Université de la Méditerrané
Faculté de Science de Luminy
Master 2 PRO SIS – Imagerie Numérique

2005/2006

Projet de Télédétection
Vidéo surveillance

Deovan Thipphavanh – Mokrani Abdeslam – Naoui Saïd

Table de matières

I. Introduction

Ce projet consiste à établir un système de vidéo surveillance afin de détecter des objets mouvants et analyser leur comportement. A partir d'une caméra de surveillance fixe, nous allons analyser les images prises et essayer de distinguer les comportements normaux des comportements anormaux des objets se trouvant dans les champs de la caméra.

Tout au long du projet, nous avons suivi l'optique d'essayer de généraliser nos algorithmes pour qu'ils puissent marcher dans plusieurs cas de figure et ne pas seulement se focaliser sur l'étude d'un endroit particulier ou la détection d'objets précis. Nous avons essayé de rendre notre application assez paramétrable pour fonctionner au mieux dans un environnement donné. Une fois paramétrée, la détection devrait s'adapter aux changements progressifs qui peuvent survenir, comme la diminution de la lumière par exemple.

Le résultat final consiste en une application qui présente les fonctionnalités suivantes :

- Pouvoir choisir une source d'images à utiliser (une vidéo par exemple)
- Pouvoir paramétrer la détection (avant ou pendant la détection)
- Lancer la détection
- Choisir le type d'affichage (objets détectés, leur trajectoire, etc.)
- Des outils divers qui peuvent servir pour diagnostiquer la détection (zoom, s'arrêter sur une image, récupérer la position et la couleur d'un pixel, calculer des distances, sauvegarder l'image affichée, etc.)
- Affichage de messages qui décrivent ce qui se passe, mais surtout, de lancer des alertes si un comportement anormal est détecté.

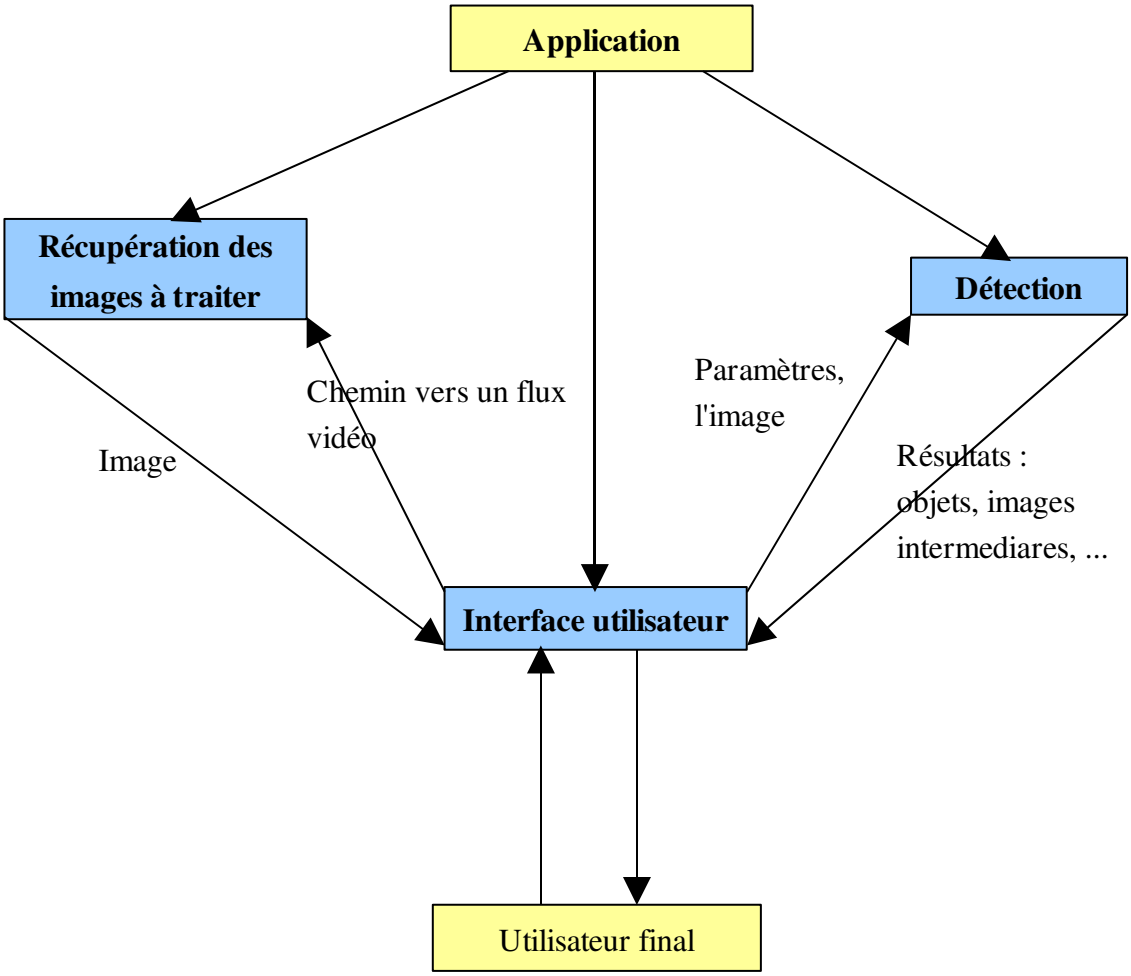
Nous avons complètement séparé la partie « interface utilisateur » de la partie, plus algorithmique, de détection. En effet, cette dernière partie a été conçue sous forme d'une bibliothèque réutilisable. Celle-ci dépend le moins possible de librairies externes ou les prend en compte seulement si elles existent. De même, son code, ne dépend aucunement du code de l'interface utilisateur. Dans la suite, nous allons nous concentrer plus sur la conception de cette bibliothèque.

II. Choix

Nous avons choisi de programmer les différents algorithmes en langage C. Ce langage est réputé pour sa puissance du fait de son rapprochement du langage machine. Pour la partie « interface utilisateur », nous avons utilisé la librairie graphique QT. Celle-ci nous permet d'intégrer notre programme avec une interface graphique facile d'utilisation. Cette librairie a l'avantage d'être disponible sous les principales plates-formes, mais aussi d'être gratuite pour des applications non commerciales. Faite avec le langage C++, notre source en C est donc accessible depuis le code de l'interface graphique.
Par manque de temps, nous avons décidé de nous concentrer sur l'analyse d'une personne.

III. Plan général de l'application

Notre application peut se subdiviser à plusieurs parties que nous allons voir plus en détail par la suite.



IV. Récupération des images à traiter

Afin d'avoir les meilleures performances possibles, nous avons donné une grande importance aux structures de données que nous utilisons. Ainsi nous avons défini au départ une structure pour représenter une image en mémoire comme indiqué ci-dessous :

Une image est un enregistrement qui contient au minimum la taille de l'image et un pointeur vers la mémoire allouée pour les données pixels (leurs couleurs). Un index vers cette mémoire de format tableau à deux dimensions permet un accès direct à la valeur d'un pixel connaissant ses coordonnées dans l'image. Nous prenons comme origine du repère image le premier pixel haut gauche et comme axe des abscisses la première ligne et d'ordonnées la première colonne. L'image est de profondeur 4 pour pouvoir représenter aussi bien les images en nivaux de gris que celles en couleur, mais aussi pour simplifier et accélérer son affichage à l'écran. Le quatrième octet est souvent utilisé pour mémoriser le niveau de gris du pixel afin de ne pas avoir à le recalculer plusieurs fois.

A partir d'une source de données (un fichier sur le disque par exemple), une image est lue en initialisant sa taille et allouant puis initialisant la mémoire qui contiendra les données des pixels.

Dans la pratique, la source d'images de notre application devrait être une vraie caméra connectée directement ou indirectement à l'ordinateur (par un réseau intermédiaire par exemple). Mais pour la conception de cette application, nous nous sommes basés sur les vidéos AVI fournies (disponibles sur Internet).

Au départ, nous avons commencé à lire les vidéos stockées sur le disque sous forme d'images PPM numérotées (après leur extraction des vidéos AVI avec un logiciel adapté). Une fonction permettant la lecture d'un fichier PPM est alors implémentée. Ces images ont l'avantage d'être simple à lire mais nous avons, de suite, constaté les inconvénients suivants :

- Les images PPM prennent un grand espace sur le disque (plusieurs Mo pour quelques secondes de flux vidéo). En effet les images PPM P6 (encore moins pour les images P3) ne sont pas compressées.
- La lecture des images PPM sur un disque dur est très lente vu la quantité de données à lire à chaque fois sur le disque.
- Pour simuler un traitement en temps réel, nous sommes obligés de lire la totalité des images, ce qui prend beaucoup de temps au départ (on atteint facilement une minute pour quelques secondes de flux vidéo). Mais l'avantage est la rapidité d'accès à chaque image puisqu'elles sont déjà dans la mémoire.
- Cet avantage peut nous constituer un piège, puisque cela nous laisse assez de temps pour traiter les images et ainsi donner moins d'importance à la rapidité de nos

algorithmes. En effet, notre application risque de fonctionner moins bien lorsque, par exemple, les images sont compressées (pour leur transfert par réseau par exemple) car cela consommera du temps supplémentaire pour la décompression.

