Lundi 3 janvier 2022

Compte-rendu 3 : Poursuite de cibles

Filtrage Particulaire

Alexandre Chaussard

Antoine Klein

Introduction : Il s'agira d'utiliser les outils bayésiens pour la poursuite de cibles. Si le dernier TP cherchait à le faire dans des modèles linéaires (filtrage de Kalman) ou dans des modèles linéarisés (filtrage de Kalman étendu), nous l'élargirons au cas non linéaire et non gaussien avec le filtrage particulaire. Nous commencerons par un modèle théorique avant de faire une étude de cas de poursuite sur vidéo.

1) Modèle de Kitagawa :

On considère dans cette partie les équations d'état suivantes :

$$\begin{cases} X_n &= 0.5X_{n-1} + 25\frac{X_{n-1}}{1+X_{n-1}^2} + 8\cos(1.2(n)) + U_n \\ Y_n &= \frac{X_n^2}{20} + V_n \end{cases}$$

Les bruits sont supposés indépendants et gaussiens. L'objectif est donc d'estimer la variable cachée X_i observant les variables Y_0 jusqu'à Y_i.

Remarquons tout d'abord qu'en faisant du filtrage, c'est-à-dire en prenant les observations que jusqu'à l'état courant, nous ne prenons pas en compte toute l'information du signal (l'information postérieur est mise de côté). Enfin, le fait qu'il n'y ait pas bijectivité entre la trajectoire et ses observations (l'observation est le carré de la trajectoire) rend encore plus difficile l'estimation du paramètre caché. En effet, deux solutions sont possibles pour chaque estimation, n'ayant pas l'information du signe de X.

Les lois de transition actualisées sont à présent :

$$f_{n|n-1}(x_n|x_{n-1}) = N(x_n; 0.5x_{n-1} + 25\frac{x_{n-1}}{1+{x_{n-1}}^2} + 8\cos(1.2(n)), Q)$$

$$g_n(y_n|x_n) = N(y_n; \frac{{x_n}^2}{20}, R)$$

```python
# ---------- 3

Q = 10
R = 20
T = 50
N = 50


# Générer la trajectoire
def generer_trajectoire(T, Q):
    x_0 = np.random.normal(0, 1)
    X = [x_0]

    for i in range(1, T):
        u_n = np.random.normal(0, Q)

        x_n = 0.5 * X[i - 1] + 25 * X[i - 1] / (1 + X[i - 1] ** 2) + 8 * np.cos(1.2 * i) + u_n
        X.append(x_n)

    return np.array(X)
```

Figure 1 : Code Python qui génère une trajectoire

```python
# ---------- 4

# Générer les observations
def generer_observations(X, R):
    Y = []

    for i in range(0, len(X)):
        v_n = np.random.normal(0, R)

        y_n = X[i] ** 2 / 20 + v_n
        Y.append(y_n)

    return np.array(Y)
```

Figure 2 : Code Python qui génère les observations

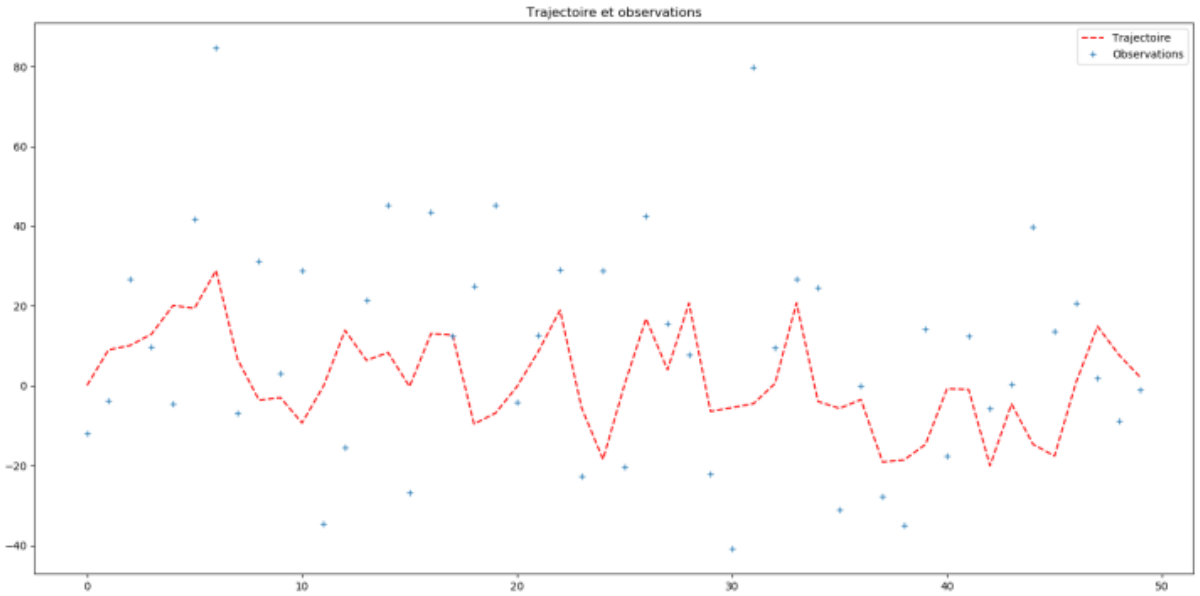

Figure 3 : Une réalisation de trajectoire et ses observations

```python
def filtrage_particulaire(Y, N):
    X_est = np.random.randn(N, 1)  # vecteur € R^Nx1
    w = g(Y[0], X_est)  # vecteur des poids
    w = w / np.sum(w)  # normalisation des poids

    for t in range(1, T):

        # tire N particules selon la pondération w
        X_t_moins_1 = np.random.choice(list(X_est[:, -1]), N, list(w[:, -1])).reshape(-1, 1)
        X_t = 0.5 * X_t_moins_1
                  + 25 * X_t_moins_1 / (1 + X_t_moins_1 ** 2)
                  + 8 * np.cos(1.2 * t) + Q**.5 * np.random.randn(N,1)  # application du modèle
        w_t = g(Y[t], X_t)
        w_t = w_t / np.sum(w_t)
        X_est = np.hstack([X_est, X_t])
        w = np.hstack([w, w_t])

    return X_est, w  # vecteurs de R^NxT
```

