Meanshift and Camshift

Prev Tutorial: How to Use Background Subtraction Methods

Next Tutorial: Optical Flow

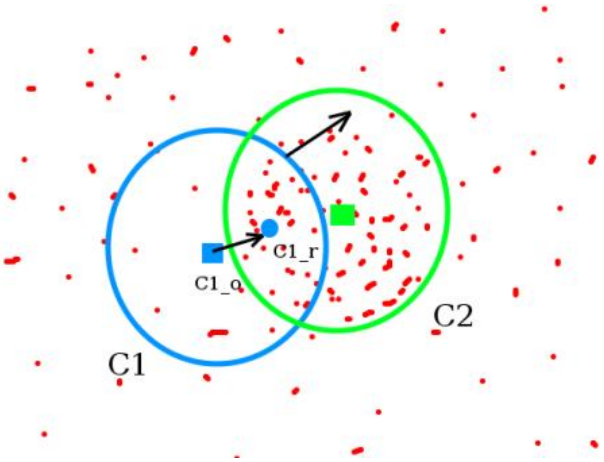
Goal

In this chapter,

- We will learn about the Meanshift and Camshift algorithms to track objects in videos.

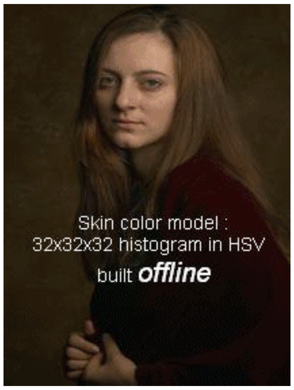
Meanshift

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). It is illustrated in the simple image given below:



image

The initial window is shown in blue circle with the name "C1". Its original center is marked in blue rectangle, named "C1_o". But if you find the centroid of the points inside that window, you will get the point "C1_r" (marked in small blue circle) which is the real centroid of the window. Surely they don't match. So move your window such that the circle of the new window matches with the previous centroid. Again find the new centroid. Most probably, it won't match. So move it again, and continue the iterations such that the center of window and its centroid falls on the same location (or within a small desired error). So finally what you obtain is a window with maximum pixel distribution. It is marked with a green circle, named "C2". As you can see in the image, it has maximum number of points. The whole process is demonstrated on a static image below:



image

So we normally pass the histogram backprojected image and initial target location. When the object moves, obviously the movement is reflected in the histogram backprojected image. As a result, the meanshift algorithm moves our window to the new location with maximum density.

Meanshift in OpenCV

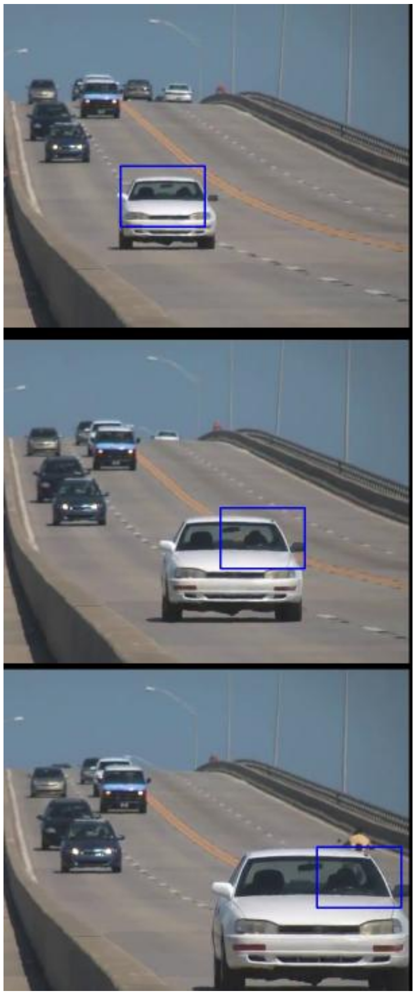
C++ Python Java

To use meanshift in OpenCV, first we need to setup the target, find its histogram so that we can backproject the target on each frame for calculation of meanshift. We also need to provide an initial location of window. For histogram, only Hue is considered here. Also, to avoid false values due to low light, low light values are discarded using **cv.inRange()** function.

- **Downloadable code**: Click here
- **Code at glance:**

```
import numpy as np
import cv2 as cv
import argparse

parser = argparse.ArgumentParser(description='This sample demonstrates the meanshift algorithm. \
                                              The example file can be downloaded from: \
                                              https://www.bogotobogo.com/python/OpenCV_Python/images/mean_shift_tracking/slow_traffic_small
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()

cap = cv.VideoCapture(args.image)

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)

# set up the ROI for tracking
roi = frame[y:y+h, x:x+w]
hsv_roi =  cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv.normalize(roi_hist,roi_hist,0,255,cv.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by at least 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret, frame = cap.read()

    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv.meanShift(dst, track_window, term_crit)

        # Draw it on image
        x,y,w,h = track_window
        img2 = cv.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv.imshow('img2',img2)

        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
    else:
        break
```

Three frames in a video I used is given below:

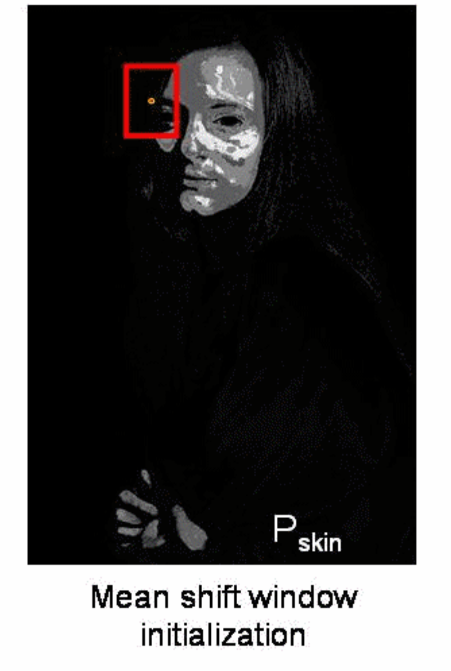


image

Camshift

Did you closely watch the last result? There is a problem. Our window always has the same size whether the car is very far or very close to the camera. That is not good. We need to adapt the window size with size and rotation of the target. Once again, the solution came from "OpenCV Labs" and it is called CAMshift (Continuously Adaptive Meanshift) published by Gary Bradsky in his paper "Computer Vision Face Tracking for Use in a Perceptual User Interface" in 1998 **[35]** .

It applies meanshift first. Once meanshift converges, it updates the size of the window as, $s = 2 \times \sqrt{\frac{M_{00}}{256}}$. It also calculates the orientation of the best fitting ellipse to it. Again it applies the meanshift with new scaled search window and previous window location. The process continues until the required accuracy is met.



image

Camshift in OpenCV

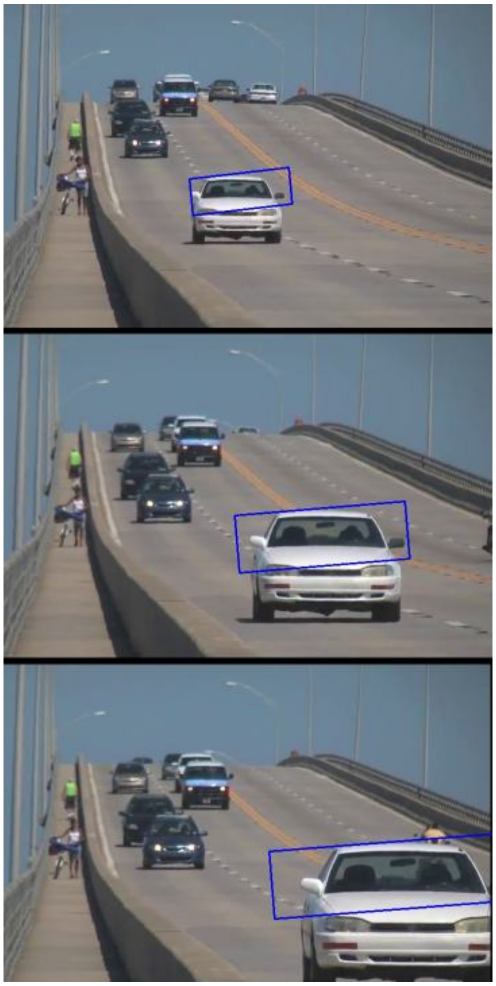
C++ | Python | Java

It is similar to meanshift, but returns a rotated rectangle (that is our result) and box parameters (used to be passed as search window in next iteration). See the code below:

- **Downloadable code**: Click here
- **Code at glance:**

```
import numpy as np
import cv2 as cv
import argparse

parser = argparse.ArgumentParser(description='This sample demonstrates the camshift algorithm. \
                                              The example file can be downloaded from: \
                                              https://www.bogotobogo.com/python/OpenCV_Python/images/mean_shift_tracking/slow_traffic_small
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()

cap = cv.VideoCapture(args.image)

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)

# set up the ROI for tracking
roi = frame[y:y+h, x:x+w]
hsv_roi =  cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv.normalize(roi_hist,roi_hist,0,255,cv.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by at least 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret, frame = cap.read()

    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply camshift to get the new location
        ret, track_window = cv.CamShift(dst, track_window, term_crit)

        # Draw it on image
        pts = cv.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv.polylines(frame,[pts],True, 255,2)
        cv.imshow('img2',img2)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
    else:
        break
```

Three frames of the result is shown below:



image

Additional Resources

1. French Wikipedia page on Camshift. (The two animations are taken from there)
2. Bradski, G.R., "Real time face and object tracking as a component of a perceptual user interface," Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Workshop on , vol., no., pp.214,219, 19-21 Oct 1998

Exercises

1. OpenCV comes with a Python sample for an interactive demo of camshift. Use it, hack it, understand it.

Generated on Sat Mar 26 2022 02:57:19 for OpenCV by doxygen 1.8.13