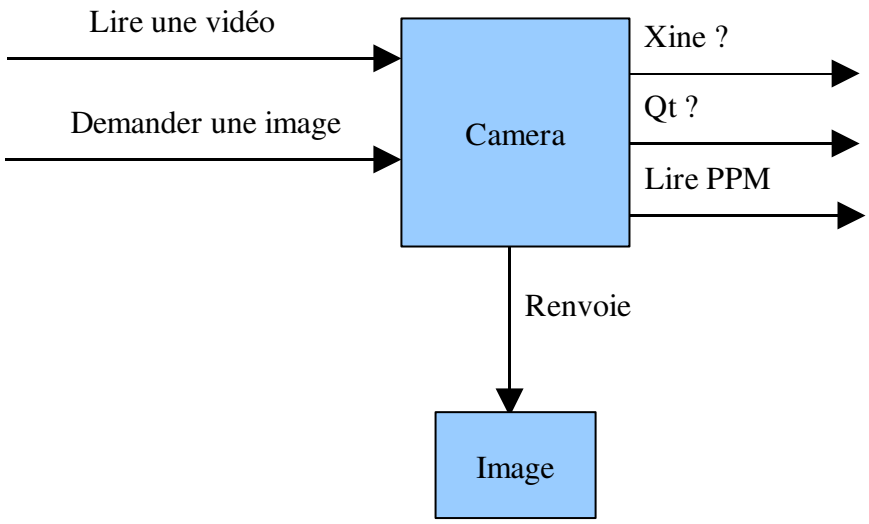
- Les images lues occupent un grand espace dans la mémoire vive (plusieurs Mo).
- Pour des vidéos assez grandes, leur traitement devient simplement impossible suite à la limitation de la mémoire.
- Un flux vidéo en temps réel ne peut pas être exploité de cette manière, puisque toutes les images sont lues dès le départ, les nouvelles images sont donc tout simplement ignorées.

Pour limiter la taille des images sur le disque et ainsi accélérer leur lecture, nous avons décidé d'utiliser des images compressées (JPEG par exemple). Nous avons donné la possibilité à la fonction de lecture d'une image d'utiliser une librairie externe (si installée) pour lire de telles images. Dans notre cas, nous avons intégré les possibilités de la librairie QT.

La plupart des inconvénients demeurent toujours, nous avons enfin décidé d'ajouter les possibilités de lire directement les vidéos compressées (beaucoup mieux que JPEG puisque on prend en compte, généralement, les transitions entre images) En effet, cela nous évitera de lire la totalité des images, mais seulement une image à la fois. Nous avons choisi d'utiliser la librairie Xine qui peut lire aussi bien des vidéos sur le disque que, directement, des flux vidéo (sur Internet par exemple). Notre programme ne dépend pas de cette libraire, elle est utilisée seulement si elle est installée sur la machine (à la compilation).

Nous sommes enfin arrivés à avoir un programme capable de récupérer en temps réel des images à partir de sources d'images variées.

Ce travail a été réalisé sous forme d'un sous module réutilisable appelé Camera :



L'utilisation de la caméra est assez simple. Son fonctionnement général peut se résumer en ces deux opérations :

- Indiquer la source d'images à utiliser (l'emplacement d'une vidéo par exemple)
- Au moment choisi, demander l'image suivante, la durée qu'il faut attendre pour demander la prochaine image est renseignée.

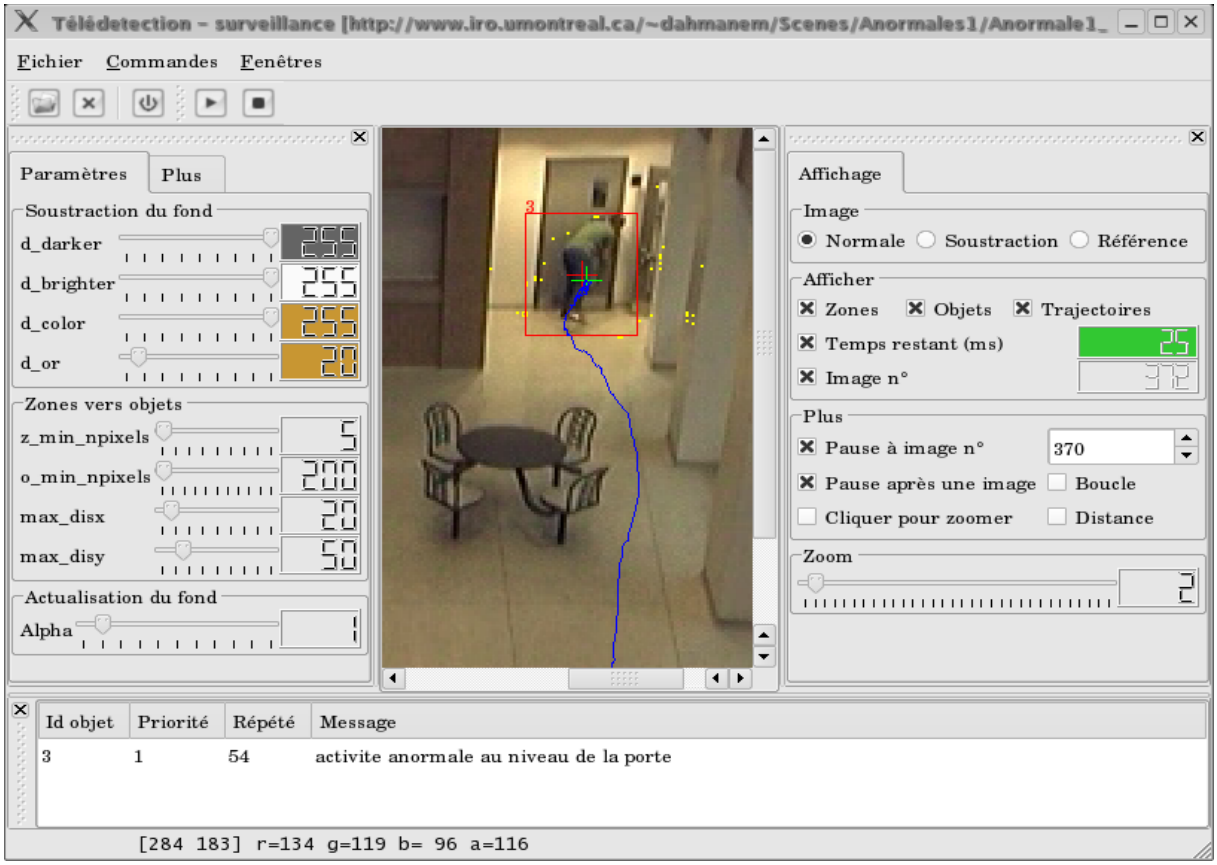
Remarque :
Dans le cas d'un ensemble d'images numérotées. Les noms des images doivent suivre la syntaxe : `<prefix>_<num><suffix>` avec :

- `<prefix>` et `<suffix>` des chaînes de caractères quelconques (les mêmes pour toutes les images) et `<suffix>` ne doit pas contenir '_'
- `<num>` le numéro de l'image qui doit être de même largeur pour toutes les images (préfixer par des 0 s'il le faut)

La lecture de telles images s'effectue en donnant le nom de la première image par laquelle doit commencer la vidéo en suivant l'ordre de numérotation. Le temps entre deux images peut être indiqué en le mettant (en unité de 1 / 90000 sec.) dans le fichier `<nom>.cfg` où `<nom>` est le nom du fichier de la première image.

V. Interface utilisateur

C'est la partie qui permet à un utilisateur d'interagir avec notre programme. Elle est basée principalement sur une interface graphique permettant de visualiser les différents résultats de la détection, mais aussi, d'autoriser l'utilisateur à modifier certains paramètres d'une manière contrôlée grâce aux composant graphiques. Nous avons réalisé une telle interface en utilisant la librairie QT dont voici un aperçu :



L'utilisation de cette interface peut se résumer aux points ci-dessous :

- Lancement de l'application : aucune source d'images n'est lue à moins que son adresse (non d'un fichier vidéo par exemple) est passée en ligne de commande.
- Ouvrir une source d'images : utiliser le bouton ouvrir et sélectionner un nom de fichier ou bien effectuer un « glisser déposer » de son adresse sur la fenêtre de l'application (du flux vidéo disponible sur Internet par exemple). La première image de la source est ensuite affichée.
- Modifier s'il le faut les différents paramètres de détection ou d'affichage.

- Lancer la détection en appuyant sur le bouton « démarrer », les images suivantes ainsi que les résultats de la détection sont affichés. Les différents paramètres peuvent être toujours modifiés.
- À tout moment, on peut arrêter la détection, fermer la source d'images, ouvrir une autre source d'images etc.

VI. Détection

C'est la partie la plus importante de notre programme mais c'est aussi la plus difficile à réaliser. Nous appelons « détection » tous les traitements effectués connaissant une image renvoyée par la caméra et éventuellement les objets détectés auparavant. Le résultat de ces traitements est récupérable sous forme d'objets détectés avec leurs caractéristiques ainsi qu'un ensemble de messages décrivant leurs comportements.

