

# Détection et suivi de mouvement

NGUYEN Quoc Khai

28 Novembre, 2013

## 1 Introduction

Ce TP est divisé par 3 grandes étapes : prendre l'image de fond, détecter des mouvements et suivre des mouvements. Pour deux premières étapes, j'ai implémenté des méthodes qu'on a vu en classe. Pour la dernière étape, j'ai utilisé le filtre de Kalman implémenté dans OpenCV. Dans ce rapport, je parle d'abord la théorie que j'ai utilisé. Je présente ensuite la programme. Enfin, je vais exprimer les résultats obtenus. Je terminerai avec une petite conclusion de mon travail.

## 2 Description du travail

### 2.1 Prendre l'image du fond

Pour détecter des mouvements, je construis d'abord une seule image du fond à partir de la vidéo. J'ai décidé de choisir une seule image du fond car chaque vidéo dans la base des images a la même image du fond et pour réduire le temps de reconstruire l'image du fond chaque l'image de séquence. Dans cette étape, j'ai converti des images de séquence en image gris, alors, l'image du fond obtenu est en gris. J'ai pris le médian chaque pixel des images pour un pixel de l'image du fond. C'est pour ça que l'image du fond peut être contient des objets de mouvement si cet objet se situe à une position dans plupart de temps de la vidéo.

Pour prendre l'image du fond d'une vidéo, on tape : `./detectvideo fond video_path fond_path`

Exemple : `./detectvideo fond video/Walk1.mpg image/test/fond_walk1.png`



Figure 1: Exemple d'image du fond

En fait, on peut ne pas utiliser toutes les images de séquence pour construire l'image du fond mais ce n'est pas faux si l'on choisi tout.

## 2.2 Détection du mouvement

La deuxième étape de ce TP est l'étape de détecter des mouvements, j'ai soustrait les images de la vidéo avec l'image de fond pour obtenir les zones en mouvement. C'est un travail simple. Cependant, le résultat obtenu contient beaucoup de bruits. Afin de réduire le bruit, j'ai mis un seuil de 50. Si le résultat inférieur ce seuil, on considère comme 0. Pour améliorer le résultat, j'ai fait aussi la convolution sur les images d'entrée (image du fond et images de la vidéo). Après la détection, j'ai utilisé l'érosion et la dilatation avec l'image du kernel de  $3x3$ . On trouve que le seuil est très utile pour réduire le bruit. Cependant, il risque de perdre des objets



Figure 2: Détection non seuil



Figure 3: Détection avec le seuil non érosion ni dilatation



Figure 4: Détection avec seuil, érosion et dilatation

de mouvement quand la couleur de l'objet près de la couleur de fond (différent inférieur à seuil). Pour que les objets de résultat soient plus complètes, on peut utiliser la dilatation plusieurs fois après l'érosion. Mais cela fait grossir l'objet et la forme de l'objet sera un peu différent.

## 2.3 Suivi du mouvement

Dans la dernière partie, on a détecté les zones de mouvement. Cette étape permet de suivre des zones détectées. Pour faire cela, j'ai fait comme le consigne du TP. J'ai calculé la boîte englobante minimale en utilisant la fonctionnalité de chercher des contours de l'OpenCV sur l'image de résultat dans la deuxième étape. Le programme dessine la boîte autour du contour de chaque objet. L'objet de suivi est le centre de chaque objet. Pour suivre les objets détectés, le filtre Kalman de OpenCV est utilisé. Les matrices utilisées :

1. Matrice de transformation

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

2. Matrice de mesure

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

On a trois couleurs pour chaque objet :

1. Bleu signifie la position de la prédiction
2. Vert représente la position de la mesure
3. Rouge représente la position de la correction

