<http://www.tmroyal.com/a-high-level-description-of-two-fingertip-tracking-techniques-k-curvature-and-convexity-defects.html>

Fingertip tracking has a number of compelling applications and, thus, is of interest to a great number of people. It is surprising that there are so few high level descriptions of techniques for fingertip tracking. Here I describe two methods for finger tip tracking: k-curvature and convexity defects. While this post is technical, it does not give implementation details. Rather, it is a high level description of these two techniques.

The first step in tracking fingertips is locating the hand. This can be accomplished through foreground-background segmentation. There are a fair number of techniques for this kind of segmentation. Some of them are rather sophisticated. I have used simple background subtraction in the past. These details are beyond the scope of this post.

When the hand is properly segmented from the background, the hand is represented by a series of points as illustrated below. One possible formulation of the goal of a fingertip tracking algorithm is the identification of the points that correspond to the location of the fingertip relative to the rest of the frame. In other words, the goal is to identify the points marked red in the image below.

#### Convexity Defects

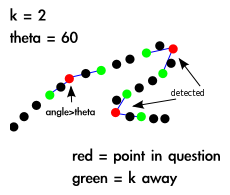
One way of finding these points is to find the convexity defects in this set of points.

A curve that is convex is one that curves outward rather than inward as is the case with a concave curve. If a curve is described as a series of points, those points corresponding to the convex parts of the curve are called its [convex hull](http://en.wikipedia.org/wiki/Convex_hull). A convexity defect is a set of points that are not also in the set comprising the convex hull.

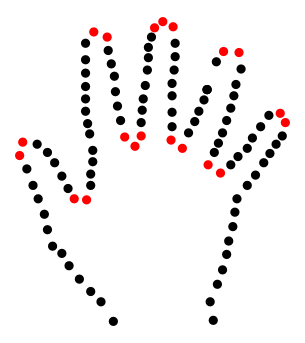
There is an important limitation of using convexity defects for fingertip tracking. Given a hand with a single finger, there are no convexity defects. This means that this approach requires more than one finger to be extended to function properly.

#### k-curvature

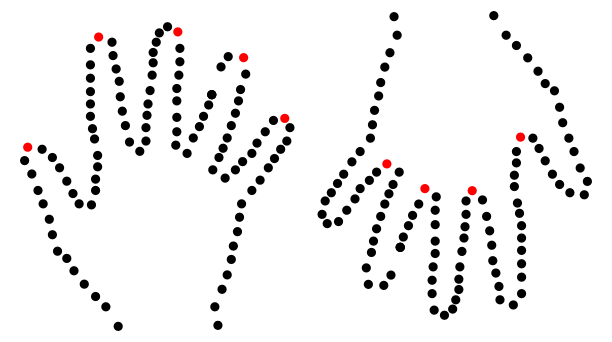
Another method for fingertip tracking is called [k-curvature](http://en.wikipedia.org/wiki/CurvatureShare). It is accomplished as follows:



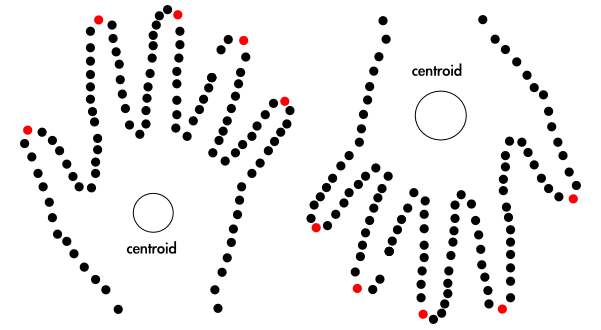
For each point in the set of points describing a curve, the algorithm determines the angle between the two lines that start at the point in question and end k points away in either direction. This is illustrated above. If the angle is determined to be below a certain threshold, often 60 degrees, the angle is marked as being of interest. This results of this process with the outline of a hand as an input is illustrated below.



