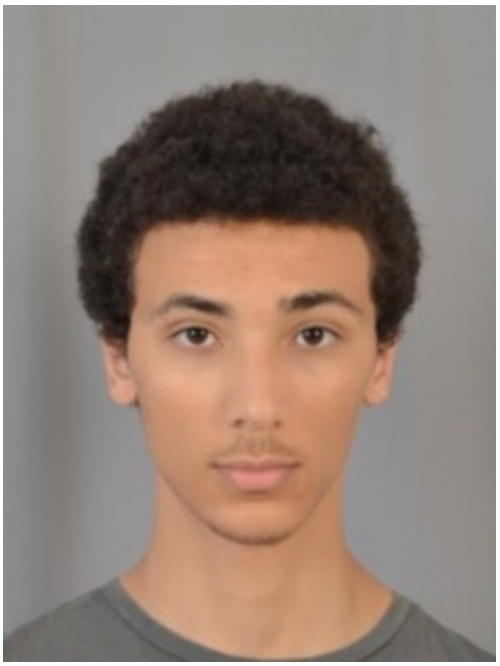
# MAT4251 - TP Filtrage Particulaire



**Taha Habib**

**Ezzahir Omar**

# Introduction :

Les données mesurées, par un radar par exemple, sont plus ou moins éloignées de la réalité en fonction du bruit de mesure, des perturbations auxquelles le système est soumis (le vent par exemple). On cherche alors à filtrer ce bruit afin d'obtenir un signal plus proche des états réels de notre système. Le TP précédent portant sur le Filtre de Kalman nous fournissait une méthode de filtrage dans un cadre linéaire. On cherche en fait à calculer $\int h(x)\pi(x)dx$, lors du premier TP on s'est arrêté au cas h linéaire (ou h peu pentue pour l'EKF). Au cours de ce TP on appliquera une autre méthode plus générale.

# Initialisation :

**1)** Nous étudions ici le modèle décrit dans le début du sujet. On décrira dans cette partie le principe du filtre particulaire pour t=0,1 puis on implémentera sur python l'algorithme correspond sur python. On souhaite calculer efficacement $E(X_t|Y_{0:t}) = \int x_0 \cdot p(dx_0|y_{0:t})$, on a :

$$p(dx_0|y_0) = \frac{p(y_0|x_0)p(dx_0)}{\int p(y_0|x_0)p(dx_0)} \tag{1}$$

commençons alors par discrétiser la mesure en remplaçant $p(dx_0)$ par $\frac{1}{n}\sum_{i=1}^{n}\delta_{X_0^i}(dx_0)$ puis on réinjecte dans (1), on obtient alors :

$$\begin{aligned} p(dx_0|y_0) &= \frac{p(y_0|x_0)p(dx_0)}{\int p(y_0|x_0)p(dx_0)} \\ &= \frac{\frac{1}{n}\cdot p(y_0|x_0)\sum_{i=1}^{n}\delta_{X_0^i}(dx_0)}{\frac{1}{n}\cdot\int p(y_0|x_0)\sum_{j=1}^{n}\delta_{X_0^j}(dx_0)} \\ &= \frac{\sum_{i=1}^{n} p(y_0|X_0^i)\delta_{X_0^i}(dx_0)}{\sum_{j=1}^{n} p(y_0|X_0^j)} \end{aligned} \tag{2}$$

Ainsi avec les notations ci-dessous on a :

$$w_0^i = \frac{p(y_0|X_0^i)}{\sum_{j=1}^{n} p(y_0|X_0^j)}$$

$$p(dx_0|y_0) = \sum_{i=1}^{n} w_0^i \delta_{X_0^i}(dx_0)$$

Notons d'ailleurs que le signe égal dans les équations ci-dessus n'est pas rigoureusement correct, il s'agit simplement d'approximations. On a alors une approximation de l'espérance pour t=0, passons donc à t=1 :

$$\begin{aligned} \int x_1 \cdot p(dx_1|y_0) &= \int\int p(dx_1, dx_0|y_0)h(x_1) \\ &= \int\int p(dx_1|x_0, y_0)p(dx_0|y_0)h(x_1) \\ &= \int\int p(dx_1|x_0)p(dx_0|y_0)h(x_1) \\ &= \int\int p(dx_1|x_0)\sum_{i=1}^{n} w_0^i\delta_{X_0^i}(dx_0)h(x_1) \\ &= \int h(x_1)\sum_{i=1}^{n} w_0^i \cdot p(dx_1|X_0^i) \end{aligned} \tag{3}$$

