Image Content Mining Assignment 2

Object Tracking in Videos

Student:
Haileleul Haile

November 7, 2022

Contents

1 Introduction **1**

2 Code Description **1**

3 Mean Shift **1**

4 Hough Transform **3**

1 Introduction

Object tracking is an application of deep learning where the program takes an initial set of object detections and develops a unique identification for each of the initial detections and then tracks the detected objects as they move around frames in a video. In other words, object tracking is the task of automatically identifying objects in a video and interpreting them as a set of trajectories with high accuracy. Often, there's an indication around the object being tracked, for example, a surrounding square that follows the object, showing the user where the object is on the screen.

2 Code Description

All of the assignment questions, except question 1, have code experimentation. Questions 3, 4, and 5 have codebases titled question3.py, question4.py, and question5.py respectively. Supporting utility functions such as reference table calculation and accumulator calculation functions are included in the utils.py file.

3 Mean Shift

Q1. Explain the principle of this algorithm and illustrate its advantages and limits by your experiments.

The idea behind meanshift is that in meanshift algorithm every instance of the video is checked in the form of pixel distribution in that frame. We define an initial window, generally a square or a circle for which the positions are specified by ourselves which identifies the area of maximum pixel distribution and tries to keep track of that area in the video so that when the video is running our tracking window also moves towards the region of maximum pixel distribution. The direction of movement depends upon the difference between the center of our tracking window and the centroid of all the k-pixels inside that window.

From the experiment, I noticed that the size of the tracking rectangle doesn't change even when the object gets closer or farther from the camera. In the videos where the ball was being passed around, initially, the tracking window covered the entire ball. But when the ball was kicked around, the size of the ball grew and the window could not cover it. However, I noticed that the algorithm is relatively faster than the hough transform that I will describe in the next section.

Another important thing to mention is that this algorithm works using the hue value of the selected template. So if there are multiple objects in the video with similar hue values such as human skin color, the back projection will match multiple areas in the frame.

Q2. Analyze more in-depth the result by displaying the sequences of hue images, and also the weight images corresponding to the back-projection of the hue histogram. Propose and program improvements, by changing the computed density and/or updating the model histogram.

Figure 1: Question 2

One improvement I tried was recalculating the histogram each time the tracking window was changed. However, the performance greatly dropped. Another proposal is using the histogram of the roi to remove(silence) pixel values with hue values that are dissimilar to that shown on the histogram.

4 Hough Transform

Q3. Calculate for each frame, the local orientation, i.e. the gradient argument of pixels, and the gradient magnitude. Use a threshold on the gradient magnitude to mask pixels whose orientation is not significant. Display the sequence of orientations, where the masked pixels appear in red.

