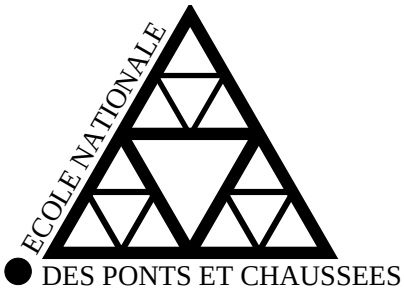

Vision - Traitement d'images

Active Shapes
Filtre de Kalman

Flavien Billard
Aurélien Boffy

27 juin 2005

Introduction

Le but du projet est le tracking d'un objet sur une vidéo (en l'occurrence un joueur de foot) grâce à 2 principales méthodes :

- **Les formes actives** (ou *Actives Shapes*) : Nous essayons de suivre l'objet en se basant sur un modèle qui est censé le représenter. Ainsi, un modèle pour le joueur de foot pourrait être le contour d'un bonhomme que l'on essaierait de faire correspondre au mieux avec le joueur qui est sur la vidéo. Le problème est que le joueur ne garde pas la même forme et qu'il ne correspond évidemment jamais au modèle *parfait* que l'on utilise. Pour essayer de contourner ce problème, on utilise les *Actives Shapes* : un modèle est dans ce cas non seulement un contour de l'objet qui est en fait ce que l'on obtient en faisant la moyenne des images utilisées pour calibrer le modèle, mais aussi quelques mouvements élémentaires qui correspondent aux modes principaux de variation que l'on obtient en faisant une "Analyse en Composantes Principales" sur les images utilisées pour la calibration. Ainsi par exemple, dans notre cas du joueur de foot, un mode de variation pourra faire écarter (ou se croiser) les jambes du joueur, un autre pourra lui faire lever les bras, etc.

- **Le filtre de Kalman** : Ceci permet d'avoir un tracking facilité par le fait que l'on prend en compte les mouvements précédents pour essayer de deviner où se trouve le joueur à l'instant t. Il faut initier le filtre, puis, en corrigeant à chaque étape sa prédiction avec la position réelle, il s'améliore au fur et à mesure de la vidéo pour donner une estimation de la position future toujours plus précise

1 Active Shapes

1.1 Notations

Commençons par donner quelques notations :

- $P = (p_1\, p_2\, \dots\, p_7)$: matrice qui contient, sur 7 colonnes, les 7 vecteurs représentant les modes de variation.
- $b = (b_1\, b_2\, \dots\, b_7)$: vecteur des 7 coefficients appliqués à chaque mode principal.
- $\bar{x}$: Contour moyen du joueur (c'est-à-dire contour qui n'a subi aucune transformation)
- $T_{X_T,Y_T,s,\theta}$: fonction qui applique une simple similitude (translation, rotation et homothétie) à un contour.

Ainsi, grâce aux 11 paramètres $(X_T, Y_T, s, \theta, b_1, \dots, b_7)$, on obtient un contour x par :

$$x = T_{X_T,Y_T,s,\theta}(\bar{x} + Pb)$$

où :

$$T_{X_T,Y_T,s,\theta}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} X_T \\ Y_T \end{pmatrix} + \begin{pmatrix} s\cos\theta & s\sin\theta \\ -s\sin\theta & s\cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

1.2 Recherche manuelle des paramètres (`interface_modes`)

Pour calibrer le filtre de Kalman, nous avions besoin de trouver les coefficients à attribuer à chaque mode principal de variation pour se rapprocher au mieux de la forme du joueur dans les 7 premières frames. Nous avons donc créé une petite interface qui permettait de régler de façon plus facile les 7 paramètres (plus la translation, la rotation et le changement d'échelle) (cf fonction `interface_modes`). En fait, nous pouvions augmenter et diminuer les coefficients de chaque mode grâce aux touches du clavier et voir le résultat directement.

1.3 Recherche automatique des paramètres (`ajusteParametres`)

Cependant, il était trop compliqué de régler ces 11 paramètres à la main. Nous n'arrivions jamais à épouser correctement la forme du joueur. Nous avons donc utilisé les *vraies* positions du joueur sur les premières images (c'est-à-dire l'abscisse et l'ordonnée des 100 points qui définissent son contour) et nous avons programmé une fonction qui permet de calculer directement les 11 paramètres cherchés pour transformer *moyenne*, qui est le contour moyen du joueur (sans translation, ni rotation, ni homothétie, ni transformation par les modes de variation), afin de se rapprocher au mieux d'une forme donnée. Cette fonction s'appelle `ajusteParametres` et utilise l'algorithme itératif suivant :

On cherche les 11 paramètres qu'il faut appliquer à x pour épouser au mieux une forme Y :