Figure 4 : Code Python qui réalise le filtrage particulaire

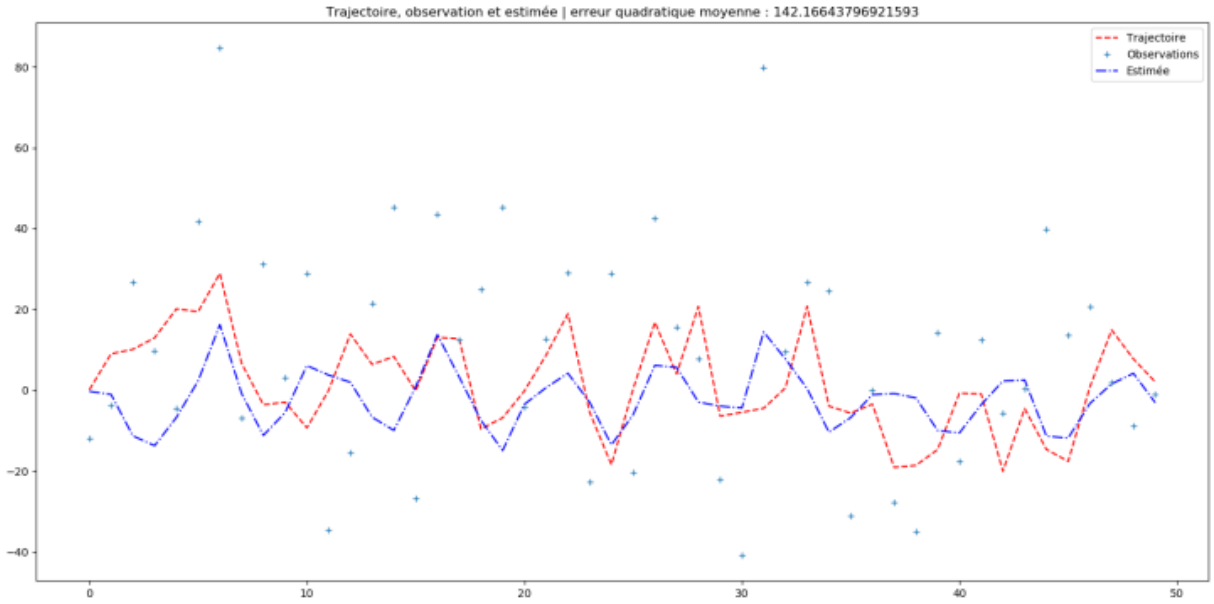

Figure 5 : Une réalisation de trajectoire, de ses observations et sa trajectoire estimée

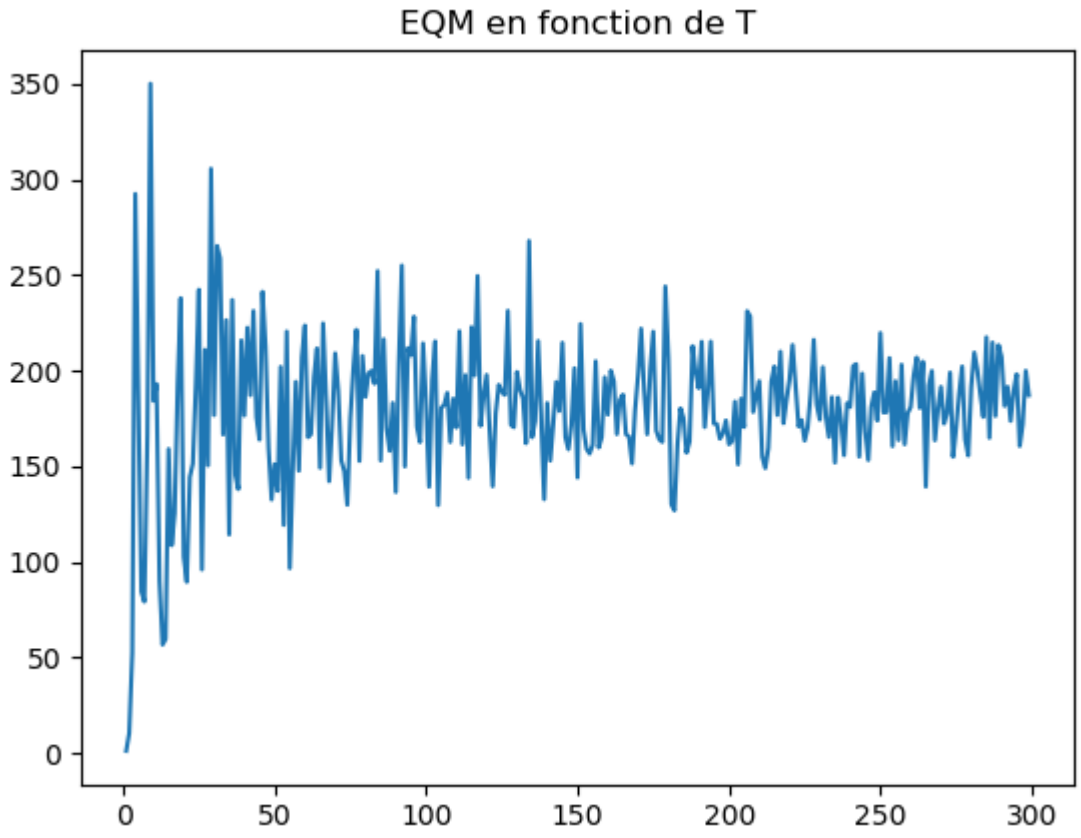

Figure 6 : Erreur quadratique en fonction de la longueur de chaîne avec 50 particules

