Hand Gesture Recognition using Python and OpenCV - Part 2

Computer Vision | 25 April 2017

33 Comments

Contents x

This is a follow-up post of my tutorial on Hand Gesture Recognition using OpenCV and Python. Please read the first

part of the tutorial here and then come back.

In the previous tutorial, we have used Background Subtraction, Motion Detection and Thresholding to segment our hand region from a live video sequence. In this tutorial, we will take one step further to recognize the number of fingers shown in a live video sequence.

> Note: This tutorial assumes that you have knowledge in using OpenCV, Python, NumPy and some basics of Computer Vision and Image Processing. If you need to setup environment on your system, please follow the instructions posted here and here.

Count My Fingers

Having segmented the hand region from the live video sequence, we will make our system to count the fingers that are shown via a camera/webcam. We cannot use any template (provided by OpenCV) that is available to perform this, as it is indeed a challenging problem.

The entire code from my previous tutorial (Hand Gesture Recognition-Part 1) can be seen here for reference. Note that, we have used the concept of Background Subtraction, Motion Detection and Thresholding to segment the hand region from a live video sequence.

We have obtained the segmented hand region by assuming it as the largest contour (i.e. contour with the maximum area) in the frame. If you bring in some large object inside this frame which is larger than your hand, then this algorithm fails. So, you must make sure that your hand occupies the majority of the

region in the frame.

We will use the segmented hand region which was obtained in the variable `hand`. Remember, this `hand` variable is a tuple having `thresholded` (thresholded image) and `segmented` (segmented hand region). We are going to utilize these two variables to count the fingers shown. How are we going to do that?

There are various approaches that could be used to count the fingers, but we are going to see one such approach in this tutorial. This is a faster approach to perform hand gesture recognition as proposed by Malima et.al. The methodology to count the fingers (as proposed by Malima et.al) is shown in the figure below.

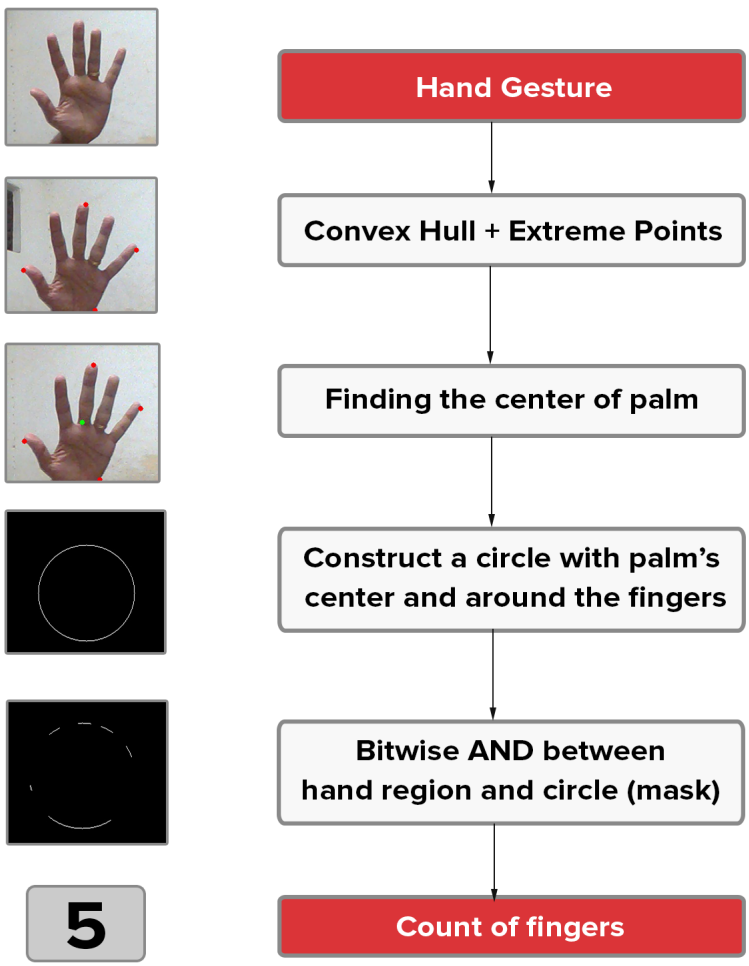

Figure 1. Hand-Gesture Recognition algorithm to count the fingers

As you can see from the above image, there are four intermediate steps to count the fingers, given a segmented hand region. All these steps are shown with a corresponding output image (shown in the left) which we get, after performing that particular step.