Dans l'implémentation, j'ai utilisé plusieurs filtres *Kalman* différents dans chaque image de séquence. Après avoir prédict, mesuré et corrigé, le programme enregistre les filtres de chaque objet dans un vecteur. L'image de séquence suivante utilise les filtres corrigés actuels. Grâce à la position des objets, l'application peut choisir des filtres corrects pour des objets suivis. Exemple :

t1 : Objet1 : Kalman1, Objet2 : Kalman2, Objet3 : Kalman3

Grâce à la position

t2 : Objet1 : Kalman2, Objet2 : Kalman2, Objet3 : Kalman1

Ici, je considère que les distances plus courtes sont plus correctes. Je trouve dans les vidéos de base, la vitesse

des objets n'est pas beaucoup différente, ce programme prédit sont base seulement de la position. Les objets changent la taille pendant le mouvement, c'est la raison pour laquelle je ne compte pas la taille des objets dans les prédictions.

Je confirme que dans l'implémentation du programme, je ne veux pas suivre des objets de taille petit car les vidéos ne contient pas d'objet petit. Et faire comme ça peut éviter des faux.

## 3 Présentation du programme

### 3.1 Structure

Dans ce TP, le programme contient 3 classes :

1. Classe AMat : considère comme une image, elle est enfant de la classe cv::Mat de l'OpenCV. Cette classe permet de réaliser des fonctionnalités telles que soustraction, dessin des boîte.
2. Classe Kalman : contient le vecteur des filtres Kalman de l'image de séquence jus avant séquence actuelle. Elle contient aussi un filtre Kalman actuel. Cette classe permet d'utiliser le filtre de Kalman de l'OpenCV.
3. Classe Control : Comme les TP avant, la classe Control permet de réaliser toutes les fonctionnalité du programme. Elle sert à contrôler le programme.

### 3.2 Utilisation du programme

1. La commande pour prendre le fond  
*./detectvideo fond video\_path fond\_path*
2. La commande pour détecter et suivre le mouvement de la vidéo  
*./detectvideo detect video\_path fond\_path temps entre deux frames*

## 4 Résultat obtenu

Voici un exemple avec quelques images d'une vidéo

Commande :

`./dectvideo detect video/Walk1.mpg image/test/fond_walk1.png 1`

Images de la vidéo



Figure 5:  $t = 1$



Figure 6:  $t = 37$



Figure 7:  $t = 55$



Figure 8:  $t = 82$

Images de la vidéo détecté

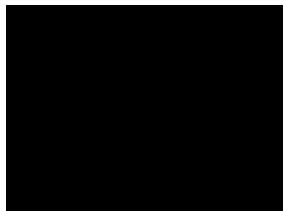


Figure 9:  $t = 1$

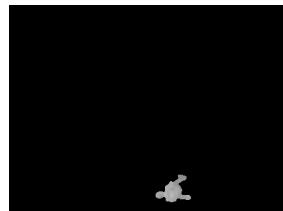


Figure 10:  $t = 37$



Figure 11:  $t = 55$



Figure 12:  $t = 82$

Images de la vidéo suivi



Figure 13:  $t = 1$



Figure 14:  $t = 37$



Figure 15:  $t = 55$

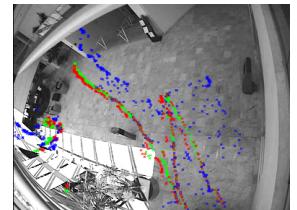


Figure 16:  $t = 611$

Explication :

On trouve que la détection marche bien sur cette vidéo, la personne dans la scène est bien détectée. La question posée ici est que ce programme détecte toujours bien des objets de mouvement dans toutes les vidéos ? Non, il marche bien sur cette vidéo car la personne dans la vidéo a la couleur très différent avec la couleur de l'image du fond. Une autre raison est que la personne se déplace, elle ne reste pas à une position pendant longtemps. Cela permet de bien détecté.