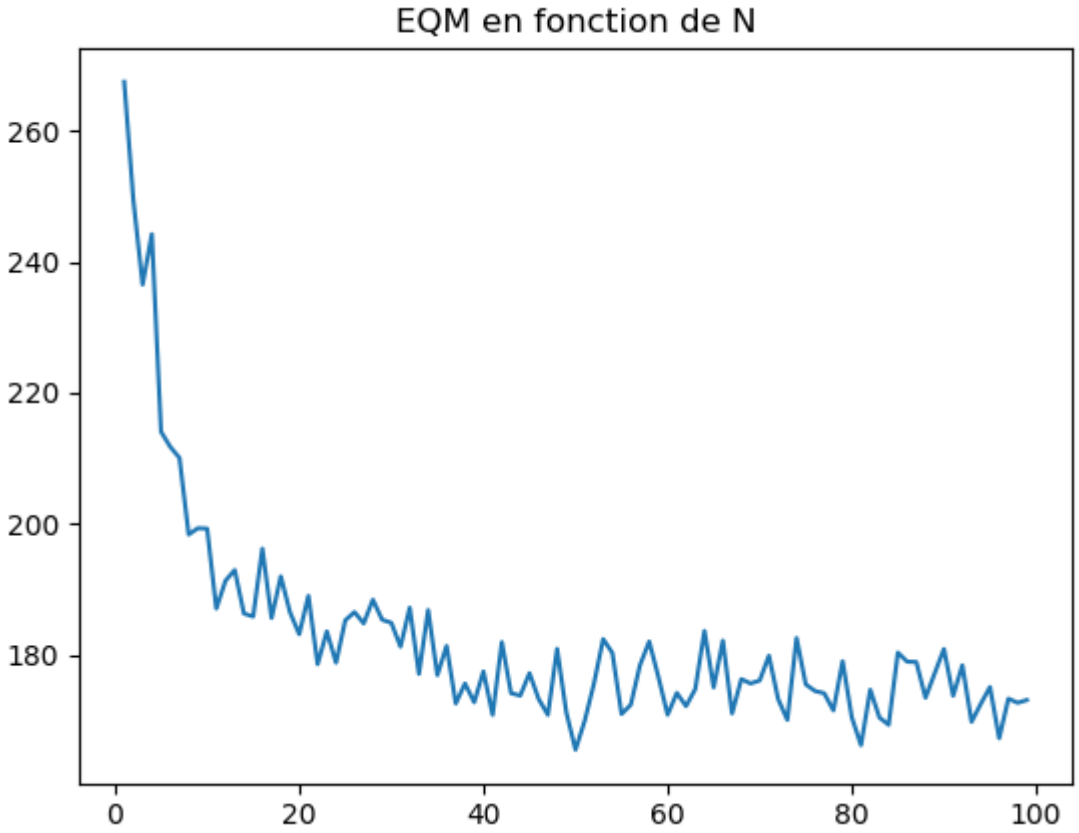

Figure 7 : Erreur quadratique moyenne en fonction du nombre de particules avec une réalisation de longueur 50

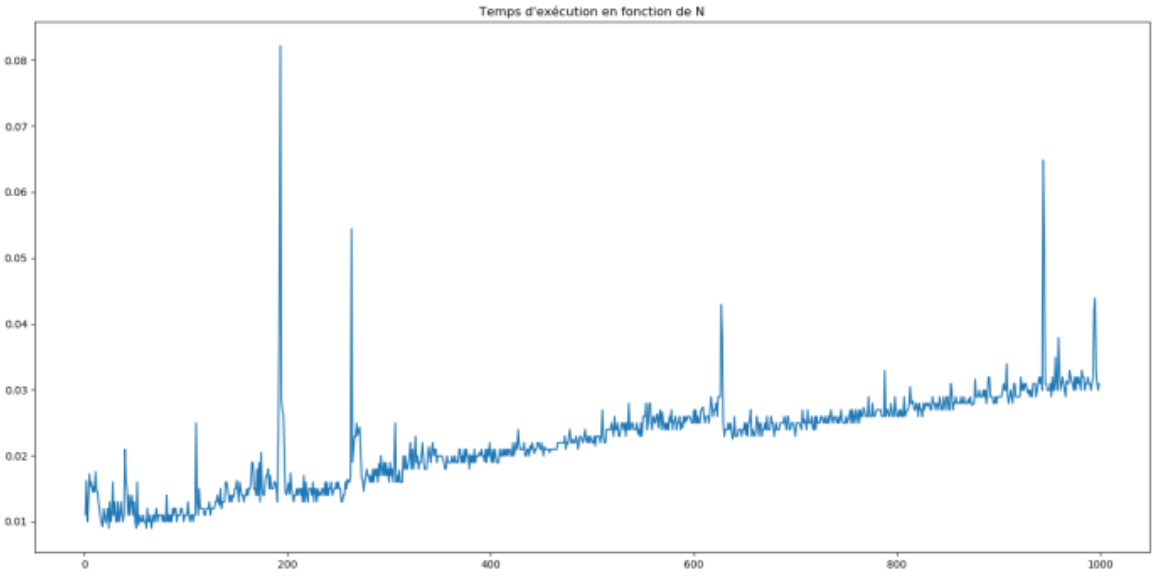

Figure 8 : Temps d'exécution en fonction du nombre de particules avec une réalisation de longueur 50

On remarque que même si les observations ne donnent pas l'information du signe de la variable cachée, le filtrage particulaire donne des résultats probants (là où le filtrage de Kalman aurait failli à cause des non-linéarités). On constate aussi qu'ajouter des particules fait diminuer significativement l'erreur quadratique moyenne (décroissance exponentielle avec pour asymptote y=26). Quant au temps d'exécution, celui-ci est linéaire en fonction du nombre de particules. Tout ceci amène le résultat suivant : pour une estimation efficace et avec un coût raisonnable, il nous faut travailler avec de l'ordre d'une centaine de particules.

2) Suivi de visage sur une séquence vidéo :

Dans cette partie, l'enjeu est de suivre un visage sur une vidéo enregistrée. Pour ce faire, nous nous proposons de le suivre au travers d'un rectangle de taille fixe qui encadrera à tout moment le visage. Implicitement, cela suppose que la personne que l'on suit n'effectue pas un mouvement avant-arrière (effet de zoom). La variable cachée est alors un couple de coordonnées caractérisant le point haut-gauche du rectangle.

L'équation d'état de la variable cachée (coordonnées du point haut gauche du rectangle) est :

$$X_n = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times X_{n-1} + U_n \qquad \text{où } U_n \sim \mathcal{N}(0_{2\times 2}, \begin{pmatrix} C_1 & 0 \\ 0 & C_2 \end{pmatrix}).$$

Quant à la distribution de la vraisemblance, nous passons par un calcul d'histogramme de couleur associé à la zone délimitée par le rectangle. Nous comparons ensuite l'histogramme de couleur courant de chaque particule avec celui défini par le rectangle initial à l'aide de la distance de Bhattacharyya-Hellinger, définie par :

$$D(q, q') = \left(1 - \sum_{i=1}^{N_b} \sqrt{q(n)q'(n)}\right)^{\frac{1}{2}}$$

Nous faisons l'hypothèse complémentaire de l'invariance de l'histogramme de couleur : le visage est filmé toujours sous le même angle, la luminosité est constante au cours du temps… Une telle hypothèse est raisonnable dans notre cas dans la mesure où l'étudiant se tient face à la caméra et effectue que des déplacements dans un même plan.

