



CERGY PARIS  
UNIVERSITÉ

## RAPPORT

Pour [UE] Systèmes de traitement d'images

**Master 1 Informatique et Ingénierie des Systèmes Complexes**

sur le sujet

# Système complet de suivi d'objets

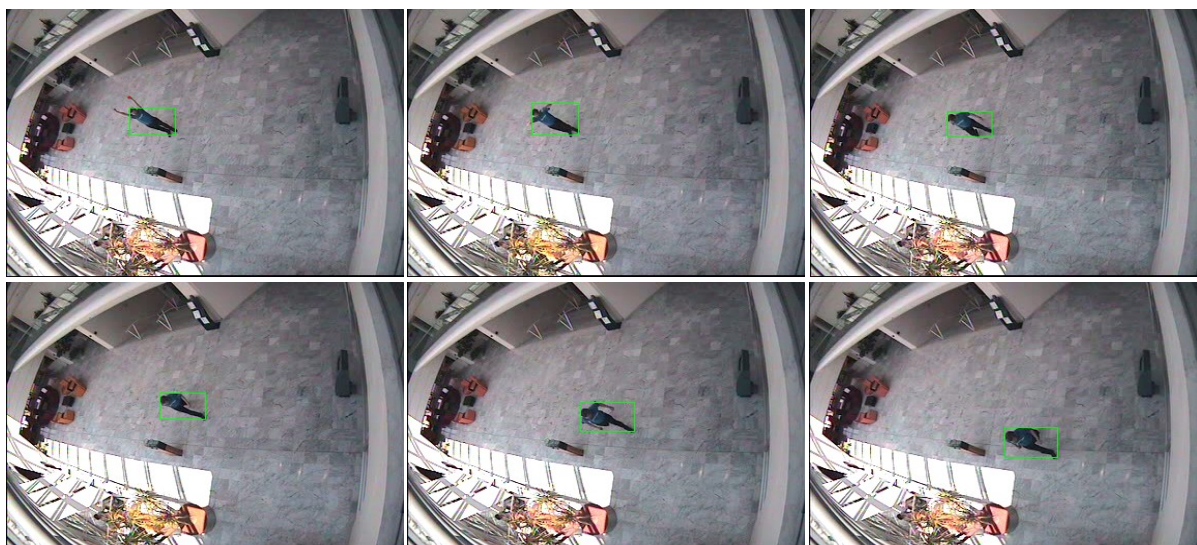
rédigé par

**Ishak Ayad**

Encadrant du projet

**Mme. Lise Aubin**

**M. Ghiles Mostafaoui**



Juin 2020

# Table des matières

<b>1</b>	<b>Réalisation</b>	<b>2</b>
1.1	Morphologie Mathématique : Érosion, Dilatation, Ouverture et Fermeture . . . .	2
1.2	Détection de mouvement par différence d'image consécutives . . . . .	4
1.3	Extraction d'image de référence . . . . .	5
1.3.1	Moyenne temporelle . . . . .	5
1.3.2	Filtre médian . . . . .	6
1.4	Étiquetage et Caractérisation des formes binaires . . . . .	7
1.5	Détection de points d'intérêt . . . . .	9
1.5.1	Méthode de Harris . . . . .	9
1.5.2	Méthode basée sur les direction de gradient . . . . .	10
1.6	Suivi d'objet avec sélection de vignettes autour des points d'intérêt . . . . .	12
1.7	Suivi d'objet Multi-critères . . . . .	13
<b>2</b>	<b>Annexe</b>	<b>17</b>

## Table des figures

1	Résultat dilatation carrée . . . . .	3
2	Résultat du débruitage du carré . . . . .	4
3	Résultat de la suppression du trou . . . . .	4
4	Résultat obtenu de l'image de référence par moyenne temporelle . . . . .	5
5	Matrice 3D utilisé pour l'implémentation du filtre médian . . . . .	6
6	Résultat obtenu de l'image de référence par filtre médian . . . . .	6
7	Résultat de l'algorithme d'étiquetage . . . . .	8
8	Résultats de la méthode de Harris avec un maximum de points . . . . .	10
9	Détection de points d'intérêt en utilisant la méthode basée sur les direction de gradient sans normalisation . . . . .	10
10	Détection de points d'intérêt en utilisant la méthode basée sur les direction de gradient avec normalisation . . . . .	11
11	Résultat du suivi d'objet avec sélection de vignettes autour des points d'intérêt .	13
12	Résultat du suivi d'objet Multi-critères, on peut remarquer sur la première photo que le taux de reconnaissance est plus élevé par le suivi avec les caractéristiques de l'image que celui avec les points d'intérêts . . . . .	16

## Listings

1	Structure de données représentant les caractéristiques pour une région . . . . .	9
2	Fonction qui calcule la convolution d'une image par un masque . . . . .	11
3	Le structure de données représentant une vignette . . . . .	12
4	Le structure de données représentant les paramètres d'une région . . . . .	13
5	Détection de points d'intérêt par la méthode de Harris . . . . .	17
6	Morphologie mathématique érosion et dilatation . . . . .	18
7	Filtre médian pour le calcul de l'image de référence . . . . .	19
8	Algorithme d'étiquetage par table de correspondance . . . . .	19
9	Mise à jour de la vignette et envoie de la nouvelle région . . . . .	21

# 1 Réalisation

Cette section fournit les différentes informations et techniques utilisées pour implémenter les différentes fonctionnalités attendues. L'organisation des différentes sous-sections suit la pertinence des différentes techniques utilisées pour réaliser le système de suivi. Je vais aussi présenter l'architecture utilisée afin de garantir une efficacité optimale dans la phase de développement du projet.

## 1.1 Morphologie Mathématique : Érosion, Dilatation, Ouverture et Fermeture

Il existe deux grands types de morphologie mathématique : la morphologie binaire (ou ensembliste) et la morphologie multiniveaux (ou fonctionnelle). La première nécessite en entrée une image binaire, la première étape consistera donc à binariser les images. Bien sûr, toutes les images ne se prêtent pas à ce type de traitement, c'est néanmoins souvent le cas des images industrielles possédant deux grandes classes de niveaux de gris : un correspondant à l'objet et un correspondant au fond). La morphologie multiniveaux s'applique elle directement sur les images.

La morphologie mathématique peut être développée dans le cadre abstrait de la théorie des treillis. Cependant, une présentation plus pratique, visant un utilisateur potentiel d'outils de traitement d'images, plutôt qu'un mathématicien, est ici adoptée.

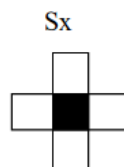
Plaçons-nous dans  $E = \mathbb{Z}^2$ , souvent utilisé comme modélisation du support des images binaires à deux dimensions, même si tout ce qui est présenté dans cette section reste valable dans  $\mathbb{R}^d$ , où  $d$  est un entier strictement positif. Soit  $B$  un sous-ensemble de  $E$ , appelé élément structurant. Si  $x$  est un élément de  $E$ , alors nous noterons  $B_x$  l'ensemble  $B$  translaté de  $x$  :  $B_x = \{b + x \mid b \in B\}$ .

- Objets :  $X = \{x \mid x = 1\}$
- Objets :  $X = \{x \mid x = 0\}$



L'élément structurant joue en quelque sorte le rôle de modèle local, ou de sonde. Il est promené partout sur l'image à traiter, et à chaque position on étudie sa relation avec l'image binaire, considérée comme un ensemble. Ces relations peuvent être du type « est inclus dans l'ensemble », ou « touche l'ensemble », par exemple.

En pavage carré, les éléments structurants les plus classiquement utilisés sont la croix, constituée de l'origine et des quatre points les plus proches, et le carré, constitué de l'origine et des huit points les plus proches. Ces deux éléments structurants correspondent respectivement à deux définitions possibles du voisinage d'un pixel ou du type de connexité de l'image. En pavage hexagonal, l'élément de base est l'hexagone centré.



**Dilatation et érosion morphologiques :** Soit  $X$  un sous-ensemble de  $E$ . La dilatation morphologique avec l'élément structurant  $B$  est définie comme la somme de Minkowski :  $\delta_B(X) = X \oplus B = \{x + b \mid b \in B, x \in X\} = \cup_{x \in X} B_x$ .

Une autre formulation plus intuitive est :  $\delta_B(X) = \{x \mid \check{B}_x \cap X \neq \emptyset\}$ .

La dilatation morphologique n'est, en général, pas inversible. L'opération qui en quelque sorte tente de produire l'inverse de la dilatation est l'érosion morphologique :  $\epsilon_B(X) = X \ominus B = \{x \mid B_x \subset X\}$ .

La dilatation et l'érosion sont les opérateurs de base de la morphologie mathématique. Pratiquement tous les autres peuvent être définis à l'aide de ceux-ci, en utilisant des compositions de fonctions et des opérations ensemblistes.