Par contre, par rapport des images de suivi, la prédiction n'est pas bien en global. Mais les trois premières images ( $t = 1, 37$  et  $55$ ) nous confirme que la prédiction et correction sont bien. La raison est que dans les première frames, il y ni de bruits ni de trou et la vidéo est simple. Dans mon implémentation, le frame actuel utilise des filtres de séquence dernière. Donc, ça marche bien si l'objet existe dans toutes les séquences de mouvement. S'il n'existe pas dans une séquence, la séquence suivant n'a pas de filtre mieux pour utiliser.

Exemple 2 :  
`./dectvideo detect video/Meet_WalkTogether1.mpg image/test/fond_meet1.png 1`  
 Images de la vidéo



Figure 17:  $t = 300$



Figure 18:  $t = 350$



Figure 19:  $t = 400$



Figure 20:  $t = 707$

Images de la vidéo détecté



Figure 21:  $t = 300$

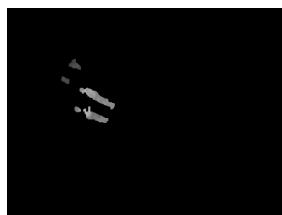


Figure 22:  $t = 350$

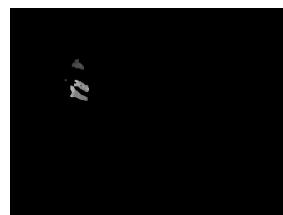


Figure 23:  $t = 400$

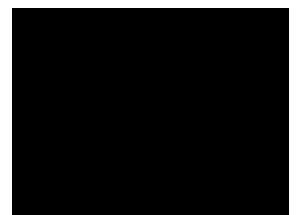


Figure 24:  $t = 707$

Images de la vidéo suivi



Figure 25:  $t = 300$

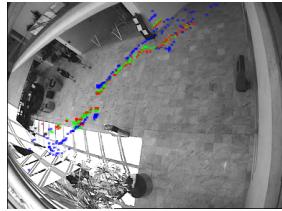


Figure 26:  $t = 350$

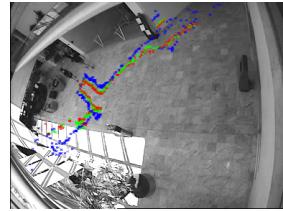


Figure 27:  $t = 400$

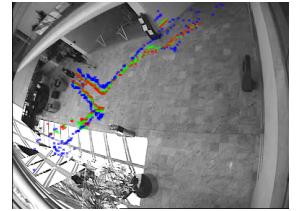


Figure 28:  $t = 707$

Explication :

Dans cet exemple, le filtre marche meilleure qu'avant. Les deux objets ont des trajectoires séparées et évite beaucoup d'erreurs. On voit que la distance entre la mesure et la correction est plus courte que celle entre la mesure et la prédiction. Cela répond bien au problème de suivi des mouvements.

Quand le temps près de  $t = 300$ , l'objet apparaît et disparaît au cours du temps, la méthode n'adapte pas bien dans ce cas.

Exemple 3 :  
`./dectvideo detect video/Meet_Crowd.mpg image/test/fond_meet_crow.png 1`  
 Images de la vidéo



Figure 29:  $t = 200$



Figure 30:  $t = 220$



Figure 31:  $t = 250$



Figure 32:  $t = 300$

Images de la vidéo détecté



Figure 33:  $t = 200$



Figure 34:  $t = 220$

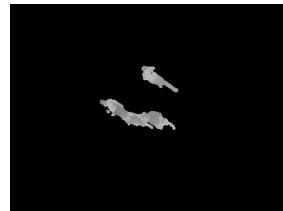


Figure 35:  $t = 250$



Figure 36:  $t = 300$

Images de la vidéo suivi



Figure 37:  $t = 200$



Figure 38:  $t = 220$

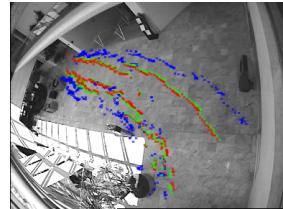


Figure 39:  $t = 300$

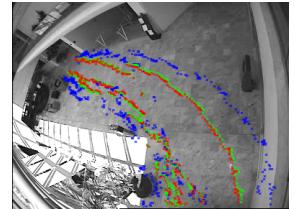


Figure 40:  $t = 484$

Explication :