On vient en fait de remplacer la mesure $p(dx_1|y_0)$ par $\sum_{i=1}^{n} w_0^i \cdot p(dx_1|X_0^i)$. Tout notre but sera de calculer les poids $w_k^i$, remarquons toutefois que les poids trop faibles provoquent une dégénérescence de notre algorithme. En effet, au bout de quelques itérations, les poids faibles vont devenir encore plus faibles (quasi nuls) ainsi nos estimations ne reposeront plus que sur quelques particules ce qui réduit l'efficacité de notre algorithme. On procède alors à un rééchantillonage dont le but est de garder uniquement les poids les plus importants.
Ainsi, pour $i \in \{1, ..., n\}$ , on tire un entier $A_0^i$ en assignant à chaque entier k la probabilité $w_0^k$ puis on tire ensuite $X_1^i$ selon la loi $p(dx_1|X_0^{A_0^i})$, ainsi, chaque $X^i$ est tiré selon $\frac{1}{n} \cdot \sum_{i=1}^{n} \delta_{X_1^i}(dx_1)$.
On en tire alors l'algorithme suivant en appliquant le même procédé pour chaque pas de temps :

**Algorithm 1** Filtrage particulaire avec SIR

1: On initialise $X_t$ et w aléatoirement à partir de Q,R et y
2: **for each** $t \in \{0, ..., T-1\}$ **do**
3: On tire une liste A de N entiers tiré selon les poids de chaque entier
4: On tire une liste de N positions, le i-ème élement de cette liste est tiré sera $X_{t-1}$[A[i]]
5: On crée $X_t$ en injectant la liste précédente dans la formule d'actualisation.
6: On normalise les poids
7: **end for**

# Application :

On se propose désormais d'étudier le modèle de la première partie du TP, servant d'indicateur qui nous permettra de détecter les failles de notre algorithme et éventuellement l'améliorer. Le modèle est le suivant :

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

On déduit ainsi la loi de transition des particules :

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

Et la loi de vraisemblance de chaque particule donnée par :

$$g_n(y_n|x_n) = \frac{x_n^2}{20}$$

Remarquons d'ailleurs qu'on ne peut déterminer $X_n$ qu'au signe près à partir des observation puisque $X_n$ est élevée au carrée dans la formule d'observation.
Après avoir codé le programme de filtrage particulaire aveec rééchantillonnage, nous avons tracer les différentes trajectoires réelle, observée et estimée par le filtrage. La figure 1 ci-dessous regroupe les 3 trajectoires, avec en vert la trajectoire réelle, en bleu la trajectoire observée et en rouge la trajectoire estimée. Les paramètres pris sont : N=50, T=50, Q=10 et R=1.