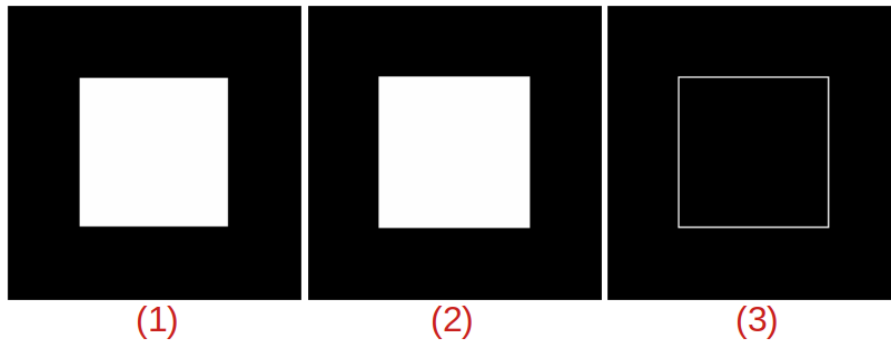


FIGURE 1 – Résultat dilatation carrée

1. Représente le carré dilaté de trois pixels
2. Image origine (en noire : le fond ; en blanc : le carré)
3. La différence entre l'image origine et l'objet dilaté

**Ouverture et fermeture :** La composition d'une dilatation morphologique avec l'érosion par le même élément structurant ne produit pas, en général, l'identité, mais deux autres opérateurs morphologiques, l'ouverture morphologique :  $\gamma_B(X) = X \circ B = \delta_B(\epsilon_B(X))$  et la fermeture morphologique :  $\phi_B(X) = X \bullet B = \epsilon_B(\delta_B(X))$ .

L'ouverture peut être caractérisée géométriquement : elle donne l'union de tous les  $B_x$  inclus dans  $X$ . Ainsi, la forme de l'élément structurant permet de choisir les structures qui peuvent le contenir.

La fermeture est le dual de l'ouverture : la fermeture du complémentaire d'un ensemble est égale au complémentaire de l'ouverture de cet ensemble.

On notera que si l'élément structurant  $B$  n'est pas symétrique, on devra utiliser l'élément symétrique  $\check{B}$  pour le second opérateur (dilatation dans le cas de l'ouverture et érosion dans le cas de la fermeture).

L'ouverture servira à la suppression des fausses alarmes :

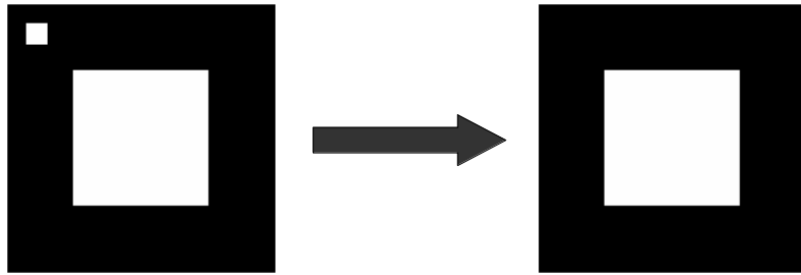


FIGURE 2 – Résultat du débruitage du carré

La fermeture supprimera les trous :

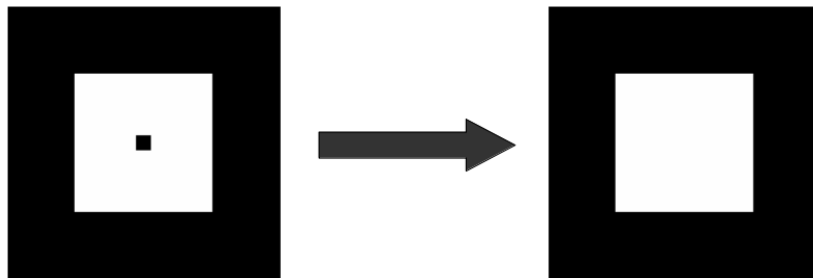


FIGURE 3 – Résultat de la suppression du trou

## 1.2 Détection de mouvement par différence d'image consécutives

La détection de mouvement par modélisation puis soustraction de l'arrière-plan (aussi connue sous les termes anglais de background soustraction ou foreground détection) permet de dissocier dans une séquence d'images les éléments ou régions en mouvement (ou mobiles) de ceux appartenant au décor de la scène (immobiles ou statiques). Les techniques de détection de mouvement utilisant cette stratégie se reposent sur l'hypothèse principale que la séquence vidéo, prise par une caméra fixe, est composée d'un background ou arrière-plan statique au-devant duquel les objets en mouvement peuvent être ainsi facilement détectés. La détection du mouvement est fondamentalement une opération de traitement bas niveau qui est en suite cruciale pour l'élaboration et la conception de techniques plus évoluées en vision par ordinateur tel que le suivi d'objets mobiles, l'analyse du mouvement humain, la reconnaissance d'actions humaines, la compression ou la compréhension de la scène.

Une façon de résoudre ce problème consiste à considérer simplement l'image précédente, dans la séquence, comme arrière-plan pour l'image courante :

$$|I(t) \ominus I(t-1)| > s, \text{ ou } s \text{ est le seuil à appliquer pour éliminer les bruits.}$$

Mais cette stratégie rend la méthode de détection de mouvement peu fiable notamment dans le cas où la vitesse des régions mobiles est trop lente ou dans le cas d'objets mobiles peu texturés possédant des régions d'intensité uniforme et pour lesquelles aucune variation d'intensité n'est visible que lorsque l'objet glisse sur lui-même.

### 1.3 Extraction d'image de référence

Une méthode simple de détection d'arrière-plan consiste à prendre une image de référence de la scène ne contenant aucun objet mobile puis, par simple différence et seuillage avec les différentes images de la séquence vidéo, identifier ensuite les pixels appartenant aux éléments mobiles n'appartenant pas au décor (statique) de la scène. Cependant, on comprend vite que l'usage d'une simple image de référence, décrivant l'arrière-plan statique, constitue un modèle trop simple et peu fiable en réalité. En effet, outre le fait qu'il est requis que la caméra soit parfaitement fixe, la luminosité de la scène change et varie dans le temps, l'image peut être entachée de bruit, un mouvement récurrent et inintéressant tel que celui d'une branche dans le vent qui peut être détectée en faux positif (tout comme la neige ou la réflexion soudaine du soleil sur une vitre peut engendrer une fausse détection), le support de la caméra peut trembler, etc. Ces erreurs possibles ont incité les auteurs à proposer des modèles de modélisation d'arrière-plan plus élaborés.

#### 1.3.1 Moyenne temporelle

Cette méthode utilise comme modèle d'arrière-plan la moyenne des  $n$  images précédant l'image courante :

$$BG(t) = \frac{1}{n} \times \sum_{k=0}^{n-1} I(t-k)$$

$$\text{et } |I(t) \ominus BG(t)| > s$$

Cependant, cette méthode montre vite des faiblesses lors de passages récurrents d'objets mobiles qui vont finir par laisser une trace dans le modèle d'arrière-plan même si le paramètre  $n$  est estimé en fonction du nombre d'images par seconde de la vidéo et de la vitesse des objets mobiles.



FIGURE 4 – Résultat obtenu de l'image de référence par moyenne temporelle

### 1.3.2 Filtre médian

Dans le cas où chaque pixel de la séquence d'images n'est pas majoritairement caché par un objet mobile, on peut aussi utiliser un filtre médian temporel calculé pour chaque pixel et utilisant les précédentes images de la séquence pour estimer un modèle d'arrière-plan. Cette stratégie va permettre d'obtenir un modèle de fond d'une meilleure qualité qu'une moyenne temporelle, permettant d'éviter les traces d'objets mobiles sur le modèle d'arrière-plan.

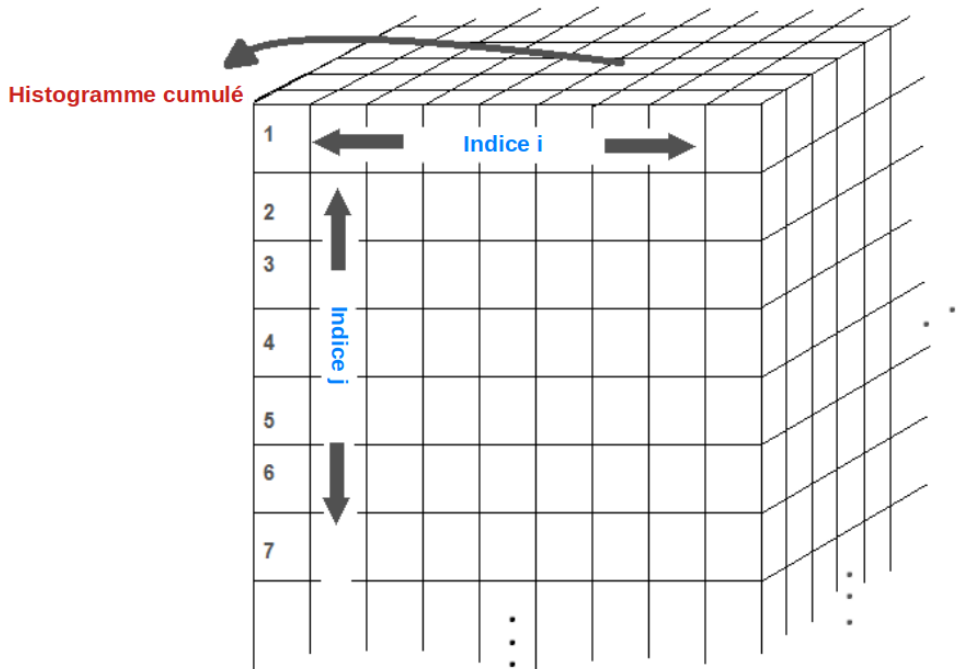


FIGURE 5 – Matrice 3D utilisé pour l'implémentation du filtre médian

L'implémentation de ce filtre est relativement simple, il suffit d'imaginer qu'on dispose d'une matrice 3D ou pour chaque pixel qui a pour coordonnées  $(i, j)$  on stocke son histogramme cumulé calculé à partir de la séquence vidéo, ainsi on peut obtenir pour tout pixel  $(i, j)$  sa valeur sur l'image référence en prenant la valeur sur l'histogramme cumulé d'indice  $\frac{n}{2}$ .

