PRACTICAL 8

Tracking

In this practical, we are going to explore different tracking algorithms: Kalman filter and Particle filter as well as association techniques between observation and trackers.

Download from QoL the provided functions to assist you during the practical as well as the clips and image sequences to be used.

TASK 1: KALMAN FILTER

In order to generate and run a Kalman Filter, 3 Matlab functions are provided. The first one, called `GeneraKalman()`, will take some input matrices no initialise the state vector, the initial covariance matrix and other parameters, such as the dynamics or the state and measurement noise. All this values will be used to create a structure where the Kalman Filter will be held.

The other 2 function, `KalmanPrediction`() and `KalmanEstimation`() will perform the prediction and the estimation of the filter, respectively, at each iteration/frame. These functions are almost empty, since only the template is provided. Your task is complete them.

STEP 1: Open `GeneraKalman` and be sure that you understand what each of the variables is by comparing them with the formulas explained in the lecture. Now open `KalmanPrediction`() and implement the 2 equations for given values to the prediction of the state vector and the prediction of the covariance matrix.

$$x_t = D \cdot x_{t-1}$$

$$C_t = D \cdot C_{t-1} \cdot D^T + Q$$

STEP 2: Complete the function `KalmanEstimation`() by implementing the correction equations of the Kalman filter

$$G_t = C_t \cdot H^T (H \cdot C_t \cdot H^T + R)^{-1}$$

$$x_t = x_t + G_t(z_t - H \cdot x_t)$$

$$C_t = C_t\,(I - G_t \cdot H)$$

Please note that the values of x_t and C_t to be corrected are the predicted ones by the previous function, not the last estimated in the previous frame.

As you can notice by the template, this function will have a different behaviour if there is measurement or not (measurement array is empty). In the first case, the prediction will be corrected, while in the second one, since no correction is possible, the prediction is considered the final estimation.

Time to test if our implementation is correct and works. We will be doing this by completing the script `KalmanTest`

STEP 3: Open the script `KalmanTest` and read through it. This script is very similar to the ones you were generating in the previous practical, but using a sequence of images instead of a proper video.

Please remember that the two hour duration of a practical class will probably not be enough time to complete all sections. There is an expectation that you will work outside of class as an independent learner.

To make the program work, you will need to add some code. Your task starts in line 60.

STEP 4: In order to initialise the filter, we will select the most likely blob to be a person in the first frame. The criterion in this simple case will be the biggest blob among the Candidates. Write a piece of code that calculates the area of the bounding boxes and finds index of the biggest candidate according to that area.

Once you know the index of the biggest blob, calculate the middle point of its bounding box, its width and its height.

STEP 5: Initialise the state vector. We are going to track the middle point of the bounding boxes, so this vector should be a 4x1 column vector containing the position of the x and y coordinate of the middle point that we just calculated, as well as their speeds.

$$x_t = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}$$

Initially, v_x and v_y will be equal to 0.

STEP 6: Fill the prediction matrix D to contain a liner motion model (space = velocity x time):

$$D = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the given value of dt in the script is 1. This means that the speed will be measure in pixels per frame instead of pixels per second. This is the most common approach.

STEP 7: Observe the rest of the initial values and parameters. Note that by giving a high value to the initial covariance we will rapidly converge towards the first measurement. Compare the values in Q and R. Are we trusting more the observation or the prediction?

By calling the function `GeneraKalman` at line 99, we are creating our Klaman filter with the given values. We can now run the filter (cicle prediction/estimation) for any other frame. The previous estimation of the Kalman filter is displayed in red on the top of the motion blobs

STEP 8: Call the function `KalmanPrediction` you completed in Step1. Then display the prediction of the bounding box in yellow.

Before running the correction, we need to decide which motion blob in the current frame will be used for updating the filter. In order to do that we need and association algorithm between the filter and the candidates. This is needed because several blob can appear at each frame.