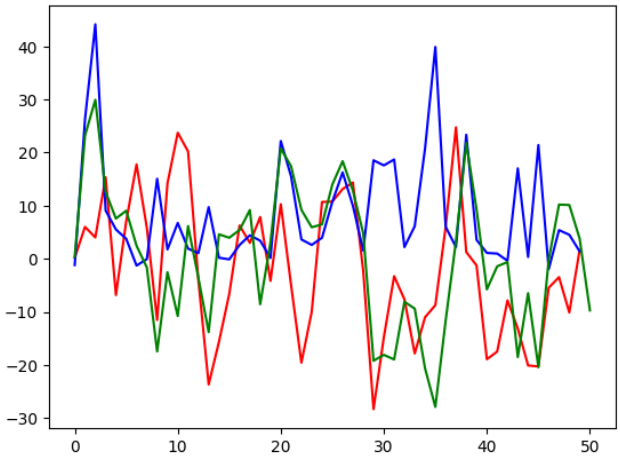



FIGURE 1 – N=50, T=50, Q=10, R=1

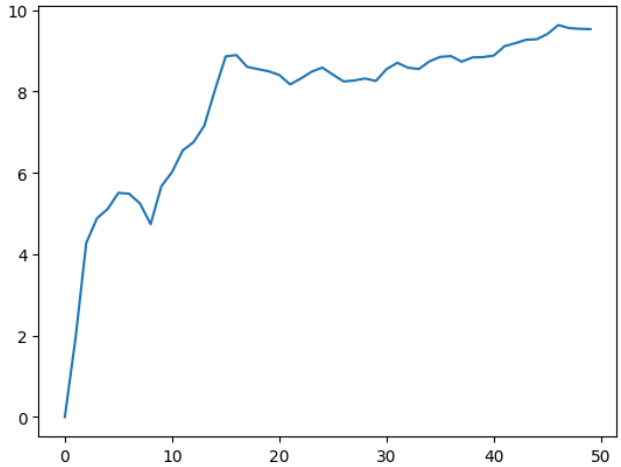



FIGURE 2 – EQM

Nous remarquons que la trajectoire estimée ne suit la trajectoire réelle que dans des portions et on a un écart trop grand entre les 2 trajectoires partout ailleurs, cela est dû à la valeur trop elevée de Q. Remarquons que dans la figure 2, le filtre converge à la fin, mais il prend plus de temps qu'un filtre moins bruité comme on verra dans la suite.
Afin de tester le filtre construit dans des conditions plus favorables, on se propose alors de réduire Q à 1, on obtient alors les figures 2 et 3. On remarque déjà des estimations plus pertinentes, les courbes vertes et rougent coïncident bien plus souvent. Toutefois notons l'impact de l'incapacité à déterminer exactement le signe de $X_n$, en effet on remarque par moment une symétrie axiale entre la courbe rouge et la courbe verte. On obtient malgré cela un filtre convergent comme le montre la figure 3.

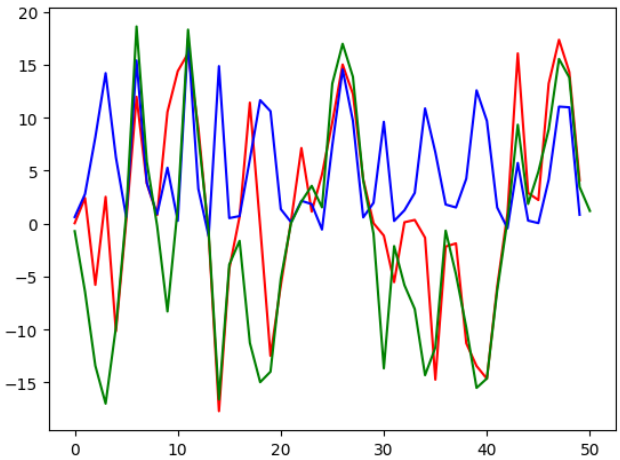



FIGURE 3 – N=1000, T=50, Q=1, R=1

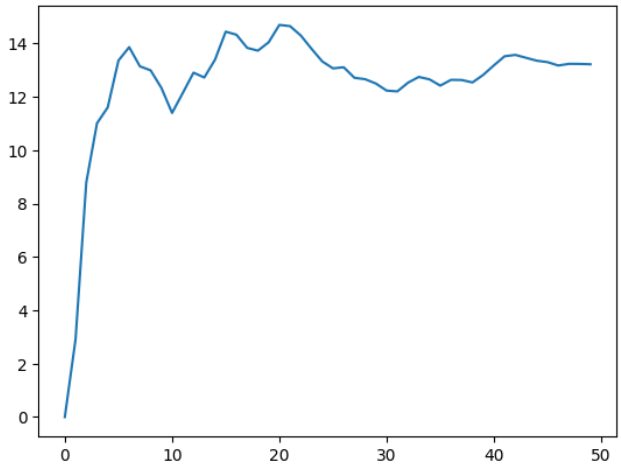



FIGURE 4 – EQM

Durant toutes ces manipulations, nous avons été contraints de relancer le programme plusieurs fois, en effet, on a souvent au bout d'un certain nombre d'itérations tout les poids qui deviennent nuls, rendant alors impossible la normalisation et bloquant l'algorithme. Pour le cas où la somme des poids est nulle, cela veut dire qu'aucune des particules n'a une probabilité non-nulle de représenter l'état réelle. Une des solutions dans ce cas où tout les poids sont nuls serait de donner à toutes les particules un poids $\frac{1}{N}$ de façon à pouvoir continuer l'algorithme.