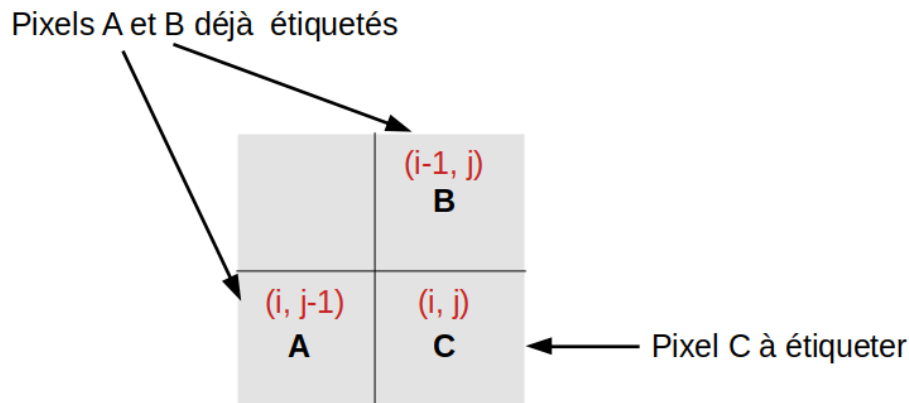


FIGURE 6 – Résultat obtenu de l'image de référence par filtre médian

## 1.4 Étiquetage et Caractérisation des formes binaires

Cette partie repose sur l'algorithme d'étiquetage, un algorithme de segmentation en région. La plupart des méthodes requièrent un algorithme d'étiquetage en composantes connexes : tous les pixels connexes possédant le même attribut doivent être affectés à la même région. Il existe plusieurs algorithmes permettant de réaliser l'étiquetage, dans notre cas j'utilise l'algorithme évolué qui offre un gain de temps, car il fait que deux parcours de l'image ont été effectués.

Supposons qu'à chaque pixel de l'image est affecté un attribut. On veut regrouper tous les pixels connexes possédant le même attribut (noté att dans la suite). Pour cela, on va balayer l'image et donner une étiquette (image notée E) à chaque nouveau pixel C.



Sur cette algorithme on utilise on une table de correspondance. Celle-ci est un vecteur ou à chaque étiquette, on affecte une autre étiquette qui lui correspond.

**Premier balayage :** On déplace un masque en L de la même façon que précédemment,

1. Si  $\text{att}(C) = \text{att}(A)$  et  $\text{att}(C) \neq \text{att}(B) \rightarrow E(C) = E(A)$
2. Si  $\text{att}(C) = \text{att}(B)$  et  $\text{att}(C) \neq \text{att}(A) \rightarrow E(C) = T[E(B)]$
3. Si  $\text{att}(C) \neq \text{att}(B)$  et  $\text{att}(C) \neq \text{att}(A) \rightarrow E(C) = \text{nouvelle étiquette}$
4. Si  $\text{att}(C) = \text{att}(B)$  et  $\text{att}(C) = \text{att}(A)$  et  $E(A) = E(B) \rightarrow E(C) = E(A)$
5. Si  $\text{att}(C) = \text{att}(B)$  et  $\text{att}(C) = \text{att}(A)$  et  $E(A) \neq E(B) \rightarrow$   
 $E(C) = \min(T[E(B)], E(A))$   
 $T[E(C)] = E(C)$   
 $T[E(A)] = E(C)$   
 $T[\max(T[E(B)], E(A))] = E(C)$

**Mise à jour de la table de correspondance :** Toutes les étiquettes telles que  $T[i] = i$  représentent de vraies régions. On leur affecte donc un numéro de région en numérotant dans l'ordre croissant.

Pour toutes les étiquettes qui ne valident pas l'hypothèse ci-dessus, on fait  $T[i] = T[T[i]]$ .

**Deuxième balayage :** A chaque pixel d'étiquette i, on lui affecte l'étiquette  $T[i]$ .

En plus de ça un petit algorithme pour retirer que la zone souhaitée ou l'entourer avec un rectangle englobant est mis en oeuvre. Ceci est fait en calculant les coordonnées minimales et maximales ( $\min_x, \min_y$ ) et ( $\max_x, \max_y$ ).



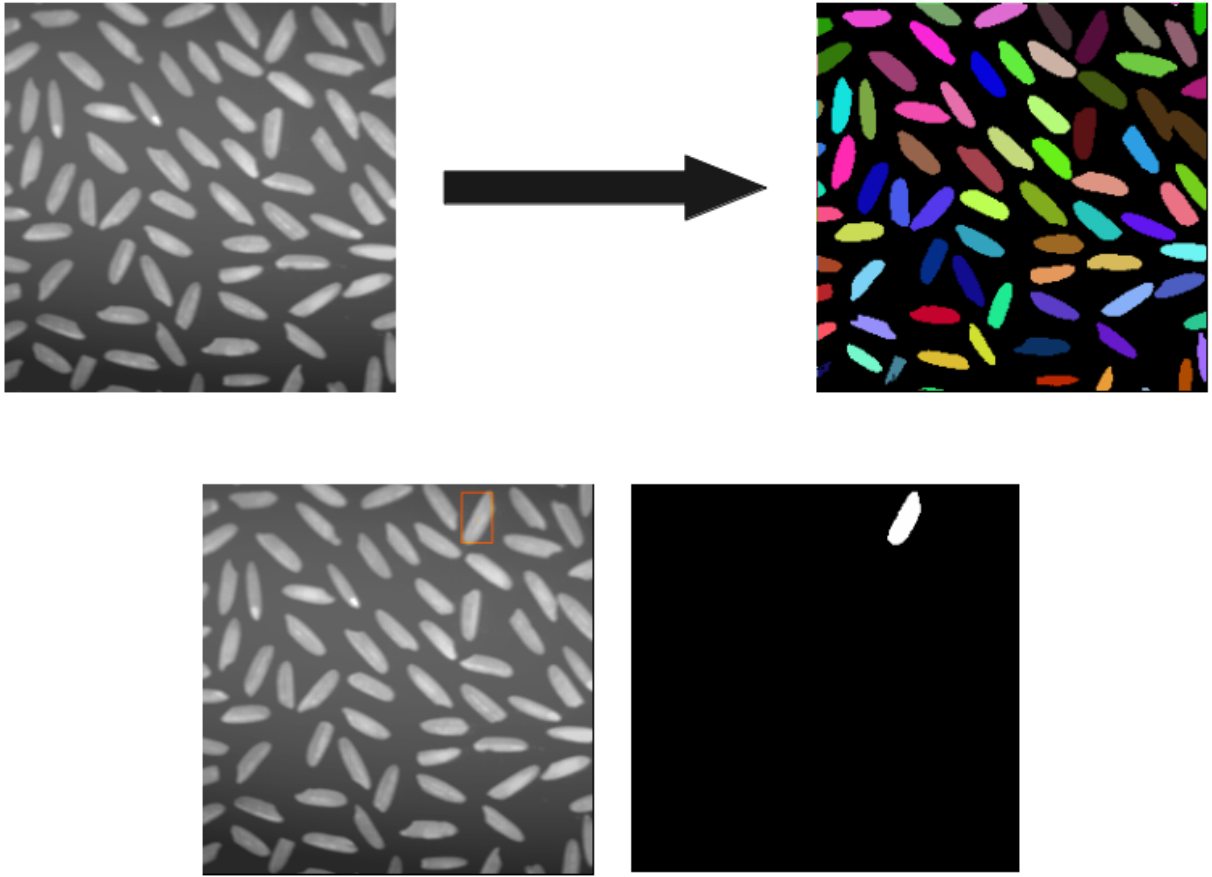


FIGURE 7 – Résultat de l'algorithme d'étiquetage

Après avoir réussi à isolé les différents régions sur une matrice  $E$ . On veut maintenant caractériser ces régions pour envoyer des vecteurs les représentant à un futur processus de reconnaissance des formes.