L'une des principales difficultés que nous avons eue, consiste en l'obtention de la région (ensemble de pixels) correspondant à un objet mouvant. Nous allons tracer un historique de tout ce qui a été essayé pour arriver à la version finale.

VI.1. Première approche

L'image de fond sans objets est relativement constante dans le temps. C'est ainsi qu'en procédant à une différence entre l'image de fond et l'image contenant les personnages, on pourra les détecter.
Notre première idée fut de faire une simple comparaison sur la différence de couleur entre les deux images : « l'image de référence » IR, image de la scène ne contenant aucun objet et l'image courante, IC, qui est lue à chaque instant.

Préalablement, on effectuait un traitement sur IC en appliquant un filtre Gaussien 3x3 afin d'éliminer les bruits parasites dus à la variation de la lumière, aux éventuelles poussières...
Après avoir effectué ce traitement sur IC, on compare la couleur de chacun de ses pixels à ceux de IR (à la même position). Si la couleur est différente et est supérieur à un certain seuil, cela signifie que le pixel appartient à un objet.

Deux majeurs problèmes se sont posés pour ce genre de traitement d'image :

- mauvaise détection de l'objet : détection de l'ombre du personnage ou mauvaise détection de l'objet (voire quasi inexistante) dans les zones sombres ou lorsque la différence de couleur entre l'objet et la scène est faible.
  Résultat : l'objet détecté n'est pas uniforme et se compose de plusieurs zones plus ou moins éparpillées, ou encore on n'obtenait qu'une partie de l'objet ;

- pas de prise en compte du facteur vidéo/temps. En effet, IR restait inchangé et donc on ne gardait aucune trace de la variation de IR dans le temps, puisque IR restait égale à la première image.
  Résultat : on obtenait une mauvaise segmentation, assez bruitée, lorsqu'il y avait un changement dans la scène au niveau de lumière, qu'il soit progressif ou brutal. Par exemple, le changement progressif de la luminosité peut ne pas être détecté par l'oeil

humain mais la machine est sensible à ce genre de changement aussi faible soit-il. Donc on se retrouve avec une image finale différente de la première.

Algorithme

Initialisation

- *image de référence IR = 1ère image,*
- *application du filtre Gaussien 3x3 sur l'image courante IR,*

Traitement des images

- *application du filtre Gaussien 3x3 sur l'image courante IC,*
- *extraction de l'objet par différence des couleurs des deux images selon un seuil de tolérance (le même quelque soit la position du pixel),*

Résultat

- *image binaire contenant l'objet uniquement,*

L'exemple ci-dessous illustre le problème d'éparpillement des zones après différenciation des images IR et IC.
L'interprétation de cette image pour une machine serait qu'il y a 3 objets dans la scène, bien qu'ils appartiennent tous au même.

VI.2. Seconde approche

Afin de remédier aux problèmes précédents, il a fallu trouver une solution pour obtenir une segmentation plus uniforme de l'objet et maintenir à jour l'image de référence IR.
Dans cette seconde approche, nous nous sommes basés sur la première partie de l'article *« Automated Visual Surveillance Using Hidden Markov Models »*, expliquant la méthode d'extraction de personnage dans une scène.

Il s'agit de toujours faire une différenciation entre IR et IC mais dans ce cas, il faut utiliser une matrice de seuil S et une matrice de référence IR qu'il faut mettre à jour à chaque itération.

Algorithme

Initialisation

- *IR = moyenne des 5 premières images sans objets ;*
- *S(x,y) = 50, pour tous les pixels (x, y) de l'image*

Traitement des images

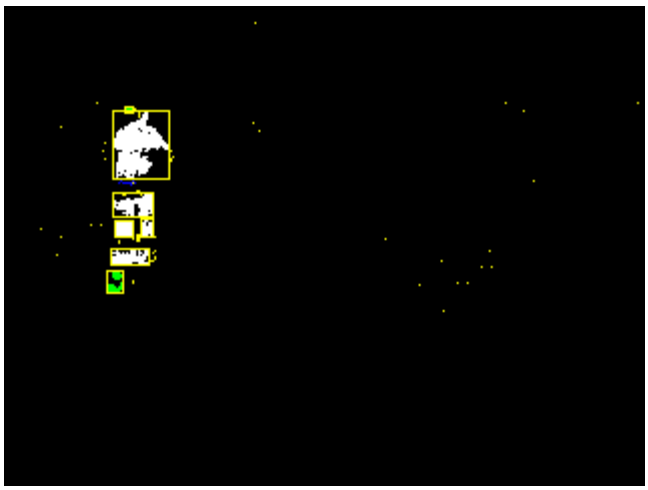
- *si* $| IC_n(x,y) - IR_n(x,y) | > S_n(x,y)$ *alors* $IC_n(x, y) = objet;$
- *si* $IC_n(x,y) = objet$ *:*
  - $IR_{n+1}(x,y) = IR_n(x,y);$
  - $T_{n+1}(x,y) = T_n(x,y);$

  Sinon :
  - $IR_{n+1}(x,y) = \alpha.IR_n(x,y) + (1- \alpha).IC_n(x,y);$
  - $T_{n+1}(x,y) = \alpha.T_n(x,y) + 2(1- \alpha).| IC_n(x,y) - IR_n(x,y) |$;

Résultats

- *Image binaire moins bruitée que la précédente et plus précise du l'objet à extraire.*

La valeur α [0..1] est la constante qui permet aux matrices de seuil S et de référence IR de se mettre à jour à chaque itération.
D'après plusieurs tests, on s'est rendu compte que la valeur $\alpha=1$ était la meilleure.



VI.2.1. Minimiser le nombre de zones

Une idée aurait été d'établir une matrice de seuil pour les zones particulièrement sombres afin de détecter un maximum de l'objet mais cela nous amenait à rajouter d'autres paramètres spécifiques à la scène et à faire l'étude d'un cas particulier.

VI.2.1.1. 1ère idée

Dans un premier temps, on a voulu détecté les contours de l'objet et en extraire sa forme. Ainsi, on a eu recourt à l'utilisation de la dilatation. Globalement, on obtenait une meilleure segmentation mais les résultats restaient assez insatisfaisants car il y avait toujours cette difficulté de détection de l'objet dans les zones sombres même après dilatation. La première idée fut d'augmenter le rayon d'opération de la dilatation mais cela prenait beaucoup trop de temps (exponentiel). Une deuxième idée fut d'appliquer la dilatation plusieurs fois avec un même rayon (le plus petit possible), ce qui est plus efficace qu'augmenter le rayon de la dilatation et plus rapide.
Résultats : meilleur définition du contour de l'objet, réduction du nombre de zones. Mais après de nombreux essais, la dilatation s'est avérée trop coûteuse en temps d'exécution et son utilisation inutile par la suite.

VI.2.1.2. 2ème idée

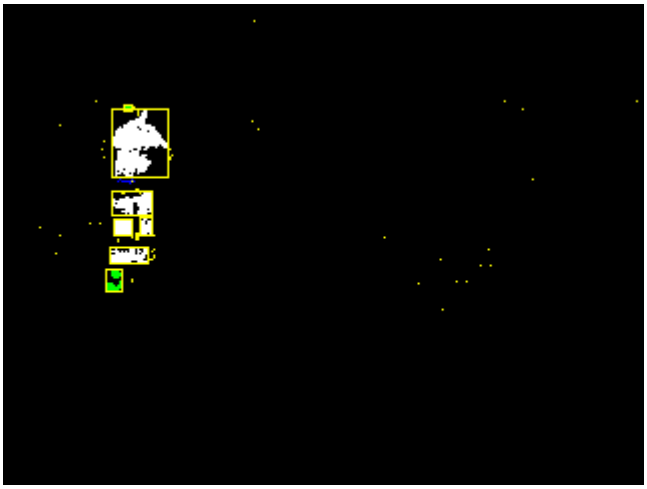
Notre deuxième idée fut de détecter l'objet lui-même et non plus les contours/forme des objets. Pour ce faire, on a décidé de regrouper toutes les régions voisines afin de ne former qu'un seul objet. On fusionne deux zones si elles répondent aux 3 critères suivants : SEUIL_X, SEUIL_Y et SEUIL_PIXELS :

- SEUIL_X et SEUIL_Y représentent respectivement les distances maximums autorisées par rapport à l'axe des x et l'axe des y entre les deux zones pour qu'elles soient considérées comme étant voisines sinon elles n'appartiennent pas au même objet;
- SEUIL_PIXELS représente le nombre de pixels (contenus dans une zone) minimum requis pour qu'une zone soit reconnue comme appartenant à un objet et non pas à du bruit ;

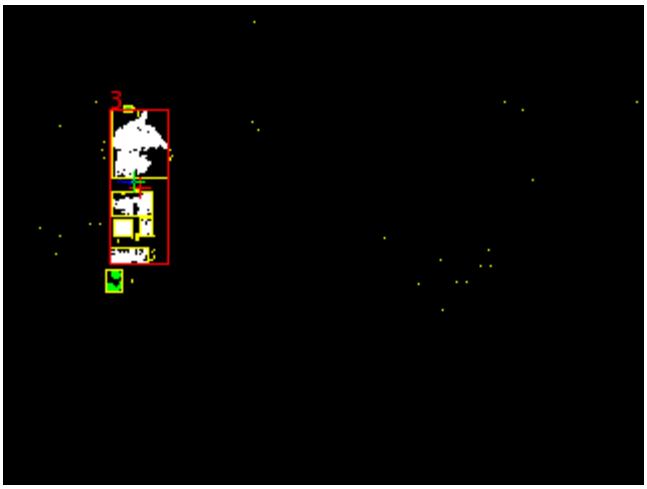
Résultats : nettement meilleurs et la détection de l'objet est quasi systématique. Les problèmes persistants sont la détection de l'ombre comme faisant partie intégrante de l'objet et possibilité de ne pas réunir deux zones à cause des variables seuils alors qu'elles appartiennent au même objet ou au contraire détecter une zone comme appartenant à l'objet alors qu'elle appartient à un autre objet. Les variables SEUIL_X, SEUIL_Y et SEUIL_PIXELS trop petites entraînerait la détection du bruit comme faisant partie de l'objet et trop grandes agrandirait le champs de détection et donc possibilité de trouver une partie d'un autre objet comme étant la partie d'un objet auquel il n'appartient pas.