1. On commence avec $b = 0$
2. On calcule $x = \bar{x} + Pb$
3. On trouve les meilleurs X_T, Y_T, s, θ qu'il faut appliquer à x pour se rapprocher de Y (cf. plus bas)
4. On calcule $y = T^{-1}_{X_T,Y_T,s,\theta}(Y)$ pour se replacer dans le système de coordonnées où est défini $\bar{x}$
5. On applique à y une homothétie de facteur $\frac{|\bar{x}|^2}{y.\bar{x}}$ pour que les échelles soient comparables
6. On met à jour b avec : $b = P^T(y - \bar{x})$
7. Retourner en 2 si ça n'a pas converger (plusieurs tests sont ici possible)

1.4 Recherche automatique des paramètres de la similitude (`ajusteSimilitude`)

Pour la troisième étape, on a besoin d'une fonction qui ajuste les 4 paramètres X_T, Y_T, s et θ pour s'ajuster à une forme. Cette fonction a donc un

objectif similaire à `ajusteParametres` sauf qu'elle ne règle que 4 paramètres. On l'a appelé `ajusteSimilitude` et voici ce qu'elle fait :

Étant donné 2 formes x et x', nous cherchons les paramètres de $T_{X_T,Y_T,s,\theta}$ pour que x s'aligne au mieux avec x'.

Pour calculer X_T et Y_T, il suffit de faire coïncider les centres de gravité des 2 formes.

Supposons maintenant que les centres de gravité des 2 formes soient tous deux situés à l'origine. Il reste à calculer s et θ. On utilise en fait tout simplement une méthode des moindres carrés dont voici le résultat :

On note :
- $a = \frac{x.x'}{|x|^2}$
- $b = \frac{\sum_{i=1}^{n}(x_i y'_i - x'_i y_i)}{|x|^2}$

Et alors :
- $s^2 = a^2 + b^2$
- $\theta = tan^{-1}\frac{b}{a}$

1.5 Tests (`transforme` et `affiche`)

Nous avons pu facilement tester ces fonctions :

Nous avons pris *moyenne* que nous avons déformé avec 11 paramètres pris au hasard (en utilisant nos fonctions `transforme`) et nous avons essayé de voir si `ajusteParametres` retrouvaient ces paramètres : nous avons constaté que c'était bien le cas.

Nous avons ensuite pris les formes avec les coordonnées précises des premières images de la vidéo et avons testé `ajusteParametres` pour voir si nous pouvions retrouver les contours du joueur. Nous avons tracé les formes obtenus (avec la fonction `affiche`) et nous pouvions remarquer que la correspondance entre les formes étaient très bonnes.

Ainsi, on peut conclure cette partie en disant que la partie "*Active Shapes*" du projet a été correctement réalisée : nous sommes capables de trouver les 11 paramètres pour ajuster une forme à une autre forme quelconque. Reste à faire fonctionner le filtre de Kalman sur ces 11 paramètres.

2 Filtre de Kalman

2.1 Principe

Pour chaque frame de la video, nous voulons estimer le vecteur état du système (les paramètres de la similitude plus les 7 coefficients appliqués aux 7 modes de variation). A partir de ce vecteur état, nous pouvons, grâce à la fonction `transforme` et à partir de la forme *moyenne*, retrouver le vrai contour du joueur.

Le vecteur état du système est : $x = (x_G\ y_G\ \theta\ s\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7)$

Comme expliqué précédemment, le filtre de Kalman nous permettra d'obtenir une estimation *a posteriori* du vecteur état à l'instant k (notée $\hat{x}_k^+$), à partir

- d'une estimation *a priori* à l'instant k (notée $\hat{x}_k^-$) calculée avec les estimations *a posteriori* des vecteurs états aux instants précédents ($k-1$ et $k-2$)
- d'une *mesure* (notée z_k) calculée avec l'estimation *a priori* à l'instant k

L'estimation *a posteriori* est une pondération entre cette estimation *a priori* et la mesure :

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{1}$$

où K_k est la matrice de Kalman à l'instant k, calculée à partir de la matrice de Kalman à l'instant $k-1$ et des matrices Q et R, qui représentent les matrices de covariance des erreurs sur l'estimation *a priori* et sur la mesure, dont nous parlerons plus bas.

2.2 Equations régissant le mouvement du joueur et la mesure

L'équation qui régit le **mouvement du joueur** est :

$$x_k = x_{k-1} + u_k + w \tag{2}$$

où :

- u_k est la "commande"
- w est le bruit (l'erreur) qui suit une loi normale centrée de matrice de covariance Q

Les **mesures** z_k que l'on va utiliser suivent une loi du type :

$$z_k = x_k + v$$

où v est le bruit (l'erreur) qui suit une loi normale centrée de matrice de covariance R.

Les matrice d'erreur Q et R seront calculées lors de la phase d'apprentissage. Elles sont supposées constantes, mais on pourrait éventuellement plus tard décider de les mettre à jour au cours du temps en considérant les erreurs successives obtenues.

2.3 Obtention de l'estimation *a priori* $\hat{x}_k^-$ (`calc_estim_apriori`)

Pour chaque nouvelle frame, on obtient une estimation a priori en utilisant l'équation 2 :

$$\hat{x}_k^- = A\hat{x}_{k-1}^+ + Bu_k \tag{3}$$

où $u_k = \hat{x}_{k-1}^+ - \hat{x}_{k-2}^+$.

