# Tutorial - Object Tracking (Based on ROI) and Re-recognizing the object if it escapes and comes back into the frame

e-Yantra Team

July 11, 2016

## Contents

## 1 Objective

This objective of the tutorial is ”How to track the object without errors based on ROI selection and How to re-recognize the object if it escapes and comes back inside the frame”.

## 2 Prerequisites

User should have handy knowledge of the following for understanding this tutorial.

- Basics of Python.
- Basics of OpenCV.
- Basics of Image processing.
- CAMShift Algorithm (Refer to tutorial 2 to understand Camshift Algorithm).

## 3 Hardware Requirement

- A Computer with an internal or external webcam.

## 4 Software Requirement

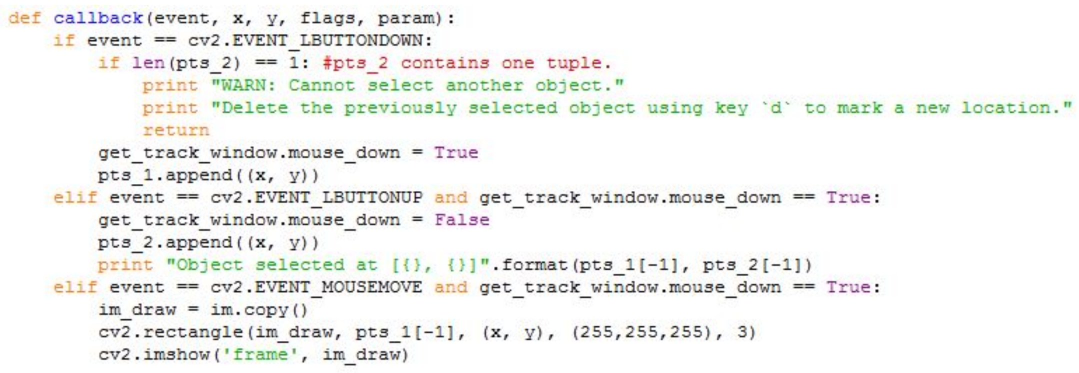
- Python 2.7.11
- OpenCV 2.4.13
- numpy 1.7.1
- **Note :** These are the versions we were working on while creating this tutorial.

## 5 Theory and Description

The base of algorithm used in this tutorial is based on Camshift algorithm and this research paper. Details of Camshift algorithm can be found here.

The working of algorithm used for object tracking is described below.

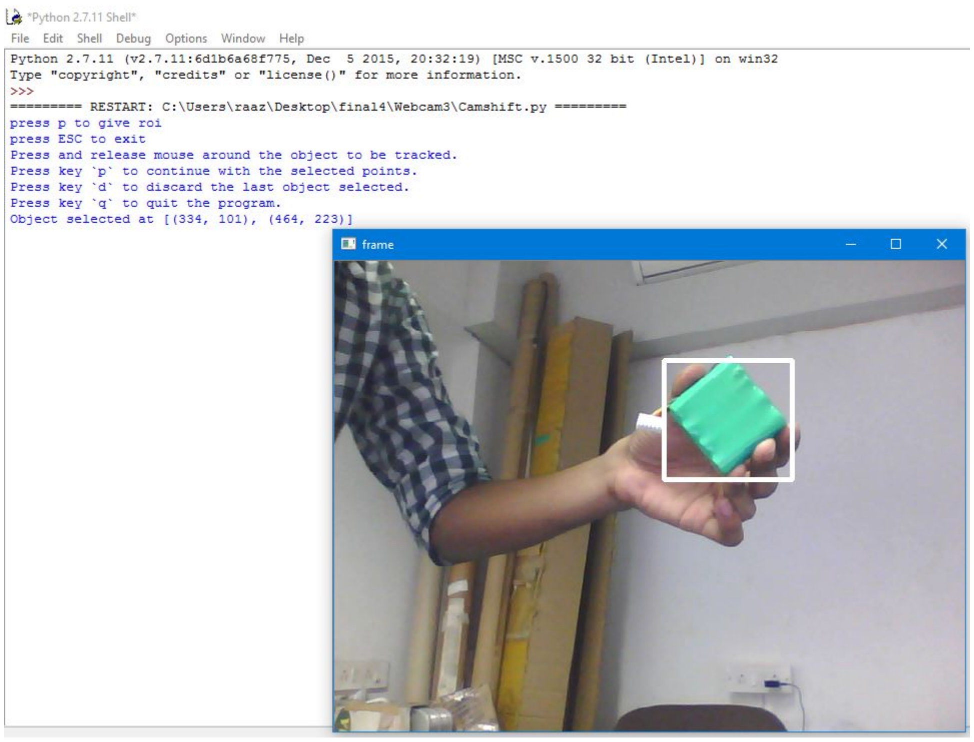
- First of all, it selects Region of Interest around Object, which you want to track using Mouse Callback function in Python. Mouse Callback function is shown in Figure 1.



```python
def callback(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        if len(pts_2) == 1: #pts_2 contains one tuple.
            print "WARN: Cannot select another object."
            print "Delete the previously selected object using key `d` to mark a new location."
            return
        get_track_window.mouse_down = True
        pts_1.append((x, y))
    elif event == cv2.EVENT_LBUTTONUP and get_track_window.mouse_down == True:
        get_track_window.mouse_down = False
        pts_2.append((x, y))
        print "Object selected at [{}, {}]".format(pts_1[-1], pts_2[-1])
    elif event == cv2.EVENT_MOUSEMOVE and get_track_window.mouse_down == True:
        im_draw = im.copy()
        cv2.rectangle(im_draw, pts_1[-1], (x, y), (255,255,255), 3)
        cv2.imshow('frame', im_draw)
```

**Figure 1**

- For selecting ROI press 'p', select region using mouse drag and then press 'p' to track that object. Figure 2 show initialization of ROI in video frame.



```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:32:19) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
========= RESTART: C:\Users\raaz\Desktop\final4\Webcam3\Camshift.py =========
press p to give roi
press ESC to exit
Press and release mouse around the object to be tracked.
Press key `p` to continue with the selected points.
Press key `d` to discard the last object selected.
Press key `q` to quit the program.
Object selected at [(334, 101), (464, 223)]
```

**Figure 2**

- On pressing p it starts tracking the object by creating a bounding box around the object. Figure 3 shows tracking of object.