Exemple :

Avant regroupement des zones



Après regroupement des zones :



VI.3. Version finale

Après différents essais et approches, dont certaines ont été vues précédemment, nous sommes arrivés aux conclusions suivantes :

- Les algorithmes liés à la détection doivent être les plus rapides possibles pour rester en temps réel. Il faut donc éviter au maximum les parcours et calculs coûteux. Nous avons ainsi écarté l'utilisation de filtres de type convolution et les Morphomaths.
- Bien que la détection peut fonctionner pour les vidéos AVI fournies en utilisant une image de fond statique initialisée avec la première image de la vidéo, nous devront penser au cas réel où des changements progressives dans de fond peuvent survenir en cours du temps. Nous devons alors absolument utiliser un système d'actualisation de l'image de fond.

Pour un souci de performance, nous avons opté pour un algorithme de type soustraction de fond. Une simple soustraction ne donne pas, en général, des régions correspondantes directement aux objets mouvants. En effet, un objet peut être subdivisé en plusieurs régions voisines. De plus, des régions peuvent ne pas appartenir à des objets, c'est le cas des bruits. Par la suite, nous allons appeler ces premières régions des zones.

Nous sommes ainsi amenés à réaliser un système d'attribution des différentes zones à des objets existants. La qualité des résultats dépend essentiellement de la soustraction de l'image de fond et, encore plus, de l'attribution des zones obtenus.

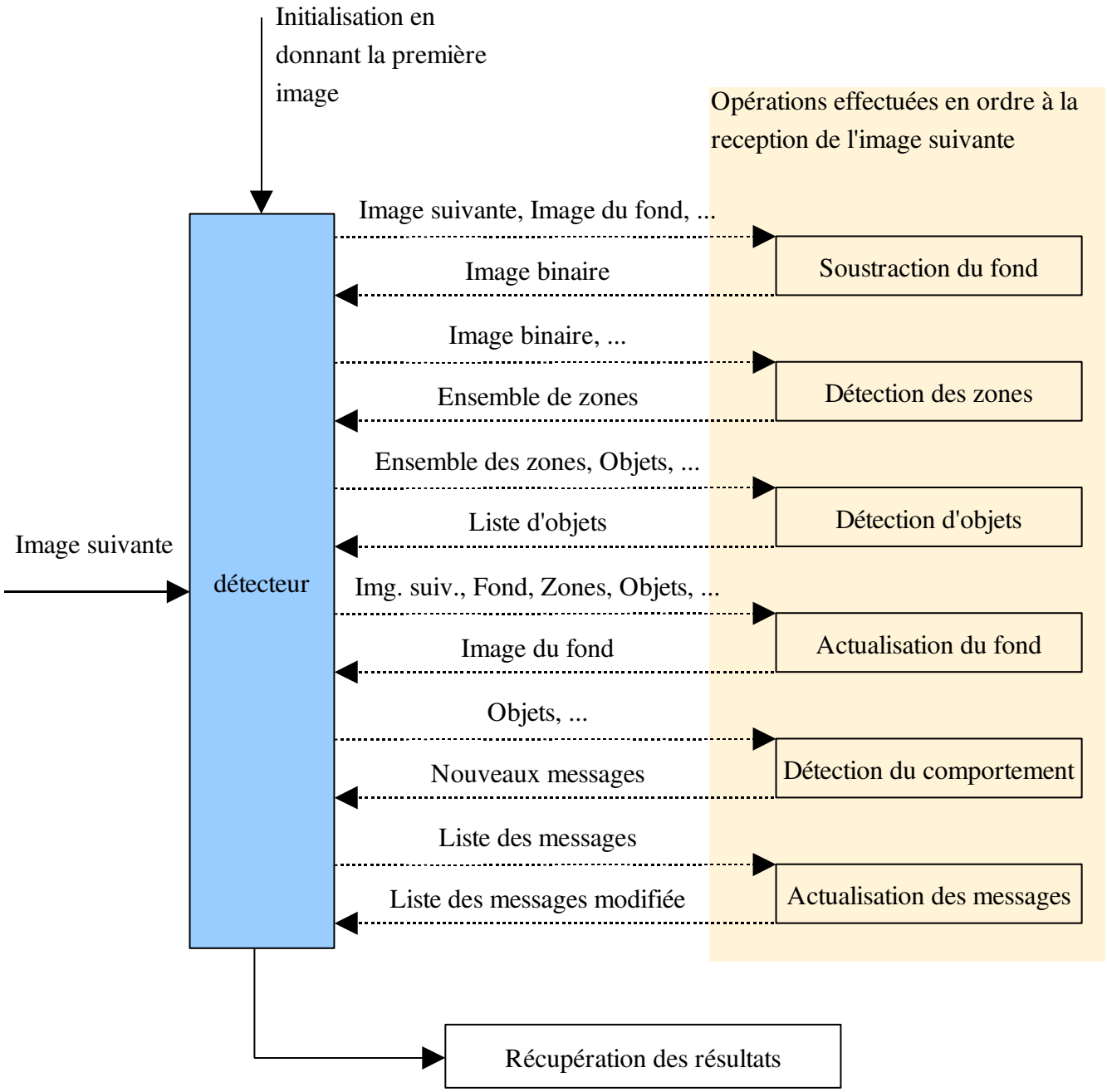
La figure ci-dessous montre les différentes phases de détection qui seront détaillées dans la suite de ce rapport.

VI.3.1. Les structures de données

Il est important de choisir les structures de données les plus performantes pour contenir les données du programme. Nous avons ainsi favorisé l'utilisation de tableaux pour certains cas et des listes chaînées pour d'autres. À cette occasion, une liste doublement chaînée générique avec accès directe aux premier et dernier éléments (plus performante qu'une simple liste chaînée) a été implémentée ainsi que les différentes fonctions qui la traitent.

Avant d'exposer les différentes étapes de la détection, nous allons énumérer les différentes structures et leurs principales contenances.

- Une image (voir partie récupération des images à traiter)



- Une zone contient :
    1. Les coordonnées x1, y1, x2, y2 du rectangle minimale qui la contient dans le repère image.
    2. Son nombre de pixels
    3. Son barycentre codé en réel
    4. Son histogramme (12 valeurs, 4 valeurs pour chaque composante couleur)
    5. Un drapeau indiquant si elle appartient à un objet

- Un objet contient
    1. Les mêmes données qu'une zone
    2. Un identifiant
    3. Un rapport largeur/hauteur avec largeur et hauteur correspondant au rectangle minimum qui contient l'objet.
    4. Un rapport largeur/hauteur estimé, correspondant à la position normale de l'objet.
    5. Un chemin qui est une liste de tous les anciens barycentres de l'objet, ordonnés chronologiquement.
    6. Le nombre de zones que l'objet contient.
- Un détecteur contient
    1. Une image de fond actualisée, une référence vers l'image actuelle et une image binaire (différence des deux premières)
    2. Un tableau de toutes les zones (nous pouvons avoir au maximum un nombre de zones égal au nombre de pixels d'une image divisé par deux)
    3. Une liste de tous les objets.
    4. Une liste de messages.
    5. Les paramètres utilisés pour la détection.

VI.3.2. Initialisation de la détection

L'initialisation de la détection s'effectue après indication d'une première image qui doit correspondre au fond. L'image du fond est alors initialisée en effectuant une simple copie. La connaissance de la taille de cette image nous permet d'allouer la mémoire qu'il faut pour les images et tableaux utilisés.