Four Intermediate Steps

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 shown.

Below you could see the entire function used to perform the above four steps.

- Input - `thresholded` (thresholded image) and `segmented` (segmented hand region or contour)
- Output - `count` (Number of fingers).

`recognize.py` **code**

```
#--------------------------------------------------------
# To count the number of fingers in the segmented hand reg
#--------------------------------------------------------
def count(thresholded, segmented):
    # find the convex hull of the segmented hand region
    chull = cv2.convexHull(segmented)

    # find the most extreme points in the convex hull
    extreme_top    = tuple(chull[chull[:, :, 1].argmin()][
    extreme_bottom = tuple(chull[chull[:, :, 1].argmax()][
    extreme_left   = tuple(chull[chull[:, :, 0].argmin()][
    extreme_right  = tuple(chull[chull[:, :, 0].argmax()][

    # find the center of the palm
    cX = int((extreme_left[0] + extreme_right[0]) / 2)
    cY = int((extreme_top[1] + extreme_bottom[1]) / 2)

    # find the maximum euclidean distance between the cent
    # and the most extreme points of the convex hull
    distance = pairwise.euclidean_distances([(cX, cY)], Y=
    maximum_distance = distance[distance.argmax()]

    # calculate the radius of the circle with 80% of the m
    radius = int(0.8 * maximum_distance)

    # find the circumference of the circle
    circumference = (2 * np.pi * radius)

    # take out the circular region of interest which has
    # the palm and the fingers
    circular_roi = np.zeros(thresholded.shape[:2], dtype="

    # draw the circular ROI
    cv2.circle(circular_roi, (cX, cY), radius, 255, 1)

    # take bit-wise AND between thresholded hand using the
    # which gives the cuts obtained using mask on the thre
    circular_roi = cv2.bitwise_and(thresholded, thresholde

    # compute the contours in the circular ROI
    (_, cnts, _) = cv2.findContours(circular_roi.copy(), c

    # initalize the finger count
    count = 0

    # loop through the contours found
    for c in cnts:
        # compute the bounding box of the contour
        (x, y, w, h) = cv2.boundingRect(c)

        # increment the count of fingers only if -
```

```
            # 1. The contour region is not the wrist (bottom a
            # 2. The number of points along the contour does n
            #     25% of the circumference of the circular ROI
            if ((cY + (cY * 0.25)) > (y + h)) and ((circumfere
                count += 1

    return count
```

Each of the intermediate step requires some understanding of image processing fundamentals such as Contours, Bitwise-AND, Euclidean Distance and Convex Hull.

Contours

The outline or the boundary of the object of interest. This contour could easily be found using OpenCV's `cv2.findContours()` function. Be careful while unpacking the return value of this function, as we need three variables to unpack this tuple in OpenCV 3.1.0 - Contours.

Bitwise-AND

Performs bit-wise logical AND between two objects. You could visually think of this as using a mask and extracting the regions in an image that lie under this mask alone. OpenCV provides `cv2.bitwise_and()` function to perform this operation - Bitwise AND.

Euclidean Distance

This is the distance between two points given by the equation shown here. Scikit-learn provides a function called `pairwise.euclidean_distances()` to calculate the Euclidean distance from *one point* to *multiple points* in a single line of code - Pairwise Euclidean Distance. After that, we take the

maximum of all these distances using NumPy's `argmax()` function.

Convex Hull

You can think of convex hull as a dynamic, stretchable envelope that wraps around the object of interest. To read more about it, visit this link.

Results

You can download the entire code to perfom Hand Gesture Recognition here. Clone this repository using

Command shell

```
git clone https://github.com/Gogul09/gesture-recognition.gi
```

in a Terminal/Command prompt. Then, get into the folder and type

Command shell

```
python recognize.py
```

> Note: Do not shake your webcam during the calibration period of 30 frames. If shaken during the first 30 frames, the entire algorithm will not perform as we expect.

After that, you can use bring in your hand into the bounding box, show gestures and the count of fingers will be displayed accordingly. I have included a demo of the entire pipeline below.