Nous choisissons ensuite une distribution exponentielle pour la vraisemblance :

$$g_n(y_n|x_n) \propto \exp\{-\lambda D^2(q, q'(x_n))\},$$

Une particule est d'autant plus invraisemblable (au sens exponentiel) que l'histogramme de couleur associé à son rectangle diffère de celui initial. En revanche, nul besoin de connaître la constante de proportionnalité dans la mesure où les poids qui en résultent sont normalisés.

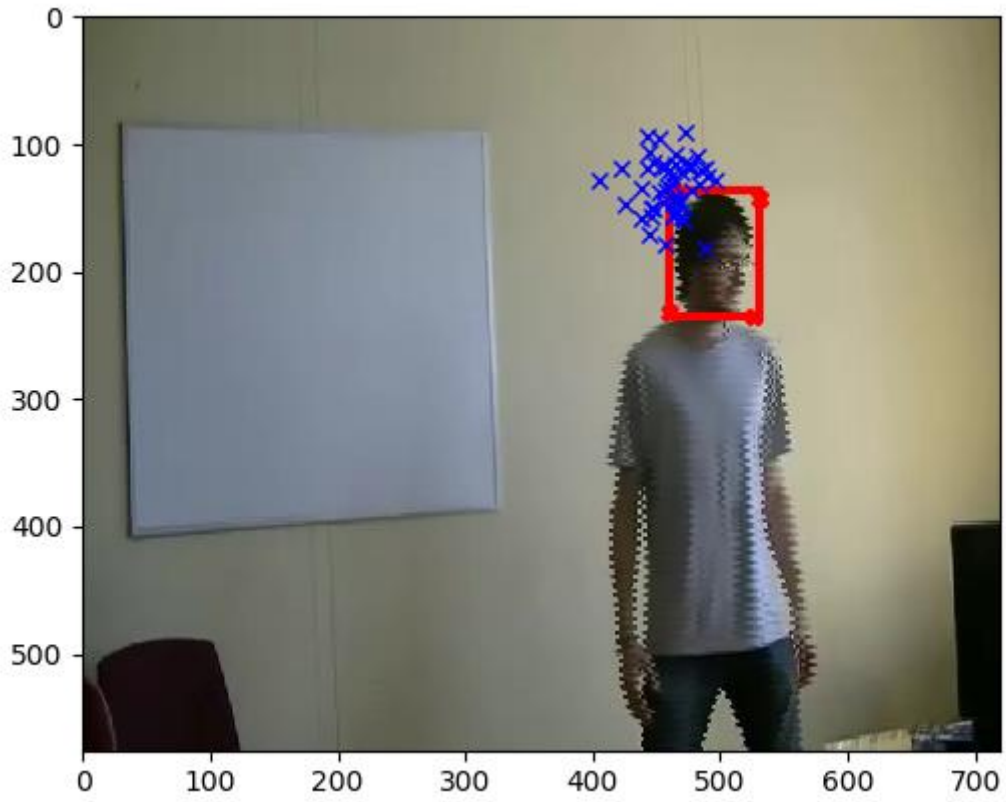

Figure 9 : Tracer du rectangle initial et réalisations des premières 50 particules

La matrice de passage du modèle étant l'identité, le modèle caractérise une situation où le visage se déplace peu d'une image à l'autre ; les petits mouvements étant capturés par le bruit gaussien dont la variance caractérise la marge de déplacement d'une image à l'autre. De plus, l'hypothèse d'un histogramme de couleur constant à tout instant est discutable. Si le visage effectue une rotation ou si l'éclairage change, l'histogramme est amené à changer. En bref, le modèle est pertinent que si le visage à suivre « ne bouge pas trop » et dans des situations où le cadre ne varie pas trop. Pour ajouter plus de robustesse à notre modèle, nous pouvons considérer des variations de l'histogramme de couleur et l'estimer à chaque itération.

Si l'objet à suivre connaît une forte accélération, la matrice de passage identité n'est plus pertinente (en ce qu'elle décrit une immobilité). Pour y remédier, nous pouvons inclure dans le vecteur X la vitesse en x et la vitesse en y de sorte à amener une matrice de passage qui caractérise un mouvement accéléré (matrice triangulaire supérieure).

Quand λ augmente, les particules sont d'autant discriminées, le rééchantillonnage fait que leur nombre diminue et que l'on ne sélectionne a priori que les plus pertinentes.

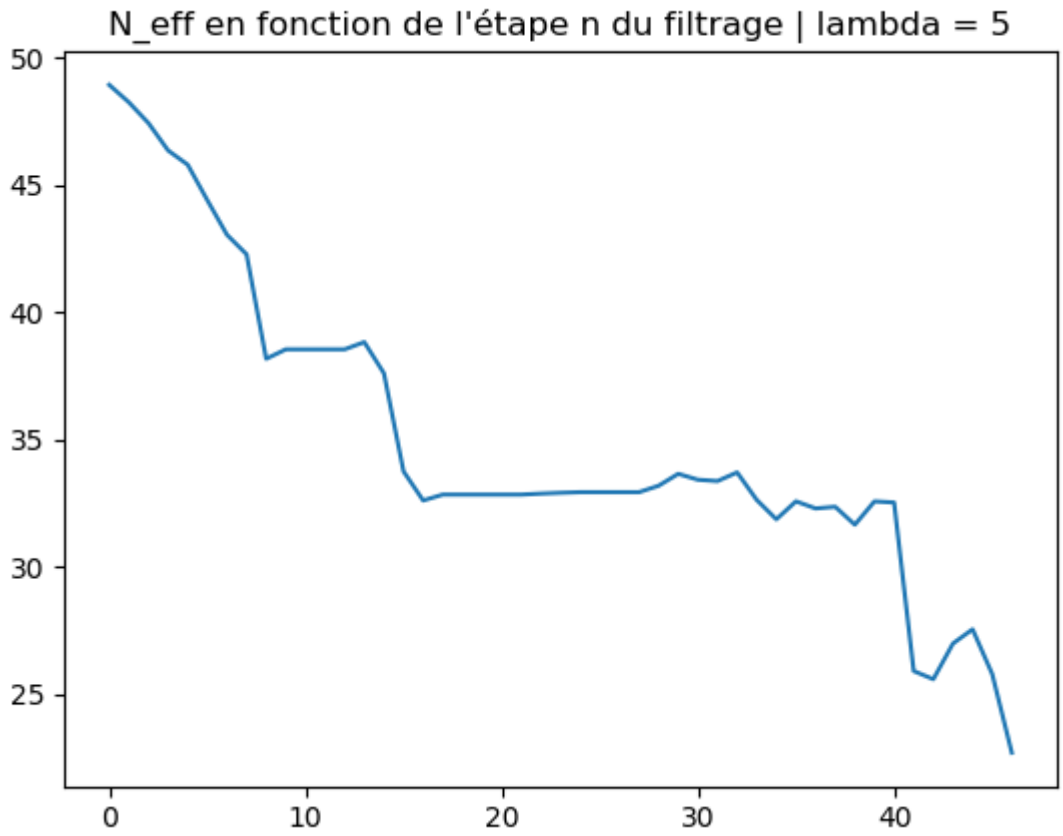

Figure 10 : N_eff en fonction du temps de la vidéo avec 50 particules initialement