## Poursuite de cible :

On souhaite désormais appliquer ce type d'algorithme à la poursuite de cible. Le but sera de construire un programme qui prend en entrée une vidéo, un certain nombre de paramètres et un rectangle que l'on voudrait traquer et le programme devra suivre cette zone tout le long de la vidéo.
Le sujet nous fournit le modèle à utiliser, un modèle qui semble assez simpliste. L'équation d'état est de la forme suivante :

$$X_n = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X_{n-1} + U_n$$

Dans toutes les figures ci-dessous le rectangle rouge correspond à la moyenne de chaque particule, le rectangle bleu correspond à la particule de poids maximal. La partie déterministe du modèle ne comprend en fait quasiment aucune information et tout repose sur la mesure des distances et le choix des bons poids. Cela est en fait assez cohérent puisque on peut difficilement avoir des informations sur la dynamique de déplacement d'une cible. On pourrait toutefois l'améliorer si l'on a déjà effectué en amont des mesures sur la dynamique de l'objet et que l'on a une idée sur sa vitesse.

On travaille désormais sur une vidéo où le mouvement est rapide afin d'évaluer l'efficacité du filtre et les problèmes auxquels il fait face dans ces situations. Comme l'illustre la fig.11, le filtre peine à traquer les mouvement rapide. En effet, en comparant à la fig.11, les rectangles ont un "retard" et ne retrouve le visage que lorsqu'il est déjà sur le point de bouger à nouveau, tandis que pour le mouvement lent, lorsque le rectangle a le temps de corriger l'erreur d'une étape à l'autre puisque le visage n'a pas trop bougé. Toutefois on peut penser à une solution assez simple : Filmer avec une caméra de bonne qualité permettant un nombre élevé d'images par seconde permettra de "lisser" le mouvement aux yeux du programme qui le verra au ralenti en quelque sorte et permettra donc un tracking efficace. On peut également affiner la partie déterministe de notre modèle comme cité plus haut.

Dans un soucis de compréhension de la construction des poids dans notre algorithme, on va désormais modifier les valeurs prises par $\lambda$ afin d'en comprendre le rôle. On change alors notre algorithme afin que l'ensemble des particules vérifiant $w_i > 0$ soient illustrées. On se propose également d'écrire à côté de chaque particule son poids afin de bien rendre compte de l'effet du changement de $\lambda$

On retrouve ainsi dans les figure 7 et 8 ci-dessous, qu'en augmentant la valeur de $\lambda$, le nombre de particules ayant un poids nul/presque nul augmente aussi, cela s'explique par le fait que $\lambda$ intervient dans la loi de probabilité de la vraisemblance comme facteur à la distance D, ainsi pour une valeur de $\lambda$ élevée, le produit $\lambda * D$ sera grand et ainsi l'exponentiel sera proche du zéro. Ce paramètre permet en fait de déterminer à quel point on sera stricte dans notre façon de déterminer les poids, si $\lambda >> 0$, alors la moindre différence d'histogramme est sanctionnée.
On remarque également que l'algorithme semble reconnaître sur le mur un histogramme qui ressemble à celui du visage traqué et autour duquel se forme un cluster de particule.

Notons que le nombre effectif de particules intervenant dans les calculs est inversement proportionnel à $\lambda$. Pour $\lambda = 1$ la totalité des particules participent aux calculs, tandis que pour $\lambda = 100$, moins de la moitié des particules participent effectivement au calcul.