At this point, the algorithm has not distinguished between the space between fingers and the fingertips themselves. Often, implementations look for those places in which the points k away are lower in height than the points under consideration. This works for upward pointing hands, but not downward pointing hands.



Another way of distinguishing between finger tips is to keep only those points that are further from the centroid than points k away. (The centroid is a point roughly in the center of the set of points.) This allows for detecting finger tips in a downward pointing hand configuration.



OpenCV/EmguCV Real time Hand and fingers tracking Simple static gesture recognition using webcam only, it works on kinect too but I am lazy to record another video Summary of algorithm/method: 1)background subtraction using YCrCb color channel, but process each channel separately and combine them together to use as a mask 2)skin color extraction using YCrCb color channel 3)contour extraction, remove noises using size threshold, also uses erosion and dilation, also uses gaussian smoothing 4)use Haar-cascade to detect the face region, then remove it 5)find convex hull and convexity defects, also find the K-curvature of convexity defects points, then determine it as fingertips if fullfill certain assumption 6)find the max inscribed circle and min enclosing circle 6)from those convexity defects that fullfill fingertips assumption, find the largest one, it is the thumb, then find the vector to determine whether it is left or right hand 7)then recognize the hand gesture based on simple assumption (too much to list here)

<https://www.youtube.com/watch?v=MmZc1REg6uw>

For left picturebox algo: 1) use convex hull find the corner points 2) calculate these points k-curvature angle 3) if lower than certain value, assume it is finger 4) start from the finger point, expand to left and right, find middle of the 2 points, then plot it blue color (u can see blue color line, it is for getting finger direction, stil under progress) For right picture box algo (smaller) 1) find convex hull and convexity defects 2) if the depth point of convexity defect is lower than start/end point + if the start point of convexity defect is higher than middle point of whole contour = assume it is finger (this algo dont work well if hand is not vertical upright) 3) find middle point of palm, draw line from middle point to finger point (red line) this finger direction is not as good as the previous 1

<https://www.youtube.com/watch?v=an8j3IcTDec>

was comparing kcurvature and convexity defects, gonna improve a algo which i use previously (bottom 1), maybe will combine both of it to achieve a more accurate hand tracking mechanism

SHOW MORE

<https://engineerslaboratory.wordpress.com/2016/11/13/appliances-control-led-toggle-with-raspberry-pi-and-opencv-c/>

<https://www.youtube.com/watch?v=kQxiFaZbOfA>

# Real Time Hand Posture/Gesture Recognition with OpenCV

# Hand recognition full source code OpenCV

<https://www.youtube.com/watch?v=5zgLtQh5_lk>

<https://github.com/SinghalHarsh/OpenCV-Projects/blob/master/Hand_Gesture_Recognition.ipynb>

<https://www.youtube.com/watch?v=f2H6I2xq09s>

<https://forum.openframeworks.cc/t/hand-and-fingers-detection/1916>

# [Hand and fingers detection](https://forum.openframeworks.cc/t/hand-and-fingers-detection/1916)

# [k-curvature / template matching](https://forum.openframeworks.cc/t/k-curvature-template-matching/1581)

<https://forum.openframeworks.cc/t/k-curvature-template-matching/1581>

<https://github.com/MikkelKim/HandMade>

<http://tespunclingti.tk/Resume/k-curvature-algorithm-open-cv-imshow.html>

<http://deraconste.tk/Resume/k-curvature-algorithm-open-cv-tutorial-3020.html>

<https://www.mathworks.com/matlabcentral/linkexchange/links/3210-real-time-static-and-dynamic-hand-gesture-recognition-system>

<https://picoledelimao.github.io/blog/2015/11/15/fingertip-detection-on-opencv/>

# GESTURE RECOGNITION USING OPENCV + PYTHON

http://vipulsharma20.blogspot.ca/2015/03/gesture-recognition-using-opencv-python.html

Repository for project code of CE264 Computer Vision

<https://github.com/RobinCPC/CE264-Computer_Vision>

<https://github.com/luizfranca/hand-tracking-and-gesture-recognition>

Implementation of the system described in the paper Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. This project is for the class of Computer Vision taken at UFRPE. This project is an implementation of the paper Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware by Yeo et al 2015.

