fingers
      if count_defects == 0:
          cv2.putText(frame, "ONE", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,(0,0,255),2)
      elif count_defects == 1:
          cv2.putText(frame, "TWO", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,(0,0,255), 2)
      elif count_defects == 2:
          cv2.putText(frame, "THREE", (5, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,(0,0,255), 2)
      elif count_defects == 3:
          cv2.putText(frame, "FOUR", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,(0,0,255), 2)
      elif count_defects == 4:
          cv2.putText(frame, "FIVE", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,(0,0,255), 2)
      else:
          pass
  except:
      pass

  # Show required images
  cv2.imshow("Gesture", frame)
  all_image = np.hstack((drawing, crop_image))
  cv2.imshow('Contours', all_image)

  # Close the camera if 'q' is pressed
  if cv2.waitKey(1) == ord('q'):
      break

apture.release()
v2.destroyAllWindows()
```



Output Screenshots

VI .Reference

B. Nandwana, S. Tazi, S. Trivedi, D. Kumar and S. K. Vipparthi, "A survey paper on hand gesture recognition," *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, Nagpur, 2017, pp. 147-152, doi: 10.1109/CSNT.2017.8418527.

B. Nandwana, S. Tazi, S. Trivedi, D. Kumar and S. K. Vipparthi, "A survey paper on hand gesture recognition," *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, Nagpur, 2017, pp. 147-152, doi: 10.1109/CSNT.2017.8418527.

. https://docs.opencv.org/3.0-beta/doc/...

https://www.jetbrains.com/pycharm/dow...

. https://www.python.org/