HW3: Particle Filter Tracking

Due date: See in Moodle

- **Read the enite exercise before you solve it / submit it.**
- We use conda as our virtual environment and package management system. create and access conda **python 3.11** environment using the commands:

```
conda create -n vp python=3.11 -y

conda activate vp
```

- We have supplied a **requirements.txt** file that we will use to install the environment in which your code will run. you can install using pip:

```
pip install -r requirements.txt
```

Part A:

Please answer the following questions. Keep your answers short as possible and show the general direction of your solution. There is more than one correct answer.

1. We saw that $posterior \propto prior \times likelihood$. Explain the formula and how it relates to the Kalman/Particle filter. Start from explaining what is the posterior, prior and likelihood.
2. For the measurement step in the particle filter, we used the histogram of the two patches (the patch around the new particle and the original patch of the object).
    a. Why do we use the histogram to score the patches? what are the pros and cons of using this method?
    b. We took the patches and computed the histogram and their distances. Why didn't we use the SSD (sum square difference) between the patches (in the image space)?
    c. Can you think of another method to compare the patches? What are the pros and cons of the new method?

    You can describe an example if it helps you.

3. Will particle filter work when the tracked object changes its scale (like a person walking towards and away from the camera)? What about the viewpoint of the object (like a person dancing)?
4. Can you think about a way to know when to update the template? Think of a scenario where we track a person. We keep a patch of the object's appearance in the first frame, and then compare each new patch to it. One of the problems is that, as time passes, the original patch no longer represents the object (light, viewpoint, …). Can you think of a way to determine when to update the patch of the object? And how to determine the new patch (based on the previous frame/ based on the last two frames/…)?

Part B:

Basic theory (just a reminder):

In the Kalman filter, the density propagation changes from one Gaussian to another Gaussian distribution. What changes is the mean and variance.

This is very good if our system is linear and the noise is Gaussian. But if it isn't (i.e. cluttered data etc.) – Kalman fails.

This is where the particle filter enters - in every stage we create a new probability density function which can take any shape (not limited to being a Gaussian).

This shape is formed by sampling particles, each with a given weight and cumulative density function. This will be further elaborated in the continuation of this worksheet.

In short – we're still computing likelihood and prior probabilities and derive the posterior from them (similar to Kalman), but we're using more dynamic (non-Gaussian) probability functions to model the system.

Definitions:

For a given time stage, the system is described using the state vectors, their weights and their CDF's.

For N particles this is comprised of $\left\{s_t^{(n)}, w_t^{(n)}, c_t^{(n)}\right\}_{n=1}^{N}$

And in short writing we'll say that:

$$S_t = \left\{s_t^{(1)}, s_t^{(2)}, \dots, s_t^{(N)}\right\}\ W_t = \left\{w_t^{(1)}, w_t^{(2)}, \dots, w_t^{(N)}\right\}\ C_t = \left\{c_t^{(1)}, c_t^{(2)}, \dots, c_t^{(N)}\right\}$$

$$c_t^{(1)} = w_t^{(1)}, c_t^{(2)} = w_t^{(2)} + w_t^{(1)}, \dots, c_t^{(N)} = \sum_{i=1}^{N} w_t^{(i)}$$

and the weights are normalized such that $\sum_{i=1}^{N} w_t^{(i)} = 1$

For example - $s_{t1}^{(5)}, w_{t1}^{(5)}, c_{t1}^{(5)}$ denotes that state vector $s_{t1}^{(5)}$ has a weight of $w_{t1}^{(5)}$ and a CDF of $c_{t1}^{(5)}$ and all this is true for time $t = t1$

Algorithm stages you'll implement in the assignment:

For brevity, we won't always write the time step indices. But keep in mind we're starting with the particle filter for time t-1 ('before the first image') and each new image is the next time step (t, t+1, t+2 etc. but because the images are discrete this doesn't mean much.)

Step A:

Here we will create an initial particle filter (composed of 100 particles) for the previous time stage (t-1).

For this assignment each state vector has the following form:

$$s_t^{(n)} = \left[X_c^{(n)}, Y_c^{(n)}, \frac{Width^{(n)}}{2}, \frac{Height^{(n)}}{2}, X_{velocity}^{(n)}, Y_{velocity}^{(n)}\right] \text{ @ } time\ t \text{ and } n \in [1,2,\dots,100]$$

$$X_c, Y_c = coordinates\ to\ the\ center\ of\ object\ (the\ person\ we're\ going\ to\ track)$$

$$\frac{Width}{2}, \frac{Height}{2} = half\ of\ the\ width\ and\ height\ of\ our\ tracked\ person$$

$$X_{velocity}, Y_{velocity} = the\ speed\ of\ our\ person\ in\ each\ coordinate$$

Hint – you don't have to change the width and height between time steps.