Le nombre de particules va décroissant

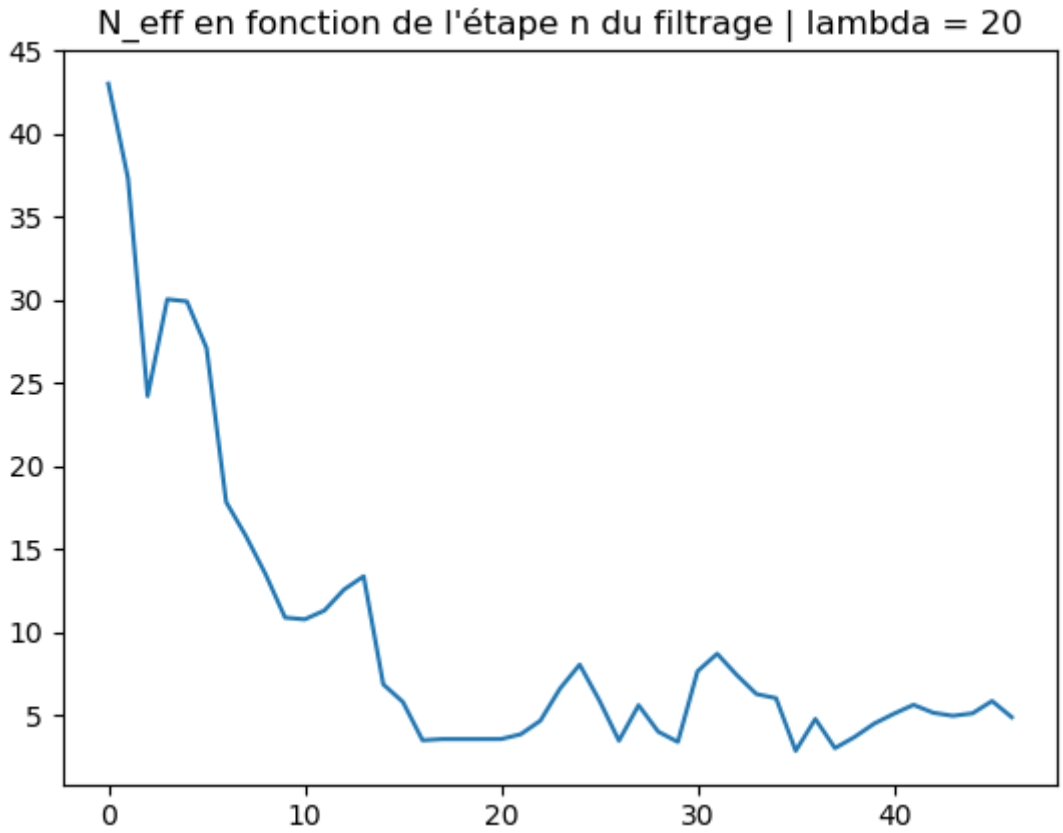

Figure 11 : N_eff en fonction du temps de la vidéo avec 50 particules initialement

Le nombre de particules va décroissant

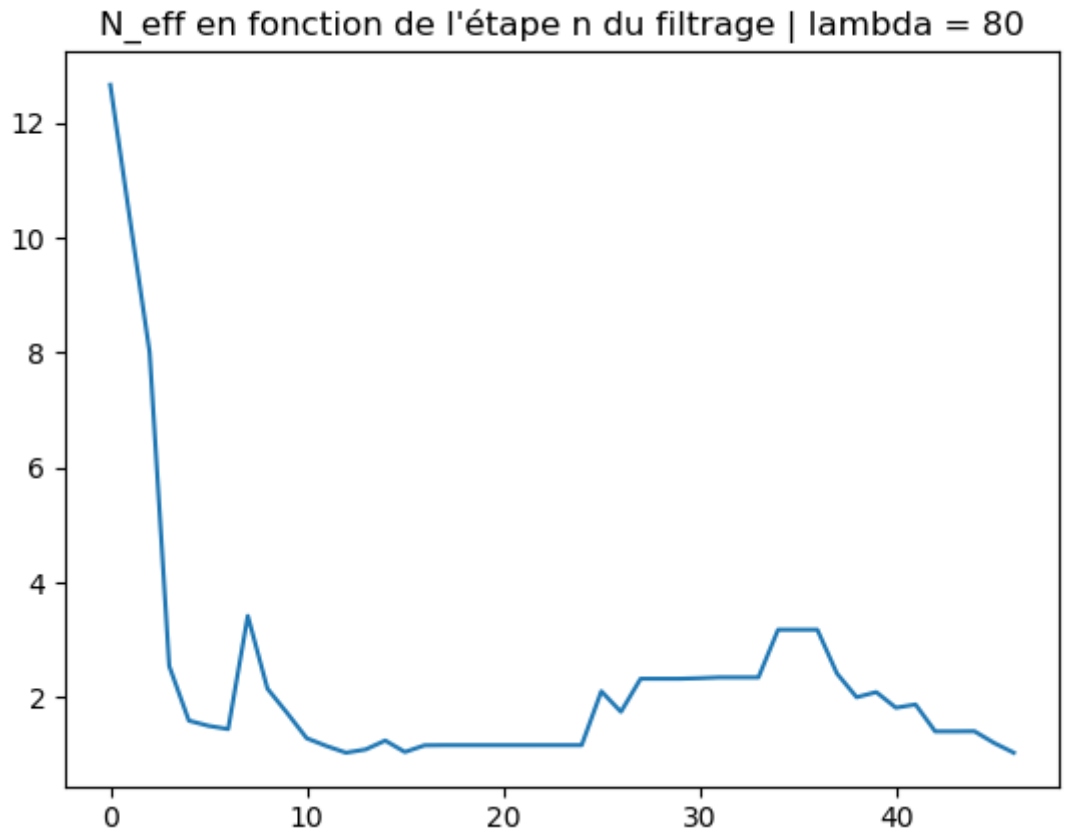

Figure 12 : N_eff en fonction du temps de la vidéo avec 50 particules initialement