STEP9: Implement an association algorithm using as criterion the Euclidean distance between the KF prediction and the centroid (or middle point) of the candidate:

$$dist(A,B) = \sqrt{(A(x) - B(x))^2 + (A(y) - B(y))^2}$$

The algorithm to implement will be as follow:

```
for each candidate

    calculate centroid of the candidate

    calculate  dist(KF prediction, candidate centroid)
end

chosen candidate  will be the candidate with minimum dist
```

Store the middle point of chosen candidate in a 2x1 vector called `measurement`

STEP 10: Call the function `KalmanEstimation` for both cases when there is a measurement and when there isn't. In the first case, you will need two input parameters: the kalman filter structure and the `measurement` array. For the second case, you will only need the KF structure as only parameter.

STEP 11: Run the code and observe how the KF evolves and tracks the subject in the image.

TASK 2: KALMAN FILTER WITH VARIABLE SIZE

As you can see in the previous results, the target position is tracked successfully over time. However the size of the bounding box is not nicely surrounded the subject.

This happens because our Kalman Filter is only tracking the position, not the size. Therefore the size (`width` and `height`) is only calculated when the filter is initialised and never updated.

Since, KF allow us to track any parameter, we can solve this issue by tracking, not only the position of the middle point, but also the size of the BB.

STEP 1: Save the previous script with a different name. You will be modifying this new script in this task. Functions `GeneraKalman, KalmanPrediction` and `KalmanPrediction` do not need to change are they will still be valid.

STEP 2: Replace the previous state vector initialisation by another one including the size and their speed

$$x_t = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \\ width \\ height \\ v_{width} \\ v_{height} \end{bmatrix}$$

STEP 3: Modify the sizes and values of Matrices R (4x4), D (8x8), C(8x8), Q(8x8) and H (4x8).

STEP 4: Extend the array measurement to store also the width and height of the bounding box of the chosen candidate.

$$measument = \begin{bmatrix} centroid_x \\ centroid_y \\ width \\ height \end{bmatrix}$$

STEP 5: Run the script and observe how the KF evolves and tracks, not only the subject position, but also its size over time.

TASK 3: PARTICLE FILTER

In this task we are going to use a Particle filter to track a face based on the skin colour. To assist you in this task, several complete and incomplete functions.

STEP 1: Open the video juanjo.avi and observe the type of sequence and movement we want to follow and the strong illumination changes happening.

STEP 2: Read and rub the code and comments in the script `ParticleFIlterTest`.m up to line 56. Try to follow the logic of it. Observe how the program allow the user to select a bounding box using the mouse and how this BB is converted in the state vector.

STEP 3: Analyse the Prediction matrix D. Is the model suitable for the type of movements you have observed in the video? Pay attention to the way the width and height propagates? Do you reckon the width and height of our BB will change over time?

STEP 4: Read through the code corresponding to the initialisation of the colour histogram, including the function `obtaining_histogram`. This function will create a 3D matrix containing a histogram for each of the channels.

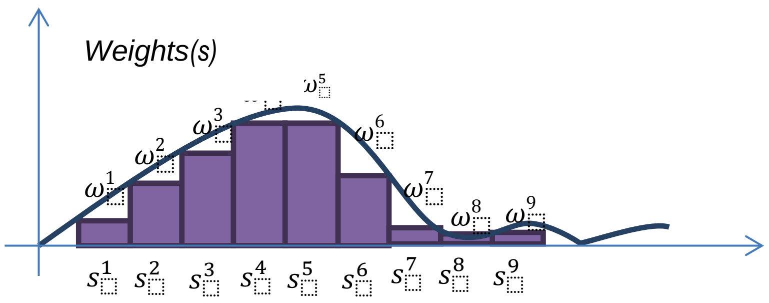
Open new figure and display the histograms.

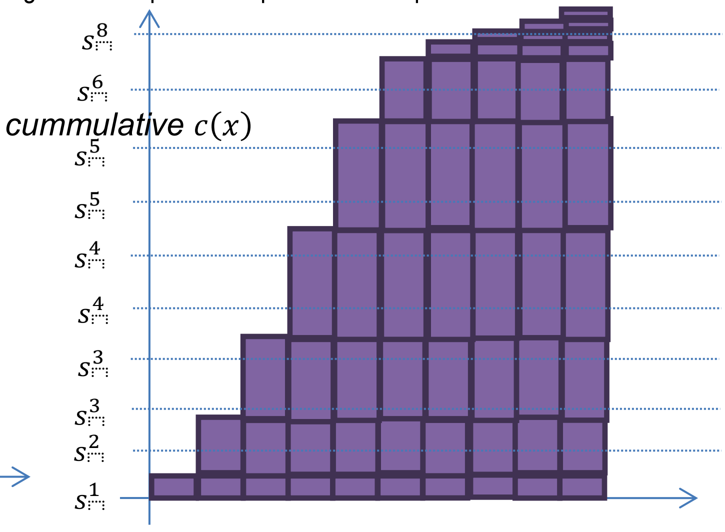
The space colour HSV is used in this program due to its ability to handle better illumination changes.

