**Object tracking using Raspberry Pi and OpenCV**

**Object tracking** is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing.

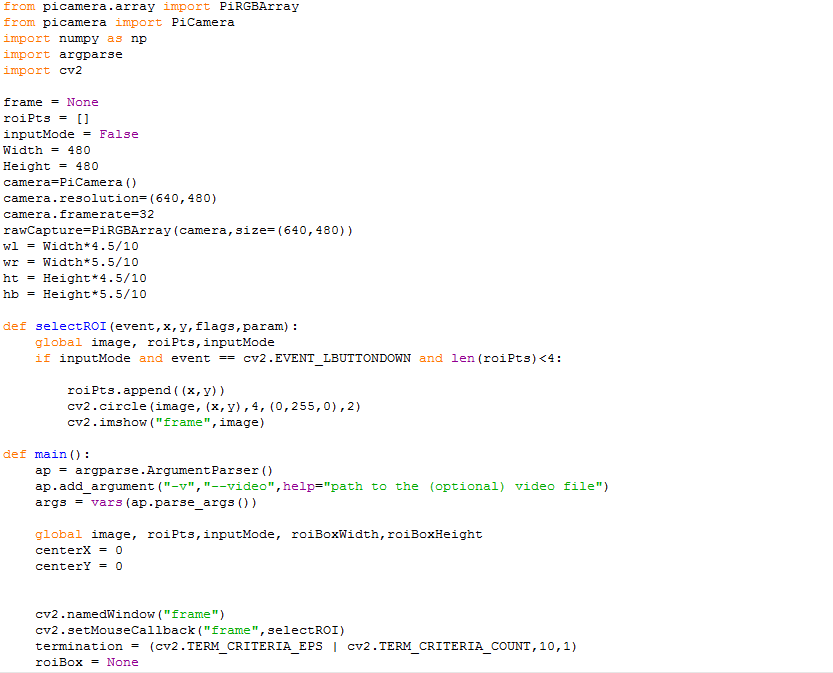
Computer vision and object tracking has for long been an active research area. There are a multitude of algorithms developed to efficiently detect and track an object, each with its own advantages and disadvantages. However, to implement an object tracking algorithm on a computationally restricted system like the **Raspberry Pi**, we need a fast and efficient algorithm. It must be able to track in real time and yet not absorb a major share of computational resources.

One such algorithm is the Continuously Adaptive Mean shift (CAMSHIFT) algorithm. CAMSHIFT tracks objects using a probability distribution of the object in a video scene. The closest existing algorithm to CAMSHIFT is known as the mean shift algorithm. The mean shift algorithm is a non-parametric technique that climbs the gradient of a probability distribution to find the nearest dominant mode (peak).

For our purposes we are using the OpenCV open source library along with Python as our programming language to interact with the Raspberry Pi. The installation instructions for OpenCV on Raspberry Pi has been given in an alternate file.

After installing OpenCV on the Pi, follow the following tutorial to implement color-based object tracking using the Pi Camera module on Raspberry Pi

**The CODE:**



In **Lines 1-5** we simply start by importing the packages that we’ll need. We’ll use NumPy for numerical processing, argparse to parse our command line arguments, and cv2 to bind with the OpenCV library.

In **Lines 7-9** we define three global variables that we’ll be using throughout the rest of this script.

The first is frame, which is the current frame of the video that we are processing.

The second is roiPts, which is the list of points corresponding to the Region of Interest (ROI) in our video.

Finally, we have inputMode, which is simply used as a boolean flag, indicating whether or not we are currently selecting the object we want to track in the video.

In **Lines 12-15** we just initialize the camera module of the Raspberry Pi and set its resolution and frame rate.

The PiRGBArray function returns a 3-dimensional RGB array from an RGB capture and stores it in our rawcapture variable.

The rest of the variables are just the specifications of our bounding box and can be arbitrarily initialized to any value.

Finally, to select the Region of Image (ROI) to track, we define a separate function to do that named as selectROI()

**Line 22** simply grabs reference to our current frame, list of ROI points, and input mode indicator.

Then, on **Line 23**, we make a check for three conditions. In order to select the ROI, these three conditions must hold:

1. We are currently in input mode (which allows us to select the ROI of the object we want to track in the video).
2. The left button of the mouse was clicked, as indicated by the cv2.EVENT\_LBUTTONDOWN constant, giving us the *(x, y* coordinates of the click.
3. We must have less than four points currently in our list of ROI points. In this example, we’ll assume that our ROI can be representing as a bounding box with four points.