1. **Taille** : n des paramètres le plus intuitif est la taille de la région représentée dans l'image par son nombre de pixels.

$$size = \sum_{k=0}^n E(i, j) == label \mid_1$$

2. **Position** : La position de la région (barycentre).
3. **Direction principale** : Elle correspond au premier vecteur propre de la matrice :

$$\begin{pmatrix} \sigma_x^2 & \sigma_x \sigma_y \\ \sigma_x \sigma_y & \sigma_y^2 \end{pmatrix}$$

L'axe principal de la région est défini par l'angle  $\theta$  qu'il fait avec l'axe des  $x$  :

$$\theta = \arctan\left(\frac{2\sigma_y\sigma_x}{\sigma_y^2 - \sigma_x^2}\right)$$

Pour faciliter la manipulation ces valeurs sont stockées dans une structure de données, ainsi pour chaque image on aura un vecteur de la représentant.

```

1 typedef struct
2 {
3     int area_label; // numero de la region
4     int size; // taille de la region
5     float OX, OY; // barycentre
6     float E_x, E_y, E_xy; // orientation
7     float theta; // direction principale
8     float average_greyscale; // la moyenne des niveaux de gris
9     float avrage_R, avrage_G, avrage_B; // les moyennes sur les composante R, G, B
10    int* H; // histogramme des niveaux de gris
11 } Parameters;

```

Listing 1 – Structure de données représentant les caractéristiques pour une région

## 1.5 Détection de points d'intérêt

### 1.5.1 Méthode de Harris

La méthode consiste à partir des images  $I_x$  et  $I_y$ , calculer  $I_x^2$ ,  $I_y^2$  et  $I_{xy} = I_x^2 \times I_y^2$ , puis lisser les trois images en utilisant un filtre Gaussien 3x3 :

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Ensuite, en chaque pixel on calcule la fonction Harris :  $H = \det(\mathcal{C}) - \alpha(\text{trace}(\mathcal{C}))^2$ . La valeur de cette fonction dépend de la matrice d'autocorrelation  $\mathcal{C}$  :

$$\begin{pmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{pmatrix}$$

où  $\alpha$  est un paramètre modifiable du détecteur (valeur par défaut : 0,04).

Pour extraire les maxima locaux : après avoir calculé Harris en tout point de l'image, on doit faire une boucle sur chaque point de l'image, du type :

```

A B C
D X E
F G H

if(harris[X] > 0
&& harris[X] > harris[A]
&& harris[X] > harris[B]
&& harris[X] > harris[C]
&& harris[X] > harris[D]
&& harris[X] >= harris[E]
&& harris[X] >= harris[F]
&& harris[X] >= harris[G]
&& harris[X] >= harris[H]
) { inserer_le_point_dans_un_tableau_trie; }

```

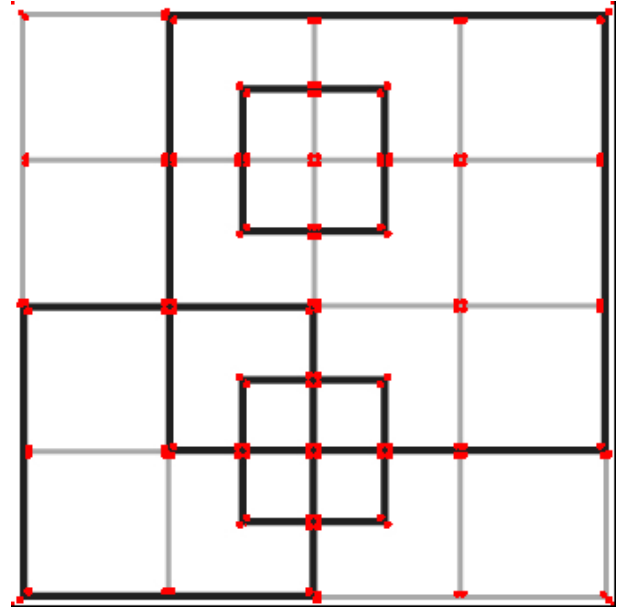
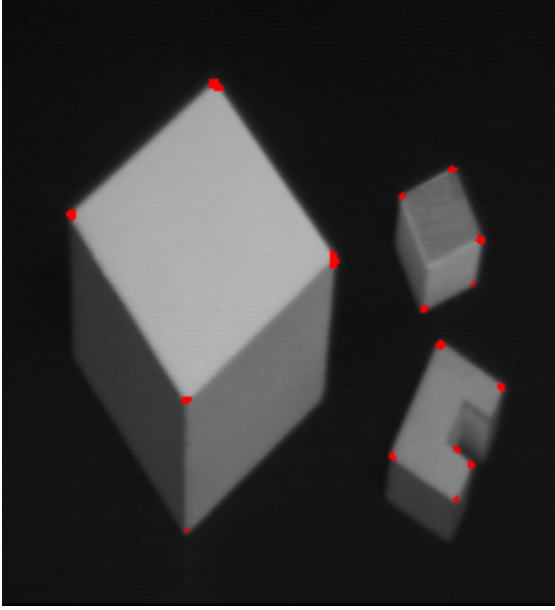


FIGURE 8 – Résultats de la méthode de Harris avec un maximum de points

### 1.5.2 Méthode basée sur les direction de gradient

Cette méthode consiste à chercher les valeurs maximales locales de :

$$k = I_x^2 < I_y^2 > + I_y^2 < I_x^2 > - 2I_x^2 I_y^2 < I_x I_y >.$$

Où  $\langle X \rangle$  représente la convolution de X par le masque suivant :

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

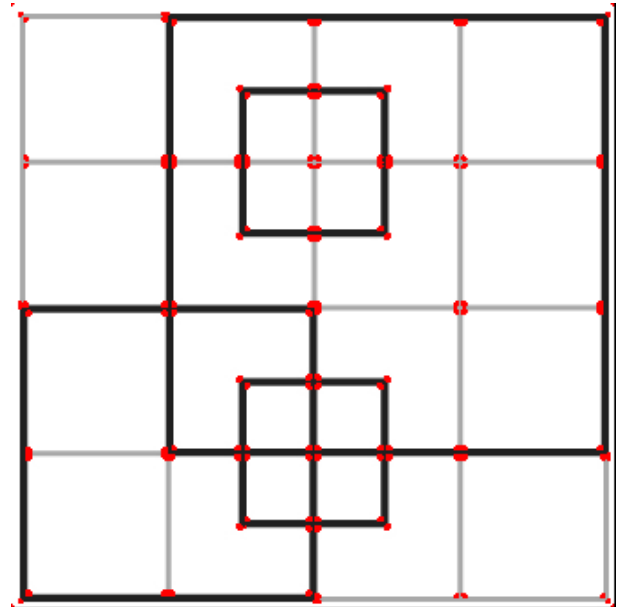
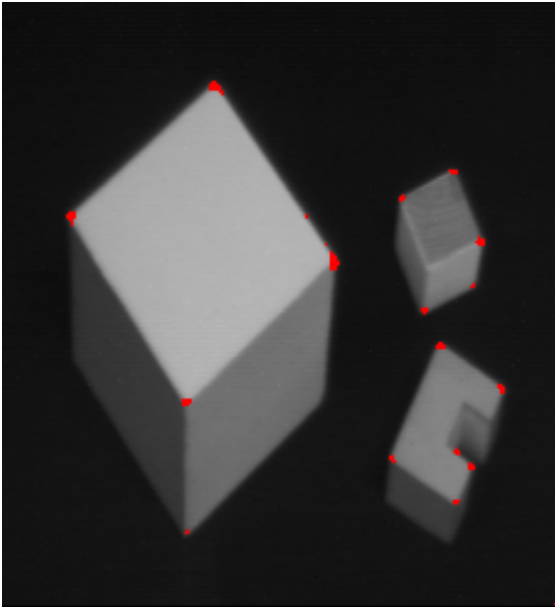


FIGURE 9 – Détection de points d'intérêt en utilisant la méthode basée sur les direction de gradient sans normalisation

Une fois qu'on a notre matrice des valeurs de  $k$ , on peut normaliser les valeurs par la norme du gradient en le divisant par :  $\langle I_x^2 \rangle + \langle I_y^2 \rangle$ .