Le nombre de particules va décroissant

Lors d'un croisement, il y a superposition de deux histogrammes de couleur différents ce qui brouille notre manière de discriminer les particules. Notre algorithme n'est donc pas robuste à ces situations. Le même problème se pose d'ailleurs lorsque le visage que l'on suit passe devant un décor de même couleur que son teint. Les conditions optimales de notre manière de poursuivre sont obtenues pour un fond démarqué du visage et ne changeant pas de couleur spatialement.

Un algorithme de reconnaissance d'un appareil photo est capable de détecter sans action humaine un visage puis de le suivre (dans notre cas, nous lui indiquons au travers du rectangle initial où se trouve le visage à suivre). Un tel algorithme est capable de garder en mémoire le visage pour continuer à le détecter sur d'autres images, même si elles comportent d'autres visages. Enfin, un tel algorithme sait traiter les situations de zoom/éloignement de l'individu. Autant de fonctionnalités que notre algorithme, à ce stade-là ne sait pas résoudre.

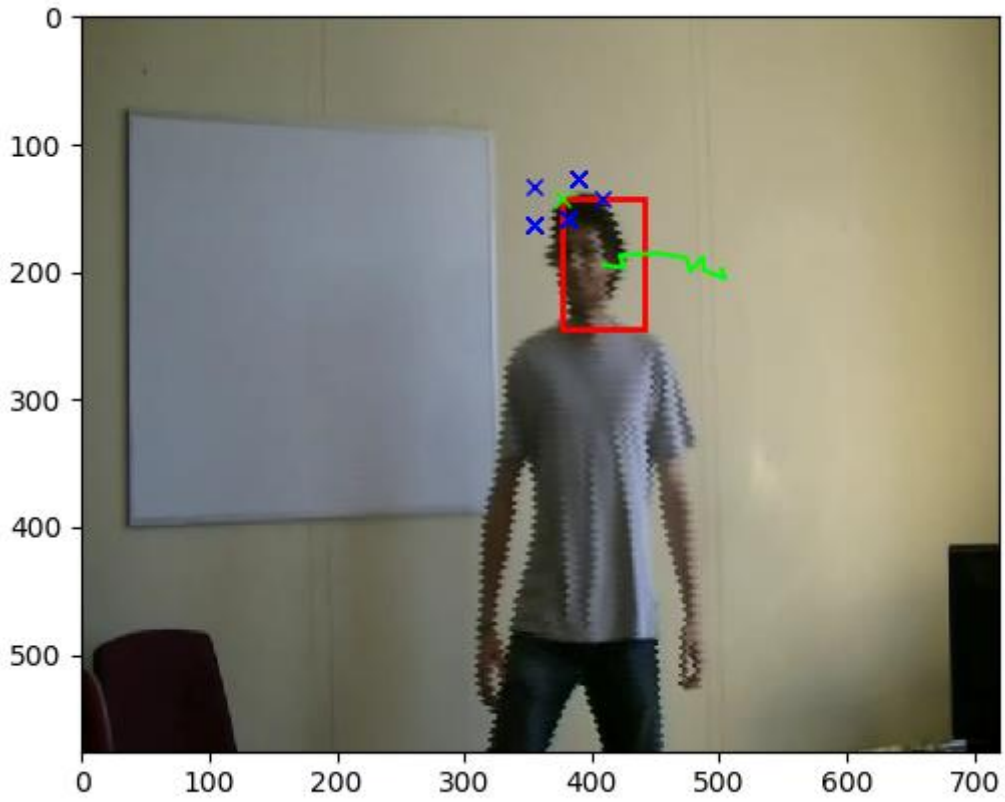

Figure 13 : Estimation de la position pour une étape intermédiaire de la vidéo

En vert, la trajectoire suivie du visage ainsi que la particule de plus grand poids

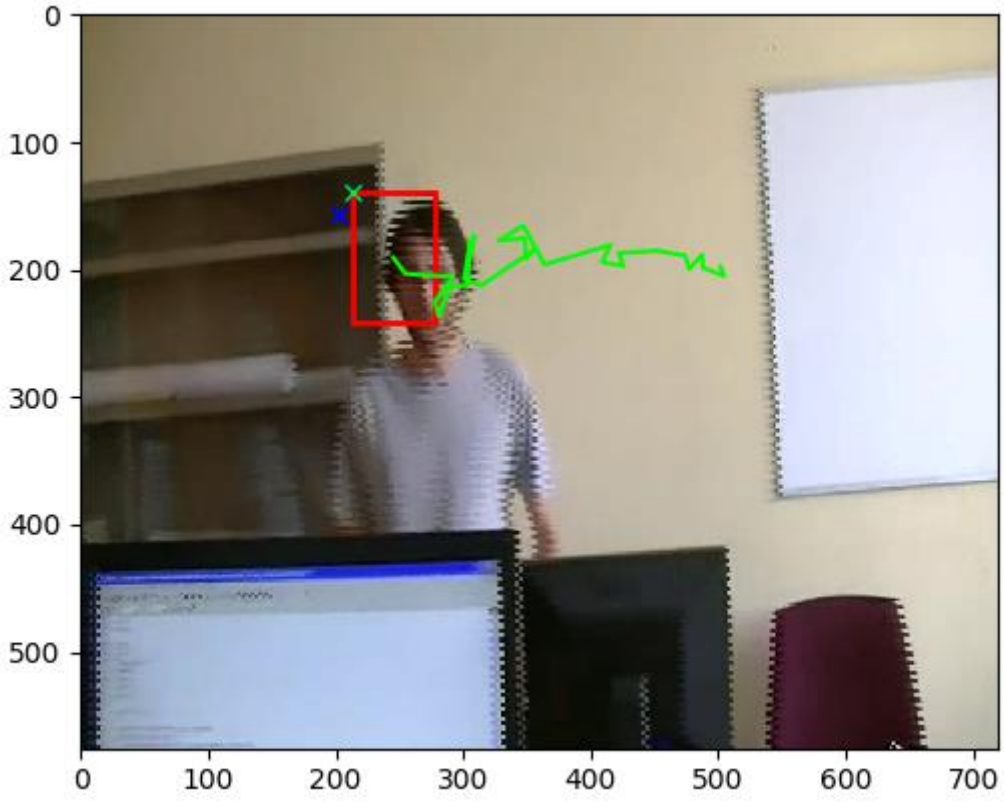

Figure 14 : Estimation de la position pour une étape intermédiaire de la vidéo

En vert, la trajectoire suivie du visage ainsi que la particule de plus grand poids