PYTHON IMPEMENTATION:

https://github.com/moxon6/OpenCV-Hand-Recognition

Based on original C++ Code From: <https://github.com/jujojujo2003/OpenCVHandGuesture>

Changed palm detection to maximum inscribed circle (improved accuracy)

Restructured into Python App using OpenCV 3.0

|  |
| --- |
| # Gesture Recognition |
|  | def gesture\_recognize(self, cnt, defects, count\_defects, crop\_res): |
|  | # use angle between start, end, defect to recognize # of finger show up |
|  | if type(defects) is not None and cv2.contourArea(cnt) >= 5000: |
|  | for i in range(defects.shape[0]): |
|  | s, e, f, d = defects[i, 0] |
|  | start = tuple(cnt[s][0]) |
|  | end = tuple(cnt[e][0]) |
|  | far = tuple(cnt[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/math.pi |
|  | if angle <= 90: |
|  | count\_defects += 1 |
|  | cv2.circle(crop\_res, far, 5, [0, 0, 255], -1) |
|  | cv2.line(crop\_res, start, end, [0, 255, 0], 2) |
|  |  |

For left picturebox algo: 1) use convex hull find the corner points 2) calculate these points k-curvature angle 3) if lower than certain value, assume it is finger 4) start from the finger point, expand to left and right, find middle of the 2 points, then plot it blue color (u can see blue color line, it is for getting finger direction, stil under progress) For right picture box algo (smaller) 1) find convex hull and convexity defects 2) if the depth point of convexity defect is lower than start/end point + if the start point of convexity defect is higher than middle point of whole contour = assume it is finger (this algo dont work well if hand is not vertical upright) 3) find middle point of palm, draw line from middle point to finger point (red line) this finger direction is not as good as the previous 1

1. Depth of each defects (ld) must be longer than palm center radius (ra) but shorter than min enclosing circle radius (rb), ra < ld < rb
2. Angle (θa) between start point (ps) and end point (pe) must be less than 90°, θa<90

iii. Local minimal K-curvature (θk) of point must be lower than 60°, θk<90

First, we find convexity defects that have a depth (ld) longer than the palm radius (ra) but shorter than (rb) and the angle (θa) between start point (ps) and end point (pe) lower than 90°. Then we store its start point (ps), depth point (pd) and end point (pe) into an array (Ap). We remove one of the neighbor points that are very close to each other. From this point, we search bi-direction (forward and backward) along the hand contour and compute the K-curvature (θk) of each point, which is the angle between the two vectors [Ci(j), Ci(j − k)] and [Ci(j), Ci(j + k)]. (20 points each in direction and k is a constant of 30). The idea is that contour point with a small K-curvature (θk) will represent a potential peak or valley. Since our method only computes K-curvature for points near to convex hull points, we can reduce the computational cost and effectively remove the valley points. Then we find the point with local maxima peak angle from these 20 points to ensure that a more accurate fingertip location can be determined. Finally, we can determine the finger’s direction by finding the line between the peak point and the middle point of Ci(j − k) and Ci(j + k).

To find the thumb finger, we find the convexity defects region with the largest area (area C) between all the convexity defects that have depth (ld) longer than palm center radius (ra) and an angle (θa) lower than 90°. Using the hand image in Fig. 8 as an example, it will be the shaded area C. The point with the shorter distance (la) to its depth point will be the thumb while the point with the longer distance (lb) will be the index finger.

The three assumptions above work well in determining fingers, except in one case, which is when only one finger is available. If there are no convexity defects with an appropriate depth distance (ld) in assumption (i), and no angle (θa) between start point (ps) and end point (pe) less than 90° in assumption (ii), then we proceed to assumption (iii) to find the K- curvature (θk) of the points of convexity defects that has (θa) larger than 90°.