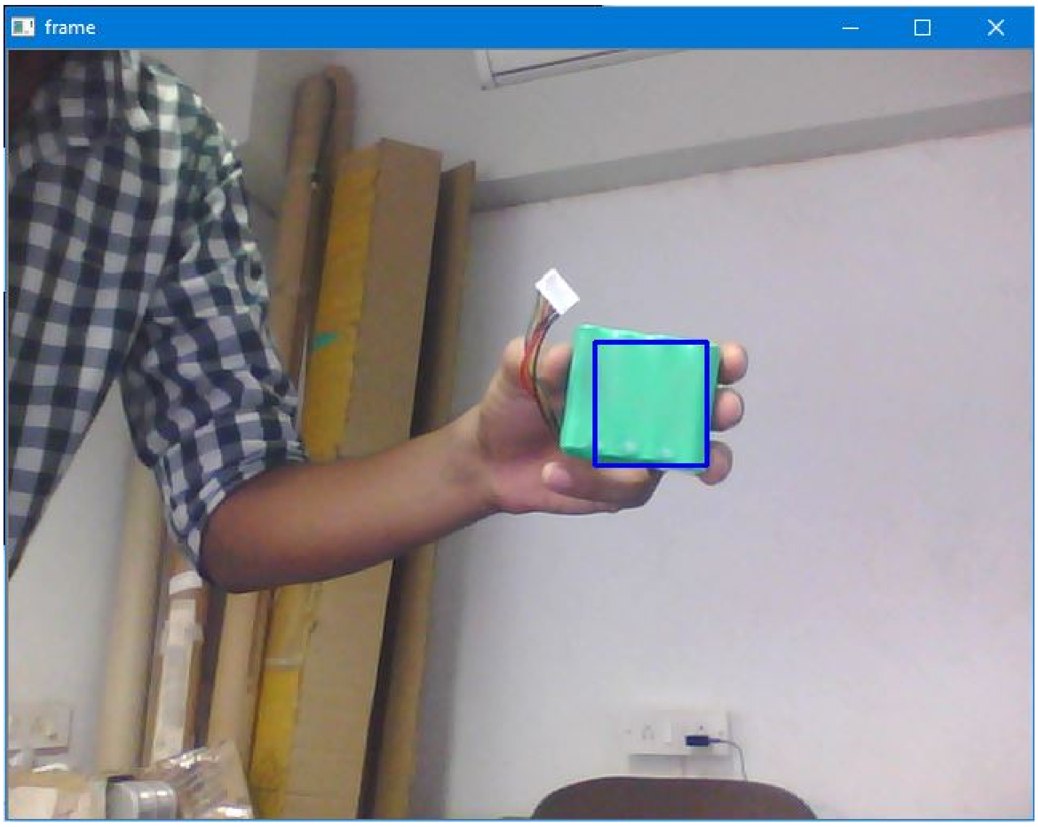
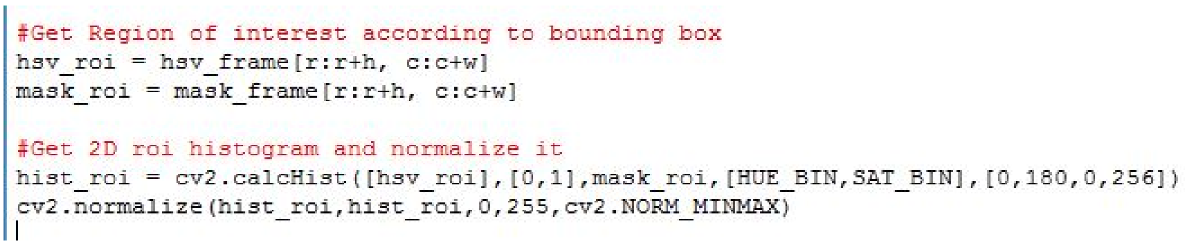


**Figure 3**

- For tracking, it first calculate 2D histogram of selected Region Of Interest (HSV color space) only one time and stores it. Histogram means it is a graph which contains Hue values from 0 to 179 as y axis and Saturation values 0 to 255 as x axis and put number of pixels for each color. Here we uses normalized histogram in which all the values of number of pixels scaled to 0 to 1 using OpenCV functions. Code for calculation of histogram is shown in Figure 4 and calculated Normalized Histogram is shown in Figure 5.



```
#Get Region of interest according to bounding box
hsv_roi = hsv_frame[r:r+h, c:c+w]
mask_roi = mask_frame[r:r+h, c:c+w]

#Get 2D roi histogram and normalize it
hist_roi = cv2.calcHist([hsv_roi],[0,1],mask_roi,[HUE_BIN,SAT_BIN],[0,180,0,256])
cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX)
```

**Figure 4**

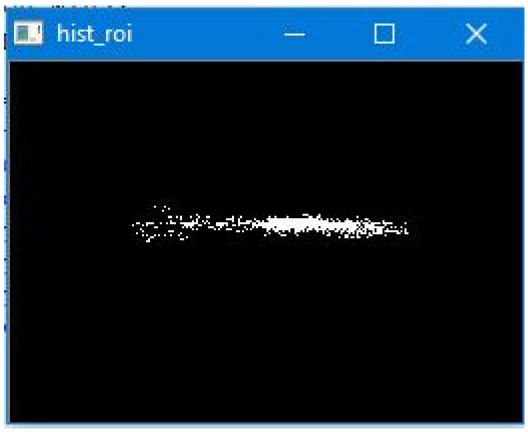


**Figure 5**

- Now it takes next frame and changes its color space from RGB to HSV and then eliminates the region of frame which was too far from the object in previous window using previous bounding box because sometimes Camshift Algorithm converges to other object of similar color of target object. Figure 6 shows code and Figure 7 shows Output image. We have considered that object movement will be real (not very fast).

```
#Change color space of frame from RGB to HSV
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

#If roi is not lost, Track ROI
if not isLost:
    #Increment frame number
    total_frames = total_frames + 1

    #Use 50 pixels margin from bounding box for tracking
    #Make white other area

    if r > 50:
        hsv_frame[:r-50,:] = [255,255,255]
    if r+h+50 < height_frame:
        hsv_frame[r+h+50:,:] = [255,255,255]
    if c > 50:
        hsv_frame[:,:c-50] = [255,255,255]
    if c+w+50 < width_frame:
        hsv_frame[:,c+w+50:] = [255,255,255]
```