VI.3.3. Soustraction du fond

À l'arrivée de l'image suivante IA du flux vidéo, on effectue la soustraction de l'image du fond IF pour obtenir l'image binaire IB en utilisant l'algorithme suivant :

```
Pour tout pixel IA(i,j) de IA
  si IA(i, j) est assez différent de IR(i, j) alors
     IB(i, j) = 1
  sinon
     IB(i, j) = 0
fin pour
```

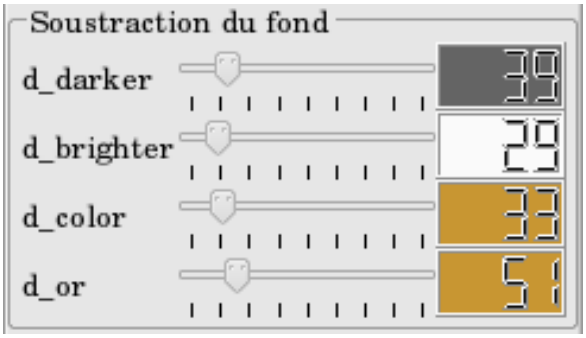
La différence entre deux pixels peut être calculée et paramétrée de plusieurs façons différentes. Les paramètres utilisés sont des seuils qui peuvent être modifiés par l'utilisateur. Supposant que (ra, va, ba, aa) et (rf, vf, bf, af) sont respectivement les composantes rouge, vert, bleu et niveau de gris (pré-calculé) d'un pixel PA de l'image actuelle et du même pixel PR dans l'image de fond.

Voici les différentes façons utilisées pour différencier ces deux pixels :

- PA est au moins plus lumineux que PR suivant un seuil aa – af > d_brighter
- PA est au moins plus sombre que PR suivant un seuil af – aa > d_darker
- (|ra – rf| + |va – vf| + |ba – bf|) / 3 > d_color
- |ra – rf| > d_or ou |va – vf| > d_or ou |ba – bf| > d_or

Il suffit que l'une des 4 conditions soit vraie, que les deux pixels soient considérés différents. On pourrait aussi ajouter d'autres paramètres et d'autres liens logiques entres les différentes conditions.

Voici une image binaire obtenue par soustraction et les paramètres utilisés :



Nous pouvons modifier ces paramètres à l'aide de l'interface graphique et voir en même temps ce que cela donne pour l'image binaire.

VI.3.4. Détection des zones

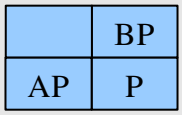
À partir d'une image binaire IB, nous pouvons retrouver l'ensemble des zones en procédant par étiquetage des pixels suivant un algorithme d'étiquetage qui parcourt l'image binaire seulement deux fois tout en générant les zones.

On possède, préalablement, un tableau E de taille égale au nombre de pixels d'une image divisé par deux. Ce tableau contiendra des étiquettes qui peuvent être des entiers. Un autre tableau EP à deux dimensions de taille égale à celle d'une image contiendra l'étiquette de chaque pixel de l'image.

Supposant que pour un pixel donné P : AP et BP sont respectivement le pixel à gauche et le pixel en haut de P dans l'image (sauf pour les pixels qui sont sur la première ligne ou la première colonne).

Algorithme :

```
NE=0 // Le nombre d'étiquettes

Traiter les pixels appartenant à la première ligne et ceux appartenant
à la première colonne comme ci-dessous mais sans prendre en
considération, respectivement, AP ou BP.

// Premier parcours

Pour chaque pixel P(i, j) de IB n'appartenant ni à la première
                          ligne ni à la première colonne parcourus
                          de gauche à droite et de haut en bas

  si IB[P] = 1 alors // pixel allumé
    si IB[AP] alors // pixel à gauche allumé
      si IB[BP] alors // pixel en haut allumé
        si EP[A] > E[EP[B]] alors
          EP[P] = E[EP[BP]]
          E[EP[A]] = E[EP[BP]]
        sinon
          EP[P] = EP[A]
          E[EP[B]] = EP[A]
        fin si
      sinon
        EP[P] = EP[A]
      fin si
    sinon
      si IB[BP] alors // BP allumé
        EP[P] = E[EP[B]]
      sinon
        NE = NE + 1 // nouvelle étiquette
        E[NE] = NE
        EP[P] = NE
      fin si
    fin si
  sinon // le pixel est éteint
    EP[P] = 0
  fin si
fin pour


// actualisation de la matrice des étiquettes (nombre d'itération
// égale au nombre d'étiquettes différentes attribuées
pour i de 1 à NE
  si i = E[i] alors // nouvelle zone
```



```
    initialiser une nouvelle zone et incrémenter le
    nombre de zones
  sinon
    E[i] = E[E[i]] // On choisit l'étiquette la plus petite
  fin si
fin pour


// Actualiser les étiquettes des pixels et générer les zones (second
// parcours)
pour tout pixel p de l'image
  si IB[p] = 0 alors aller à l'itération suivante
  EP[p] = E[EP[p]]
  actualiser la zone numéro EP[p] en ajoutant le pixel p
  (incrémenter le nombre de pixels, recalculer le barycentre,
   l'histogramme, les coordonnées etc.)
fin pour
```

Cet algorithme nous a permis en un minimum de parcours et de calculs, d'étiqueter chaque pixel de l'image par le numéro de la zone (ou son index dans le tableau de zones) qui le contient, ou par 0 s'il est éteint. Parallèlement, les différentes zones sont générées.

Voici deux images montrant les différentes zones détectées à partir d'une image binaire (dessinées sur l'image actuelle puis l'image binaire correspondante)

VI.3.5. Détection des objets

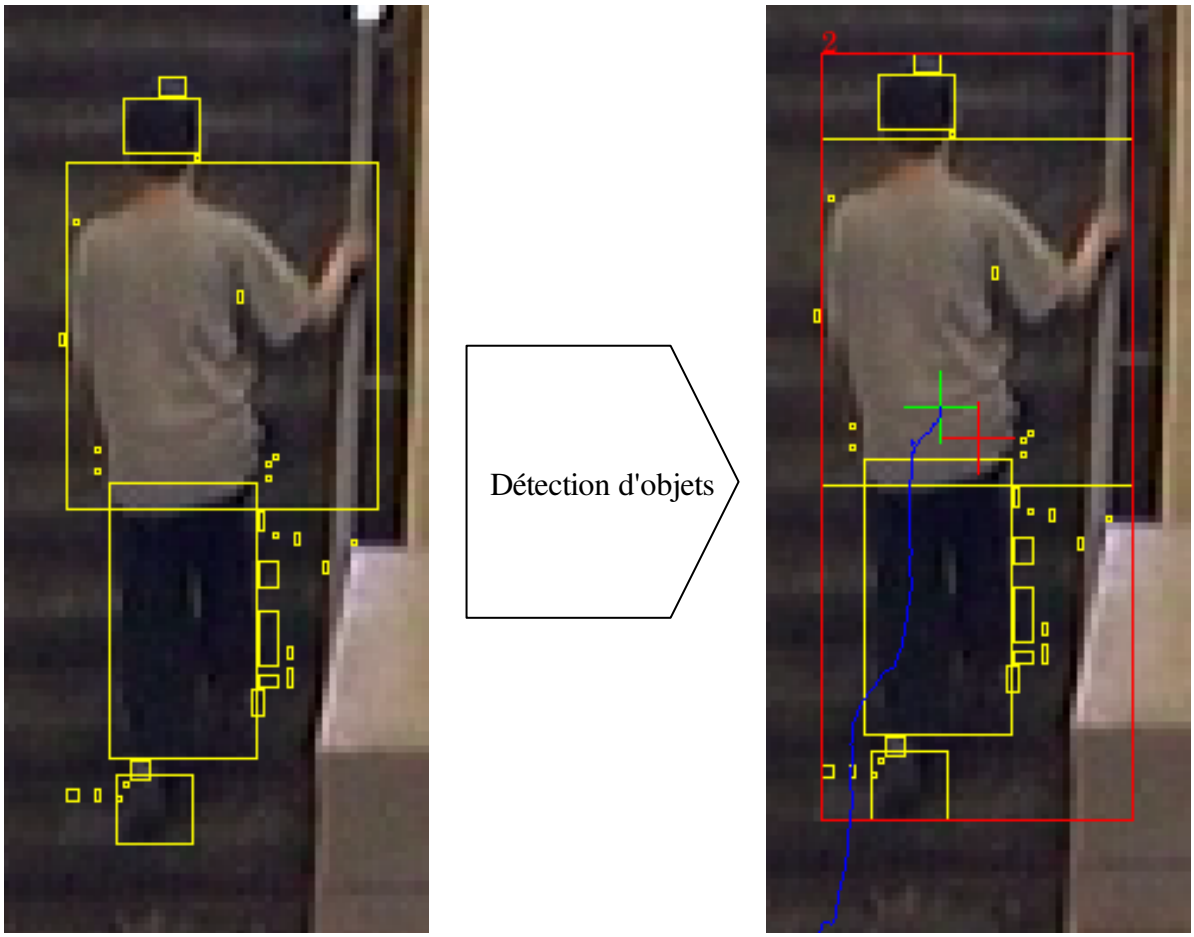
À partir de l'ensemble des zones calculée précédemment, nous pouvons actualiser les éventuels anciens objets ou en générer des nouveaux. De point de vu performance, nous n'avons plus besoins d'effectuer des parcours des pixels, mais des zones beaucoup moins nombreuses. Cela nous permet d'effectuer des calculs lourds et divers afin de mieux détecter les objets. De plus, les zones sont des indications très précises, car l'image binaire utilisée n'a pas été altérée auparavant (par des filtres, Morphomaths, ... par exemple)