Conclusion :

Les derniers TP œuvraient à la poursuite de cibles dans le cas linéaire ou linéarisé avec le filtre de Kalman et le filtre de Kalman étendu. Nous avons à présent une méthode efficace de suivi de visage dans les cas non linéaires au travers du filtrage particulaire. Nous savons à présent que pour un suivi pertinent au coût raisonnable nous devons travailler avec une centaine de particules. Notre méthode, efficace dans la situation présente, comporte tout de même des limites : ne tient pas compte des mouvement avant/arrière (ce qui peut être résolu grâce à un paramètre d'échelle), reste sensible aux changements de décors, surtout s'ils sont de même couleur que le visage, diverge dans le cas de rencontres… Il n'empêche qu'il s'agit d'une application puissante lorsque les hypothèses de modèles sont respectées.

Code - Filtrage particulaire

Alexandre Chaussard, Antoine Klein

January 2022

1 Exercice 1

```python
import numpy as np
import matplotlib.pyplot as plt
import time

start_time = time.time()

# ----------- 3

Q = 10
R = 20
T = 50
N = 50


# G n rer la trajectoire
def generer_trajectoire(T, Q):
    x_o = np.random.normal(0, 1)
    X = [x_o]

    for i in range(1, T):
        u_n = np.random.normal(0, Q)

        x_n = 0.5 * X[i - 1] + 25 * X[i - 1] / (1 + X[i - 1] ** 2) + 8 * np.cos(1.2 * i) + u_n
        X.append(x_n)

    return np.array(X)


# ----------- 4

# G n rer les observations
def generer_observations(X, R):
    Y = []

    for i in range(0, len(X)):
        v_n = np.random.normal(0, R)
```

```
37
38          y_n = X[i] ** 2 / 20 + v_n
39          Y.append(y_n)
40
41      return np.array(Y)
42
43
44  # ----------- Tracer graphe
45
46  def erreur_quadratic(X, E):
47      sum = 0
48
49      for i in range(0, len(X)):
50          sum = sum + (X[i] - E[i]) ** 2
51
52      return sum/T
53
54  # Plot les graphes
55  def tracer(X, Y, Z=None):
56      abscisses = []
57      for i in range(0, len(X)):
58          abscisses.append(i)
59
60      fig, ax = plt.subplots()
61      ax.plot(abscisses, X, "r--", label="Trajectoire")
62      ax.plot(abscisses, Y, "+", label="Observations")
63
64      if Z is not None:
65          ax.plot(abscisses, Z, "b-.", label="Estim e")
66
67      ax.legend()
68
69      if Z is not None:
70          error = erreur_quadratic(X, Z)
71          plt.title("Trajectoire, observation et estim e | erreur
        quadratique moyenne : " + str(error));
72      else:
73          plt.title("Trajectoire et observations")
74      plt.show()
75
76
77  X = generer_trajectoire(T, Q)
78  Y = generer_observations(X, R)
79
80  print("--- %s seconds ---" % (time.time() - start_time))
81
82  tracer(X, Y)
83
84  # ----------- 5
85
86  from scipy import stats
87
88
89  # Fais un tirage selon f n | n-1
```

```
90 def tirage_f(x_n_prev, n, Q):
91     return np.random.normal(0.5 * x_n_prev + 25 * x_n_prev / (1 +
   x_n_prev ** 2) + 8 * np.cos(1.2 * n), Q)
92
93
94 # Evalue f n | n-1 en x_n, x_n_prev
95 def f(x_n, x_n_prev, n, Q):
96     return stats.norm.pdf(x_n, 0.5 * x_n_prev + 25 * x_n_prev / (1 +
   x_n_prev ** 2) + 8 * np.cos(1.2 * n), Q)
97
98
99 # Evalue g en y_n | x_n
100 def g(y_n, x_n):
101     return stats.norm.pdf(y_n, x_n ** 2 / 20, R)
102
103
104 def filtrage_particulaire(Y, N):
105     X_est = np.random.randn(N, 1)  # vecteur     RˆNx1
106     w = g(Y[0], X_est)  # vecteur des poids
107     w = w / np.sum(w)  # normalisation des poids
108
109     for t in range(1, T):
110
111         X_t_moins_1 = np.random.choice(list(X_est[:, -1]), N, list(w[:,
   -1])).reshape(-1, 1)  # tire N particules selon la pond ration w
112         X_t = 0.5 * X_t_moins_1 + 25 * X_t_moins_1 / (1 + X_t_moins_1 **
   2) + 8 * np.cos(1.2 * t) + Q**.5 *np.random.randn(N,
113                                                 1)  # application du
   mod le
114         w_t = g(Y[t], X_t)
115         w_t = w_t / np.sum(w_t)
116         X_est = np.hstack([X_est, X_t])
117         w = np.hstack([w, w_t])
118
119     return X_est, w  # vecteurs de RˆNxT
120
121
122 def esperance(X, W):
123     return (X * W).sum(0)
124
125
126 end_time = time.time()
127
128 X_est, W = filtrage_particulaire(Y, N)
129 E = esperance(X_est, W)
130
131 print("--- %s seconds ---" % (time.time() - end_time))
132
133 tracer(X, Y, E)
134
135 # EQM en fonction de N
136
137 T = 50
```

```
abscisse = []
error_results = []
for i in range(1, 100):
    N = i
    abscisse.append(i)
    error = 0

    for j in range(0, 100):
        X = generer_trajectoire(T, Q)
        Y = generer_observations(X, R)
        X_est, W = filtrage_particulaire(Y, N)
        E = esperance(X_est, W)
        error += erreur_quadratic(X, E)
    error = error/100
    error_results.append(error)

plt.title("EQM en fonction de N")
plt.plot(abscisse, error_results)
plt.show()

# Temps d'exec en fonction de N

abscisse = []
results = []
X = generer_trajectoire(T, Q)
Y = generer_observations(X, R)
for i in range(1, 1000):
    N = i
    abscisse.append(i)
    start_time = time.time()
    X_est, W = filtrage_particulaire(Y, N)
    E = esperance(X_est, W)

    spent = time.time() - start_time
    results.append(spent)

plt.title("Temps d'ex cution en fonction de N")
plt.plot(abscisse, results)
plt.show()

# EQM en fonction de T

abscisse = []
error_results = []
for i in range(1, 300):
    T = i
    abscisse.append(i)
    X = generer_trajectoire(T, Q)
    Y = generer_observations(X, R)

    X_est, W = filtrage_particulaire(Y, N)
    E = esperance(X_est, W)

    error = erreur_quadratic(X, E)
```

```
    error_results.append(error)

plt.title("EQM en fonction de T")
plt.plot(abscisse, error_results)
plt.show()
```