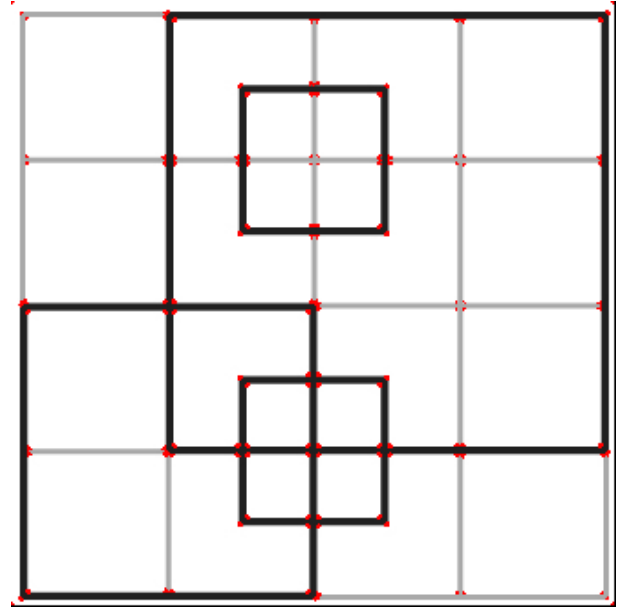
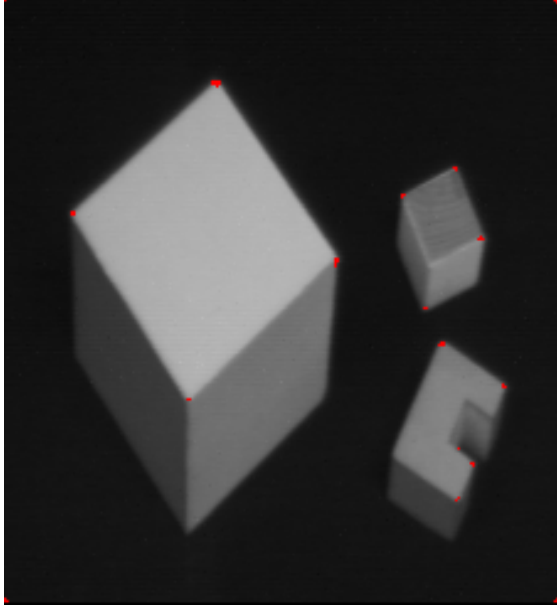


FIGURE 10 – Détection de points d'intérêt en utilisant la méthode basée sur les direction de gradient avec normalisation

On remarque avec la normalisation on peut éliminer quelques points faussé, mais aussi on peut en supprimer des points d'intérêt détecte sans normalisation.

```

1  int** conv2(int** f, long nrl, long nrh, long ncl, long nch,
2             float** mask, long maskw, long maskh)
3  {
4      int** out = imatrix(nrl, nrh, ncl, nch);
5
6      for (int x = nrl; x < nrh; x++) {
7          for (int y = ncl; y < nch; y++) {
8              double acc = 0.0;
9              int n = 0;
10
11             for (int u = 0; u < maskw; u++) {
12                 for (int v = 0; v < maskh; v++) {
13                     int nx = x + u - (maskw / 2);
14                     int ny = y + v - (maskh / 2);
15
16                     if(nx >= nrl && nx < nrh && ny >= ncl && ny < nch) {
17                         acc += f[nx][ny] * mask[u][v];
18                         n++;
19                     }
20                 }
21             }
22             out[x][y] = acc / n;
23         }
24     }
25
26     return out;
27 }

```

Listing 2 – Fonction qui calcule la convolution d'une image par un masque

## 1.6 Suivi d'objet avec sélection de vignettes autour des points d'intérêt

Afin de pouvoir suivre une région en mouvement il faut d'abord la caractériser, la technique est d'associer à un point d'intérêt une vignette de taille paramétrable. La vignette est un sort de matrice que l'on caractérise avec l'une des caractéristiques de l'image la moyenne des couleurs ou la texture (voir cours) ou l'histogramme etc.

Une fois qu'on calcule les points d'intérêt de la première image, on sélectionne les tops quatre des points d'intérêt et on les stock dans un tableau de vignette, puis un calcul est fait en fonction du barycentre de la région la plus proche vignette est considéré comme la vignette à suivre et le reste est ignorée.

Dans notre cas j'ai décidé de caractériser mes vignettes par l'histogramme des niveaux de gris ainsi que la taille de la région.

```
1 typedef struct {
2
3     int OX, OY; // cordonnees de la vignette
4
5     // utilise pour faciliter l'affichage du rectangle englobant
6     int min_x, max_x;
7     int min_y, max_y;
8 } Pose;
9
10 typedef struct {
11
12     int size; // taille de la vignette
13     Pose pose; // position de la vignette
14     int area_size; // taille de la region a suivre
15     int* H; // histogramme des niveaux de gris
16 } Vignette;
```

Listing 3 – Le structure de données représentant une vignette

Et ainsi a chaque nouvelle itération on regarde autour de la vignette et on détecte les régions autour d'elle, pour chaque région on calcule sa taille et son histogramme des niveaux de gris et on décide de la région la plus pertinente et finalement on fait déplacer la vignette vers ce point où on a détecté cette région.

Le déroulement de l'algorithme se fait comme suit :

1. Détection des régions en mouvement et amélioration du résultat
2. Étiquetage des régions détecté
3. Sur la premier image :
  - Détection des points d'intérêt
  - Initialisation des vignettes et choix de la vignette à suivre
4. Sur le reste de la séquence :
  - On parcourt le voisinage de la vignette en fonction de sa taille
  - On décide de la prochaine étiquette
  - On déplace la vignette vers sa prochaine position

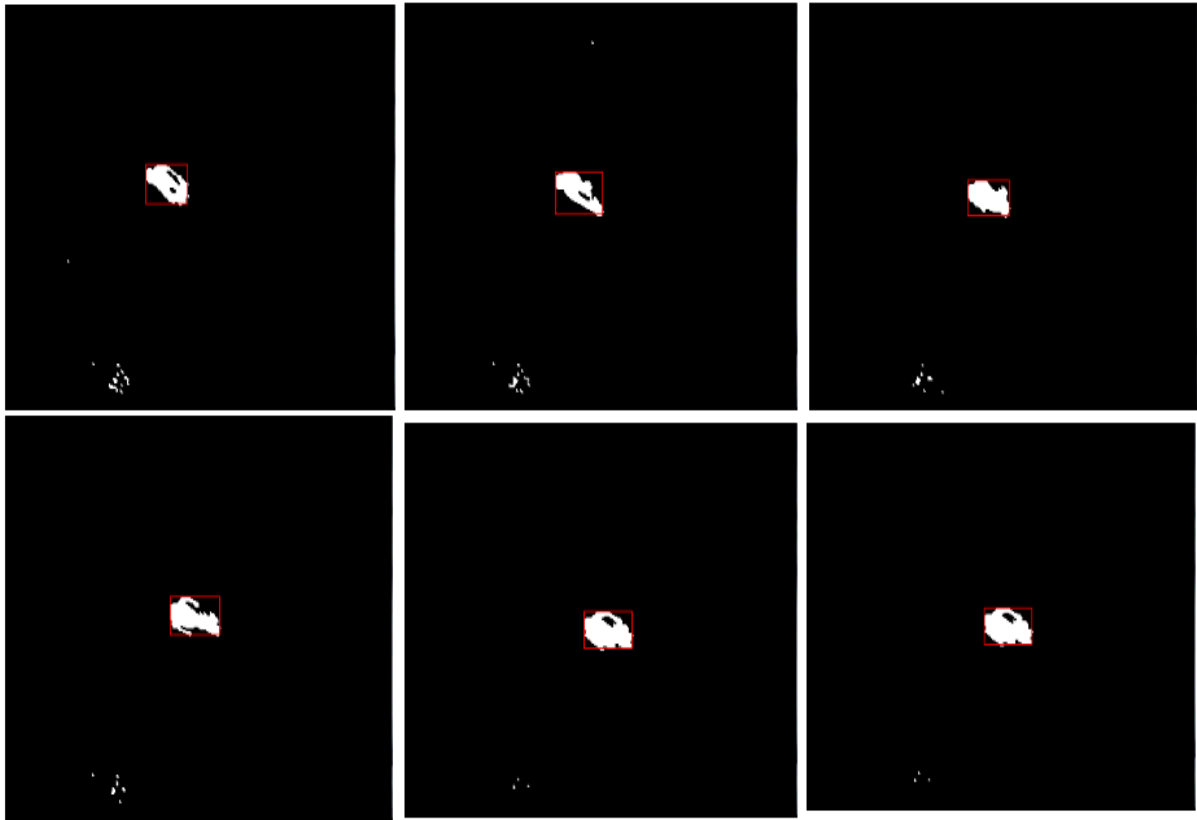


FIGURE 11 – Résultat du suivi d’objet avec sélection de vignettes autour des points d’intérêt

## 1.7 Suivi d’objet Multi-critères

Sur cette partie nous allons utiliser les caractéristiques des formes binaires déjà illustré sur la section 1.4, pour chaque image on les stock dans un tableau et ainsi pour chaque région on associe un ensemble de paramètres.

```

1 typedef struct
2 {
3     int area_label; // numero de la region
4     int size; // la taille
5     float OX, OY; // les coordonnees du centre de gravit
6     float E_x, E_y, E_xy; // les ecarts types
7     float theta; // la direction principale
8     float average_greyscal; // la moyenne des niveaux de gris
9     float avrage_R, avrage_G, avrage_B; // les moyennes sur les composantes R, G
10     et B
11     int min_x, max_x, min_y, max_y;
12     int* H; // histogramme des niveaux de gris
13 } Parameters;

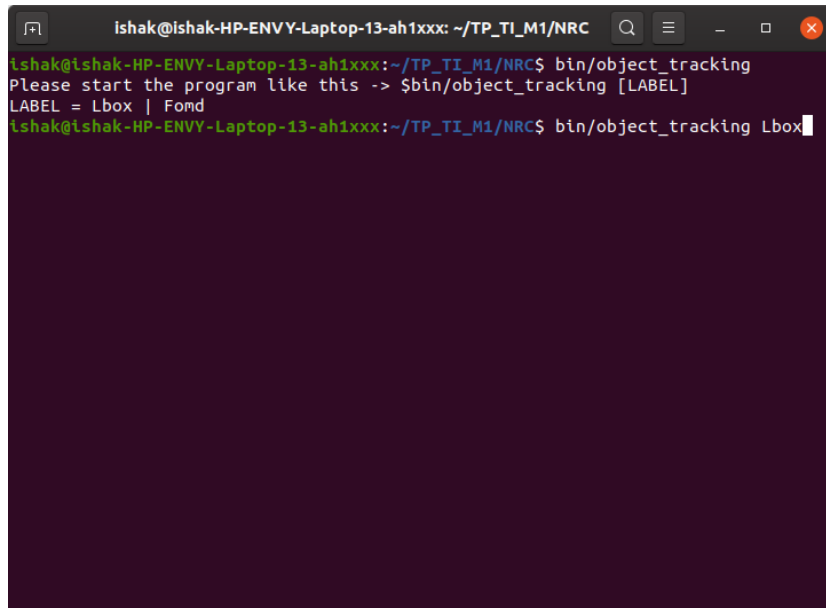
```

Listing 4 – Le structure de données représentant les paramètres d’une région

Ainsi a chaque itération du processus, on aura un tableau de paramètres et pour décider quelle région faut-il suivre il faut comparer le paramètre sélectionné précédemment avec le tableau et sélectionner celui le plus proche en comparant les histogrammes des niveaux de gris.