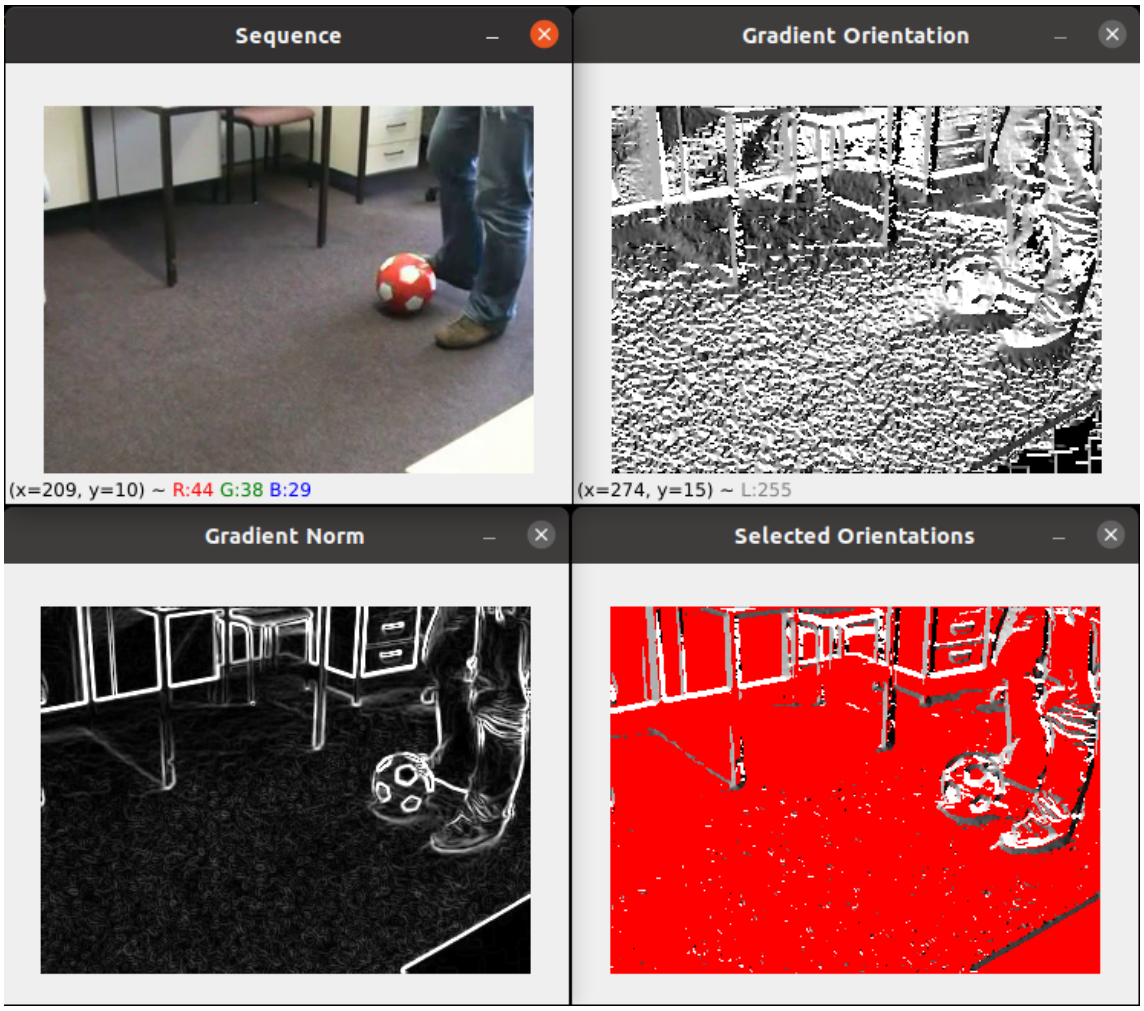

Figure 2: Question 3

Q4. Build a model of the initial object under the form of an implicit model indexed on the orientation (R-Table), calculated on significant (unmasked) pixels. Then, calculate the associated Hough transform on all the images of the sequence. Calculate the straightforward tracking, corresponding to the maximal value of the Hough transform at each image. Comment and criticise the obtained results, by showing some examples of Hough Transform and corresponding detection results.

In the utility library 'utils.py', you will find the functions I used to build the reference table and the accumulator.

For the generalized hough transform, I first build a reference table for the template image which is the area selected by a rectangle. The reference table is a table in which each row corresponds to an edge orientation. Each entry in the table represents an edge pixel location (r,α) with respect to a center point. Following that, we will look at all the edge pixels in the target image, where each edge pixel will have an edge orientation. Now, all we have to do is to look up that specific orientation in the r-table which will lead us to the (r,α) pairs. Each (r,α) will tell us a "candidate center point" and will contribute to one "vote" to that candidate center point.

Here I have displayed the hough transform tracking performance on two videos given along with the assignment. In both cases, we see the accumulator matrix with the pixel areas having the higher number of votes with brighter pixel values. In the case of the second test, we can see that many parts of the frame resemble the orientations in the template image and hence are shown with bright values in the accumulator matrix.

```python
def getRTable(template, degrees):
    template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(template,50,255)
    orientations = edgeOrientation(edges)
    r_table = [[] for i in range(degrees)]
    center_point = [int(template.shape[0]/2), int(template.shape[1]/2)]

    for (i, j), value in np.ndenumerate(edges):
        if value:
            # compute r and alpha for rtable
            r = np.sqrt((center_point[0] - i) ** 2 + (center_point[1] - j) ** 2)
            alpha = np.arctan2(i - center_point[0], j - center_point[1]) + np.pi

            # find index of rtable
            index = (int(degrees * orientations[i, j] / (2 * np.pi)))%degrees
            # print(index)
            r_table[index].append((r, alpha))

    return r_table
```

Figure 3: Question 4 - Code snippet to calculate the reference table for template edges' orientations

The main issue I faced with this method is the time it takes to compute the accumulator at each frame. This makes the tracking slow with machines that have good performance with the mean-shift algorithm. One solution to this problem can be increasing the threshold values used earlier when computing the orientation. That is because the vote is done only for the significant pixels which have passed through the thresholding. But this can not be raised too much because we will start to lose important information.