D'après la figure 10, on observe que le filtre détecte bien le visage de la personne pistée au début de la séquence, cependant, une fois la personne retourne son visage et s'éloigne de la caméra, le filtre a des difficulté à detécter son visage. Les particules restent autour de la même zone et dès que la deuxième personne arrive à cet endroit elles se mettent à la traquer.
Finalement, nous améliorons notre code pour qu'il s'adapte avec la distance du visage pisté et adapte

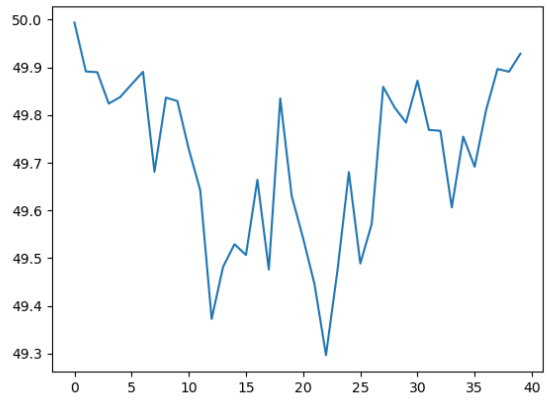



FIGURE 5 – $N_{eff}$ pour $\lambda = 1$

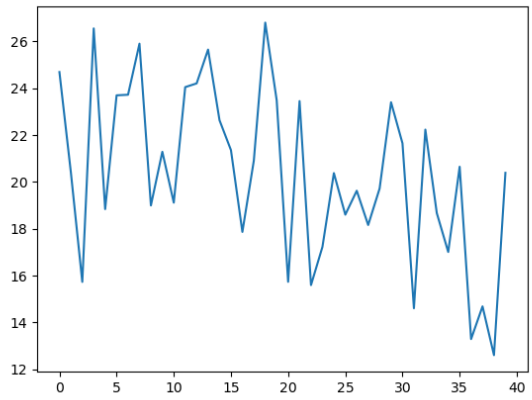



FIGURE 6 – $N_{eff}$ pour $\lambda = 100$

ainsi la taille du rectangle associée avec le zoom. La figure 12 démontre bien cela. En effet, on remarque bien que la taille du rectangle varie avec la distance entre le visage et la caméra.

En conclusion, en général, les algorithmes de reconnaissance faciale utilisés dans les appareils photo numériques sont plus simples et moins complexes que l'algorithme de suivi de visage présenté dans ce TP. Ils sont conçus pour fonctionner rapidement et efficacement sur des images individuelles, plutôt que sur une séquence d'images en temps réel comme c'est le cas pour l'algorithme de suivi de visage.