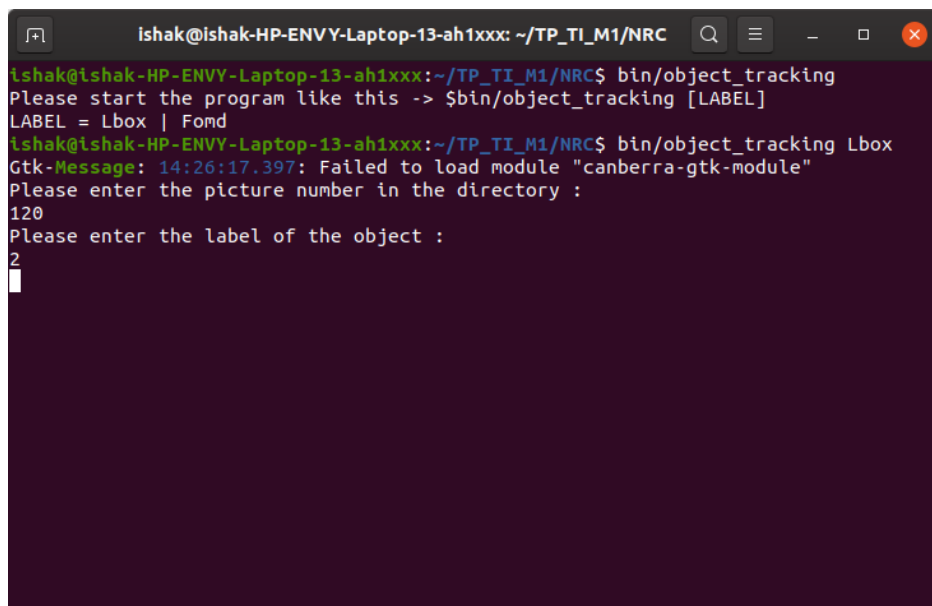
## Illustration de l'utilisation de l'application :

1. Au lancement l'utilisateur doit préciser quelle séquence il veut utiliser



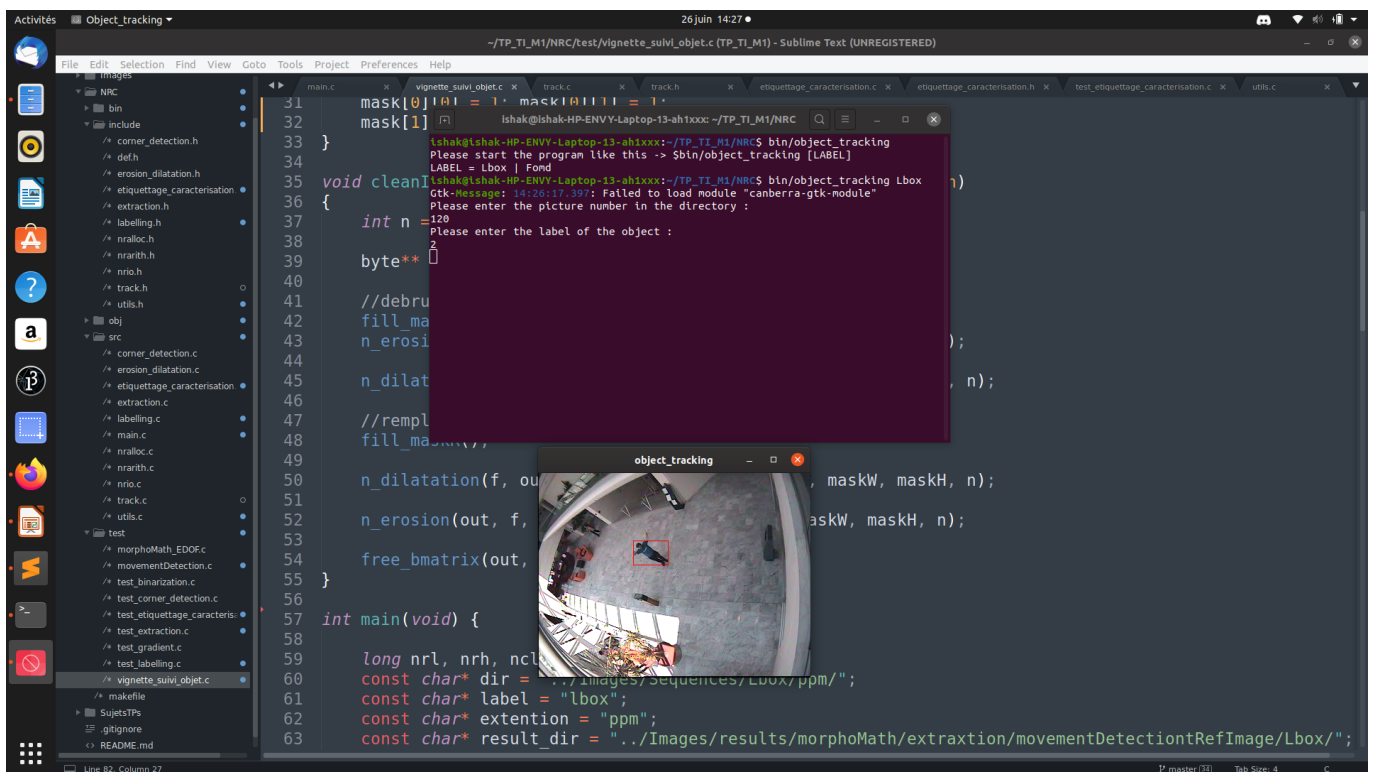
```
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx: ~/TP_TI_M1/NRC
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx:~/TP_TI_M1/NRC$ bin/object_tracking
Please start the program like this -> $bin/object_tracking [LABEL]
LABEL = Lbox | Fomd
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx:~/TP_TI_M1/NRC$ bin/object_tracking Lbox
```

2. Ensuite, l'utilisateur doit rentrer le numéro de l'étiquette à tracker ainsi que le numéro de la photo dans laquelle se trouve

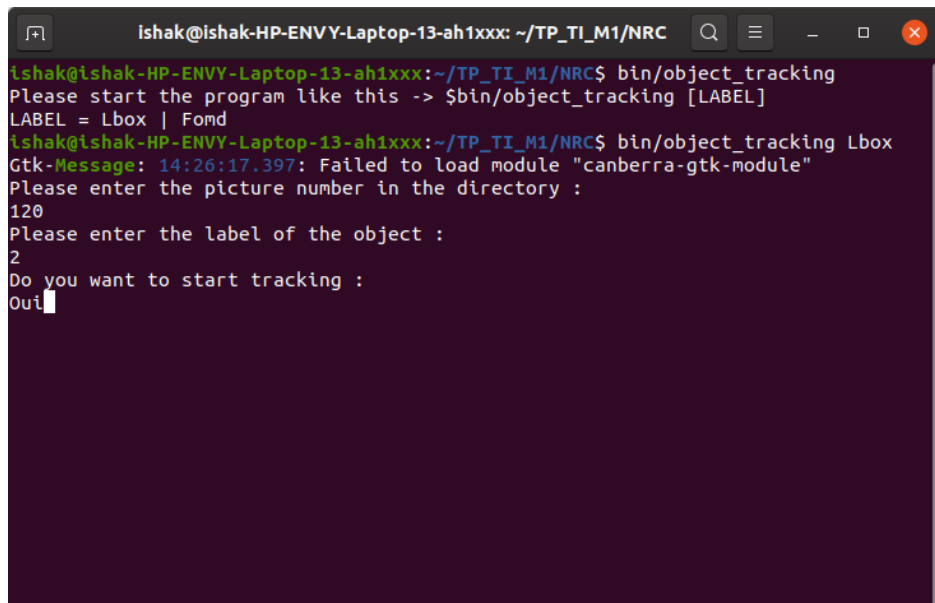


```
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx: ~/TP_TI_M1/NRC
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx:~/TP_TI_M1/NRC$ bin/object_tracking
Please start the program like this -> $bin/object_tracking [LABEL]
LABEL = Lbox | Fomd
ishak@ishak-HP-ENVY-Laptop-13-ah1xxx:~/TP_TI_M1/NRC$ bin/object_tracking Lbox
Gtk-Message: 14:26:17.397: Failed to load module "canberra-gtk-module"
Please enter the picture number in the directory :
120
Please enter the label of the object :
2
```