Nous avons défini plusieurs stratégies d'attribution d'une zone à un objet, de l'ajouter à un nouveau objet ou de la considérer tout simplement comme étant un bruit. Ci-dessous l'algorithme utilisé.

```
Pour chaque objet o, initialiser un objet no avec le même id le même
chemin mais avec les autres champs (nombre de pixels par exemple) tous
à zéro
Pour toutes les zones z
  si z < z_min_npixels alors // nombre de pixels minimum
  aller à l'itération suivante;

  pour tous les objets no
    si o = ancien objet(no) existe (créé dans une autre image)
    alors
      max_disx = (largeur(o) + largeur(o) / r_max_dis) / 2
      max_disY = (hauteur(o) + hauteur(o) / r_max_dis) / 2
      // r_max_dis est un paramètre permettant de régler le
      // taux des déformations acceptables par rapport aux
      // dimensions actuelles de l'objet

      dx, dy = distance entre le barycentre de l'objet et celui
      de la zone + ddx, ddy avec ddx, ddy le déplacement
      effectué par l'objet en l'image précédente et l'image
      actuelle.
    sinon (l'objet vient d'être créé)
      max_disx = max_disx // paramètre
      max_disy = max_disy // paramètre
      dx, dy distance entre les deux barycentres de z et no
    fin si

    // Condition indispensable
    si dx <= max_disx et dy <= max_disy alors
      dis = distance entre les deux parycentres
```

```
      // no doit être le plus proche
      si dis > distance min alors aller à l'objet suivant

      z_no = z U no (union de la zone et de l'objet no)
      si o existe alors // ancien
      max_d_npixels = nombre pixels(o)/r_max_npixels // param
      max_d_histo = nombre pixels(o)/r_max_histo // param

      // Il faut s'assurer qu'il s'agit bien de cet objet
      si |npixels(z_no) – npixels(o)| > max_d_npixels alors
        aller à l'objet suivant
      si au moins histogram(z_no)[i] – histogram(o)[i]
          > max_d_histo alors

        // i détermine une valeur des 12 valeurs de
        // l'histogramme
        aller à l'objet suivant
      fin si
      // Ici, on peut, pour l'instant attribuer z à no
      distance min = dis
      meilleur_o = no
    fin si
  fin pour // pour tous les objets

  si meilleur_o calculé alors
    ajouter z à meilleur_o

  sinon si nombre pixels(z) > min et // min paramètre
        z ne se trouve pas au milieu
    on crée un nouveau objet no
    on ajoute z à no
    on ajouter no à la liste des nouveaux objets
  sinon
    z est un bruit
fin pour // toutes les zones
on supprime les objets no non mis à jours et ne répondant à certaines
conditions
on met à jours les autres champs des objets no (chemin, etc.)
on supprimer les anciens objets
```

Ci-dessous un exemple montrant comment des zones sont fusionnées pour actualiser un objet (personne) déjà existant.



Légende :

- les zones
- l'identifiant de l'objet, le rectangle minimal qui le contient et son centre.
- le barycentre de l'objet.
- la trajectoire de l'objet.

VI.3.6. Actualisation du fond

Cette opération est très importante pour que la détection s'adapte aux changements progressifs qui peuvent survenir sur l'image du fond. Le principe est d'actualiser chaque pixel, dans l'image du fond, n'appartenant pas à un objet. La valeur du pixel peut être calculée en utilisant sa valeur dans l'image du fond IF et celle dans l'image actuelle IA et un paramètre réel alpha entre 0 et 1 pour choisir le degré d'actualisation (0 : aucune actualisation, 1 le pixel est remplacé par sa valeur dans l'image actuel)

```
Pour chaque pixels p de IF
  si p n'appartient pas à un objet alors
     IF(p) = alpha * IA(p) + (1 – alpha) IF(p)
  fin si
fin pour
```

Deux problèmes ont été rencontrés lors de l'implémentation de cet algorithme.

- L'utilisation du réel alpha diminue les performances du programme
- IF(p) étant un entier, il est alors arrondi vers l'entier le plus petit si l'on considère directement la partie entière du résultat du calcule et non l'entier le plus proche. Par ailleurs, l'utilisation d'une fonction mathématique d'arrondi, ne fait que diminuer les performances encore.

Voici les solutions apportées :

- Il faut éviter de travailler avec des réels quand on est au niveau pixel. Nous avons transformé le paramètre alpha à un entier qui peut varier entre 0 et 100. Il représente le pourcentage d'actualisation.
- Nous ajoutons la valeur 0.5 avant de considérer la partie entière du résultat. Puisqu'il faut éviter les réels, la valeur ½ est ajoutée en l'intégrant dans le calcul.

Algorithme final :

```
Pour chaque pixels p de IF
  si p n'appartient pas à un objet alors
     IF(p) = ((alpha * IA(p) + (1 – alpha) IF(p)) * 2 + 100) / 200
  fin si
fin pour
```

VI.3.7. Détection des comportements

Cette partie est détaillée dans les sections « Études de la trajectoire » et « Détection des anomalies » ci-dessous. Nous considérons qu'après cette opération, les messages concernant les anomalies ou la description du comportement des objets sont ajoutés à l'ensemble des messages.

VI.3.8. Actualisation des messages

Un message est définit par :

- L'objet concerné.
- Sa priorité (suivant l'importance du message) allant de 0 à 255.

- Le nombre de fois que le message est répété.
- La durée qu'il doit rester affiché en nombre d'images consécutives.
- Une chaîne de caractères décrivant ce qui se passe.

Tous les messages sont stockés dans une liste chaînée. L'ajout d'un message m est soumis à la règle suivante :

Si le message existe déjà (même objet et même message), alors il est actualisé avec la nouvelle priorité et la nouvelle durée, et son nombre de répétition est incrémenté. Il est ensuite inséré à la première position en partant du début de la liste, tel que tous les messages se trouvant après cette position sont d'une priorité inférieure à celle du message en question. Ceci permet de trier les messages par leur ancienneté (les plus récents au début) puis par priorité (les plus prioritaires au début)

L'actualisation des messages consiste à supprimer les messages qui ont dépassé la durée qui leur est affectée et de décrémenter la durée des autres messages.

VI.3.9. Améliorations

L'algorithme de détection d'objets à partir de l'ensemble de zones pourrait être amélioré. En effet, si deux objets sont assez rapprochés, une zone peut appartenir en même temps aux deux objets. Une solution serait de regrouper momentanément les deux objets en un nouveau objet complexe qui contient la référence de chacun de ses sous objets. Les deux objets fusionnés ne doivent pas être supprimés mais doivent contenir une référence vers l'objet complexe, une manière de savoir qu'ils sont momentanément non détectable. À leur séparation, on pourrait les retrouver assez facilement, grâce à leurs informations, comme le nombre de pixels et l'histogramme.

VII. Étude de la trajectoire

VII.1. Phase préliminaire : initialisation de la scène

Une phase très importante consiste en l'initialisation de la scène par la décomposition du lieu que l'on désire surveiller. Il nous est ainsi nécessaire d'établir la liste des endroits « interdits » (rampe de l'escalier, fenêtre …) ainsi que les zones qui pourraient présenter d'éventuel agissement suspect. Dans notre cas, il s'agit de la porte.
D'après les vidéos dont nous disposons, nous avons pu isoler six zones qui présentaient des activités plus ou moins normales:

- l'escalier,
- la rampe de l'escalier,
- la porte,
- la table,
- la fenêtre,
- le sol,

Dans un cadre plus général et de façon plus automatique, il suffira juste de définir les zones à inspecter et les zones interdites ou qui ne devraient pas présenter d'activités.

VII.2. Données des objets

Lorsqu'une personne est détectée, on garde en mémoire un certain nombre de données nécessaires à l'analyse de ses agissements, allure et comportement.
Chaque personne possédera plusieurs caractéristiques qui leur sont propre dont les plus importantes sont les suivantes :

- **zone** : rectangle qui délimite la personne,
- **path** : liste des points (x, y) représentants la trajectoire de la personne,
- **barycentre** : centre de gravité de la personne qui nous permet de tracer la trajectoire qu'elle emprunte,
- **ratio** : rapport entre la largeur et la hauteur,

VII.2.1. Zone

La **zone** est le rectangle créé après application de l'algorithme de détection d'objets à partir des zones obtenues par différenciation. Chaque rectangle doit contenir une seule personne et la délimiter.

VII.2.2. Path

Path est l'ensemble des points qui forment la trajectoire que la personne emprunte. En reliant les points un à un, on obtient la courbe de sa trajectoire. Initialement, les points étaient représentés par le centre du rectangle « zone », mais on s'est aperçu que cette donnée n'était pas assez précise et nous rendait une courbe trop bruitée et pas assez lisse. Cette courbe est entièrement dépendante de la détection de l'objet, ainsi si l'on a une mauvaise segmentation alors on aura une mauvaise courbe et par la suite, une mauvaise analyse de la situation qui pourra entraîner de fausses alertes.

VII.2.3. Barycentre

Barycentre représente le centre de gravité de la personne. Deux barycentres consécutifs équivalent à un déplacement du personnage. Ces données nous sont utile dans le processus d'analyse de la vitesse ainsi que la détection des comportements anormaux.

