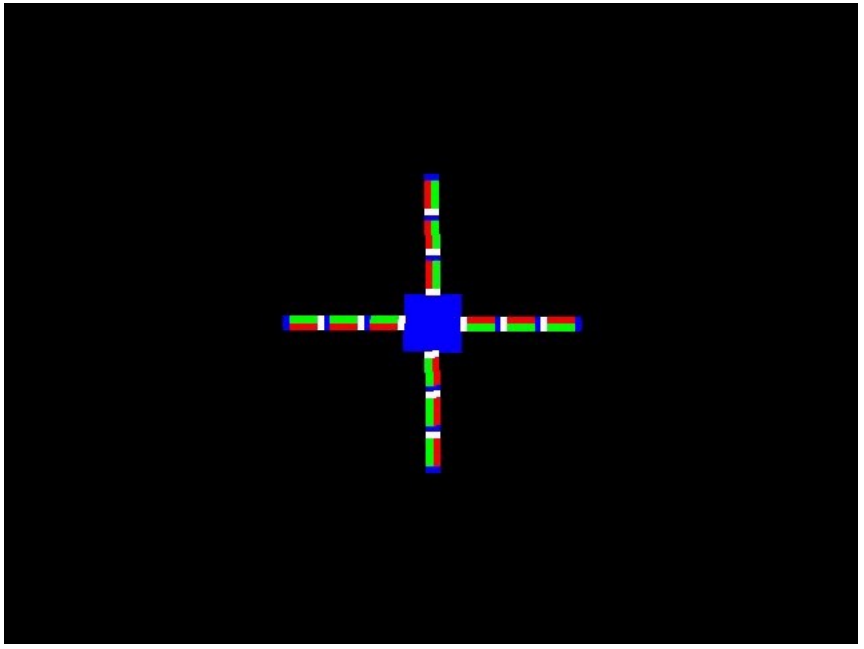
Traitement d’images et vidéos

TP3-4 : Object tracking

SIA_TP1
TRAN Jérémy

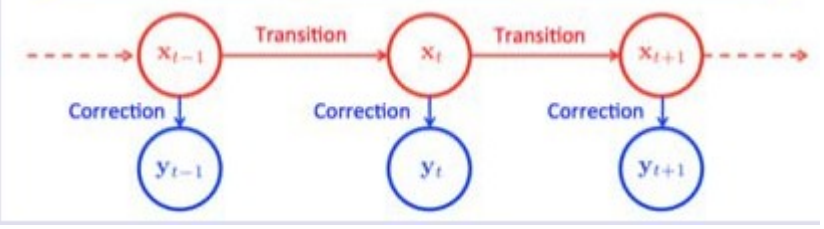
Objective : Simple object tracking using particle filters



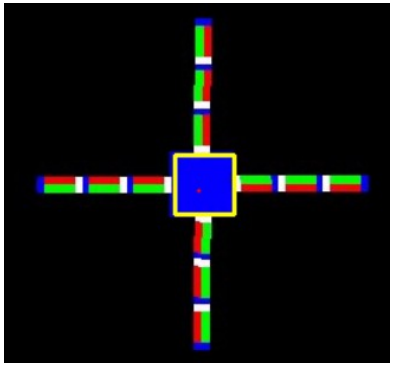
We want to track the movement of the blue square in the center of this 640x380 image.

This object moves in a 2D space and rotates so there are 3 values to track, but I ignored the rotation at the start.

We will be using a **state space model** for tracking.
This Markov model allows us to use strong hypotheses :
- *The current state only depends on the previous state*
- *Observations are conditionnaly independent with respect to states and only depend on the current state*



Tracking algorithm



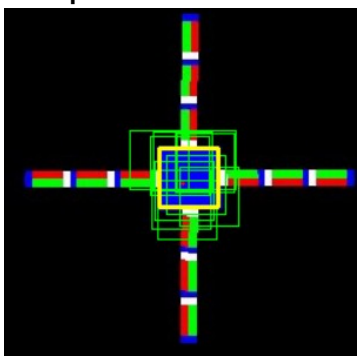
Step 1 : Initialize the tracked area by determining the position of the center and its area.

Step 2 : Calculate the normalized reference histogram href associated with this area.
For this we define a subimage and a bin number Nb

```
"""Histogram calculation"""
def initialize_href(img,init_center,size_rectangle=[20,20],Nb_bins = 25):
    sub_image = img[init_center[0]-size_rectangle[0]:init_center[0]+size_rectangle[0],
    init_center[1]-size_rectangle[1]:init_center[1]+size_rectangle[1]]
    href, bin_edges = np.histogram(sub_image, bins=Nb_bins, range=(0, 255))
    nb_pixels = size_rectangle[0] * size_rectangle[1] * 4 * 3
    href = href / nb_pixels
    return href
```

The color histogram is normalized

Step 3 : Initialize a set of N particles around the center of the tracked area :
Here N = 10.