Cet exemple n'est pas facile de détecter. On voit que quand le temps = 250 ( $t = 250$ ), trois objets deviennent 1 objet (crois), c'est difficile pour suivre dans la séquence après. Dans l'implémentation, je compte seulement la position, ne compte pas des cas comme ça, donc, le programme ne connaît pas quelle trajectoire est pour quel objet pour ce cas.

Solution proposée : Une part, on compte la vitesse et la direction des objets pour séparer dans les cas comme ce cas. Une autre part, on peut améliorer l'étape de détecter l'objet pour distinguer ces trois objets. Car en réalité, ce sont les personnes différentes.

**Exemple 4 :**  
`./dectvideo detect video/Fight_RunAway1.mpg image/test/fond_fight_run1.png 1`  
 Images de la vidéo



Figure 41:  $t = 1$



Figure 42:  $t = 300$



Figure 43:  $t = 400$



Figure 44:  $t = 551$

Images de la vidéo détecté



Figure 45:  $t = 1$



Figure 46:  $t = 300$



Figure 47:  $t = 400$

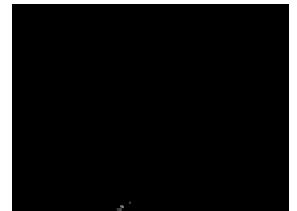


Figure 48:  $t = 551$

Images de la vidéo suivi



Figure 49:  $t = 1$



Figure 50:  $t = 300$

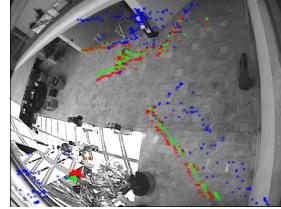


Figure 51:  $t = 400$

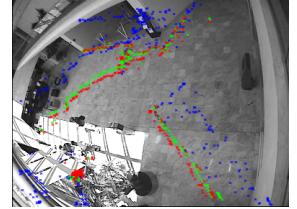


Figure 52:  $t = 551$

#### Explication :

Dans cet exemple, la prédition n'est pas bien. Par contre, la correction est très bien.

Après avoir expérimenté des résultat, on trouve que la distance entre la mesure et la correction est plus courte que celle entre la mesure et la prédition. Cela respecte le principe de la méthode de suivi de mouvement. Ici, la mesure est la position en réalité (ne pas compter des erreurs de traitement).

## 5 Conclusion

En global du résultat, la principe du filtre de Kalman est respecter (la prédition a plus erreurs que la correction). Dans ce TP, Je trouve que le programme marche bien pour les deux premières étapes? Par contre, je me semble que la troisième étape il ne marche pas très bien malgré mes efforts. Dans la méthode Kalman, on peut simple utiliser un seul filtre mais ce n'est pas possible pour plusieurs objets. J'ai fait des efforts pour utiliser plusieurs filtres et choisir la meilleurs pour chaque objet. Bien que le résultat n'est pas très bien, le programme peut résoudre le problème de multiple objets.

Pour ce TP, je me concentre à la position des objets. Il y a plusieurs façon pour améliorer le résultat de suivi de mouvent telles que :

1. Utilisation de la vitesse

2. Utilisation de la taille des objets
3. Utilisation des méthodes de reconnaissance des objets pour distinguer des objets différents. Cette façon a l'air intéressant mais c'est difficile pour réaliser à cause du changement des caractères des objets dans les étapes avant.

## References

- [1] NGUYEN Thi Oanh, *TP3 : Détection et suivi de mouvement*. IFI, Hanoi, Vietnam, 2013.
- [2] OpenCV *OpenCV Sample source code*. <https://github.com/Itseez/opencv/blob/master/samples/cpp/kalman.cpp>