**Figure 6**

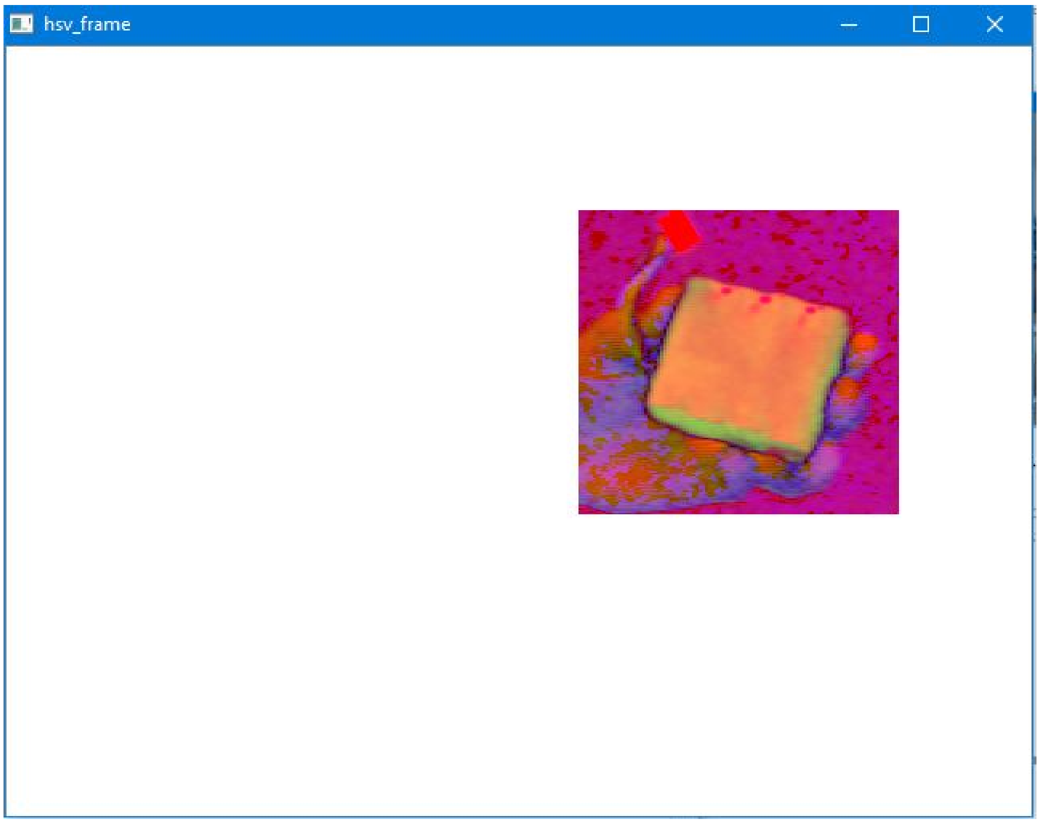
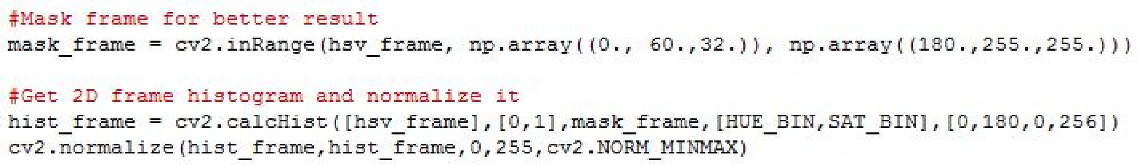


**Figure 7**

- Now it finds 2D Histogram of Figure 7, shown in Figure 9 using code shown it Figure 8.



```
#Mask frame for better result
mask_frame = cv2.inRange(hsv_frame, np.array((0., 60.,32.)), np.array((180.,255.,255.)))

#Get 2D frame histogram and normalize it
hist_frame = cv2.calcHist([hsv_frame],[0,1],mask_frame,[HUE_BIN,SAT_BIN],[0,180,0,256])
cv2.normalize(hist_frame,hist_frame,0,255,cv2.NORM_MINMAX)
```

**Figure 8**

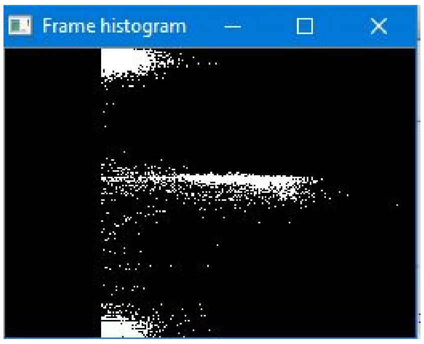
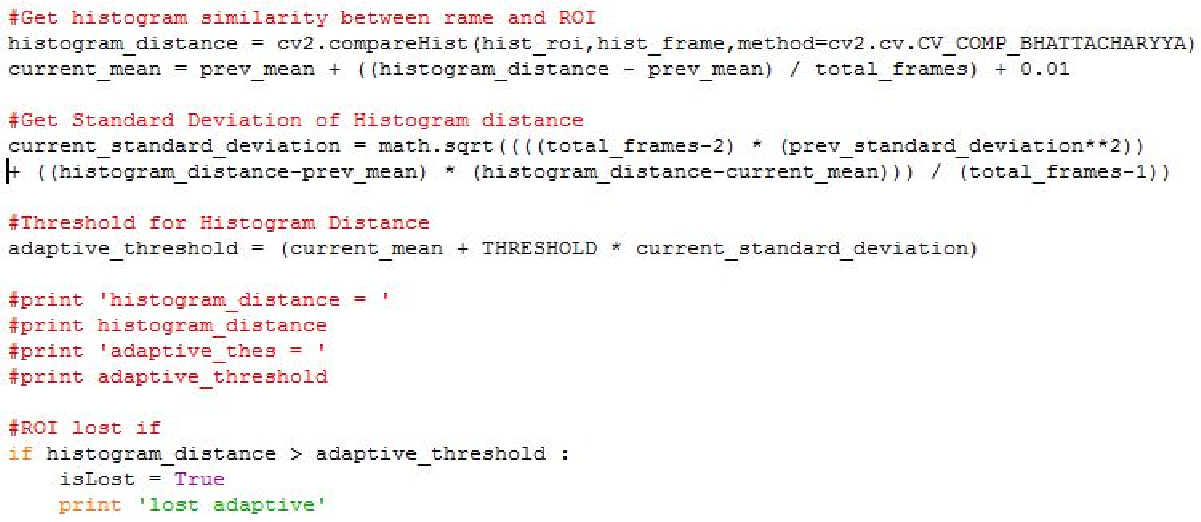