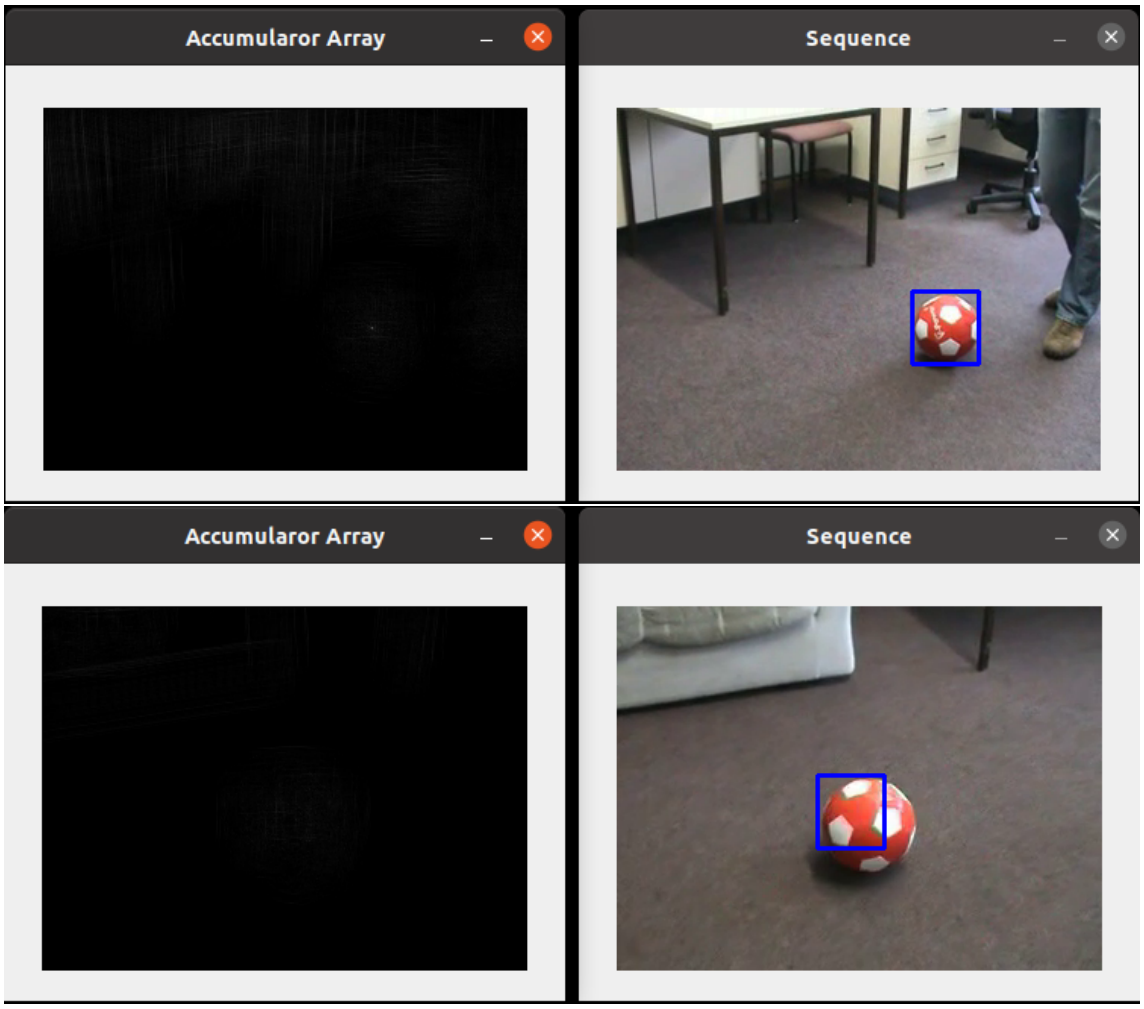

Figure 4: Question 4 - Football Video

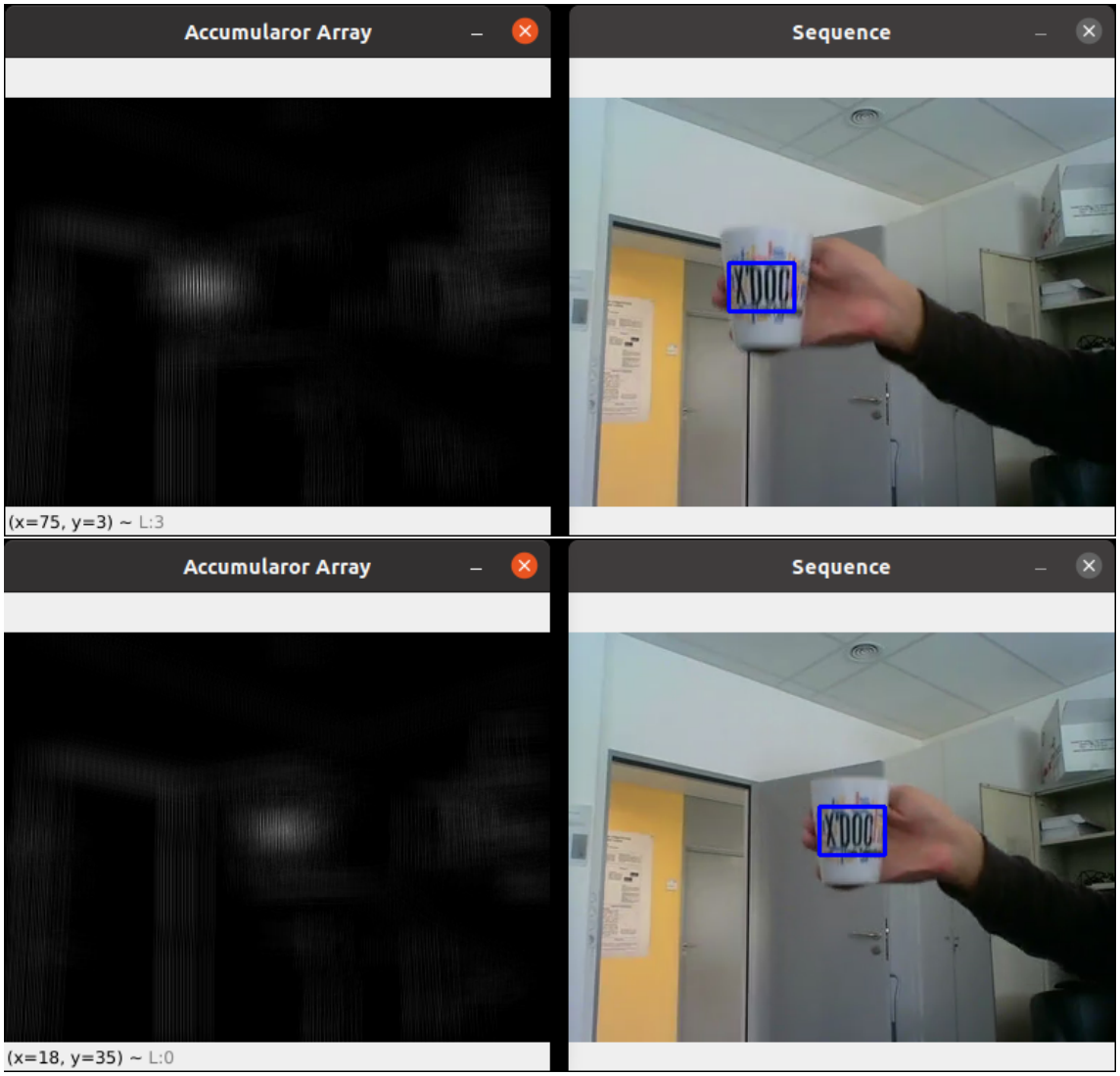

Figure 5: Question 4 - Mug Video

Q5. Propose a way to combine the two previous approaches (i.e. histogram and Hough based). Interpret the result and compare it with the previous ones. Propose an update strategy of the model that would allow to be robust to aspect changes of the object.

My proposal is multiplying the accumulator found using the hough transform with the back-projected weights of the histogram. This will mix the power of both using the hue and orientation to find the similarity between the template object and the identified object. Practically, we can see that the area with similar orientation and hue values will have very high pixel values. Then passing this new multiplied weight matrix to the mean shift algorithm will yield better accuracy in object tracking.

```
if ret == True:
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

    acc, rstart, rend, cstart, cend = genHoughTransformMatch(gray, roi, rTable)
    acc = cv2.convertScaleAbs(acc)

    multiplied_weights = np.multiply((0.25*dst),acc)
    multiplied_weights = cv2.convertScaleAbs(multiplied_weights)

    ret, track_window = cv2.meanShift(multiplied_weights, track_window, term_crit)

    # Draw a blue rectangle on the current image
    r,c,h,w = track_window
    frame_tracked = cv2.rectangle(frame, (r,c), (r+h,c+w), (255,0,0) ,2)
```

Figure 6: Question 4 - Mug Video

Note that the multiplication between the back projection weights and the accumulator is used to highlight the areas with similar orientations with the template and also remove similarly oriented areas with different hue histograms. To this end to limit the effect of the back projection weights, I quartered their values before multiplying with the accumulator so that similar hue histogram areas without similar orientations will not be magnified.