Their initial weights are all equal : $w_0^i = \frac{1}{N}, i = 1, \ldots, N.$

Step 4 : Calling the particule filter, made up of 3 steps : **prediction**, **correction** and **resampling**

Particule filter

Prediction: We define a transition function in a scenario where we don't have any prior information on the object trajectory. This model is given by the equation : $$X_k = X_{k-1} + V_k,$$

Where $V_k \sim \mathcal{N}(0, \mathbf{\Sigma})$ is a gaussian white noise and Σ is a diagonal matrix.

```
"""Transition function"""
#Adds a random noise to the previous positions to try to predict the current location
def prediction_step(position,deviation=12 ) :
    position = [int(random.gauss(position[0],deviation)),int(random.gauss(position[1],deviation))]
    return position
```

Likelihood function : We calculate the color histogram of the tracked area defined by the particules and we calculate the distance between the tracked histogram and the reference histogram following this equation :

$$\mathrm{dist}(h, h') = \left(1 - \sum_{i=1}^{N} \sqrt{h(i)h'(i)}\right)^{\frac{1}{2}},$$

Where h(1) and h'(i) are the i-th bin of histograms h and h' respectively.

The likelihood is then approximated by :

$$g(y_k^i | x_k^i) \propto \exp(-\lambda \mathrm{dist}^2(h_{ref}, h(x_k^i))),$$

where href is the reference histogram, $h(x^i_k)$ is the histogram associated with particle x^i_k, and λ is a constant

```
"""Likelihood function"""
#Corrects the prediction using the observation of the object
def distance(href,hpred) :
    dist = 0
    hsum=0
    for i in range(len(href)) :
        hsum += (href[i]*hpred[i])**0.5
    dist = (1-hsum)**0.5
    return dist

def likelihood(dist,lmbd=10):
    lklh = np.exp(-lmbd*dist**2)
    return lklh
#lmbd not too high to keep enough particles
```

Particule filter

Correction: This step is used to correct the predicted position obtained from the previous step using the observation from the current image. The particle weights w^i_k are updated after calculating the likelihood function :

$$w^i_{k+1} \propto g(y^i_k|x^i_k)w^i_k.$$

```
for position in predicted_positions :
    predicted_hist =initialize_href(img,position,size_rectangle)
    hist_dist.append(distance(href,predicted_hist))
    estimated_likelihood.append(likelihood(hist_dist[indice]))
    new_weights[indice] = new_weights[indice]*estimated_likelihood[indice]
    indice+=1
new_weights = new_weights/sum(new_weights)
estimated_new_center = [0,0]
for index in range(len(predicted_positions)) :
    estimated_new_center[0] +=new_weights[index]*predicted_positions[index][0]
    estimated_new_center[1] +=new_weights[index]*predicted_positions[index][1]
estimated_new_center[0] = int(estimated_new_center[0])
estimated_new_center[1] = int(estimated_new_center[1])
```

The new object center is estimated as the weighted average of the particles :

$$\bar{x}_k = \sum_{i=1}^{N} w^i_k x^i_k.$$

Resampling : This step is implemented to counteract the sample degeneracy problem :

```
"""Resampling function"""
#Uses systematic resampling when the weights values start to dip : Replaces the lower weight value
def resampling(new_weights,predicted_positions,threshold) :
    resampled_weights = new_weights.copy()
    resampled_positions = predicted_positions.copy()
    stable_weights = np.where(new_weights>=threshold)[0]
    for resampled_indices in np.where(new_weights<threshold)[0] :
        random_weight_indice = random.choices(stable_weights,new_weights[stable_weights],k=1)[0]
        resampled_weights[resampled_indices]=new_weights[random_weight_indice]
        resampled_positions[resampled_indices]=predicted_positions[random_weight_indice]
    resampled_weights = resampled_weights/sum(resampled_weights)
    return resampled_weights,resampled_positions
```

I used systematic resampling to replace particles with a weight lower than the threshold with particles with a weight higher than the threshold.

Results, observations, comments...

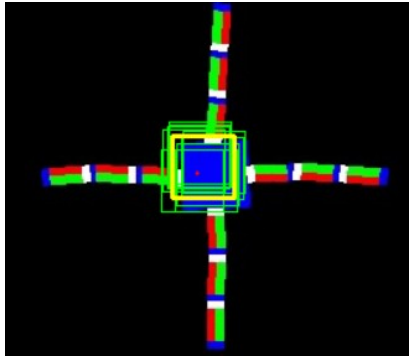
Parameters selection :

- I first started using a subimage size higher than the object size to add information from the « arms » and the background which works a bit in this scenario but isn't viable when the background is not uniform.

- The stride Σ is an important parameter, too low and the tracking won't be able to keep up with the object, too high and the average quality of the particles will decrease.

- The λ parameter in the likelihood function must not be too high to keep some lower weight particles. At each time step no more than 50 % of the particles should be dropped because the filter relies on the average of the particles.

- The tracking should put more importance on the weighted average of the particles rather than the particle with the highest weight because the variance of the values might introduce incorrect particles with high weights .



The weighted average of the particles is highlighted in yellow with a red dot at its center.

Results, observations, comments...

Results :

- Unfortunately, the tracking is not very good. With the right set of parameters, the object is decently tracked at first but it tends to easily get lost.
See demo_tracking.mkv for an example : Nb bins = 25, N=10, Σ = 8, λ = 10.

Ideas for improvement :

- Change the image characterization, rather than calculating the color histogram, I could implement a methos to count the subimage pixels that match with the object's pixels the distance would then be based on the number of pixels that are different from the tracked object.

- The object rotates during the video sequence, to track this new dimensionality, I have to add a way to track the angle in the hidden state so adapt the prediction and correction step.

- Tried the tracking with the occlusion problems and with the right parameters, it manages to somewhat track the object's movement.
See tracking_cluster.mkv for reference : Nb bins = 25, N = 10, Σ=15, λ = 5