**Figure 9**

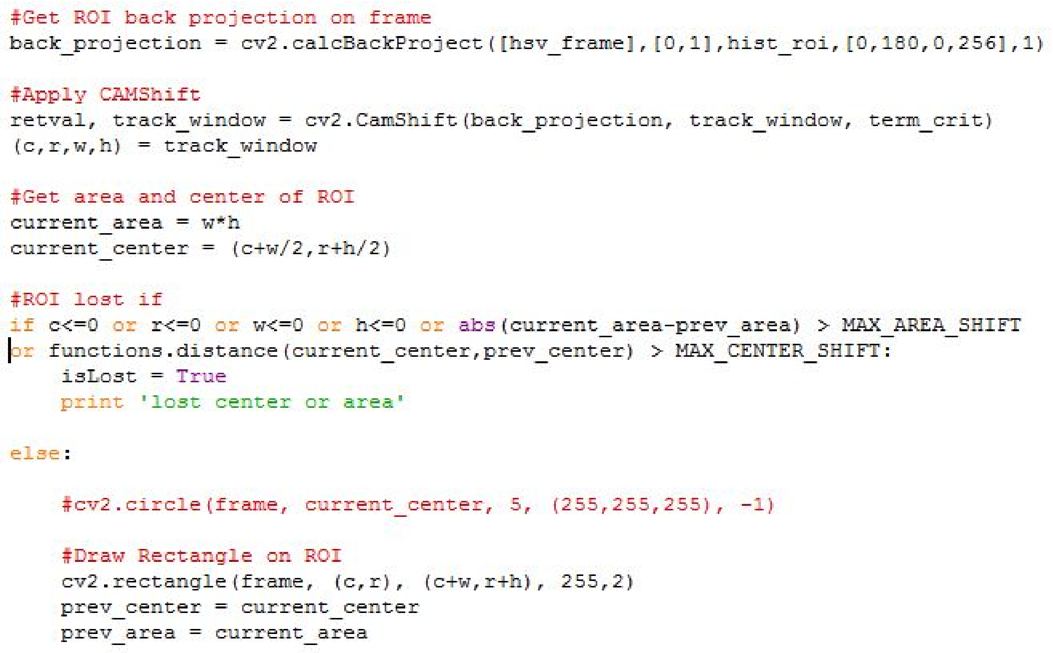
- Now it compares histogram of ROI and histogram of frame using Bhattacharya coefficient (OpenCV has inbuild function for it) and it gives histogram distance. Then calculates mean of the all the histogram distances. Then calculates standard deviation of mean. If current Histogram Distance follows the condition given in code (Figure 10), it means object is lost from window. Figure 10 shows code for this procedure.



```
#Get histogram similarity between rame and ROI
histogram_distance = cv2.compareHist(hist_roi,hist_frame,method=cv2.cv.CV_COMP_BHATTACHARYYA)
current_mean = prev_mean + ((histogram_distance - prev_mean) / total_frames) + 0.01

#Get Standard Deviation of Histogram distance
current_standard_deviation = math.sqrt((((total_frames-2) * (prev_standard_deviation**2))
+ ((histogram_distance-prev_mean) * (histogram_distance-current_mean))) / (total_frames-1))

#Threshold for Histogram Distance
adaptive_threshold = (current_mean + THRESHOLD * current_standard_deviation)

#print 'histogram_distance = '
#print histogram_distance
#print 'adaptive_thes = '
#print adaptive_threshold

#ROI lost if
if histogram_distance > adaptive_threshold :
    isLost = True
    print 'lost adaptive'
```

**Figure 10**

- If condition is false back projection is calculated of HSV frame with respect to histogram of ROI. OpenCV provides function for back projection calculation. Then Camshift algorithm is applied to calculate new bounding box and minimum area ellipse. For more information about Camshift algorithm refer to Tutorial 2. Figure 11 shows code for it and Figure 12 shows back projection.



```
#Get ROI back projection on frame
back_projection = cv2.calcBackProject([hsv_frame],[0,1],hist_roi,[0,180,0,256],1)

#Apply CAMShift
retval, track_window = cv2.CamShift(back_projection, track_window, term_crit)
(c,r,w,h) = track_window

#Get area and center of ROI
current_area = w*h
current_center = (c+w/2,r+h/2)

#ROI lost if
if c<=0 or r<=0 or w<=0 or h<=0 or abs(current_area-prev_area) > MAX_AREA_SHIFT
or functions.distance(current_center,prev_center) > MAX_CENTER_SHIFT:
    isLost = True
    print 'lost center or area'

else:

    #cv2.circle(frame, current_center, 5, (255,255,255), -1)

    #Draw Rectangle on ROI
    cv2.rectangle(frame, (c,r), (c+w,r+h), 255,2)
    prev_center = current_center
    prev_area = current_area
```