2 Exercice 2

```
import math
import scipy.stats
from fonctions_TP_python import *

# Nombre de particules
N = 50
# Nombre de couleurs dans l'histogramme
N_b = 10
# Variable de discrimination des particules par l'histogramme de couleur
    associ
lamb = 20
# Valeurs de la matrice de bruit pour la variable al atoire U
C1 = 300
C2 = 300

# On charge la 1 re  image
im, filenames, T, SEQUENCE = lecture_image()
# On s lectionne la zone    suivre
zoneAT = selectionner_zone()

# On d finit les dimensions de l'image extraite
h_width = zoneAT[2]
h_high = zoneAT[3]


# Permet d'initialiser les particules
def init_particles(sigma, mu):
    return sigma * np.random.randn(N, 2) + mu


# On initialise les particules
particles = init_particles(math.sqrt(300), [zoneAT[0], zoneAT[1]])


# D finie la distance BH entre deux histogrammes
def D(q, q_prime):
    somme = 0

    for i in range(0, N_b):
        somme = math.sqrt(q[i] * q_prime[i])

    return math.sqrt(1 - somme)


```

```
# Loi de y sachant x selon les histogrammes
def g(q, q_prime):
    return math.exp(- lamb * (D(q, q_prime) ** 2))


# On calcule l'histogramme de r f rence , en supposant qu'il n' volue pas beaucoup lors du suivi
q_img, q_kmeans, q_histo_original = calcul_histogramme(im, zoneAT, N_b)
plt.pause(0.01)


# Fonction de filtrage particulaire
def filtrage_particulaire():
    X_est = particles  # initialisation bruit e , vecteur     RˆNx1
    estimated_trajectory_X = []  # initialisation du tableau des X estim s
    estimated_trajectory_Y = []  # initialisation du tableau des Y estim s
    N_eff_list = [] # initialisation de la liste des N_eff

    w = np.array([1 / N] * N).reshape(N, 1)  # matrice des poids, initialement uniformes

    X_t = X_est  # X_0 correspond aux particules initiales

    # On plot les particules
    for i in range(0, N):
        plt.plot(X_t[i][0], X_t[i][1], marker='x', color='blue')

    plt.pause(0.01)

    # On parcourt les images enregistr es
    for t in range(1, T):

        # Application de la loi de transition -> on propage les particules
        U = math.sqrt(C1) * np.random.randn(N, 2)
        X_t = X_t + U  # application du mod le

        W_t = []  # On pr pare la liste des poids    l'instant t

        # On r cup re l'image de l'instant t
        new_img = Image.open((SEQUENCE + filenames[t]))
        for i in range(0, N):
            rect_x = X_t[i][0]
            rect_y = X_t[i][1]
            q_prime_img, q_prime_kmeans, q_prime_histo = calcul_histogramme(new_img,

                                                            [rect_x, rect_y, h_width, h_high],

                                                            q_kmeans)

            w_value = w[i][-1] * g(q_histo_original, q_prime_histo)  # On
```

```
    calcule le poids de la particule
90              W_t.append(w_value)
91
92          W_t = np.array(W_t)
93          W_t = W_t / np.sum(W_t)  # On normalise les poids
94
95          N_eff_inv = 0 # On pr pare le calcul de N_eff
96          for i in range(0, N):
97              w_value = W_t[i]
98              N_eff_inv += w_value**2
99
100         N_eff_list.append(1 / N_eff_inv)
101
102         indice = np.random.choice(N, N, p=W_t)  # tire N particules selon
    la pond ration W_t
103
104         # r  chantillonnage
105         X_t = X_t[indice]  # On r cup re depuis les indices tir s un
    vecteur X_t r  chantillon s
106
107         plt.cla()  # Efface le plot
108         plt.clf()  # Efface la figure
109
110         EX = 0  # Calcul de l'esp rance en X
111         EY = 0  # Calcul de l'esp rance en Y
112
113         for i in range(0, N):
114             plt.plot(X_t[i][0], X_t[i][1], marker='x', color='blue')
115
116             x_value = X_t[i][0]
117             y_value = X_t[i][1]
118             EX += x_value
119             EY += y_value
120
121         EX = EX / N
122         EY = EY / N
123
124         estimated_trajectory_X.append(EX + h_width/2)
125         estimated_trajectory_Y.append(EY + h_high/2)
126
127         X_est = np.hstack([X_est, X_t])  # On ajoute X_t    la matrice
    X_estim  en derni re colonne
128
129         W_t = W_t.reshape(W_t.shape[0], 1)
130         w = np.hstack([w, W_t])  # On ajoute W_t    la matrice des poids
    w
131
132         # Plot les particules, le rectangle et la trajectoire
133         plt.plot(estimated_trajectory_X, estimated_trajectory_Y, color="
    lime")
134         plt.imshow(new_img)
135         rect = ptch.Rectangle([EX, EY], h_width, h_high, linewidth=2,
    edgecolor='red', facecolor='None')
136         plt.plot(EX, EY, marker='x', color='lime')
```

```
        currentAxis = plt.gca()
        currentAxis.add_patch(rect)

        plt.pause(0.01)

    return X_est, w, N_eff_list  # vecteurs de R^NxT


X_est, w, N_eff_list = filtrage_particulaire()

abscisse = []
for i in range(0, len(N_eff_list)):
    abscisse.append(i)

plt.cla()
plt.clf()

plt.title("N_eff en fonction de l' tape  n du filtrage | lambda = " + str
    (lamb))
plt.plot(abscisse, N_eff_list)
plt.pause(100)
```