# Tutorial - Re-recognizing the object if it escapes and comes back into the frame.

e-Yantra Team

July 5, 2016

## Contents

## 1 Objective

This tutorial will teach you "How to track the object without errors and How to re-recognize the object if it escapes and comes back into 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.

## 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

CAMSHIFT applies traditional mean shift to the backprojection of the appearance model (color histogram) and adds an adaptive region sizing step. The tracking phase simply uses CAMSHIFT, and maintains an adaptive histogram distance threshold. When the target object leaves the scene or is occluded, switch to the detection phase, which proceeds as follows.

- It backprojects the appearance model to determine the consistency of each pixel with the appearance model.
- It binarizes the backprojection to eliminate weakly consistent image regions.

- It generates the connected components using the morpahological filters.
- It generates a set of candidate regions by eliminating inconsistent connected components.
- It finds the most consistent region using histogram comparison between candidate region and target histogram.

To re-recognize the object, two additions to CAMShift Algorithm are, **Suspend Tracking**
CAMSHIFT works extremely well so long as the target appearance remains consistent and distinctive with respect to the background. However, when the target object leaves the scene, is occluded, or impinges a background region with a similar color distribution, the tracking region tends to change rapidly in size, growing into background regions or moving to a different location completely. In these situations, it is important to suspend tracking and attempt to re-detect the target object.

- To achieve this, as a first measure it imposes simple constraints on the target detection windows location and size. If the target objects estimated size or location changes by an amount inconsistent with robot and target dynamics, clearly, the tracker is lost and needs to be reinitialized.
- And before committing to CAMSHIFTs estimate of the target at time, it verifies the quality of the candidate detection region using an adaptive threshold applied to the dissimilarity between the candidate regions color histogram and the appearance model. It uses the default OpenCV histogram comparison function which returns a distance based on the Bhattacharyya coefficient. The resulting distance varies between 0, for identical histograms, to 1, for non-overlapping histograms. The histogram comparison threshold is computed adaptively. Which contains combination of distance measures mean and standard deviation.

**Target Re-detection**
While tracking is suspended, on every frame, It executes the target redetection algorithm. Which is as follows,

- **Backproject the appearance model**: Backprojection of the appearance model simply means that it generates a new image such that, it will have in general several clusters of pixels with high values, indicating some degree of consistency of the region with object histogram.
- **Binarize likelihood image**:

## 6 Experiment

The Python code is given below

```
#import necessary modules
import numpy as np
import cv2
import math
import functions

#defining maximum possible shift in area and center admissible.
MAX_CENTER_SHIFT = 100
MAX_AREA_SHIFT = 5000

#This threshold will be used to not forget sliding window
THRESHOLD = 3

#Variable used to store standard deviation in hostogram distance in previ
prev_standard_deviation = 0

#Variable is used to count number of frames
total_frames = 1

#This kernel will be used in erosion and dilation (opening)
kernel = np.ones((3,3),np.uint8)

#Flag to indicate first frame
first_frame = True

#Termination criteria for CAMShift to stop.
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

#Window to show tracking
cv2.namedWindow('frame', cv2.WINDOW_NORMAL)

###########
#Pass 0 for webcam or Video url for video file.
cap = cv2.VideoCapture(0)
###########

#Loop to get first bounding box
while(1):
    #Reading the frame
    ret,frame = cap.read()
    #Stop program if frame can not be read.
    if not ret:
        cap.release()
        cv2.destroyAllWindows()
        exit(0)

    #Show frame
    cv2.imshow('frame',frame)

    #Show some messages
    if first_frame:
        print 'press p to give roi'
        print 'press ESC to exit'
        first_frame = False

    k = cv2.waitKey(1)
    #If user presses p, get track window
    if k == ord('p'):
        (c,r,w,h) = functions.get_track_window(frame.copy())
        break

    #If user presses escape, exit program
    elif k == 27:
        cap.release()
        cv2.destroyAllWindows()
        exit(0)

#print w*h
#print (c+w/2,r+h/2)

#Get height, width of the frame.
(height_frame,width_frame,channels) = frame.shape

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

#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]

#Mask frame for better result
mask_frame = cv2.inRange(hsv_frame, np.array((0., 60.,32.)), np.array((18

#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,[16,20],[0,180,0,256])
cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX)

#Get 2D frame histogram and normalize it
hist_frame = cv2.calcHist([hsv_frame],[0,1],mask_frame,[16,20],[0,180,0,2
cv2.normalize(hist_frame,hist_frame,0,255,cv2.NORM_MINMAX)

#Get mean of histogram distance
prev_mean = cv2.compareHist(hist_roi,hist_frame,method=cv2.cv.CV_COMP_BHAT

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

#Get track window and apply CAMShift
track_window = (c,r,w,h)
retval, track_window = cv2.CamShift(back_projection, track_window, term_c
(c,r,w,h) = track_window

#Get center and roi
prev_center = (c+w/2,r+h/2)
prev_area = w*h

#print '******'
#print prev_center
#print prev_area
#print prev_mean
#print '******'

#Flag to get whether roi is lost or not.
isLost = False

#Main loop
```

```
while(1):
    #Reading the frame
    ret ,frame = cap.read()

    #Stop program if frame can not be read
    if not ret:
        break

    #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]

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

        #Get 2D frame histogram and normalize it
        hist_frame = cv2.calcHist([hsv_frame],[0,1],mask_frame,[16,20],[0
        cv2.normalize(hist_frame,hist_frame,0,255,cv2.NORM_MINMAX)

        #Get histogram similarity between rame and ROI
        histogram_distance = cv2.compareHist(hist_roi,hist_frame,method=c
        current_mean = prev_mean + ((histogram_distance - prev_mean) / to

        #Get Standard Deviation of Histogram distance
        current_standard_deviation = math.sqrt((((total_frames-2) * (prev

        #Threshold for Histogram Distance
        adaptive_threshold = (current_mean + THRESHOLD * current_standard
        #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'

        else:
            #Circulating variables
            prev_mean = current_mean
            prev_standard_deviation = current_standard_deviation

            #Get ROI back projection on frame
            back_projection = cv2.calcBackProject([hsv_frame],[0,1],hist_

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

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

            #print '**********'
            #print current_area
            #print current_center
            #print '**********'

            #ROI lost if
            if c<=0 or r<=0 or w<=0 or h<=0 or abs(current_area-prev_area
                isLost = True
                print 'lost center or area'

            else:
                '''
                #Reinitialize ROI histogram after each 50 frames
                if total_frames % 500 == 0:
                    if h > 50 and w>50:
                        hsv_roi = hsv_frame[r+15:r+h-15, c+15:c+w-15]
                        mask_roi = mask_frame[r+15:r+h-15, c+15:c+w-15]
                    else:
                            hsv_roi = hsv_frame[r-10:r+40, c-10:c+40]
                            mask_roi = mask_frame[r-10:r+40, c-10:c+40]

                        hist_roi = cv2.calcHist([hsv_roi],[0,1],mask_roi,[16,
                        cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX
                    '''

                    #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

        #Procedure to re-recognize the object
        else:
            #Get ROI back projection on frame
            back_projection1 = cv2.calcBackProject([hsv_frame],[0,1],hist_roi

            #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.

            #Apply erosion and dilation
            back_projection1 = cv2.morphologyEx(back_projection1, cv2.MORPH_C

            #Show back_projection
            cv2.imshow('back_projection_recognization',back_projection1)

            #Get contours
            contours, hierarchy = cv2.findContours(back_projection1,cv2.RETR_

            #Get good contours
            good = []
            for i in range(len(contours)):
                #area = cv2.contourArea(contours[i])
                #if area > 0.1 * prev_area:
                good.append(i)

            '''
            #if there is only one contour that is ROI
            if len(good) == 1:
                print 'a'
                c,r,w,h = cv2.boundingRect(contours[good[0]])
                track_window = (c,r,w,h)
                prev_area = w*h
                prev_center = (c+w/2,r+h/2)
                if c > 0 and r > 0 and w > 0 and h > 0:
                    isLost = False
                    print 'found'
            '''

            #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
                for i in good:
                    bounding_box[i][0],bounding_box[i][1],bounding_box[i][2],
                    roi_detected = hsv_frame[bounding_box[i][1]:bounding_box[
                    mask_roi_detected = cv2.inRange(roi_detected, np.array((0
                    hist_roi_detected = cv2.calcHist([roi_detected],[0,1],ma
                    cv2.normalize(hist_roi_detected,hist_roi_detected,0,255,c
                    hist_dist[i] = cv2.compareHist(hist_roi,hist_roi_detected

                #Get closest histogram distance
                min_hist_dist = min(hist_dist)

                #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]
                    track_window = (c,r,w,h)
                    prev_area = w*h
                    prev_center = (c+w/2,r+h/2)
```

```
                #If valid bounding box
                if c > 0 and r > 0 and w > 0 and h > 0:
                    isLost = False
                    print 'found'

        #Show frame
        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xff == 27: #ascii value for escape key
            break

#Release camera
cap.release()
cv2.destroyAllWindows()
```

## 7 Exercise

Re-detection and tracking of an object is shown below.

