OpenCV
Part 3

OpenCV

35. Object Tracking

Goals

- Learn basic object tracking techniques
  - Optical Flow
  - MeanShift and CamShift
- Understand more advanced tracking
  - Review Built-in Tracking APIs

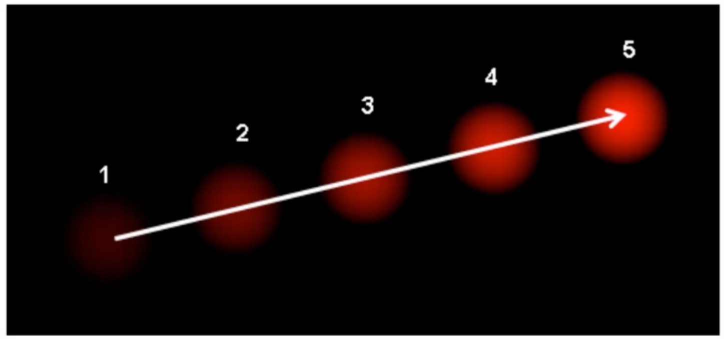
https://nanonets.com/blog/optical-flow/

35. Object Tracking

- Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera
- Optical Flow Analysis has a few assumptions:
  - ***The pixel intensities of an object do not change between consecutive frames***
  - ***Neighboring pixels have similar motion***
- The optical flow methods in OpenCV will first take in a given set of points and a frame
- Then it will attempt to find those points in the next frame
- It is up to the user to supply the points to track

35. Object Tracking

- Consider the following image
- Here we display a five frame clip of a ball moving up and towards the right



- Note that given just this clip, we can not determine if the ball is moving, or if the camera moved down and to the left
- Using OpenCV we pass in the previous frame, previous points and the current frame to the ***Lucas-Kanade function***

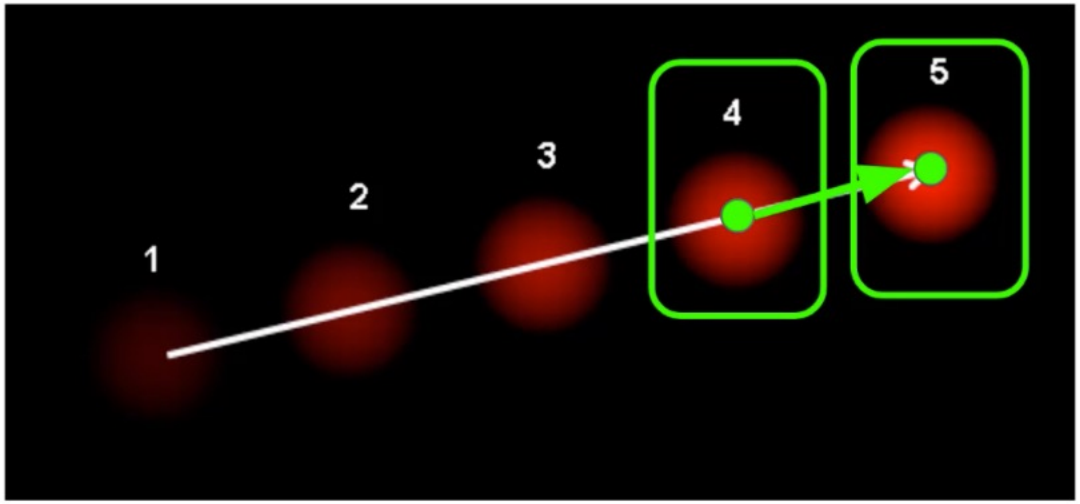
$$I(x,y,t) = I(x+\delta x, y+\delta y, t+\delta t)$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t +$$

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

35. Object Tracking

- Using OpenCV we pass in the previous frame, previous points and the current frame to the Lucas-Kanade function



- The function then attempts to locate the points in the current frame

35. Object Tracking

- The Lucas-Kanade computes optical flow for a sparse feature set
  - Meaning only the points it was told to track
- But what if we wanted to track ***all the points*** in a video?

- We can use Gunner Farneback's algorithm (also build in to OpenCV) to calculate ***dense*** optical flow
- This ***dense*** optical flow will calculate flow for all points in an image
- It will color them ***black*** if no flow (no movement) is detected

36. Object Tracking – Lucas Kanade Optical Flow

```
# Parameters for ShiTomasi corner detection (good features to track paper)
corner_track_params = dict(maxCorners = 10,
                           qualityLevel = 0.3,
                           minDistance = 7,
                           blockSize = 7 )
```

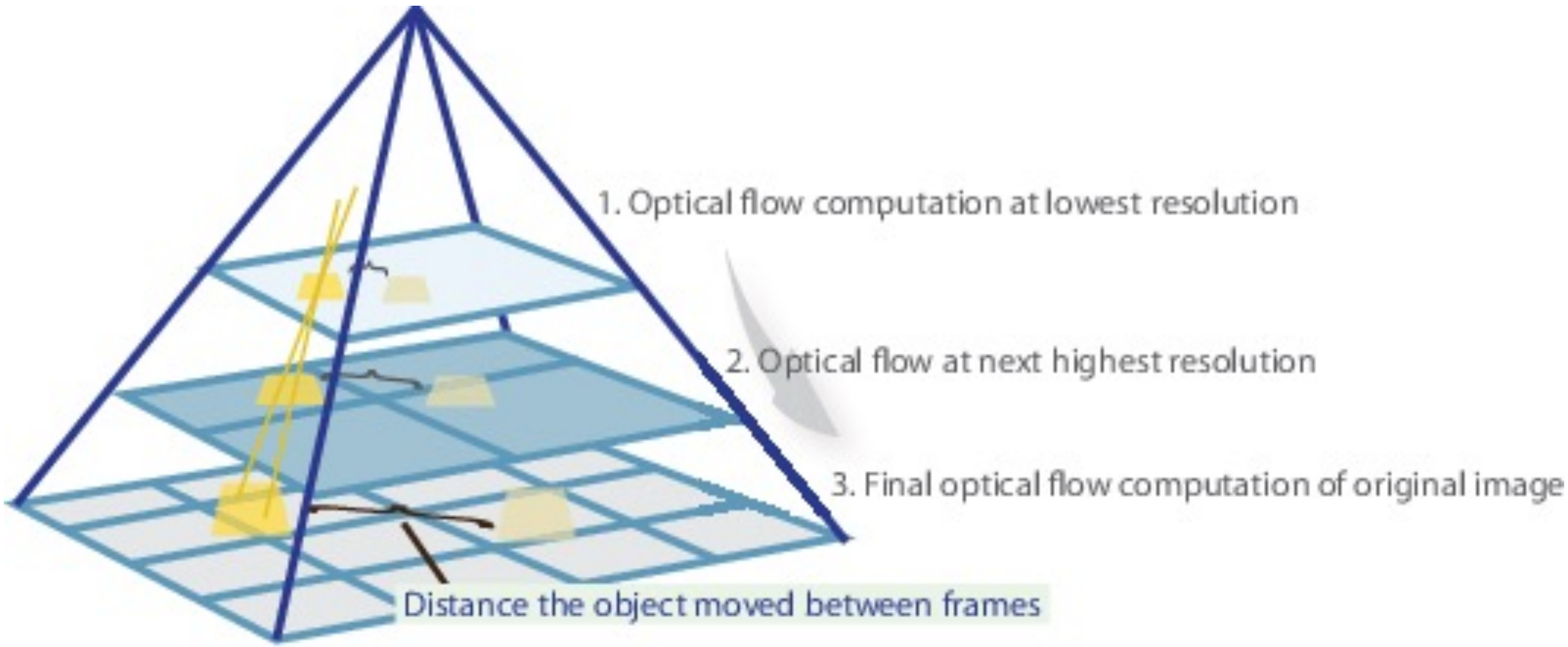
```
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (200,200),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10,0.03))
```

```
# Grabbing the corners
prevPts = cv2.goodFeaturesToTrack(prev_gray, mask = None, **corner_track_params)
# Calculate the Optical Flow on the Gray Scale Frame
nextPts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, frame_gray, prevPts, None,
                                                **lk_params)

# Using the returned status array (the status output)
# status output status vector (of unsigned chars); each element of the vector is set to 1 if
# the flow for the corresponding features has been found, otherwise, it is set to 0.
good_new = nextPts[status==1]
good_prev = prevPts[status==1]
```