Assuming that these conditions hold, we then append the (x, y) location of the click to our list of ROI points on **Line 25**, draw a circle representing the location of the mouse click on **Line 26**, and then display the updated frame on **Line 27**.

We then use this selectROI function inside our main function to select the region to track during the live video feed.

Inside the main function, **Lines 30-32** we parse our command line arguments. We’ll need only a single (optional) switch here, --video, which is the path to where our video resides on disk. Otherwise we can also run the file directly from the shell and use live camera feed instead.

After we have parsed our command line arguments, we grab our reference to our global variables of the current frame, list of ROI points, and input mode indicator on **Line 34**

**Line 35-36 is used to initialize two new variables namely the X,Y coordinates of the center of the bounding box.**

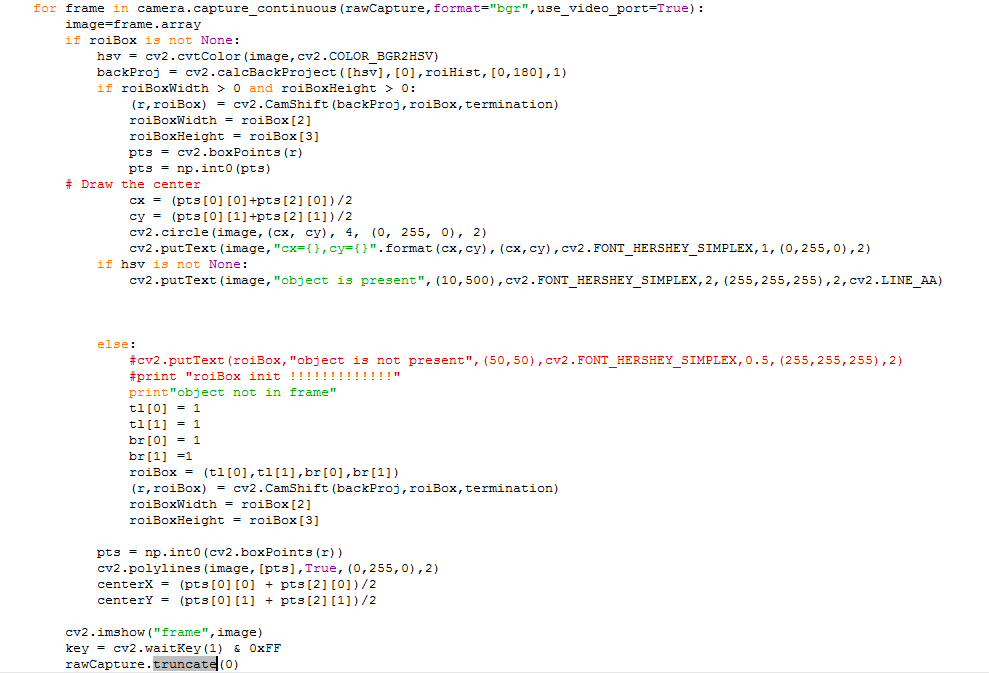
We are going to have to select the bounding box of the object we want to track in our video. In order to do this, we’ll need to register an event callback. We do this on **Line 39 and 40** by creating a window named “frame” and indicating that any mouse events applied to the “frame” window should be handled by our selectROI function.

Finally, we initialize a tuple containing our termination criteria for the CamShift algorithm on **Line 41.**

The CamShift algorithm is iterative, meaning that it seeks to optimize the tracking criterion. In this case, we’ll set the termination criterion to perform two checks.

The first check is the epsilon associated with the centroids of our selected ROI and the tracked ROI according to the CamShift algorithm. If the tracked centroid has not changed by at least one pixel, then terminate the CamShift algorithm.

The second check controls the number of iterations of CamShift. Using more iterations will allow CamShift to (ideally) find a closer centroid match between the selected ROI and the tracked ROI; however, this comes at the cost of runtime. If the iterations are set too high, then we will drop below real-time performance, which is substantially less than ideal in our situation.