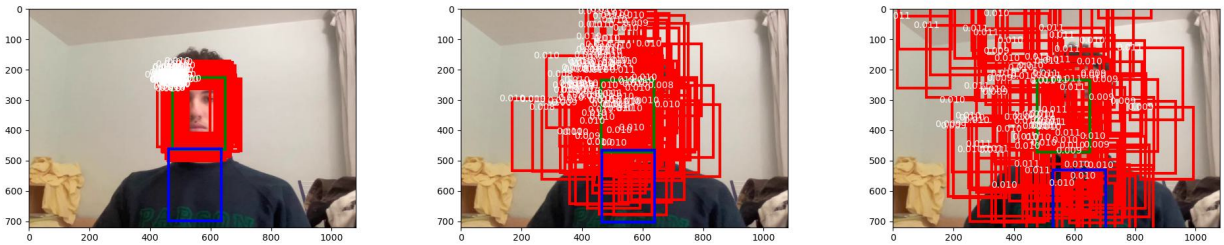


FIGURE 7 – $\lambda = 1$

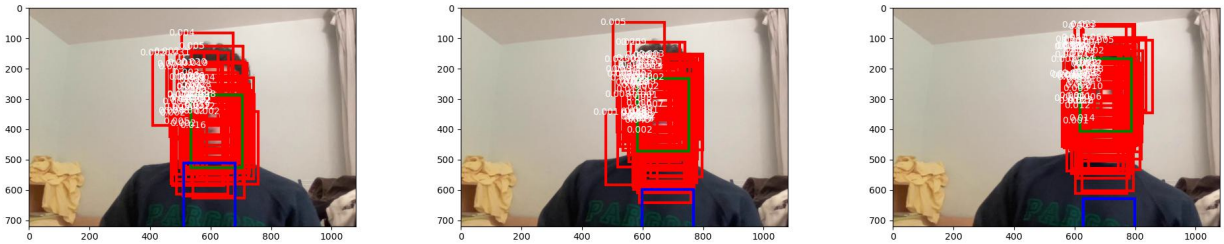


FIGURE 8 – $\lambda = 100$