**Figure 10**

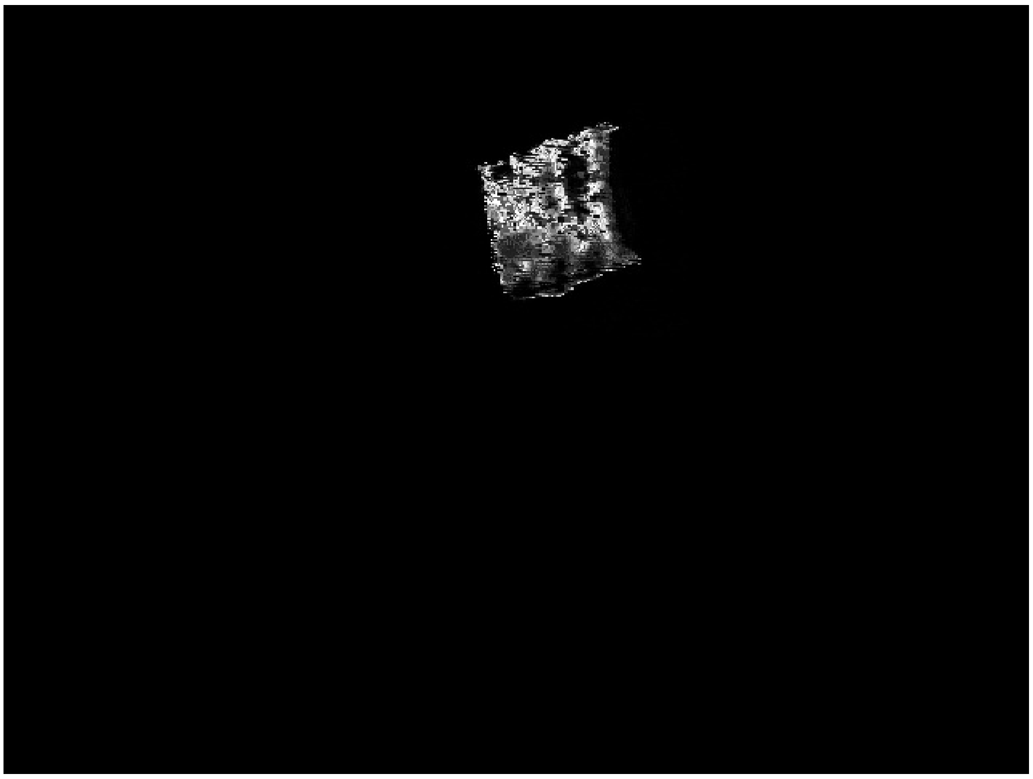
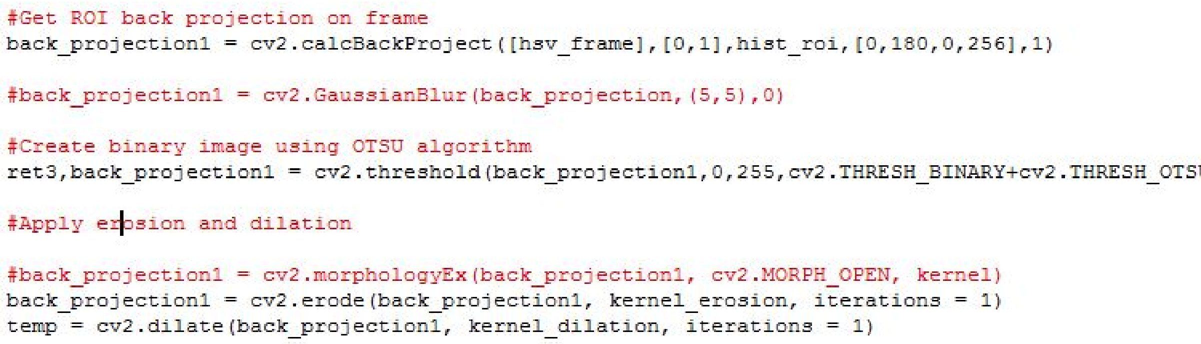


**Figure 11: Back projection**

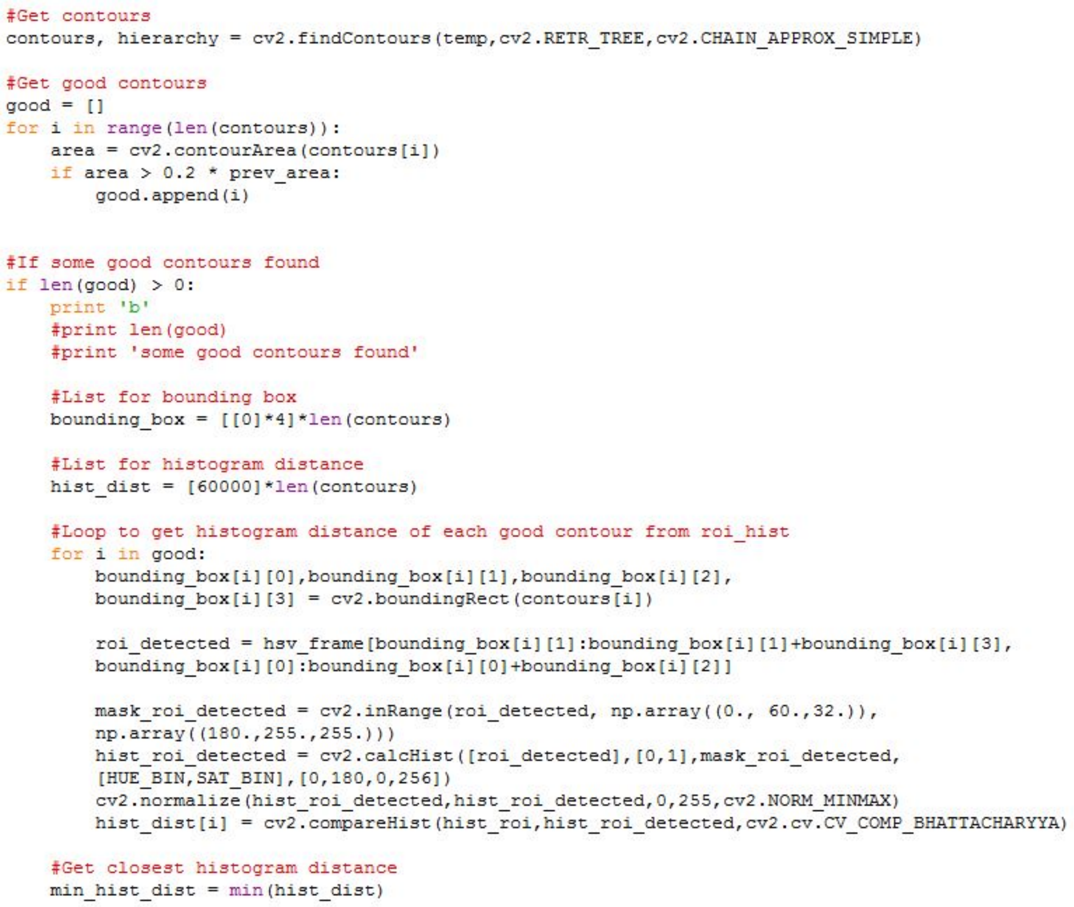
- If object escapes from the frame then it show object is lost and continuously calculates back projection of frame, then binaries it and then uses erosion and dilation to remove errors and get big blobs of objects which has similar color as target object. Figure 12 show this back projection than we take contours in ]the back projection and get histogram of each contour in original frame. then compares these histograms with ROI histogram if one follows given condition given in Figure 13 code then it may be our object. Figure 12 shows code for redetection of object.