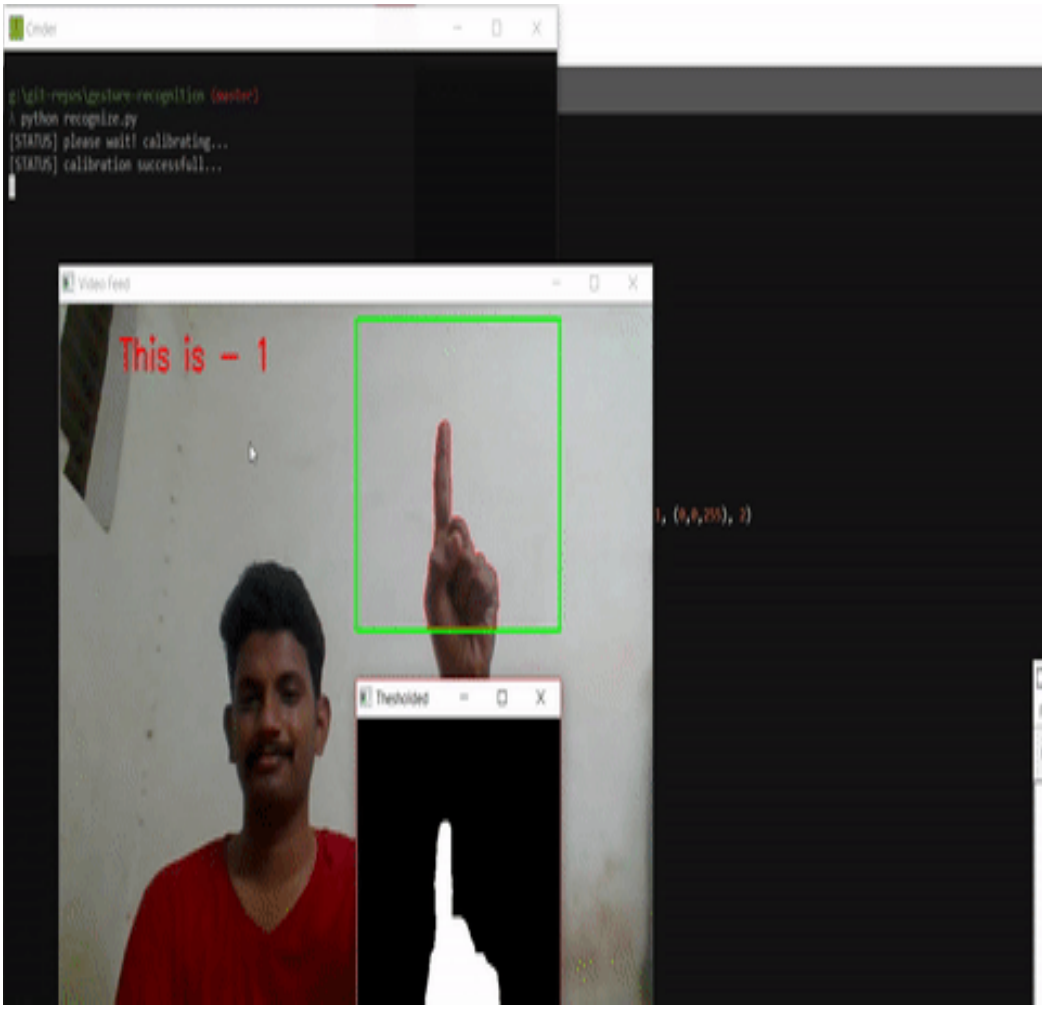

Figure 2. Hand-Gesture Recognition | Counting the fingers | Demo

Summary

In this tutorial, we have learnt about recognizing hand gestures using Python and OpenCV. We have explored Background Subtraction, Thresholding, Segmentation, Contour Extraction,

Convex Hull and Bitwise-AND operation on real-time video sequence. We have followed the methodology proposed by Malima et al. to quickly recognize hand gestures.

You could extend this idea by using the count of fingers to instruct a robot to perform some task like picking up an object, go forward, move backward etc. using Arduino or Raspberry Pi platforms. I have also made a simple demo for you by using the count of fingers to control a servo motor's rotation here.

In case if you found something useful to add to this article or you found a bug in the code or would like to improve some points mentioned, feel free to write it down in the comments. Hope you found something useful here.

33 Comments