Now for the actual steps:

1. Set $s_{initial} = [297, 139, 16, 43, 0, 0]$
2. Create a matrix of vector states sized 6x100 by adding random additive noise to $s_{initial}$. Repeating this will create 100 particle state vectors (each column describes one state vector, each row describes one of the state vector components for 100 different particles). We'll call this S (with a capital S).
   You can use step C (prediction) using the initial state vector for this part.
3. Load the first image and compute the normalized histogram for this image using $s_{initial}$. We'll call this normalized histogram 'q'.
   This step can be confusing, so please read this carefully:
   Your current frame 'I' is described in 3 dimension (width x height x 3 channels, RGB). $s_{initial}$ is 2 dimensional sized 6x1.
   The histogram is computed for the pixels in the rectangle sized width x height with the centers xc and yc. You receive this information from $s_{initial}$.
   Looking at this sub-portion of I, you'll notice the pixel intensities vary between 0 and 255 (8-bit). We want to quantize this to 4 bits (0 to 15).
   Next we look at each new combination of I_subportion(R,G,B) and we have a total of $16^3 = 4096$ combinations of RGB values. We'll build a histogram vector such that each element in the vector describes the number of times this RGB combination appears. Do this for 16x16x16 values and reshape it into a 4096x1 vector.
   Normalize this vector such that the sum of all elements is equal to 1.
4. Repeat step 3, but this time use the first image and use the first column of S
   We'll call this normalized histogram 'p'. p is computed exactly like q was computed in the long explanation in step 3.
   Now we compute the weights of each particle using the Bhattacharyya distance between p and q. This similarity index will be our way to determine the weights.
   Repeat this step for all 100 columns of S and you'll end up with $W = \{w_1, w_2, \dots, w_{100}\}$ where each element is the single Bhattacharyya distance value. The equation is: $w = dist(p,q) = exp^{20*\sum_{i=1}^{4096}\sqrt{p_i * q_i}}$
5. Normalize vector W such that $\sum_{i=1}^{100} w(i) = 1$
6. Compute vector $C = \{c_1, \dots, c_j, \dots, c_{100}\} = \{w_1, \dots, \sum_{i=1}^{j} w(i), \dots, \sum_{i=1}^{100} w(i) = 1\}$.
   This means that the j'th entry in C denoted is c_j is: $c_j = \sum_{i=1}^{j} w(i)$

We now have a complete set for our initial particle filter, normalized, weighted and with the CDFs.

Next, we must:

- sample the previous particle filter (step B)
- predict the next particle filter and update the weights + CDF (step C)
- display the results (step D)

Step B (sampling, deterministic 'drift'):

- Repeat the following steps 1-3 for each value of $n \in [1, N]$
1. Generate a random value $r \in [0,1]$ from a uniform distribution.
2. Find the smallest value 'j' such that $c_{t-1}^{(j)} \geq r$
3. Set $s_t^{\prime(n)} = s_{t-1}^{(j)}$ where index 'j' corresponds with the 'j' we found in step 2.

We now have N sampled particles $S_t' = \left\{s_t^{\prime(n)}\right\}_{n=1}^{N}$

Step C (prediction, random 'diffusion'):

Project the sampled particles from the previous step to their new location using a dynamic model and adding noise. This is done by setting:

$$S_t = AS_t' + noise$$

The noise is white additive noise (you decide what noise!).

A is a dynamic model (like in Kalman) – you may simply add the previous velocity components such that

$$S_t = \left[X_c^{(n)} + X_{velocity}^{(n)}, Y_c^{(n)} + Y_{velocity}^{(n)}, \frac{Width^{(n)}}{2}, \frac{Height^{(n)}}{2}, X_{velocity}^{(n)}, Y_{velocity}^{(n)}\right]_t + noise$$

Step D (show results):

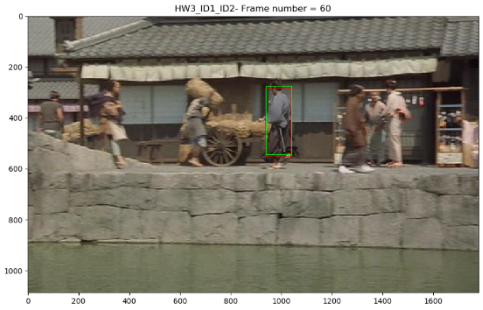
You are required to plot the **average** particle filter rectangle in green and to plot the **maximal** particle filter (with the largest weight) in red.

There are numerous ways to achieve this type of plot.

The title of the image plot must be HW3_ID1_ID2- Frame number = xxx.

When you display a figure to the screen, make sure the open figure window won't block the progress of the code (use the block=False flag, in plt.show).

For example:


HW3_ID1_ID2- Frame number = 60
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

WHAT YOU SUBMIT FOR THIS HOMEWORK ASSIGNMENT:

- PDF file with answers to part A.
- particle_filter.py (change the IDs to your IDs. Fill in the code where requested). We will run and test your code using this file. The file should contain the implementation for all the following functions:
    - bhattacharyya_distance
    - predict_particles
    - compute_normalized_histogram
    - sample_particles
    - show_particles
- results directory – containing the results of your implementation after running the particle_filter.py script

SEE THE INPUT/OUTPUT REQUESTS WITHIN THE ATTACHED EMPTY FUNCTIONS AND READ THE COMMENTS FOR FURTHER CLARIFICATIONS.

Attach all the requested files in a zip file named vp2025_ex3_ID1_ID2.zip

Enjoy and Good Luck! ☺️