Figure 12: Eroded Dilated Binary Back projection



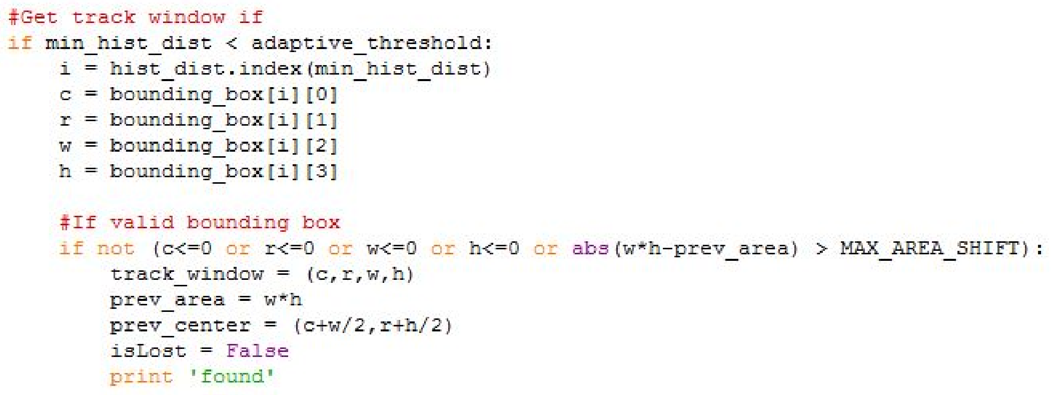
```
#Get ROI back projection on frame
back_projection1 = cv2.calcBackProject([hsv_frame],[0,1],hist_roi,[0,180,0,256],1)

#back_projection1 = cv2.GaussianBlur(back_projection,(5,5),0)

#Create binary image using OTSU algorithm
ret3,back_projection1 = cv2.threshold(back_projection1,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTS

#Apply erosion and dilation

#back_projection1 = cv2.morphologyEx(back_projection1, cv2.MORPH_OPEN, kernel)
back_projection1 = cv2.erode(back_projection1, kernel_erosion, iterations = 1)
temp = cv2.dilate(back_projection1, kernel_dilation, iterations = 1)
```

Figure 13(a): Redection part 1



```
#Get contours
contours, hierarchy = cv2.findContours(temp,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

#Get good contours
good = []
for i in range(len(contours)):
    area = cv2.contourArea(contours[i])
    if area > 0.2 * prev_area:
        good.append(i)


#If some good contours found
if len(good) > 0:
    print 'b'
    #print len(good)
    #print 'some good contours found'

    #List for bounding box
    bounding_box = [[0]*4]*len(contours)

    #List for histogram distance
    hist_dist = [60000]*len(contours)

    #Loop to get histogram distance of each good contour from roi_hist
    for i in good:
        bounding_box[i][0],bounding_box[i][1],bounding_box[i][2],
        bounding_box[i][3] = cv2.boundingRect(contours[i])

        roi_detected = hsv_frame[bounding_box[i][1]:bounding_box[i][1]+bounding_box[i][3],
        bounding_box[i][0]:bounding_box[i][0]+bounding_box[i][2]]

        mask_roi_detected = cv2.inRange(roi_detected, np.array((0., 60.,32.)),
        np.array((180.,255.,255.)))
        hist_roi_detected = cv2.calcHist([roi_detected],[0,1],mask_roi_detected,
        [HUE_BIN,SAT_BIN],[0,180,0,256])
        cv2.normalize(hist_roi_detected,hist_roi_detected,0,255,cv2.NORM_MINMAX)
        hist_dist[i] = cv2.compareHist(hist_roi,hist_roi_detected,cv2.cv.CV_COMP_BHATTACHARYYA)

    #Get closest histogram distance
    min_hist_dist = min(hist_dist)
```

**Figure 13(b): Redection part 2**



```
#Get track window if
if min_hist_dist < adaptive_threshold:
     i = hist_dist.index(min_hist_dist)
     c = bounding_box[i][0]
     r = bounding_box[i][1]
     w = bounding_box[i][2]
     h = bounding_box[i][3]

     #If valid bounding box
     if not (c<=0 or r<=0 or w<=0 or h<=0 or abs(w*h-prev_area) > MAX_AREA_SHIFT):
         track_window = (c,r,w,h)
         prev_area = w*h
         prev_center = (c+w/2,r+h/2)
         isLost = False
         print 'found'
```

**Figure 13(c): Redection part 3**

- Then largest area contour is calculated and a bounding box is formed around it and center is given to us.In this way the object will be re-tracked.Refer Figure 13(b)(c).
- So this algorithm continues above process iteratively and tracks the object frame by frame.

## 6 Experiment

The Python code can be found here.

To run code run run.py file and pause video by pressing 'p' and give roi and then track object by pressing 'p' again.

## 7 Exercise

Re-detection and tracking of an object is shown below. In Figure 14 object is inside the frame. In Figure 15 object escapes from the frame. Then in Figure 16 It is again tracking the object.