36. Object Tracking – Lucas Kanade Optical Flow

- Consider the following image
- Here we display a five frame clip of a ball moving up and towards the right



36. Object Tracking – Lucas Kanade Optical Flow

```
# Calculate the Optical Flow on the Gray Scale Frame
nextPts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, frame_gray, prevPts, None,
                                                **lk_params)

# Using the returned status array (the status output)
# status output status vector (of unsigned chars); each element of the vector is set to 1 if
# the flow for the corresponding features has been found, otherwise, it is set to 0.
good_new = nextPts[status==1]
good_prev = prevPts[status==1]


# Use ravel to get points to draw lines and circles
for i,(new,prev) in enumerate(zip(good_new,good_prev)):

    x_new,y_new = new.ravel()
    x_prev,y_prev = prev.ravel()

    # Lines will be drawn using the mask created from the first frame
    mask = cv2.line(mask, (x_new,y_new),(x_prev,y_prev), (0,255,0), 3)

    # Draw red circles at corner points
    frame = cv2.circle(frame,(x_new,y_new),8,(0,0,255),-1)

# Display the image along with the mask we drew the line on.
img = cv2.add(frame,mask)
cv2.imshow('frame',img)
```

36. Object Tracking – Dense Optical Flow

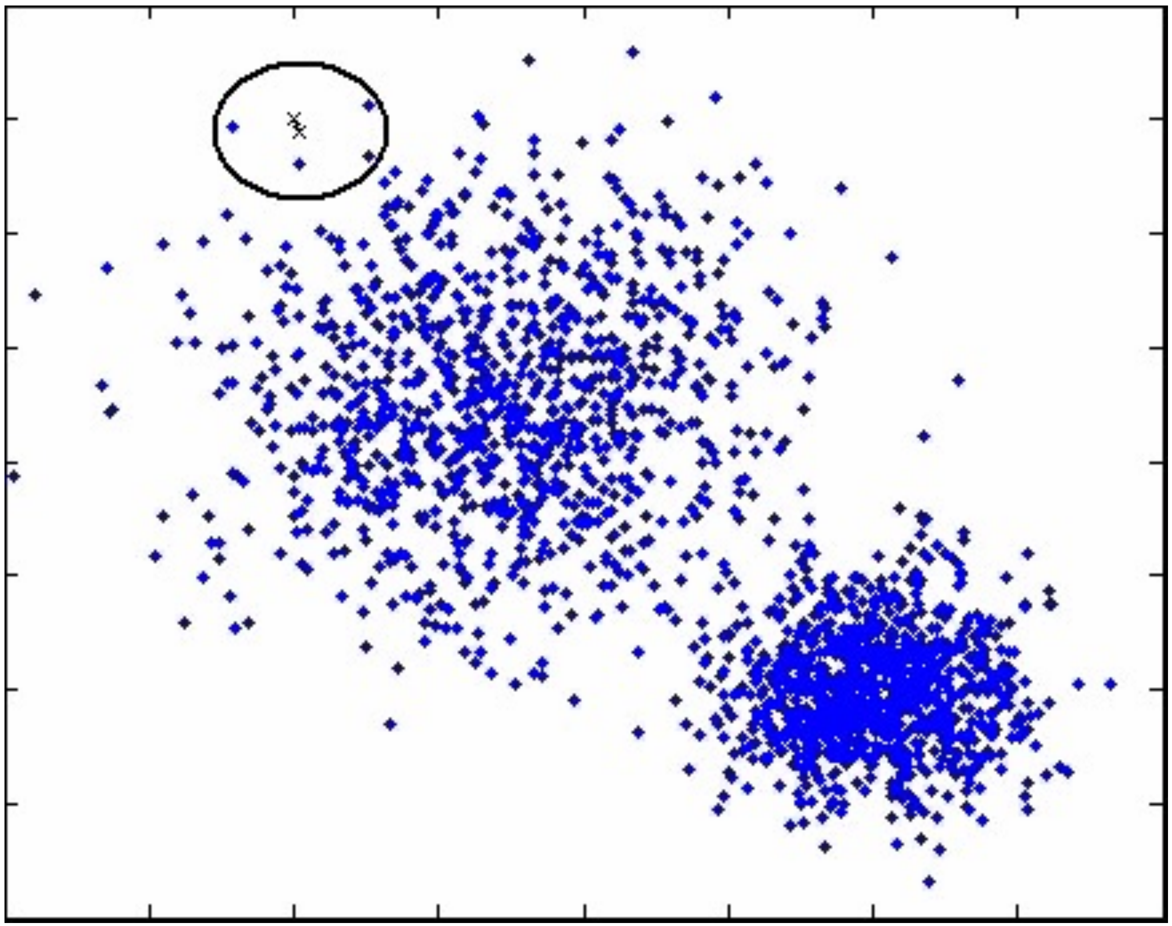
```
# Check out the markdown text above for a break down of these paramters, most of these are just
flow = cv2.calcOpticalFlowFarneback(prvsImg,nextImg, None, 0.5, 3, 15, 3, 5, 1.2, 0)

# Color the channels based on the angle of travel
# Pay close attention to your video, the path of the direction of flow will determine color!
mag, ang = cv2.cartToPolar(flow[:,:,0], flow[:,:,1],angleInDegrees=True)
hsv_mask[:,:,0] = ang/2
hsv_mask[:,:,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
```

```
# Set the Previous image as the next iamge for the loop
prvsImg = nextImg
```