Performance: The result is very encouraging and can be seen in the following figures. The 'multiplied weights' image shows the final matrix used to perform the mean shift tracking. We can see that for 'VOT-Ball.mp4' video, the ball is clearly identified and the initial weight matrix is cleaned to remove ambiguity with the background.

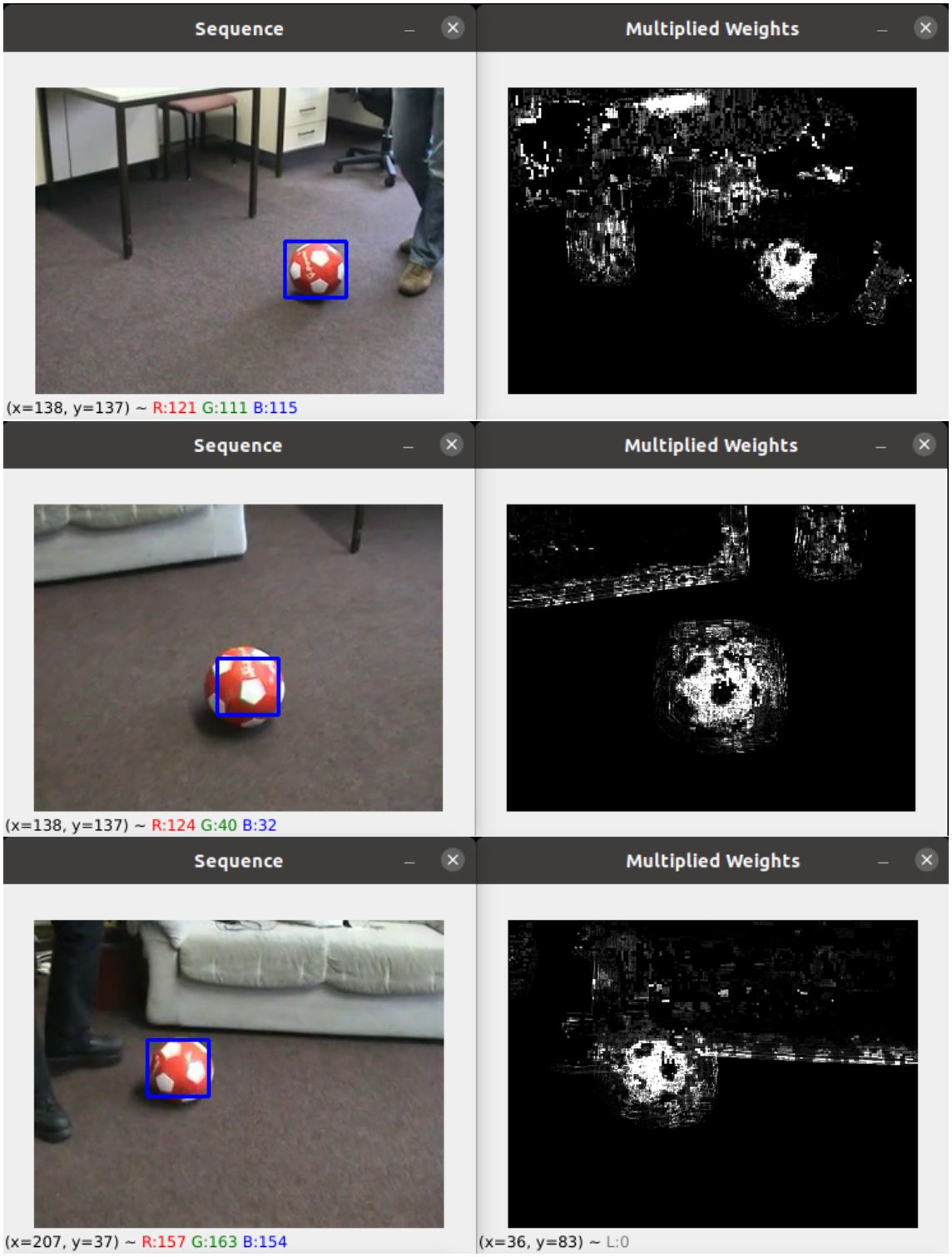

Figure 7: Question 5 - Performance of multiplied weights on object tracking for VOT-Ball.mp4. Multiplicated weights show higher accuracy in identifying the selected template area as compared to the back projection weights