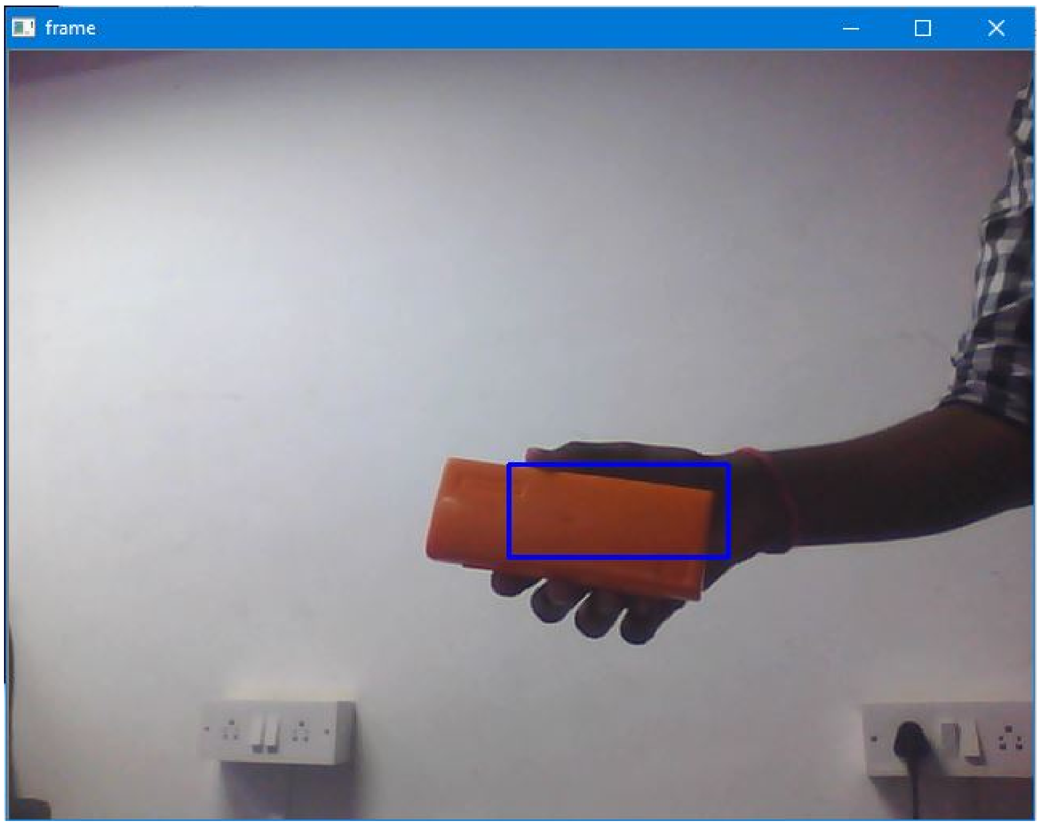


**Figure 14: Object is inside the frame**

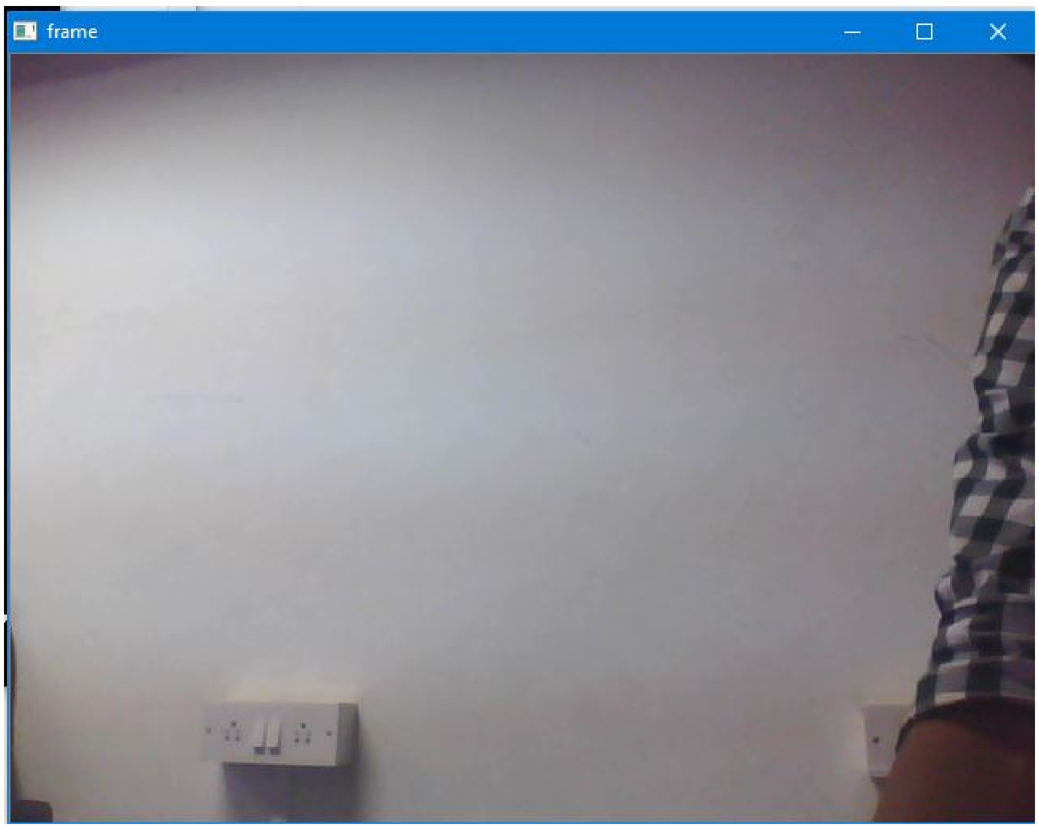


**Figure 15: Object lost in the frame**

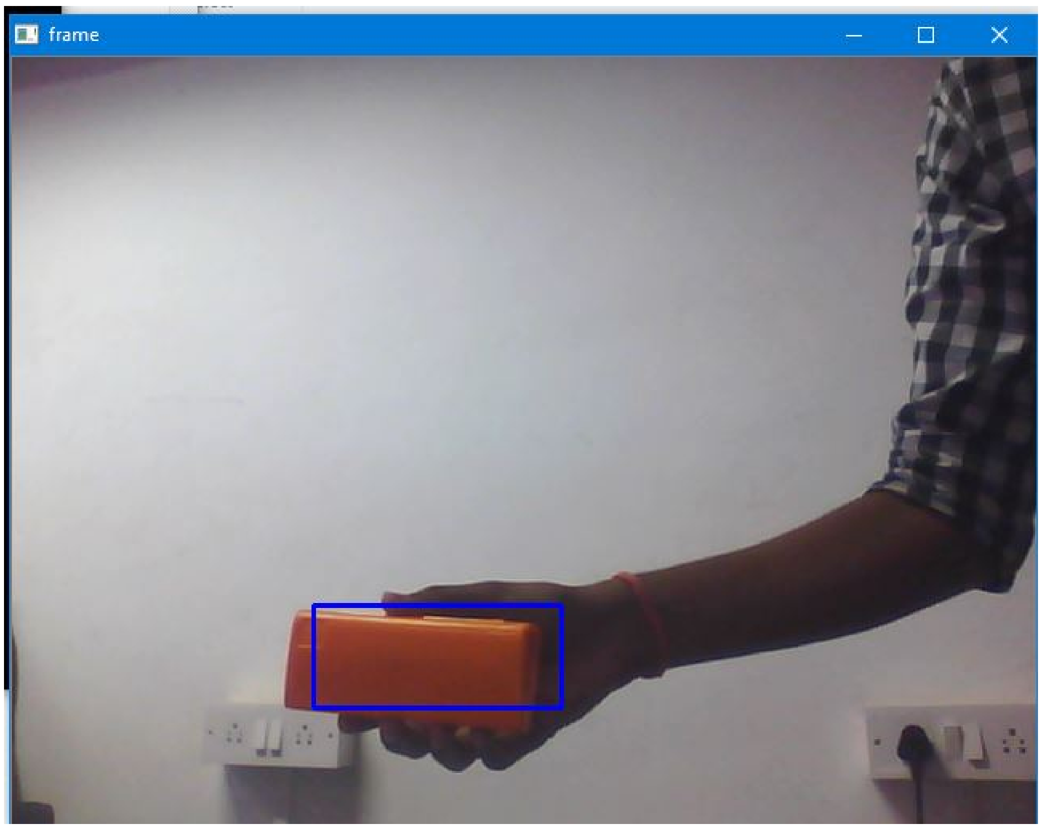


**Figure 16: Re-detection of object in the frame**

In Figure 17 algorithm is not being confused because of the presence of similar colored object.

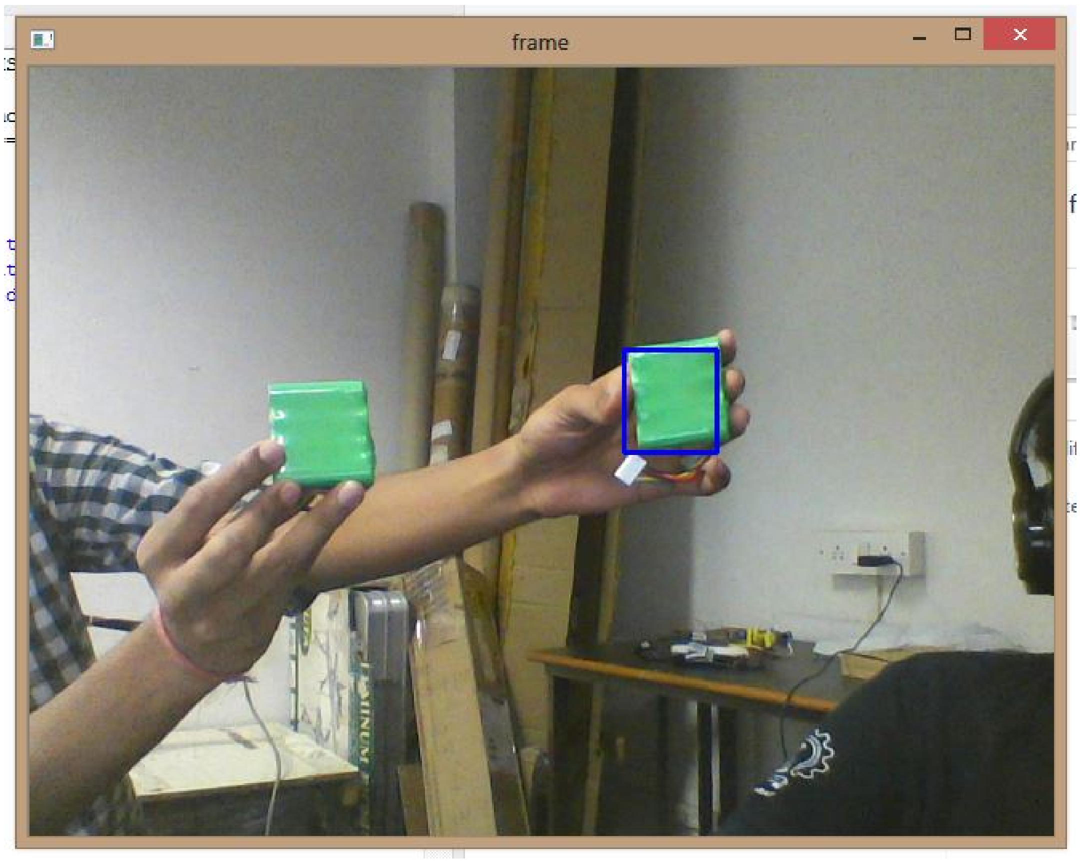


**Figure 17: Testing algorithm with similar colored object.**

## 8 Limitations

- Suppose there are two similar objects P and Q,and we are tracking object P and it goes out of the frame,If in a case Object Q comes into the frame,the algorithm may mistake Q as P as they are both similar objects
- if the Object P comes back with large scale changes,then the algorithm may not re-track it.

## 9 References

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
2. http://vgl-ait.org/mdailey/uploads/publication_file/filename/106/Basit-CAMSHIFT.pdf