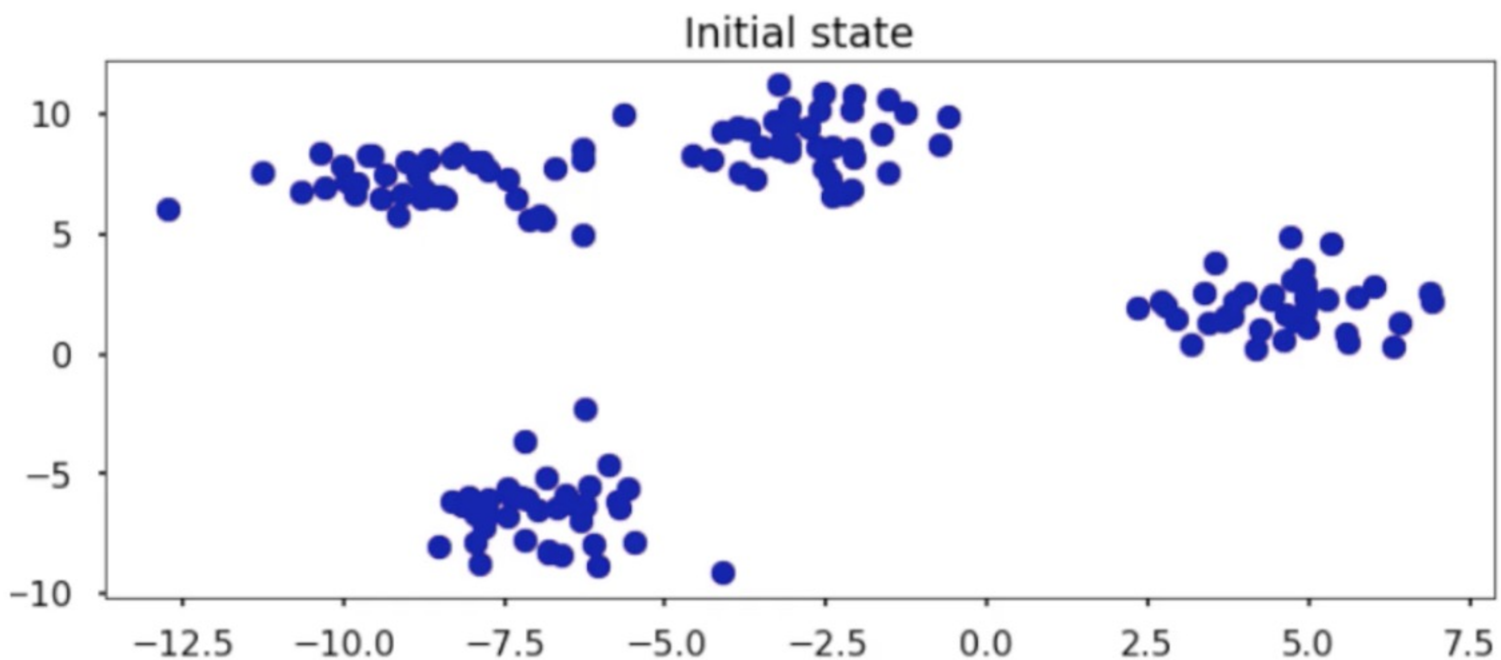
37. Object Tracking – MeanShift and CamShift

- Some of the most basic tracking methods are ***MeanShift*** and ***CAMShist***
- https://youtu.be/RG5uV_h50b0
- https://www.youtube.com/watch?v=iBOlbs8i7Og



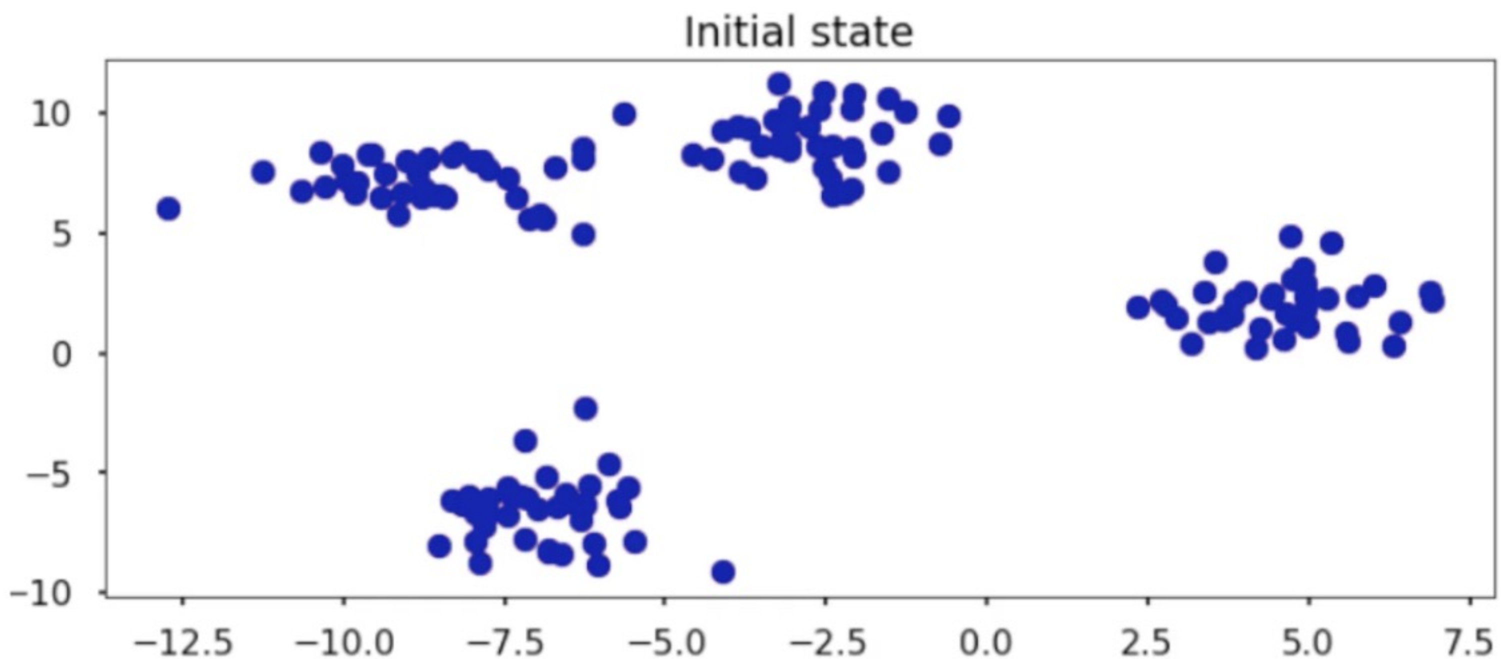
37. Object Tracking – MeanShift and CamShift