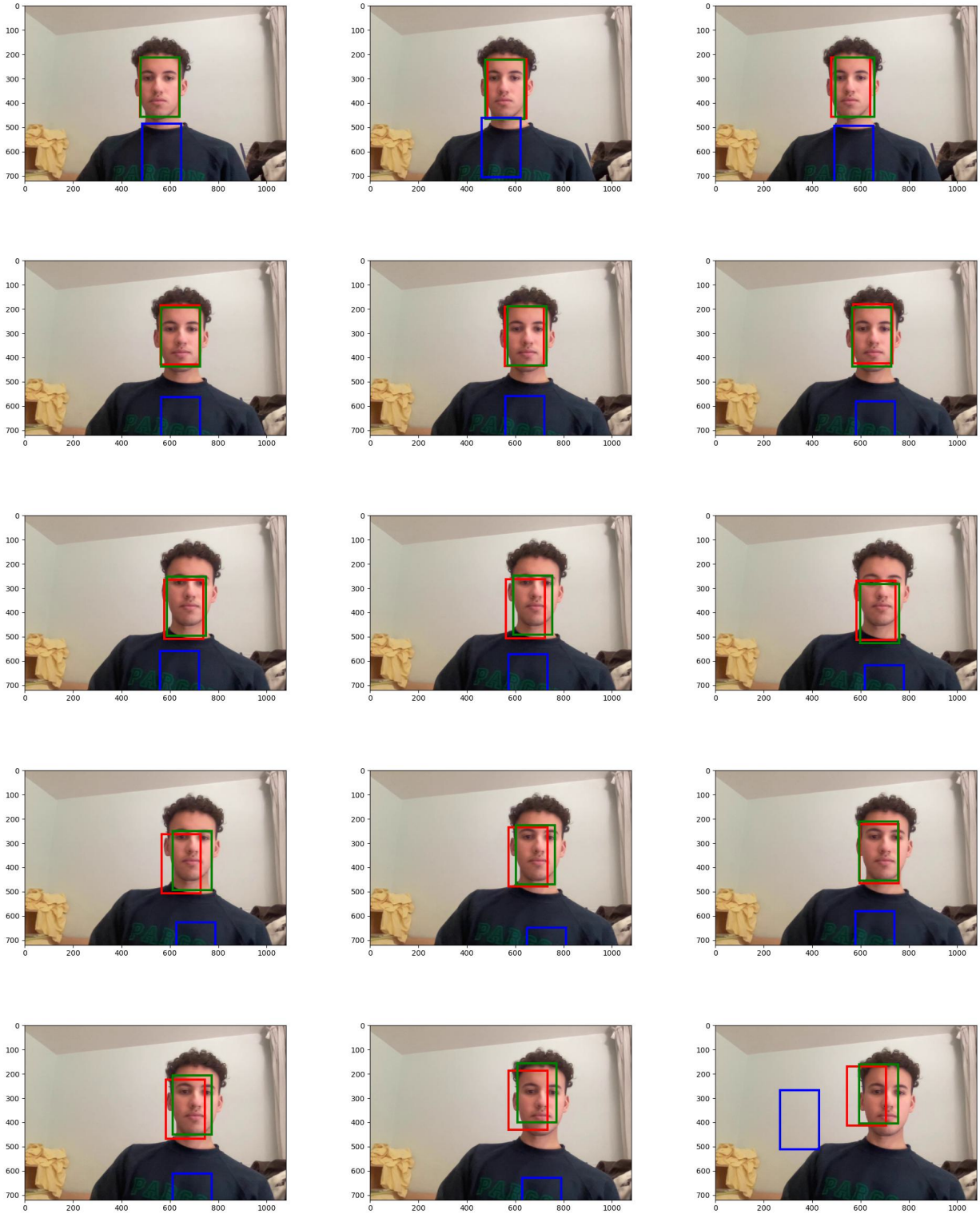


FIGURE 9 – Poursuite d'un mouvement lent pour référence

FIGURE 10 – Poursuite d'un mouvement avec croisement

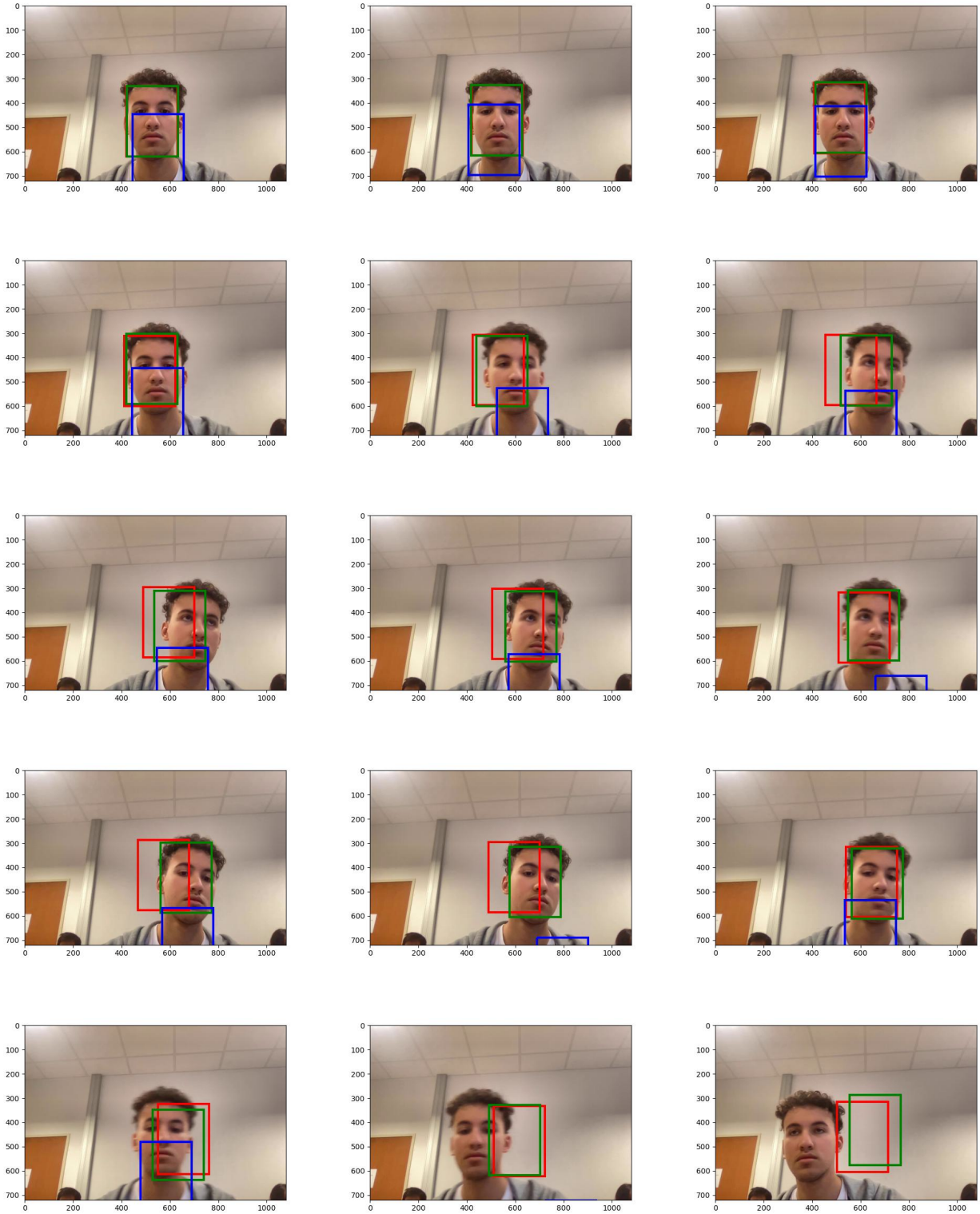