STEP 5: Go through the code. Main particle filter algorithm is embedded in the function condensation. Parts to pay attention are how the particles are initialised and the main steps of the particle filter:

- Resampling
- Prediction
- Observation

STEP 6: Open the function `rsampleN` function, responsible of performing the resample and implement the equation that calculated the cumulative weigths (1st step of the resampling)





STEP 7: Complete the function `prediction` by inserting the propagation equation:

$$s^i{}_t = D \cdot s^i{}_{t-1} + \text{noise}$$

STEP 8: Complete the observation process within the function `condensation`. You will need to extract first the histogram for the position and size the particle is (HINT: use the provided function to obtain the histogram). Then, add the code to calculate the Bhattacharyya distance between 2 histograms:

$$d_{Bh} = 1 - \rho[p(s^i{}_t), q]$$

$$\rho[p(s^i{}_t), q] = \sum_{u=1}^{N_u} \sqrt{p(s^i{}_t)(u) \cdot q(u)}$$

Please note that the Bhattacharyya distance will be calculated for each of the histogram's colour channels independently and then accumulated into one.

After that step, we just need to convert the distances into weights (probabilities) by using a negative exponential, as you can see already in the code.

With that, we have already finished the implementation of our PF. However, in order to visualise the estimation of our face location at each frame, it is a common practice to calculate the mean estimated state vector:

$$E[s_t] = \sum_{i=1}^{N} s^i{}_t \cdot \widetilde{\omega^i}_t$$

STEP 9: Open the function `new_State` and complete the last line to estimate the new state values using the previous formula. You can implement this functionality usng a for loop or in a single line taking advantage of Matlab matrix operations.

STEP 10: Run the full script. If you implemented correctly all previous steps, you should be able to see how the particles evolve (there is a colour code to distinguish good and bad particles, see script for more details) and the estimated face location. What happen with our tracking when the first illumination change appear?

TASK 4: TUNING OUR FILTER AND THE IMPORTANCE OF PREPROCESSING

In the light of the results from previous task, we will try to improve the tracking performance so we are able to follow the face over the full video.

STEP 1: Change the number of particles from values between 10 to 1000 and evaluate the result. What conclusion can you extract? Which number do you consider optimum? Does any of those avoid the tracking to diverge against illumination changes?

As you may notice, the tracking may not be robust in spite of our efforts, or we do at the cost of an unaffordable computational cost. This is happening because, due to the illumination variation, the initial histogram that we are using as model, is not valid anymore. We could address this issue by updating the model at every frame. However this is dangerous (if the estimation is not good, we will diverge) and, although effective for slow variation, it is not very effective against strong and sudden variation ,such as this case.

Instead, we will use this opportunity to demonstrate how important preprocessing is in the success of our algorithms. By compensating (as much as possible) the illumination variation before bedding them into the algorithm, we improve the robustness of our system.

As preprocessing, we will apply the Gray World function, which tries to compensate variations in colour due to illumination changes. Assuming that we have a good distribution of colours in our scene, the average reflected colour should be the colour of the light. If the light source is assumed to be white, we know how much the whitepoint should be moved in the colour cube.

STEP 2: Create a small script where you load the 3 provided images illum1.jpg, illum2.jpg and illum3.jpg. Process them using the function `GrayWorldFun_mio`(). Dysplay all 6 images in a figure. What conclusion can you extract?

STEP 3: Add this preprocessing to your particle filter script. Please note, that you should preprocess every image that you use (both to initialise the histogram model and every frame within the loop). Apply the preprocessing before converting the image into HSV.

STEP 4: Run the particle filter with the original value N=200. How does the filter perform?