- Imagine we have a set of points and we wanted to assign them into clusters
- We take all our data points and stack red and blue points on them. (You can't see the red points underneath)



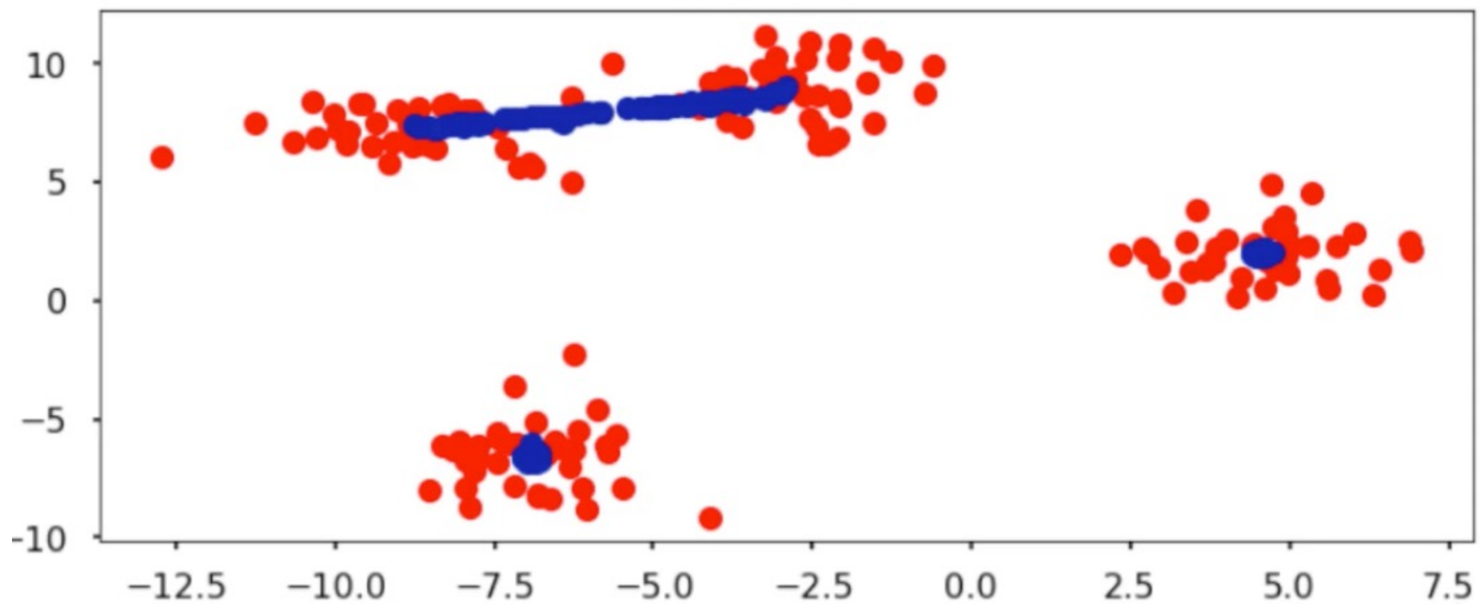
37. Object Tracking – MeanShift and CamShift

- The direction to the closest cluster centroid is determined by where most of the points nearby are at
- So each iteration each blue point will move close to where the ***most points*** are at, whi ch is or will lead to the cluster center



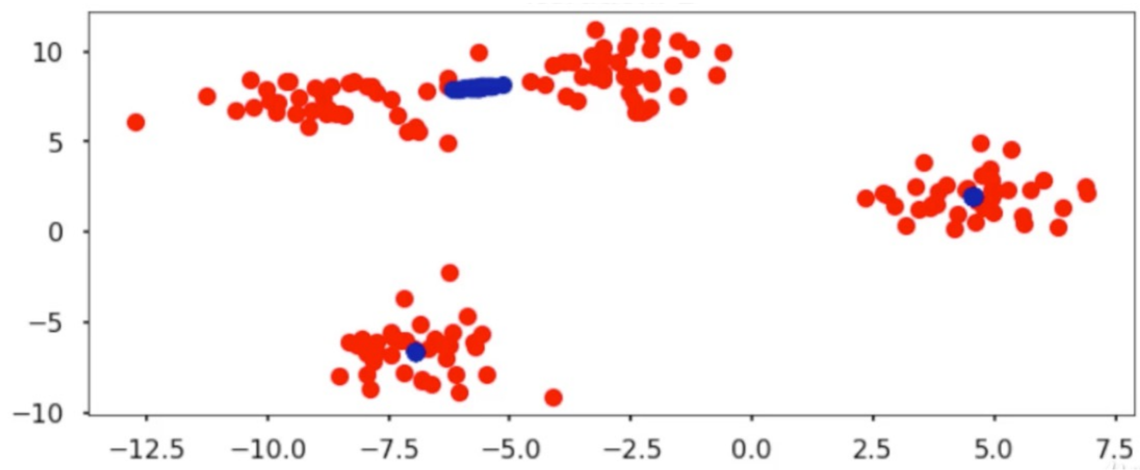
37. Object Tracking – MeanShift and CamShift

- The red and blue datapoints overlap completely in the first iteration before the Mean shift algorithm starts
- At the end of iteration 1, all the blue points move towards the clusters. Here is appears there will be either 3 or 4 clusters