FIGURE 11 – Poursuite d'un mouvement rapide

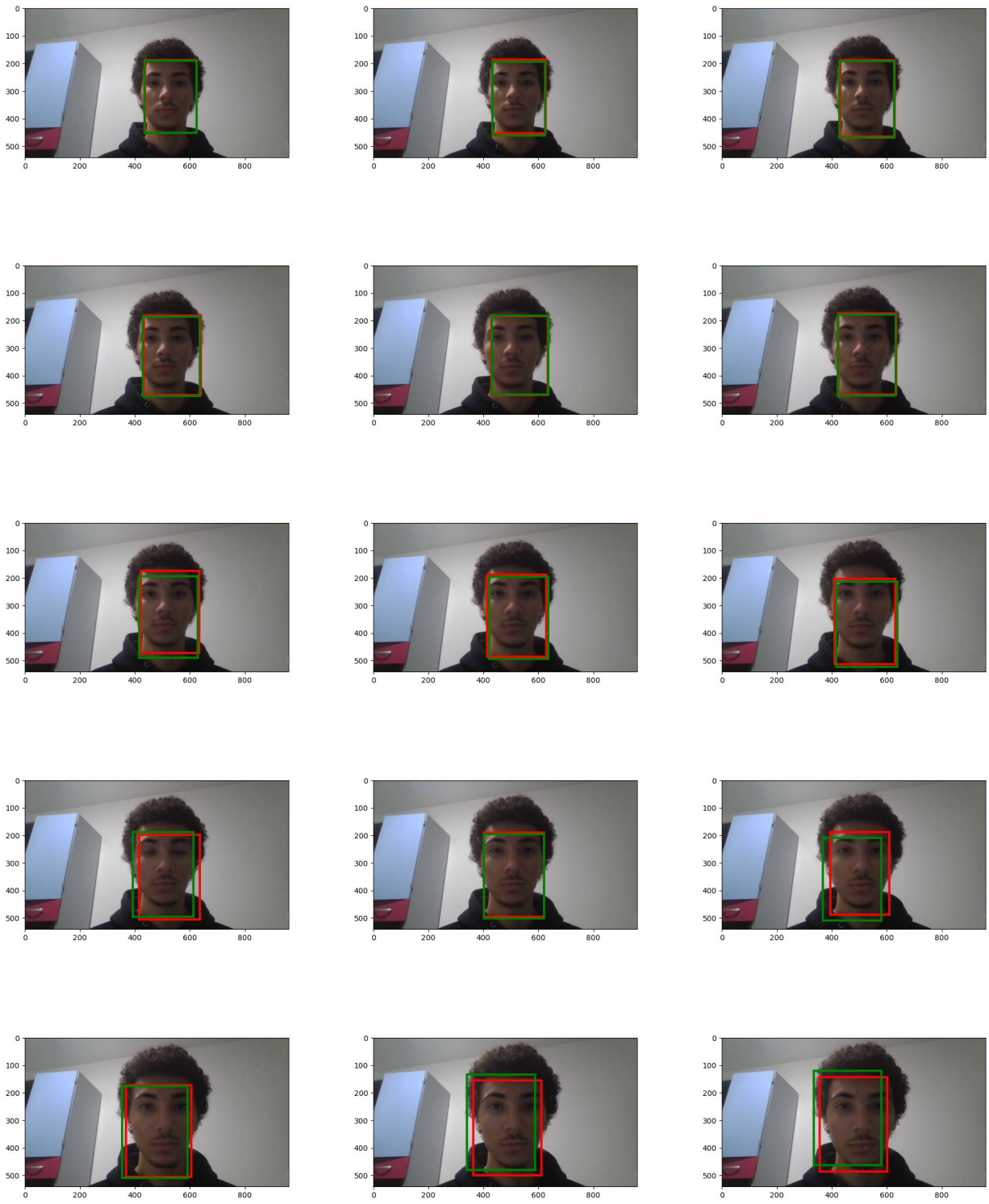


FIGURE 12 – Poursuite d'un rapprochement à la caméra