- On affiche sur cette image de la séquence un rectangle rouge englobant cette région et on demande à l'utilisateur de confirmer son choix ("oui", "non")

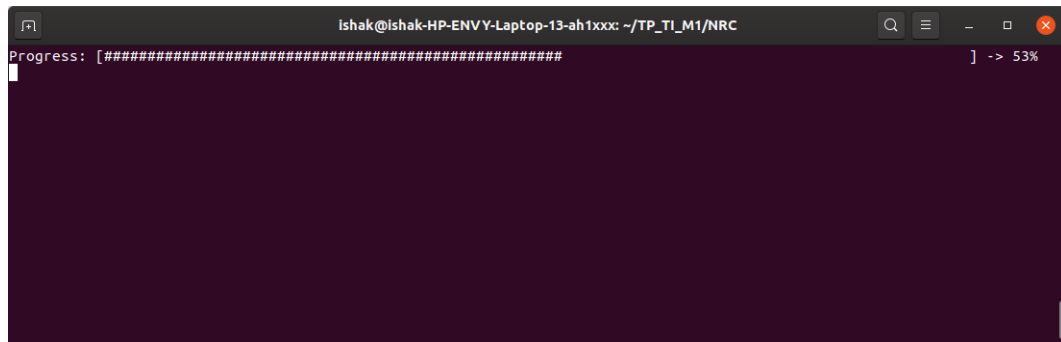


Pour cela j'ai utilisé la bibliothèque *gtk* qui offre un ensemble de fonctions permettant de réaliser des interfaces graphiques, ainsi une fois l'utilisateur quitte la fenêtre *gtk* le dialogue de confirmation est lancé





Et puis finalement le système est lancé et l'utilisateur peut observer un progressbar lui indiquant le pourcentage du nombre d'images traitées



Ici on réalise deux types de suivi, le premier en utilisant les points d'intérêt et le second avec les autres caractéristiques sauvegardé dans le tableau, ainsi pour chaque type de suivi on affiche un rectangle englobant en rouge pour le suivi par points d'intérêt et en vert pour le suivi avec les caractéristiques sauvegardées.

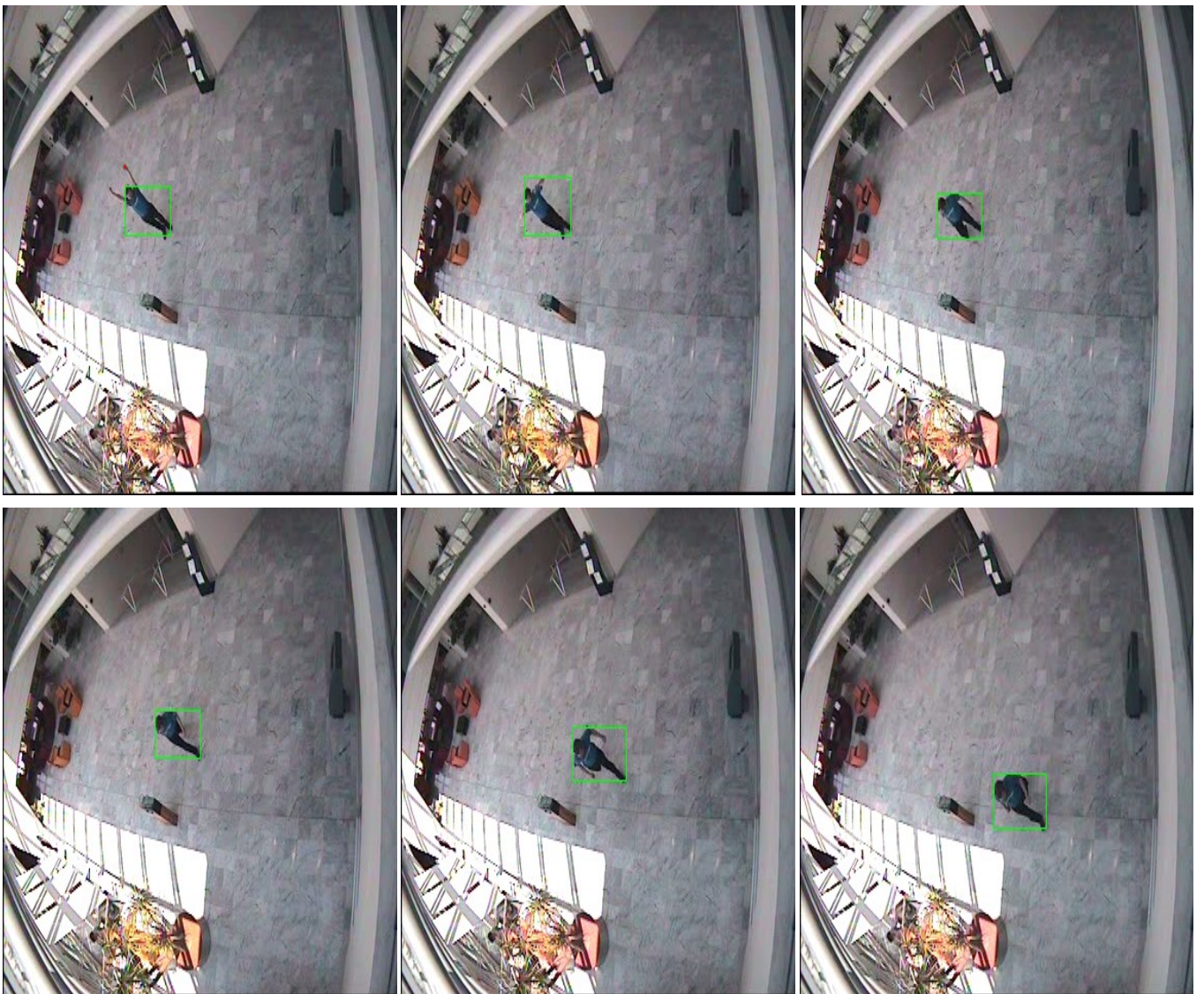


FIGURE 12 – Résultat du suivi d'objet Multi-critères, on peut remarquer sur la première photo que le taux de reconnaissance est plus élevé par le suivi avec les caractéristiques de l'image que celui avec les points d'intérêts

## 2 Annexe

```
1 void harris(int** harris_i, byte** I, long nrl, long nrh, long ncl, long nch)
2 {
3     // creation des masques mask1, mask2 et mask3
4     create_mask(1);
5
6     /* Calculate convolutions for the two masks */
7     int** ix = conv2(I, nrl, nrh, ncl, nch, mask1, 3, 3); // horizontal
8     int** iy = conv2(I, nrl, nrh, ncl, nch, mask2, 3, 3); // vertical
9
10    int** ixy = multiply(ix, iy, nrl, nrh, ncl, nch);
11
12    apply(ix, nrl, nrh, ncl, nch, pow2);
13    apply(iy, nrl, nrh, ncl, nch, pow2);
14
15    int local_x = 0;
16    int local_y = 0;
17    float local_max = -10000;
18
19    for (int x = nrl; x < nrh; x++) {
20        for (int y = ncl; y < nch; y++) {
21
22            int ixx = gaussian_filter_pixel(ix, nrl, nrh, ncl, nch, mask3, 3, 3, x, y)
23            ;;
24            int iyy = gaussian_filter_pixel(iy, nrl, nrh, ncl, nch, mask3, 3, 3, x, y)
25            ;;
26            int ixy_g = gaussian_filter_pixel(ixy, nrl, nrh, ncl, nch, mask3, 3, 3, x,
27                y);
28
29            ixy_g = ixy_g * ixy_g;
30
31            float res = (ixx * iyy) - ixy_g - LAMBDA * (ixx + iyy) * (ixx + iyy);
32            harris_i[x][y] = 0;
33
34            if(res < 0) {
35                if(local_max > 0) {
36                    harris_i[local_x][local_y] = 1;
37                    local_max = -10000;
38                }
39            } else {
40                if(res > local_max && res > 3500) {
41                    local_max = res;
42                    local_x = x;
43                    local_y = y;
44                }
45            }
46        }
47    }
48
49    free_imatrix(ix, nrl, nrh, ncl, nch);
50    free_imatrix(iy, nrl, nrh, ncl, nch);
51    free_imatrix(ixy, nrl, nrh, ncl, nch);
52 }
```