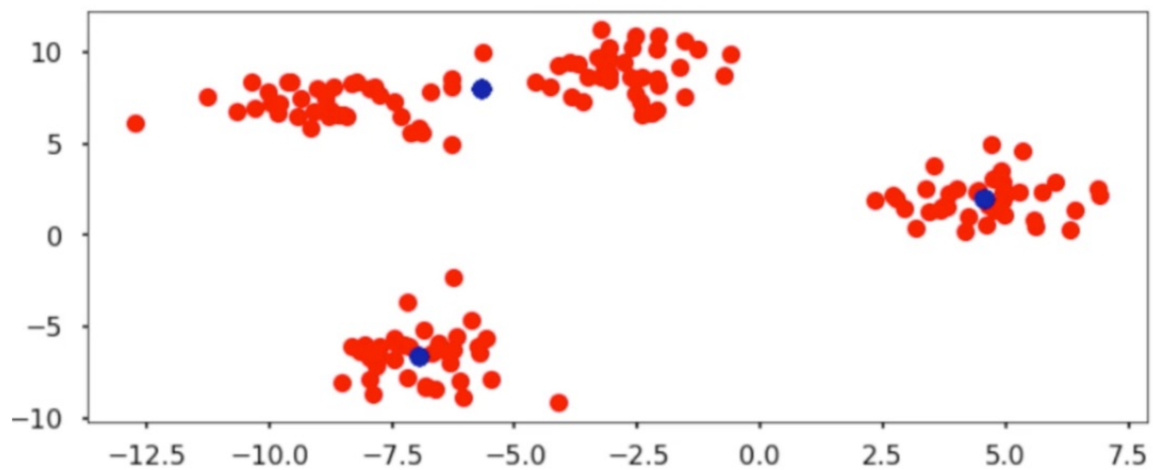


37. Object Tracking – MeanShift and CamShift

- The bottom clusters have begun to reach convergence

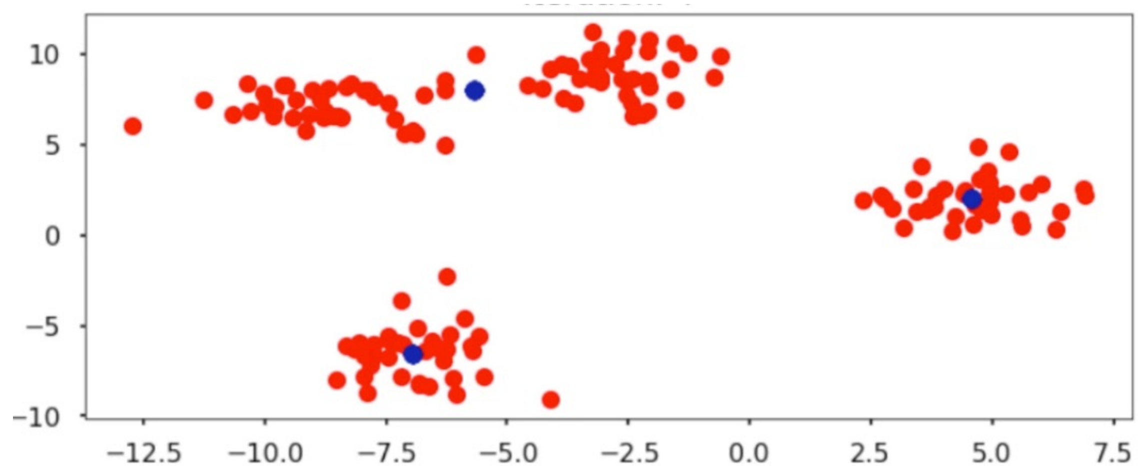


- MenShift found 3 clusters by the third iteration

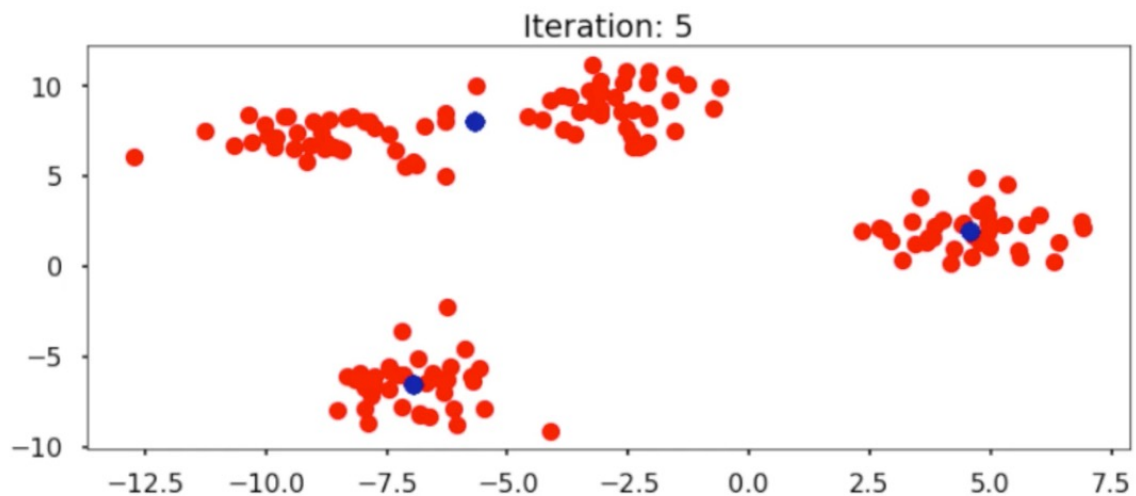


37. Object Tracking – MeanShift and CamShift

- After subsequent iterations, the cluster means have stopped moving

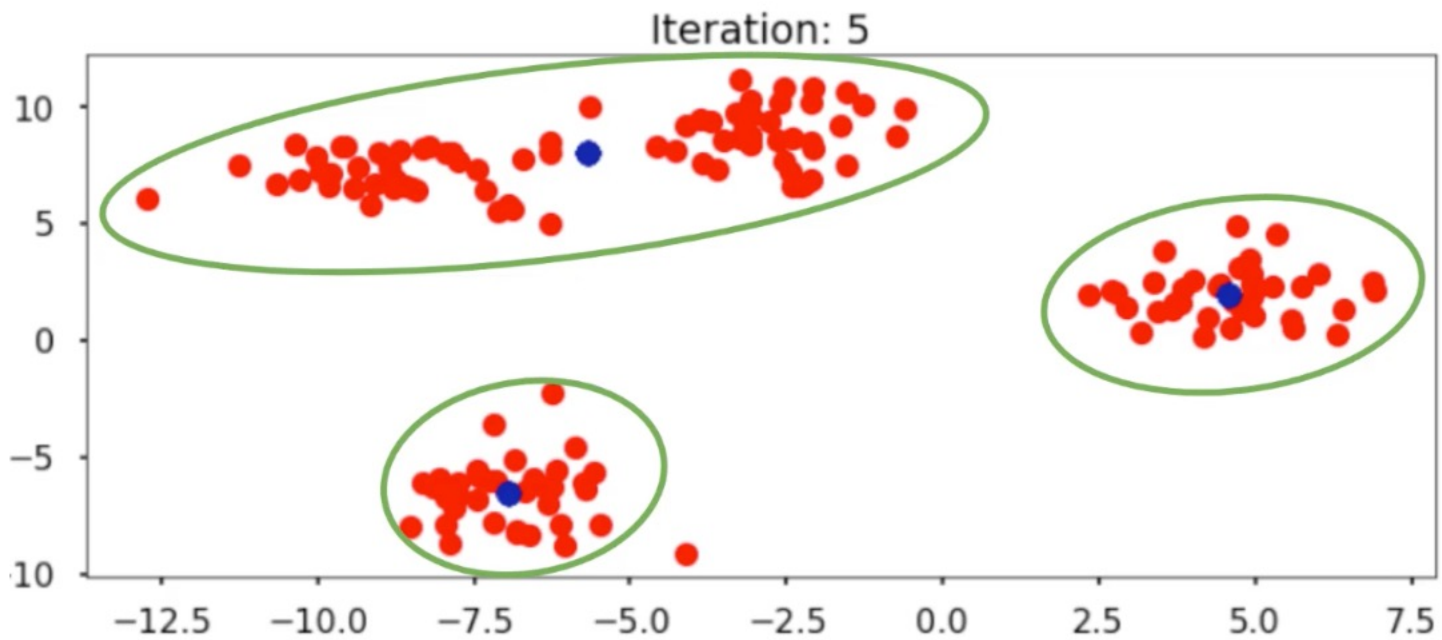


- All clusters have converged and there is no more movement



37. Object Tracking – MeanShift and CamShift