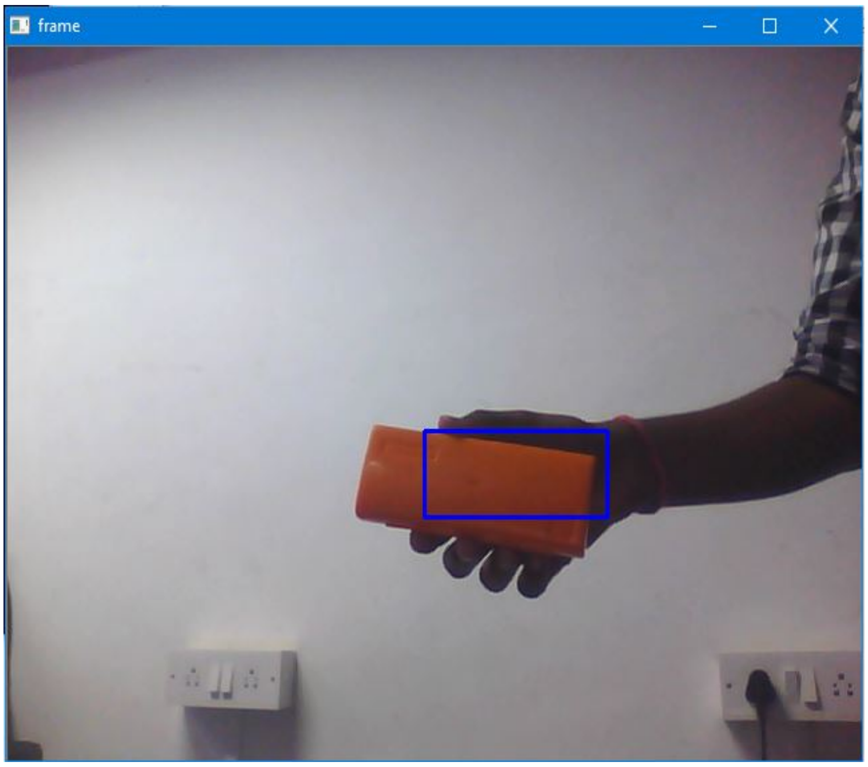

Figure 1: Object in the frame

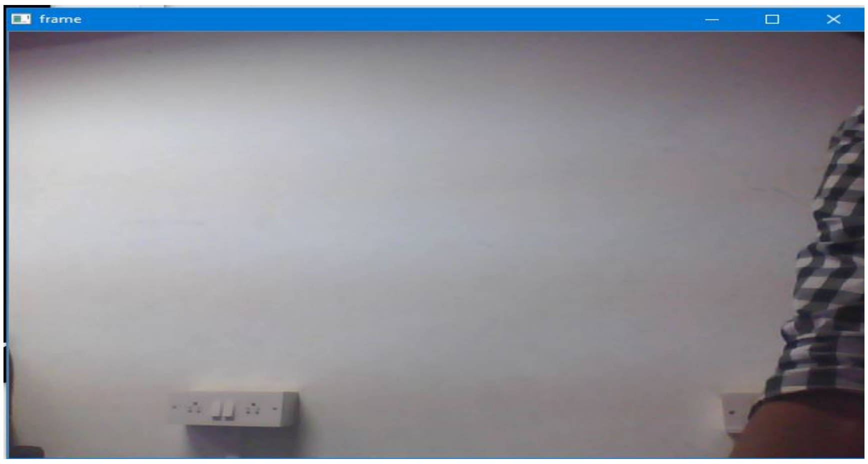

Figure 2: Object lost in the frame

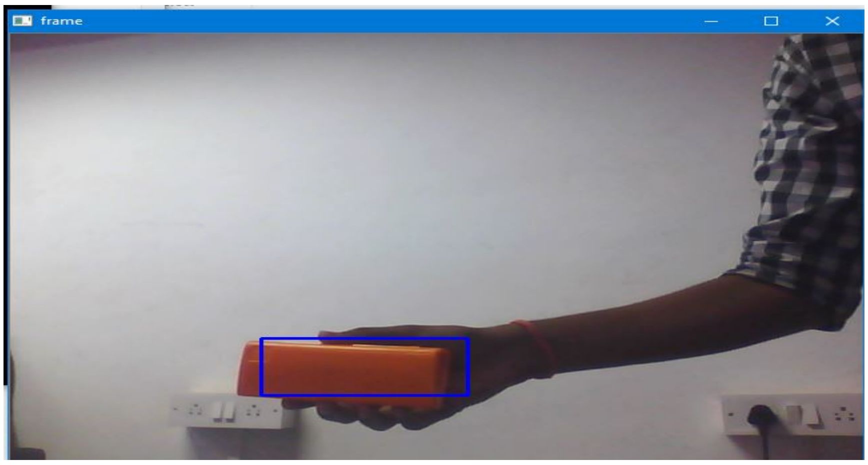

Figure 3: Re-detection of object in the frame

## 8 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