Now that our termination criteria is setup, we can start examining frames of the video:

We start by looping over our frames in **Line 44** and store the current frame in the variable “image”

**Line 45** then checks to see if we have selected a bounding box for our object to track yet.

Provided that we have selected a bounding box, the first thing we’ll do is convert our image from the RGB color space to the HSV color space (**Line 46**) and compute the histogram back projection on **Line 47**, using only the Hue component of the color space.

**Line 49** then checks if the bounding box tracking the object exists or not. This check is necessary to handle cases of object occlusions or object out of frame cases.

Once we have the back projection and have checked the box condition, we apply the CamShift algorithm on **Line 50** by making a call to cv2.CamShift. This function expects three arguments:

1. **backProj**: Which is the output of the histogram back projection.
2. **roiBox**: The estimated bounding box containing the object that we want to track.
3. **termination**: Our termination criterion which we defined on **Line 41**.

The cv2.CamShift function then returns two values to us. The first contains the estimated position, size, and orientation of the object we want to track. We then take this estimation and update our width and height of the box. We also store the four corner points in the variable “pts”

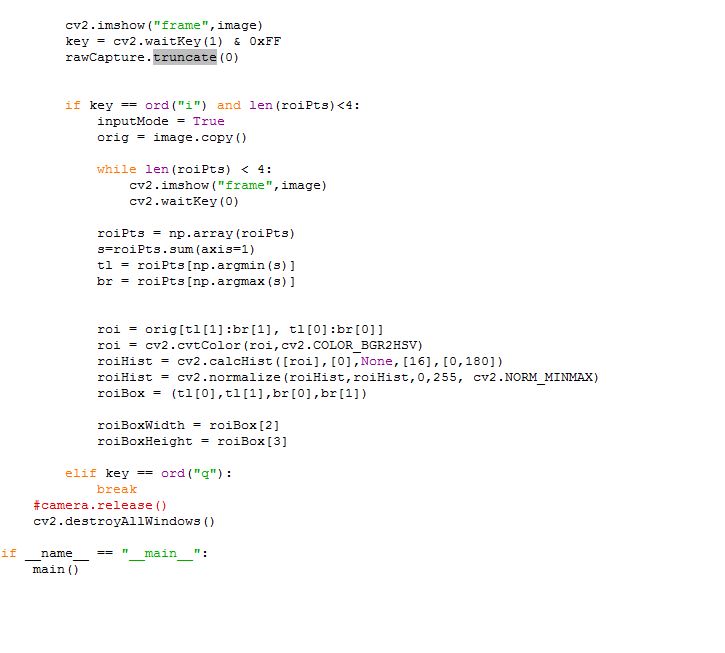
Finally, the second output of the cv2.CamShift function is the newly estimated position of the ROI, which will be re-fed into subsequent calls into the cv2.CamShift function

The **Lines 56-61** is used to update the center of the box and display it on top of the frame.

To handle the cases where the object is not present in the frame, in **lines 65-76**, we re-initialize the box at any arbitrary point and feed it again to the cv2.CamShift function to start tracking. This case is necessary as otherwise the program crashes everytime the object is occluded/out of frame.

Finally in **lines 78-81**, we draw the bounding box using the corner points stored in “pts” using the cv2.polylines function.

**Line 83** handles displaying the current frame on screen, while **Line 84** handles if a key is pressed.



If the ‘i’ key was pressed and there are less than four points in our list of points corresponding to the ROI, then we drop down into input mode (**Line 89**).

From there, we set inputMode = True to indicate that we are selecting our ROI. We also make a clone of the original frame.

In order to apply the cv2.CamShift function, we need the output of the back projection. And the only way to obtain the back projection is to create a reference histogram of the object we want to track.

We construct this histogram on **Lines 104 and 105** by making a call to cv2.calcHist and the normalizing it. Again, we are only using the Hue component of the HSV color space.

It is important to note the 4th parameter of cv2.calcHist -- the number of bins. In this case, we are using only 16 bins in the histogram. In OpenCV, hue values can fall within the range [0, 180], so tuning the number of bins for your application will certainly be important.

Finally, we construct the ROI bounding box on **Line 113** using the top-left and bottom-right points, respectively.

In order to break out of our frame loop, we’ll also monitor if the ‘q’ key is pressed on **Line 116**. If the 'q' key is pressed, we’ll break out of the loop.

The rest of the code is simple cleanup including destroying the windows and releasing the camera reference.

**OUTPUT:**

****