

Détection de la trajectoire d'une balle de tennis

Raphaël Montaud et Cyril Malbranke

8 décembre 2017

Résumé

L'objectif est de pouvoir arbitrer un match de tennis en annonçant les balles qui rebondissent en dehors du terrain. Dans un premier temps nous travaillerons avec une caméra classique (donc une seule image de la même scène) et nous nous pencherons ensuite sur le même problème avec une caméra 3D (donc deux images différentes de la même scène).

1 Benchmarking et choix des données de travail :

1.1 La technologie Hawk-Eye

L'état de l'art est déjà bien développé pour ce qui est du suivi de la trajectoire d'une balle. En effet, la technologie Hawk-Eye opère déjà dans plusieurs sports et peut remplacer les décisions de l'arbitre dans certaines compétitions officielles. Par exemple au tennis, dans certains tournois comme le Qatar Total Open, les joueurs peuvent contester la décision de l'arbitre à trois reprises au cours d'un set en faisant appel à la technologie Hawk-Eye. Le suivi de la trajectoire se fait avec l'aide de 6 caméras haute performance qui traquent la balle. Une triangulation permet ensuite de connaître la position de la balle en trois dimensions. Enfin, le nuage de points obtenus est transformé en une courbe 3D qui correspond à la trajectoire la plus probable.

1.2 Choix des données

Il est difficile de trouver des images de matchs de tennis de qualité suffisante. En effet, même une très bonne qualité d'image comme la 4k ne suffit pas à traquer la position de la balle. Si il n'y a pas assez d'images par secondes, la balle peut apparaître très floue sur les images ce qui rend la tâche très compliquée. Nous avons donc contacté l'entreprise Hawk-Eye Innovations pour être en mesure de tester nos algorithmes sur des données réelles. Cependant, l'issue de cette démarche étant incertaine, nous avons cherché une autre source de données.

Notre choix s'est finalement porté sur des images de jeux vidéos de tennis. Celles-ci sont effectivement de très bonne qualité, et il n'y a pas de flou autour de la balle lorsque l'on fait un arrêt sur image. Cela simule donc des images haute fréquence et de bonne qualité. De plus, les images varient peu (notamment en termes de lumière), ce qui permet de simplifier le problème dans un premier temps. En revanche, la caméra bouge contrairement aux images des caméras Hawk-Eye et cela va donc nous obliger à procéder à un prétraitement.

1.3 Objectifs

Dans un premier temps, nous travaillerons avec des séries d'images provenant d'un seul point de vue. Nous travaillons sur des images de jeux vidéos, le travail sur ces images se décompose de la sorte :

- On stabilise l'image, afin de fixer le terrain.
- Dans chaque image de la série d'images, on cherche à détecter la balle.
- Puis on cherche à reconstituer la trajectoire de la balle avant et après le rebond.
- On localise ainsi l'impact de la balle pour savoir si celle-ci est faute.

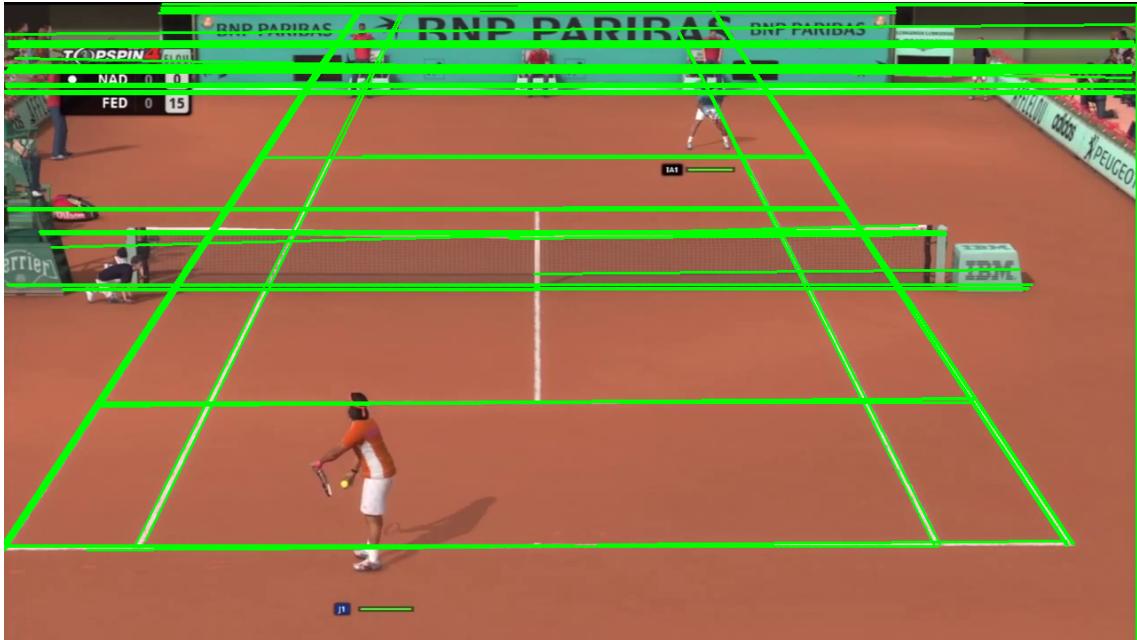


FIGURE 1 – détection de lignes, cas 1

2 Preprocessing

2.1 De la vidéo à la séquence d'image

Il faut dans un premier temps transformer la vidéo en une séquence d'images. Pour cela nous avons utilisé le logiciel ffmpeg. Le format d'une vidéo est en fait très compressé et quelques minutes de vidéo (donc quelques Mo) ont produit plusieurs Go d'images. La vidéo utilisée est contenue dans le dossier ressources.raw_files et la vidéo au format images par images est contenue dans le dossier ressources.framebyframe2D.