VII.2.4. Ratio

Ratio est le coefficient qui nous permet de connaître l'aspect du candidat. Son importance vient du fait qu'il nous aide à détecter si la personne est tombée ou pas, dans quel cas, le ratio subira un changement brutal. En effet, on peut aisément détecter l'immobilité d'un personnage, et dans ces cas d'immobilité, il faut savoir faire la différence entre un problème ou une activité normale, comme lorsqu'un personnage s'assoie sur une chaise, ce dernier reste immobile mais ne présente pas d'agissement anormal. Par contre, un personnage qui s'est écroulé sur le sol ou autre endroit doit donner suite à une alerte.
On calcule le ratio est calculé de la façon suivante ratio = width / height.
Afin de détecter une éventuelle chute, on garde en mémoire un ratio courant, celui que possède le personnage à chaque itération et le ratio moyen qui varie en fonction du ratio courant à chaque itération aussi. En effet, une simple moyenne sur l'ensemble de la vidéo serait fausse, car lorsque le personnage s'éloigne (s'avance dans le couloir vers le haut de l'image) son ratio est plus petit que celui du début (en supposant qu'il vienne du bas).

VII.3. Analyse des données

L'analyse des anomalies s'est effectuée grâce à la mise en place de variables seuils, une pour chaque cas particulier comme la vitesse, l'immobilité, chute ou encore comportement anormale... qui déterminent du comportement et allure du personnage.
La difficulté dans ce genre de procédé est qu'il faut bien jauger ces variables afin de pouvoir distinguer le normal de l'anormal et ne pas faire de fausses détections. Les valeurs données aux variables SEUIL_ ont été définies grâce à l'analyse de vidéos en situations normales et anormales.

VII.3.1. La vitesse

La vitesse sera détectée par rapport à la distance que parcours le personnage entre chaque frame. Selon un certain seuil SEUIL_VITESSE = 2, on pourra savoir s'il est en train de courir, s'il a une allure soutenue ou s'il est en train de marcher. Ce seuil a été défini par rapport à la distance entre deux barycentres consécutifs (mesure en pixel). On pourra noter qu'en temps normal, une vitesse nulle serait signification d'immobilité.

VII.3.2. L'immobilité

L'immobilité est certainement la situation la plus facile à détecter mais l'un des plus contraignants pour faire la différence entre le normal et l'anormal. Un personnage est considéré comme étant immobile du moment où son centre de gravité ne bouge plus. On peut affirmer son immobilité du moment où il vérifie le seuil de stationnarité SEUIL_STATIONNAIRE = 50, c'est-à-dire si son centre de gravité n'a pas bougé depuis 50 images.

VII.3.3. La chute

Le ratio du personnage commence à être calculé et mis à jour du moment où l'on détecte un personnage en entier. S'il est en train de rentrer dans la scène, on ne tient pas compte de son ratio. Pour le calculer, on se sert de la largeur et hauteur de la zone qui délimite le personnage. A chaque instant, on le compare avec le ratio moyen et dès que la différence entre ces deux ratios dépasse un seuil SEUIL_RATIO = 0,5 alors on détecte une chute.

VII.3.4. Le comportement anormal (non implémenté)

Une idée serait d'analyser la courbe du personnage afin de détecter des mouvements un peu désordonnés ou répétitifs comme faire les 100 pas, des allers-retours, des trajectoires en rond...

Dans un premier temps, il faudrait savoir si la personne se situe toujours dans un même espace, si oui, combien de changement de direction a-t-elle effectué? En temps normal, une personne ne devrait pas changer de direction ou avoir au maximum un seul changement de direction, si elle fait un aller-retour. Si on détecte un grand nombre de changement de direction en un petit laps de temps, cela pourrait se définir comme une bagarre entre deux personnes ou une tentative de forçage de porte ou une attitude assez étrange. S'il y a plusieurs changement de direction mais étaler sur le temps, cela marquerait les différentes situations de va-et-vient.

VII.4. Problèmes rencontrés

A cause des bruits que comporte les vidéos et d'une segmentation non parfaite, on est obligé d'accorder un certain seuil de tolérance quant aux variables SEUIL_ afin de ne pas prendre en compte les aléas des vidéos. Voici quelques exemples de cas mitigés:

La détection de l'ombre du personnage n'est pas systématique mais reste existante spécialement lorsque la luminosité est grande et que l'ombre est bien marquée. De ce fait, lorsque l'on détecte l'ombre comme faisant partie du personnage, nous obtenons un ratio totalement faussé et du moment où l'on ne détecte plus cette ombre, le ratio subit une modification brutale et on détectera une chute, qui n'existe pas.

Lorsque le personnage est tombé au sol et ne bouge plus, on constate que sur certaines vidéos notamment celles où il se trouve sur l'escalier, le centre de gravité a tendance à avoir un léger déplacement qui ne devrait pas exister puisque la personne est immobile. Afin de régler ce problème il a fallu tolérer le déplacement du centre de gravité dans un rayon de 1 pixel, au delà, le personnage a bougé.

Un problème relevant encore de la segmentation est celui où le personnage se trouve entièrement dans le champ de vision de la caméra mais son ombre étant détectée comme appartenant à la personne. Non seulement on ne détecte pas la personne comme étant entièrement dans la scène mais en plus on aura des données totalement faussées, primordiales pour l'analyse du comportement.

VIII. Détection des anomalies

Après étude des vidéos, on peut distinguer et regrouper toutes les différentes activités anormales dans trois catégories:

- **catégorie 1 : immobilité**, lorsque la personne est stationnaire et ne bouge plus,
- **catégorie 2 : vitesse**, lorsqu'elle a une allure anormale ou quand elle court,
- **catégorie 3 : comportement anormale**, tout ce qui n'est pas une trajectoire normale, mouvement non rectiligne...,

VIII.1. Listing des vidéos

VIII.1.1. Situations normales 1 personne

- **Monter et descendre l'escalier:**
  Dans ces différentes vidéos, le personnage se contente d'aller et venir sur l'escalier en passant près de la rampe, au milieu des escaliers, ou à côté du mur.
  Une des difficultés rencontrées dans cette partie a été la détection de la vitesse du personnage. En effet, pour analyser la vitesse d'un personnage, on se base sur la distance qu'il parcourt. Or, le cas de l'escalier est particulier dans le sens où le personnage avance assez rapidement du fait des marches.

- **Table**
  Le personnage va et vient en s'arrêtant au niveau de la table. Les différentes vidéos le montre à tour de rôle sur les quatre chaises.
  Plusieurs difficultés se sont présentées ici. Le personnage étant assis, il est donc immobile. Il reste à savoir s'il y a anomalie ou pas. On aurait pu seulement vérifier son ratio et sa position dans la scène, en pensant que si son ratio était normal et que s'il se situait au niveau de la table, tout allait bien. Or, nous avons des vidéos présentant le personnage se couchant sur la table. Son ratio reste normal, sa position est au niveau de la table et il est immobile.

- **Marche**
  Le personnage se contente de traverser le couloir soit en venant du bas et se dirigeant vers le haut, soit au contraire en passant par la porte d'abord et se dirige vers le bas.
  Un des problèmes rencontrés a été lorsqu'il rejoignait la porte en venant du bas. En effet, une des façons de détecter s'il essaie de passer en force par la porte est d'analyser le temps qu'il reste au niveau de la porte. Or lorsque le personnage passe au niveau de la porte normalement, il s'avère qu'il prend presque autant de temps que lorsqu'il essaie de passer en force.

VIII.1.2. Situations anormales 1 personne

- 1 : passage en force de la porte, la porte est fermée et le personnage insiste pour l'emprunter, Catégorie 1
- 2 : course, le personnage est en train de courir, Catégorie 2
- 3 : course, Catégorie 2
- 4 : allonger sur la table, le personnage allonge une partie de son corps sur la table et reste immobile pendant un certain temps, Catégorie 1
- 5 : chute dans les escaliers, le personnage tombe et reste immobile, Catégorie 1
- 6 : zone interdite, le personnage s'assoie sur la fenêtre et reste immobile, Catégorie 1
- 7 : chute derrière la table, Catégorie 1
- 8 : aller-retour, le personnage fait des allers-retours dans le couloir, Catégorie 3
- 9 : marche bizarre, le personnage longe le mur, Catégorie 3
- 10 : zone interdite, le personnage s'assoie sur le bord de la rampe des escaliers et reste immobile, Catégorie 1
- 11 : zone interdite au niveau de la fenêtre, Catégorie 1
- 12 : aller-retour, Catégorie 3
- 13 : passage en force de la porte, Catégorie 1
- 14 : aller-retour; Catégorie 3
- 15 : allonger sur la table; Catégorie 1
- 16 : chute sur les escaliers, Catégorie 1
- 17 : chute devant la table, Catégorie 1

VIII.2. Catégorie 1 : Immobilité

A plusieurs reprises, le personnage se retrouve immobile pendant un certain laps de temps et ce pour des raisons assez diverses:

- il va s'asseoir dans une zone interdite et y rester immobile comme la rampe de l'escalier ou la fenêtre,
- il va s'allonger sur la table,
- il va faire une chute au niveau de l'escalier et du sol,
- il va rester au niveau de la porte en essayant de passer en force,

VIII.2.1. Zones interdites

Il est assez facile de détecter si un personnage se trouve dans une zone illicite. Il suffit simplement d'analyser sa position et s'il se trouve dans une de ces zones illicites, alors on déclenche une alerte. Le fait qu'il reste immobile est secondaire dans ces cas là.