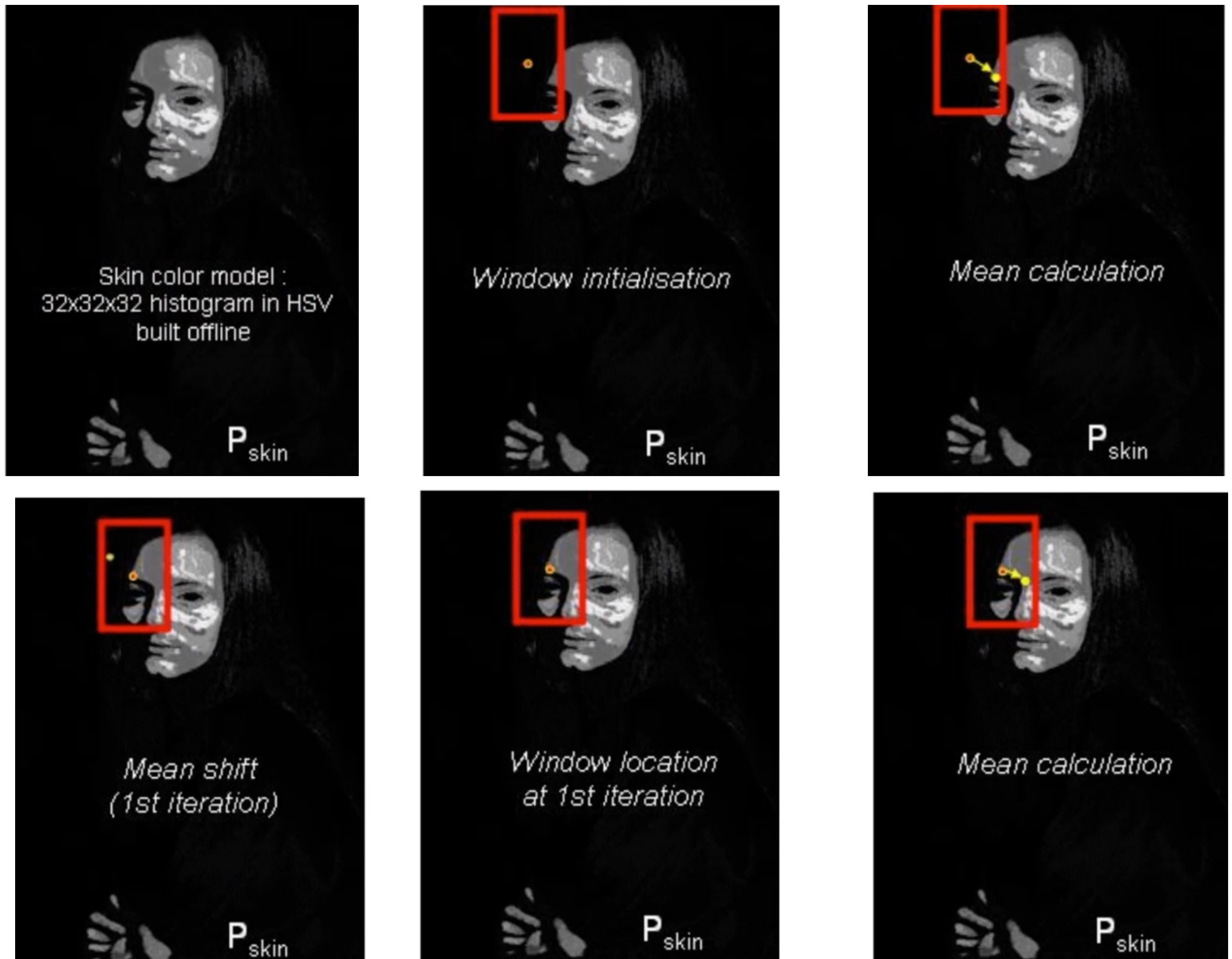
- Identified Clusters:



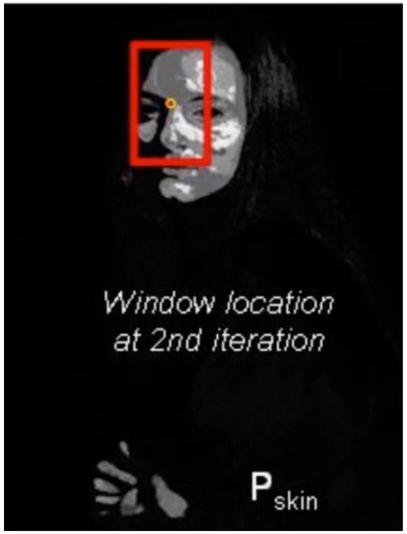
- It won't always detect what may be more "reasonable"
- It may have been more reasonable to detect 4 clusters in the previous situation

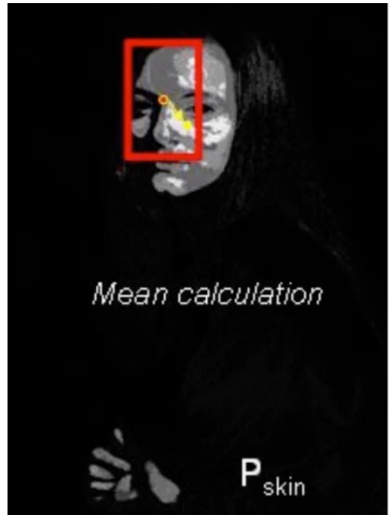
37. Object Tracking – MeanShift and CamShift

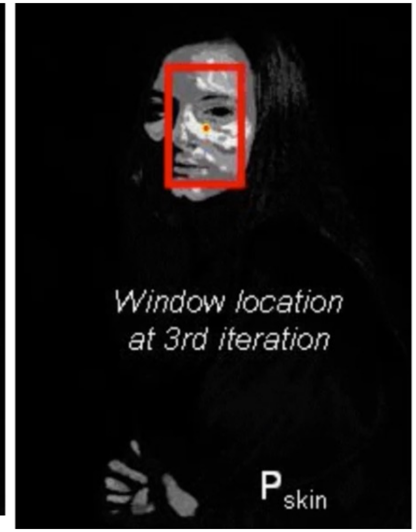
- ***MeanShift*** can be given a target to track, calculate the color histogram of the target are a, and then keep sliding the tracking window to the closest match (the cluster center)