2.2 Stabilisation de l'image

Dans le cadre de notre premier travail avec une seule image, il est nécessaire de stabiliser l'image. L'objectif est ici que les lignes du terrains aient toujours la même position dans l'image. Ainsi, cela facilite grandement le travail d'analyse de la trajectoire. Pour cela nous avons choisi une image de référence. Puis, nous calculons l'homographie qui permet de passer d'une image à l'image de référence. Pour cela, il existe plusieurs façons de faire. On peut par exemple citer les méthodes de corrélation d'images avec les points de Harris. Pour notre part, nous avons décidé de prendre avantage de la spécificité du problème. Nous faisons de la détection de lignes qui nous permet de trouver les quatre coins du terrains et ainsi calculer l'homographie qui relie l'image à celle de référence.

Pour la détection de lignes, nous utilisons d'abord un filtre de Canny qui permet de faire de la détection de contours. Puis nous appliquons l'algorithme Hough Lines de OpenCv. Cet algorithme repose sur un principe assez simple mais il fournit des résultats satisfaisants. Pour chaque droite possible (ρ , θ), on regarde combien de points appartiennent à cette ligne (nous rappelons ici que l'image est binaire à la sortie du filtre de Canny). Les droites avec le plus de votes sont retenues. Nous identifions ensuite les lignes qui nous intéressent grâce à de simples conditions sur la position et l'orientation. Cependant, le fait que la caméra bouge rend ce conditionnement difficile. Il faut garder des conditions larges, ce qui fait que certaines images renvoient des lignes avec des positions aberrantes (cf figures 1 à 4).

Pour cela nous appliquons plusieurs filtres sur les données pour éliminer des valeurs aberrantes : Un premier filtre consiste à calculer l'écart au carré de l'homographie par rapport à l'homographie moyenne. Cela permet d'avoir un tracé qui correspond aux mouvements de caméras, comme on peut le voir dans la figure 5. Les mouvements de caméras extrêmes correspondent à des valeurs autour de 25 000. Toutes les valeurs au dessus de ce seuil sont donc considérées comme aberrantes