Listing 5 – Détection de points d'intérêt par la méthode de Harris

```

1 void erosion(byte** f, byte** out, long nrl, long nrh, long ncl, long nch,
2             int** mask, long maskw, long maskh)
3 {
4     int same = 0;
5     for (int i = 0; i < maskw; i++) {
6         for (int j = 0; j < maskh; j++) {
7             if(mask[i][j] == 1)
8                 same++;
9         }
10    }
11
12    for (int x = nrl; x < nrh; x++) {
13        for (int y = ncl; y < nch; y++) {
14            int acc = 0;
15
16            for (int u = 0; u < maskw; u++) {
17                for (int v = 0; v < maskh; v++) {
18                    int nx = x + u - ((int) (maskw / 2));
19                    int ny = y + v - ((int) (maskh / 2));
20
21                    if(nx >= nrl && nx < nrh && ny >= ncl && ny < nch) {
22                        acc += ((int) f[nx][ny] / 255) * mask[u][v];
23                    }
24                }
25            }
26            out[x][y] = (acc == same) ? (byte) 255 : (byte) 0;
27        }
28    }
29 }
30
31 void dilatation(byte** f, byte** out, long nrl, long nrh, long ncl, long nch,
32                int** mask, long maskw, long maskh)
33 {
34     for (int x = nrl; x < nrh; x++) {
35         for (int y = ncl; y < nch; y++) {
36             int acc = 0;
37
38             for (int u = 0; u < maskw; u++) {
39                 for (int v = 0; v < maskh; v++) {
40                     int nx = x + u - ((int) (maskw / 2));
41                     int ny = y + v - ((int) (maskh / 2));
42
43                     if(nx >= nrl && nx < nrh && ny >= ncl && ny < nch) {
44                         acc += ((int) f[nx][ny] / 255) * mask[u][v];
45                     }
46                 }
47             }
48             out[x][y] = (acc > 0) ? (byte) 255 : (byte) 0;
49         }
50     }
51 }
52
53

```

Listing 6 – Morphologie mathématique érosion et dilatation

```

1 byte** medianFilter(const char* dir_name, const char* label, const char*
   extension, long* nrl, long* nrh, long* ncl, long* nch, int n_seq)
2 {
3     int n = strlen(dir_name) + strlen(label) + 3 + strlen(extension);
4     const char file_name[n];
5
6     int som, median;
7
8     int*** votingBox;
9
10    for(int i = 1; i <= n_seq ; i++) {
11        sprintf(file_name, "%s%s%03d.%s", dir_name, label, i, extension);
12        rgb8** I1 = LoadPPM_rgb8matrix(file_name, nrl, nrh, ncl, nch);
13
14        if(i == 1)
15            votingBox = i3D(*nrh, *nch, 256);
16
17        for (int x = *nrl; x < *nrh; x++) {
18            for (int y = *ncl; y < *nch; y++) {
19                int value = ((int)I1[x][y].r + (int)I1[x][y].g + (int)I1[x][y].b) / 3;
20                votingBox[x][y][value]++;
21            }
22        }
23
24        free_rgb8matrix(I1, *nrl, *nrh, *ncl, *nch);
25    }
26
27    byte** bOut = bmatrix(*nrl, *nrh, *ncl, *nch);
28
29    median = n_seq / 2;
30
31    for (int x = *nrl; x < *nrh; x++) {
32        for (int y = *ncl; y < *nch; y++) {
33            som = 0;
34            for(int z = 0; z < 256 && !find; z++) {
35                som += votingBox[x][y][z];
36
37                if(som > median) {
38                    bOut[x][y] = (byte) z;
39                    break;
40                }
41            }
42        }
43    }
44
45    free_i3D(votingBox, *nrh, *nch);
46
47    return bOut;
48 }

```

Listing 7 – Filtre médian pour le calcul de l'image de référence

```

1 // E represents the matrix of labels or label image
2 void LOOKUP_TABLE LABELLING(byte** I, int** E, int* CURRENTLABEL, long nrl, long
   nrh, long ncl, long nch) {
3     int att_a, att_b, e_a, e_b, att_c;
4
5     ArrayList lookup = NEW_ArrayList();
6
7     for (int y = nrl; y < nrh; y++)

```

```

8      for (int x = ncl; x < nch; x++)
9          E[y][x] = 0;
10
11      *CURRENTLABEL = 0;
12      ADD_ArrayList(&lookup, *CURRENTLABEL);
13      /*First sweep*/
14      for (int y = nrl; y < nrh; y++) {
15          for (int x = ncl; x < nch; x++) {
16              /*recuperate the attribut and the label of A and B*/
17              att_a = ATT_A(I, nrl, nrh, ncl, nch, y, x);
18              att_b = ATT_B(I, nrl, nrh, ncl, nch, y, x);
19
20              e_a = E_A(E, nrl, nrh, ncl, nch, y, x);
21              e_b = E_B(E, nrl, nrh, ncl, nch, y, x);
22
23              /*only if the attribut of C is different then 0 we'll apply the
24               algorithm*/
25              att_c = (int) I[y][x];
26              if (att_c != 0) {
27                  if (att_c == att_a && att_c != att_b) {
28                      E[y][x] = e_a;
29                  } else if (att_c == att_b && att_c != att_a) {
30                      E[y][x] = e_b;
31                  } else if (att_c != att_b && att_c != att_a) {
32                      (*CURRENTLABEL)++;
33                      ADD_ArrayList(&lookup, *CURRENTLABEL);
34                      E[y][x] = *CURRENTLABEL;
35                  } else if (att_c == att_b && att_c == att_a && e_a == e_b) {
36                      E[y][x] = e_b;
37                  } else if (att_c == att_b && att_c == att_a && e_a != e_b) {
38                      /*first step*/
39                      E[y][x] = I_min(lookup.list[e_b], e_a);
40                      lookup.list[E[y][x]] = E[y][x];
41                      lookup.list[e_a] = E[y][x];
42                      lookup.list[I_max(lookup.list[e_b], e_a)] = E[y][x];
43                  }
44              }
45          }
46      }
47      /*Updating the correspondence table*/
48      *CURRENTLABEL = -1;
49      for (int i = 0; i < lookup.current_size; i++) {
50          if (lookup.list[i] == i) {
51              (*CURRENTLABEL)++;
52              lookup.list[i] = *CURRENTLABEL;
53          } else {
54              lookup.list[i] = lookup.list[lookup.list[i]];
55          }
56      }
57      /*Second sweep*/
58      for (int y = nrl; y < nrh; y++) {
59          for (int x = ncl; x < nch; x++) {
60              E[y][x] = lookup.list[E[y][x]];
61          }
62      }
63      FREE_ArrayList(&lookup);
64  }

```

Listing 8 – Algorithme d'étiquetage par table de correspondance

```

1 int track(int** E, rgb8** I1, Vignette* vignette, long nrl, long nrh, long ncl,
2         long nch)
3 {
4     int new_label = 0;
5     int h_err = 10000;
6     int app_size = 10000;
7
8     int new_x = vignette->pose.OX, new_y = vignette->pose.OY;
9
10    int curr_min_x = vignette->pose.min_x;
11    int curr_max_x = vignette->pose.max_x;
12    int curr_min_y = vignette->pose.min_y;
13    int curr_max_y = vignette->pose.max_y;
14    int curr_area_size = vignette->area_size;
15
16    int* H = ivector(0, 255);
17
18    for (int u = 0; u < vignette->size; u++) {
19        for (int v = 0; v < vignette->size; v++) {
20
21            // The thumbnail centred on the point of interest
22            int ny = vignette->pose.OY + u - ((int) (vignette->size / 2));
23            int nx = vignette->pose.OX + v - ((int) (vignette->size / 2));
24
25            if(ny >= nrl && ny < nrh && nx >= ncl && nx < nch) {
26
27                int label = E[ny][nx];
28                if(label != 0) {
29                    int next_min_x = INFINITY, next_max_x = -INFINITY,
30                    next_min_y = INFINITY, next_max_y = -INFINITY;
31                    int next_area_size = 0;
32
33                    compute_param(I1, E, label, H, &next_area_size,
34                                &next_min_x, &next_max_x, &next_min_y, &next_max_y,
35                                nrl, nrh, ncl, nch);
36
37                    int curr_h_err = 0;
38                    int curr_app_size = 100 * (abs(next_area_size - curr_area_size)
39                                              / (float)curr_area_size);
40
41                    for(int j = 0; j < 255; j++) {
42                        if(vignette->H[j] != 0 && H[j] != vignette->H[j]) {
43                            curr_h_err++;
44                        }
45                    }
46
47                    if(curr_h_err < h_err && curr_app_size < app_size) {
48                        app_size = curr_app_size;
49                        h_err = curr_h_err;
50                        new_label = label;
51
52                        new_y = ny;
53                        new_x = nx;
54
55                        curr_min_x = next_min_x;
56                        curr_max_x = next_max_x;
57                        curr_min_y = next_min_y;
58                        curr_max_y = next_max_y;
59                    }
60                }
61            }
62        }
63    }
64 }

```

```

60     }
61 }
62 }
63
64 free_ivector(H, 0, 255);
65
66 vignette->pose.OY = new_y;
67 vignette->pose.OX = new_x;
68
69 vignette->pose.min_x = curr_min_x;
70 vignette->pose.max_x = curr_max_x;
71 vignette->pose.min_y = curr_min_y;
72 vignette->pose.max_y = curr_max_y;
73
74 return new_label;
75 }

```

Listing 9 – Mise à jour de la vignette et envoi de la nouvelle région