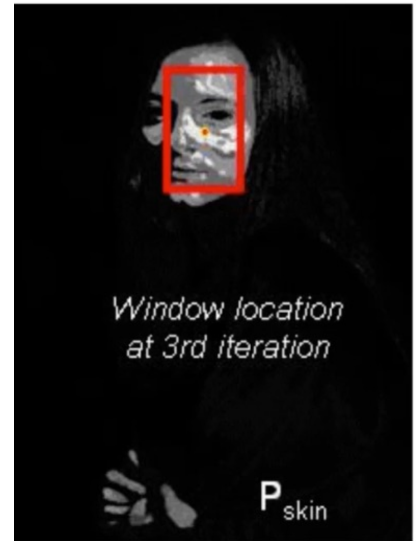


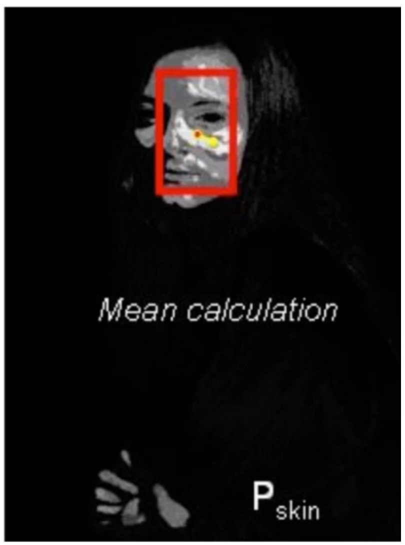
37. Object Tracking – MeanShift and CamShift

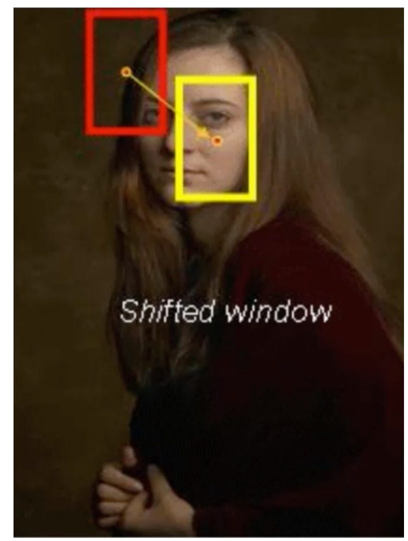










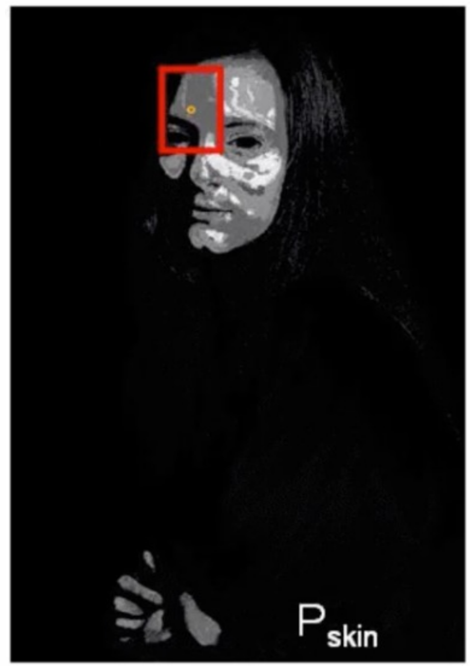


37. Object Tracking – MeanShift and CamShift

- Just using MeanShift won't change the window size if the target moves away or towards the camera
- We can use ***CAMshift*** to ***update the size*** of the window

Mean shift window initialization

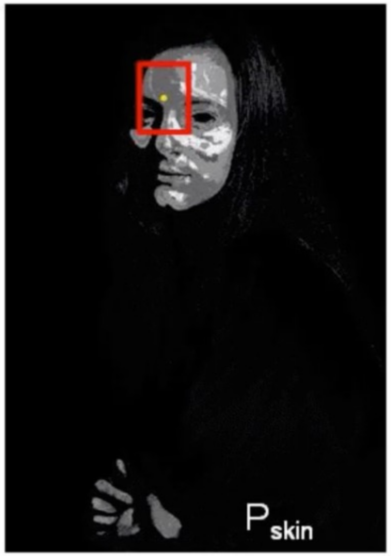
Mean shift 1st iteration



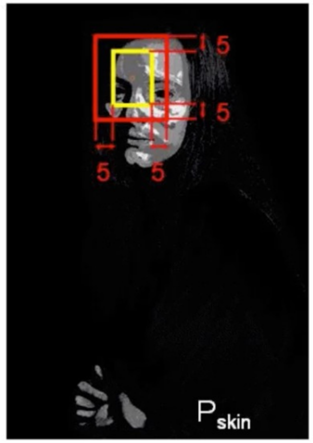
Mean shift 2nd iteration

Mean shift 3rd iteration

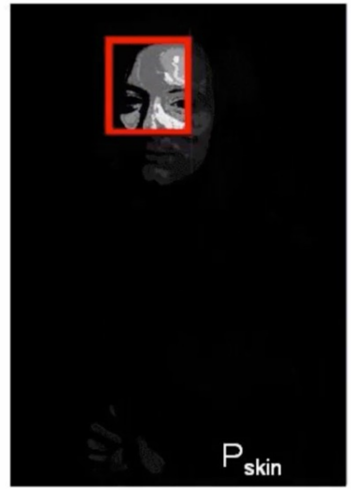
37. Object Tracking – MeanShift and CamShift



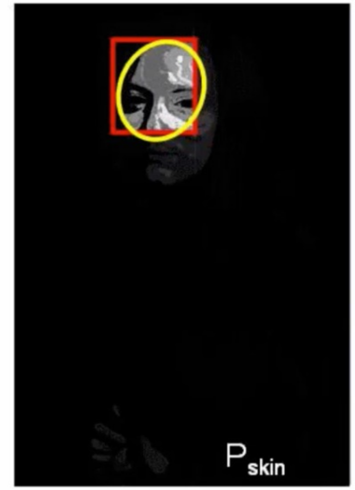
Mean shift converged



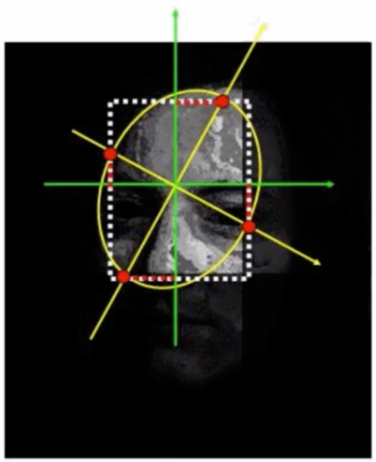
ROI for ellipse estimation :
± 5 pixels width and height