2.4 Obtention de la mesure (`calc_observation`)

Pour trouver la mesure z_k, nous allons partir de l'estimation a priori $\hat{x}_k^-$. Pour chaque point du contour estimé *a priori* (c'est-à-dire obtenu avec $\hat{x}_k^-$), nous nous déplaçons sur la normale au contour, et nous recherchons un gradient important (pour retrouver le contour exact du joueur). Nous avons dans un premier temps décidé de garder le point sur le profil qui a le gradient le plus important. Cependant, ce point n'était pas toujours le bon et nous avons décidé de privilégier les points proches de la position de $\hat{x}_k^-$. C'est la fonction `deplacements` qui est chargée de trouver la "mesure" à partir d'une estimation a priori.

Puis, par une méthode des moindres carrés (avec la fonction `ajusteParametres` évoquée plus haut), nous recherchons le vecteur état qui permette d'approcher cette forme au mieux : ce sera notre mesure z_k.

2.5 Calcul de l'estimation a posteriori grâce à la matrice de Kalman (`calc_estim_aposteriori` et `majKalman`)

Calcul de l'estimation a posteriori : Comme expliqué plus haut, l'estimation a posteriori est une pondération entre l'estimation a priori et la mesure, selon l'équation 1.

C'est la fonction `calc_estim_aposteriori` qui, dans notre programme, calcule cette pondération. C'est donc dans ce calcul que la matrice de Kalman intervient. Voici comment elle est calculée :

Calcul de la matrice de Kalman K_k : Pour déterminer K_k nous utilisons cette formule de récurrence (implémentée dans la fonction `majKalman`) :

$$K_k = P_k^-(P_k^- + R)^{-1}$$

avec

$$P_k^- = P_{k-1}^+ + Q$$

où

$$P_{k-1}^+ = (\mathbb{I}_4 - K_{k-1})P_{k-1}^-$$

Pour l'initialisation de P nous avons choisi $P_0 = 0$

2.6 La phase d'apprentissage (fonction `apprentissage`)

Dans notre algorithme nous remplissons au fur et à mesure un tableau `etats` de dimension $11 \times N$ qui sauvegarde les vecteurs états calculés pour les N frames de la vidéo.

Pour les images 1 à 7, nous avons les véritables positions des 100 points qui forment le contour. Nous pouvons donc calculer les 11 paramètres qui permettent, à partir de *moyenne*, d'obtenir ces positions (cf. section 1.3). Nous avons donc les 7 premiers vecteurs état de façon très précises, et nous pouvons remplir les 7 premières colonnes de `etats`.

Cela va nous permettre de calculer les matrices Q et R :

A partir de la 3è image, nous pouvons comparer ces vecteurs états *exacts* avec ceux que l'on aurait obtenu avec la formule (3), c'est-à-dire les estimations *a priori*. Cela nous permettra d'avoir 5 vecteurs "erreur" e_k.

On obtient aussi 5 vecteurs "erreurs" ϵ_k en calculant la différence entre l'observation obtenue à partir de l'estimation a priori (recherche du contour avec les gradients puis méthode des moindres carrés) et les valeurs exactes dont nous disposons.

Il nous reste à trouver Q et R, les covariances des bruits gaussiens : ces matrices s'obtiennent par les formules suivantes :

$$Q = \mathbb{E}(e_k^T e_k) \qquad R = \mathbb{E}(\epsilon_k^T \epsilon_k)$$

2.7 Résultats

L'implémentation du filtre Kalman n'a pas marché. Pour les valeurs de K_k nous obtenons des valeurs incohérentes. La matrice ne converge pas vers une matrice constante. Les estimations du mouvement sont donc complètement fausses.

Conclusion

Ainsi, la partie sur les *Active Shapes* est correcte et donne de très bons résultats : nous sommes capable de trouver les 11 paramètres (similitude + modes de variation principaux), qui permettent de retrouver la forme "admissible" la plus proche d'une forme quelconque.

Le problème vient de la partie concernant le filtre de Kalman : nous avons implémenté l'algorithme présenté dans [3] mais nous n'obtenons aucun résultat. Le problème vient apparemment de la matrice de Kalman qui est fausse. Nous ne savons pas si le problème est purement d'ordre "technique" et vient de notre code (problème dans les produits de matrice, etc.) ou s'il est conceptuel : peut-être que nous avons mal compris l'algorithme et avons fait une erreur dans l'interprétation que nous avons fait de la théorie sur le filtre de Kalman.

Références

[1] Cootes, T.F. et Taylor, C.J. *Statisticals Models of Appearance for Computer Vision.* University of Manchester, 2001.

[2] Maybeck, Peter S. *Stochastic Models, estimation, and control*, Volume 1. Academic Press, 1979.

[3] Welch, Greg et Bishop, Gary *An Introduction to the Kalman Filter.* University of North Carolina, 2004.