VIII.2.2. Table

Lorsque le personnage va s'allonger au niveau de la table, il est plus difficile de dire ce qu'il est en train de faire, étant donné qu'il est immobile et se trouve au niveau de la table, donc on pourrait penser qu'il est assis. Dans notre algorithme d'analyse de comportement, on prend en compte les changements éventuels au niveau de la table elle-même, si le personnage se trouve vers la table et qu'il est resté immobile pendant un certain temps. On définit une zone qui correspond au haut de la table et on compte le nombre de pixels qui ont changé. S'il dépasse un certain seuil alors on considérera que le personnage est allongé. Pour ce faire, on regarde l'image binaire, et on considère que les pixels ont changé s'ils appartiennent à un objet.
En faisant une analyse sur le personnage et sur les changements de la table, on arrive à détecter s'il est allongé ou non.

VIII.2.3. Chutes

Lorsque le personnage effectue une chute, s'il y a un problème, c'est-à-dire s'il s'est évanoui ou autre, il va rester immobile. Dans un premier temps, on étudie le ratio de la personne, s'il subit un changement brutal alors il y a détection de chute. De plus, s'il reste immobile et que son ratio reste inchangé alors on déclenche une alarme.

VIII.2.4. Passage en force de la porte

La porte est fermée mais la personne essaie tant bien que mal de passer par là.
Une façon de procéder pour détecter ce genre de situation est de regarder le temps qu'il passe au niveau de la porte. D'une façon similaire que celle faite pour la table, on regarde si la porte

a subi des changements. Si le personnage reste trop longtemps au niveau de la porte sans qu'il y ait modification de la porte, alors on déclenche une alarme.

VIII.3. Catégorie 2 : Vitesse

On détecte qu'une personne a une allure anormale s'il se déplace plus vite en peu de temps. D'après analyse des vidéos, on pourra dire en moyenne que pour une allure normale, le personnage se déplace d'un pixel entre chaque frame. Au delà de deux pixels, on considère qu'il est en train de courir. Entre ces deux valeurs, il aura une allure soutenue.
La difficulté ici a été de différencier le mouvement au niveau des escaliers du reste de la scène. En montant les marches, le personnage avance plus rapidement dans la scène que s'il était ailleurs dans le couloir. Avec nos paramètres actuels, on détectera un personnage en course alors que ce n'est pas le cas. Il faut donc changer le seuil pour les activités au niveau des escaliers et augmenter le seuil de vitesse.

VIII.4. Catégorie 3 : Comportement anormal (non implémenté)

Par manque de temps, cette partie n'a pas été implémentée. Voici la liste des différentes anomalies constatées sur les vidéos :

- allers-retours,
- longer les murs,
- bagarre,
- danse,

VIII.4.1. Allers-retours

La détection des allers-retours peut se faire grâce à l'indication du nombre de changement de direction et de sens. Pour comprendre un changement de direction, on cherche déjà à savoir la direction qu'il emprunte. Si le personnage se déplace du bas vers le haut alors son déplacement sera négatif, s'il se déplace du haut vers le bas son déplacement sera positif. Il suffit de faire la différence des barycentres des Y. On procède de façon similaire pour savoir s'il va vers la droite ou vers la gauche en faisant la différence par rapport à l'axe des X. Un déplacement vers la droite sera négatif et un déplacement vers la gauche positif. On peut connaître le sens et la direction que le personnage possède à chaque instant.
Pour détecter les allers-retours, il suffirait donc de compter le nombre de changements, par rapport au temps écoulé.

VIII.4.2. Union des deux personnages

Lorsque les personnages sont en contact, on ne distinguera qu'un seul objet au lieu de deux. Il est plus dur de comprendre ce que fait chacun des personnages indépendamment. D'après les

vidéos, il nous faudra différencier des cas comme une bagarre ou des situations telle que lorsque les deux personnes se tiennent par les bras, se mettent à tourner ensemble ou s'entrechoquent.
On pourra remarquer que lors d'une bagarre les mouvements du barycentre sont plus désordonnés que lorsque les personnages tournent ensemble, le barycentre de la zone de détection reste plus ou moins stable.

VIII.4.3. Autres

Lorsque le personnage longe les murs, sa trajectoire est rectiligne mais il a une vitesse très variable. En enregistrant les différentes vitesses qu'il a eues tout au long de son chemin, on pourra détecter son comportement bizarre.

VIII.5. Algorithme

Dans cet algorithme, on parcourt l'ensemble du chemin qu'à emprunter les personnes. Du fait que les vidéos sont de très petites tailles et qu'il n'y a pas beaucoup de personnes à détecter, on peut se permettre de faire cette étude exhaustive. Si l'on veut généraliser cet algorithme, au lieu de parcourir le chemin entier à chaque itération, on pourra mettre un compteur à partir de 300 images avant l'image en cours.

Pour chaque objet détecté

Analyse préliminaire sur l'ensemble de sa trajectoire

- *Analyse de l'immobilité*
- *Analyse de la vitesse*
- *Analyse des changements de direction et sens*
- *Analyse du ratio*

Analyse de la position en cours ainsi que des 10 dernières frames

- *Analyse de l'immobilité*
  - *Détection de l'immobilité:*
    On regroupe les informations des analyses préliminaires et de celles en cours de l'immobilité. Si ça répond au SEUIL_STATIONNAIRE alors le personnage est immobile.
    - *Analyse de la position du personnage*
      - *Table: analyse des changements de la table*
      - *Porte : analyse des changements de la porte*
      - *Zones interdites*
    - *Analyse du ratio par rapport au SEUIL_RATIO pour détecter une éventuelle chute*

- *Analyse de la vitesse*
  - *Détection de la vitesse*
    On regroupe les informations sur les vitesses, de façon similaire à celle de l'immobilité.
    Si l'on remplie le critère de la variable SEUIL_VITESSE
    - *Analyse de la position du personnage*
      Si escalier : comparaison de la vitesse avec le SEUIL_VITESSE défini pour les escaliers
- *Analyse des changements de direction et sens en procédant de façon similaires aux autres*

N.B.: Les alertes sont déclenchées pendant les analyses du comportement des personnages.

IX. Conclusion

IX.1. Détection des objets

Nous avons vu que la détection d'un objet est une opération relativement complexe à cause de plusieurs contraintes. Outre les différents bruits, les changements de l'image des fonds liés au passage d'un objet (ombres, autres objets comme une porte qui s'ouvre, ...) et la fusion de deux objets était la principale difficulté rencontrée.

Suite à la limitation du temps pour le traitement d'une image, nous avons choisi des méthodes performantes qui nous laissaient assez de temps pour détecter efficacement les objets.

IX.2. Analyse du comportement

Les analyses ont été faite par rapport à des vidéos déjà existantes dont on a pu extraire les anomalies. Le problème est que leur énumération n'est certainement pas exhaustive. On a réussi à généraliser plus ou moins certaines situations qui se ressemblaient ou se répétaient. Mais il y a de nombreux cas qui n'ont pas été étudiés et ne feront pas l'objet d'une analyse s'ils se produisent. Par exemple, une personne qui marcherait sur ses mains avec une allure normale et un ratio normal, ou encore une personne qui marcherait à reculons..., ne fera pas l'objet d'une détection anormale.

Les valeurs des paramètres ont été plus ou moins déterminées par rapport à de nombreux tests jusqu'à en trouver de satisfaisantes. La principale difficulté a résidé dans le fait que la segmentation de l'objet n'est pas toujours parfaite et de ce fait il a fallu ajuster les paramètres en fonction pour qu'il n'y ait pas fausses détections. En fait, il est assez facile de détecter les anomalies mais le problème est de savoir si l'on détectera des événements inexistants sur des situations normales.

X. Déroulement du projet

X.1. Répartition des tâches

- Lecture des vidéos et création de l'interface graphique : Abdeslam

- Extraction de l'objet
  - 1ère approche : Abdeslam, Saïd, Thipphavanh
  - 2ème approche : Abdeslam (détection des différentes zones), Thipphavanh (détection de l'objet à partir de zones données)
  - Version finale : Abdeslam

- Étude de la trajectoire : Abdeslam, Thipphavanh

- Analyse des anomalies : Thipphavanh

- Rédaction du rapport : Abdeslam et Thipphavanh

X.2. Outils utilisés

- Le développement du projet est effectué sous Linux
- GNU bash, version 3.00.16(1)-release (i386-redhat-linux-gnu)
- gcc (GCC) 4.0.1 20050727 (Red Hat 4.0.1-5)
- GNU Make 3.80
- Qt: 3.3.4
- Xine 1.1.0
- Doxygen version 1.4.4
- Editeur utilisé : jEdit 4.2final
- OpenOffice.org 2.0 pour la rédaction du rapport
- GIMP version 2.2.8 pour les captures d'écran

XI. Références

[1] Vinod Nair and James J. Clark, « Automated Visual Surveillance Using Markov Models »

[2] Jonathan Owensa, Andrew Hunterb & Eric Fletchera, « A Fast Model-Free Morphology-Based Object Tracking Algorithm »