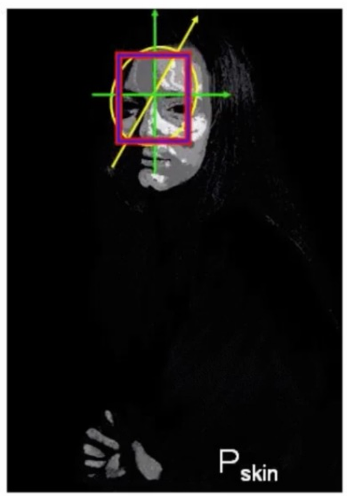
ROI : region of interest
for ellipse estimation



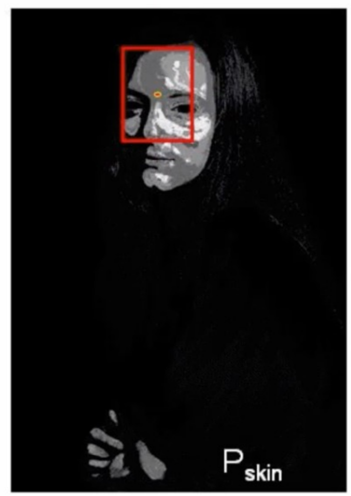
Ellipse computation based
on P_{skin} second order moments



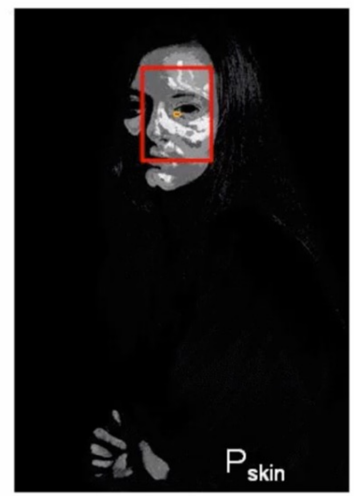
New mean shift window
from ellipse axis projection



Updated mean shift window

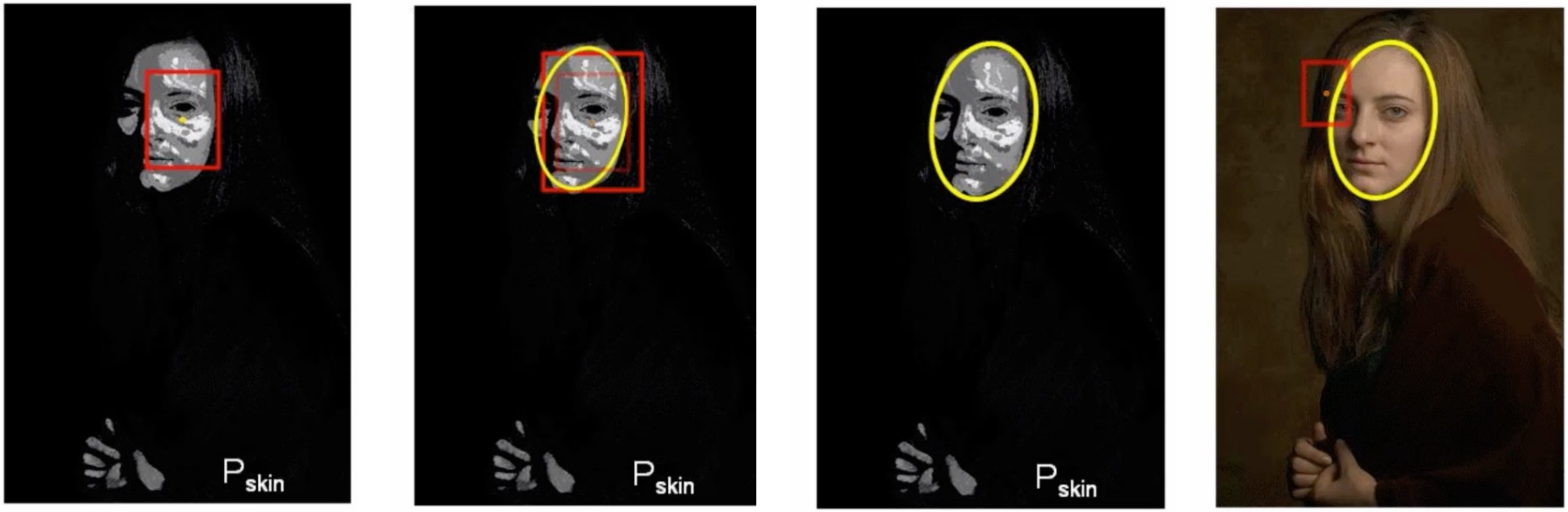


Mean shift again



Mean shift
1st iteration

37. Object Tracking – MeanShift and CamShift



Mean shift
2nd iteration = convergence

Ellipse calculation

Repeat until convergence

Result on a still image

38. Object Tracking – MeanShift Examples

```
# We will first detect the face and set that as our starting box.
face_cascade = cv2.CascadeClassifier('../DATA/haarcascades/haarcascade_frontalface_default.xml')
face_rects = face_cascade.detectMultiScale(frame)
```

```
# Convert this list of a single array to a tuple of (x,y,w,h)
(face_x,face_y,w,h) = tuple(face_rects[0])
track_window = (face_x,face_y,w,h)
# set up the ROI for tracking
roi = frame[face_y:face_y+h, face_x:face_x+w]
```

```
# Calculate the Back Projection based off the roi_hist created earlier
dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

# Apply meanshift to get the new coordinates of the rectangle
ret, track_window = cv2.meanShift(dst, track_window, term_crit)
```

38. Object Tracking – MeanShift and CamShift Examples

```
# We will first detect the face and set that as our starting box.
face_cascade = cv2.CascadeClassifier('../DATA/haarcascades/haarcascade_frontalface_default.xml')
face_rects = face_cascade.detectMultiScale(frame)
```

```
# Convert this list of a single array to a tuple of (x,y,w,h)
(face_x,face_y,w,h) = tuple(face_rects[0])
track_window = (face_x,face_y,w,h)
# set up the ROI for tracking
roi = frame[face_y:face_y+h, face_x:face_x+w]
```

```
# Calculate the Back Projection based off the roi_hist created earlier
dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

# Apply meanshift to get the new coordinates of the rectangle
ret, track_window = cv2.meanShift(dst, track_window, term_crit)
```

```
# Apply Camshift to get the new coordinates of the rectangle
ret, track_window = cv2.CamShift(dst, track_window, term_crit)

# Draw it on image
pts = cv2.boxPoints(ret)
pts = np.int0(pts)
img2 = cv2.polylines(frame,[pts],True, (0,0,255),5)
cv2.imshow('img2',img2)
```