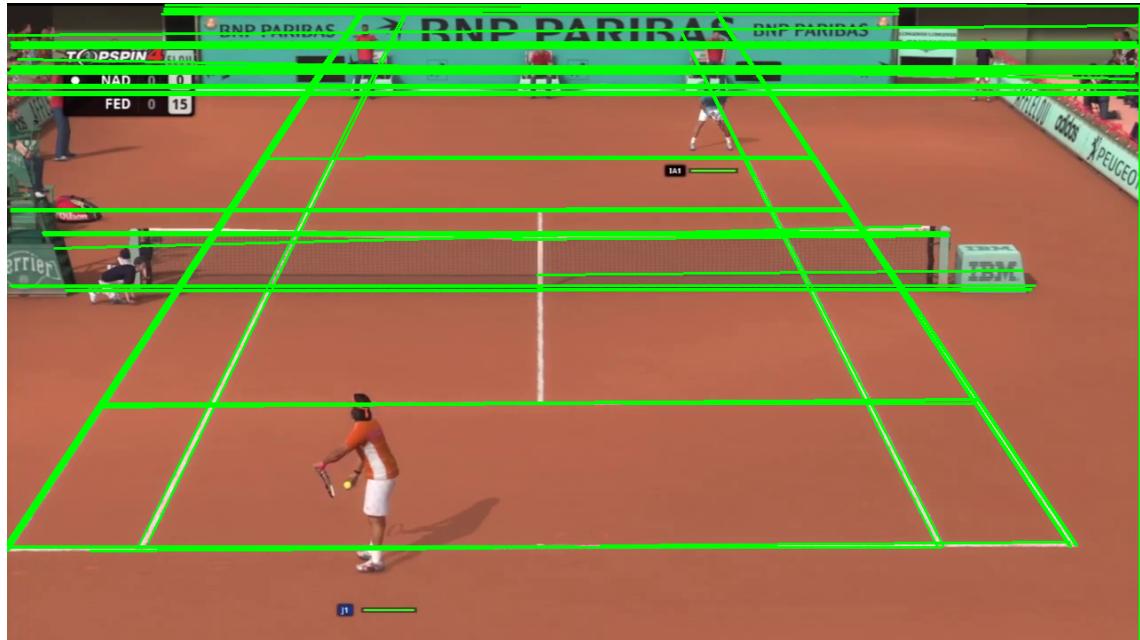


FIGURE 2 – sélection de lignes, cas 1 : succès

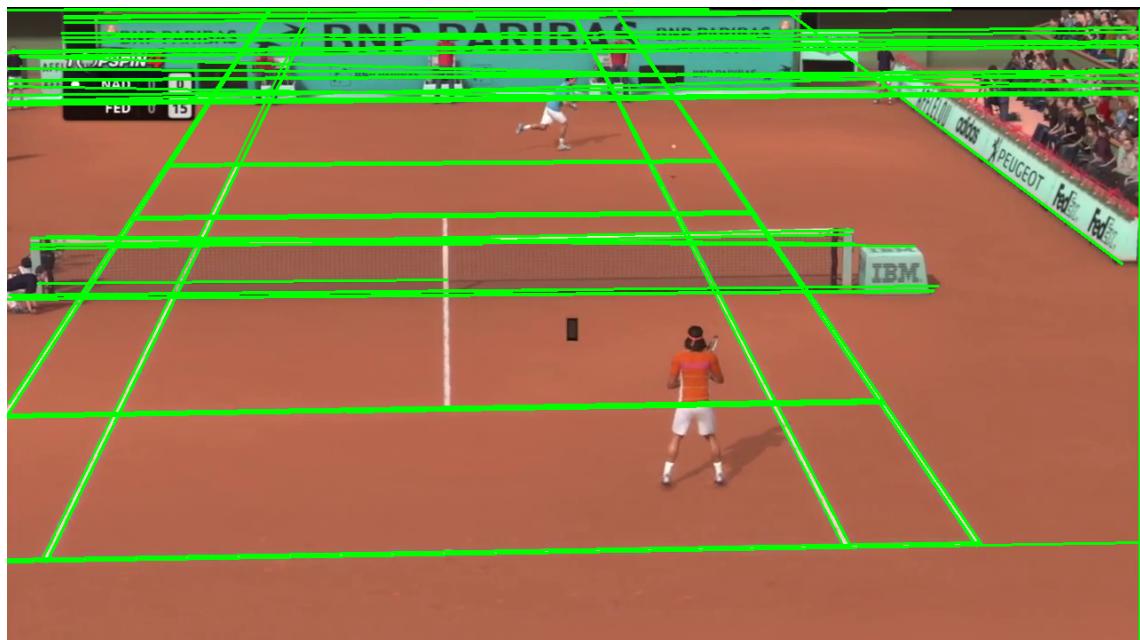


FIGURE 3 – détection de lignes, cas 2

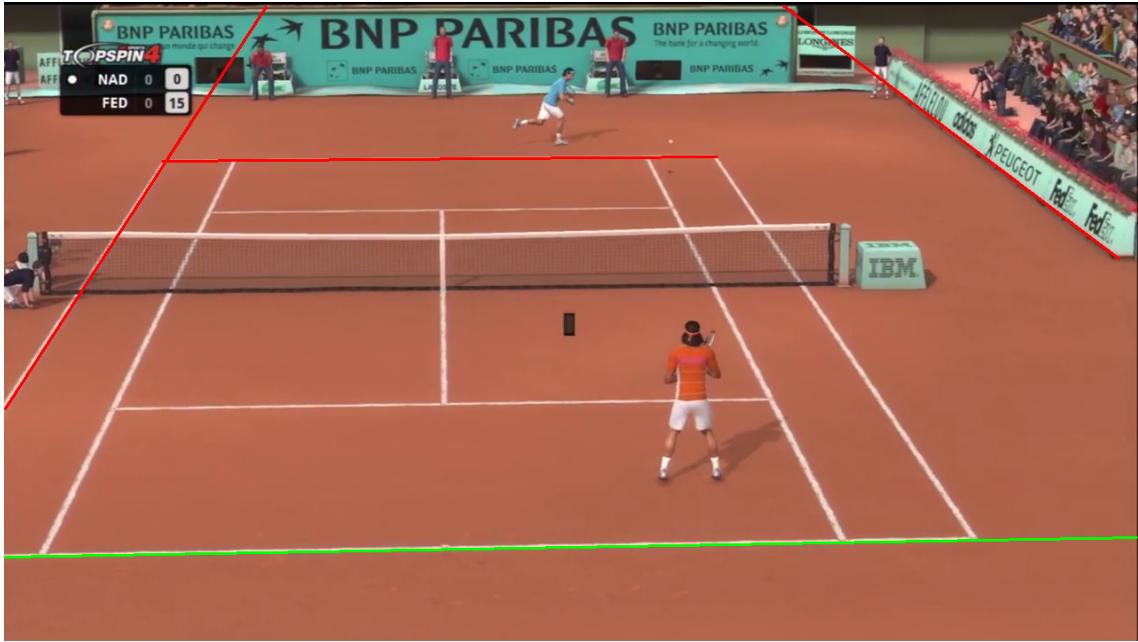


FIGURE 4 – sélection de lignes, cas 1 : échec

et l'homographie calculée est remplacée par l'homographie de l'image précédente.

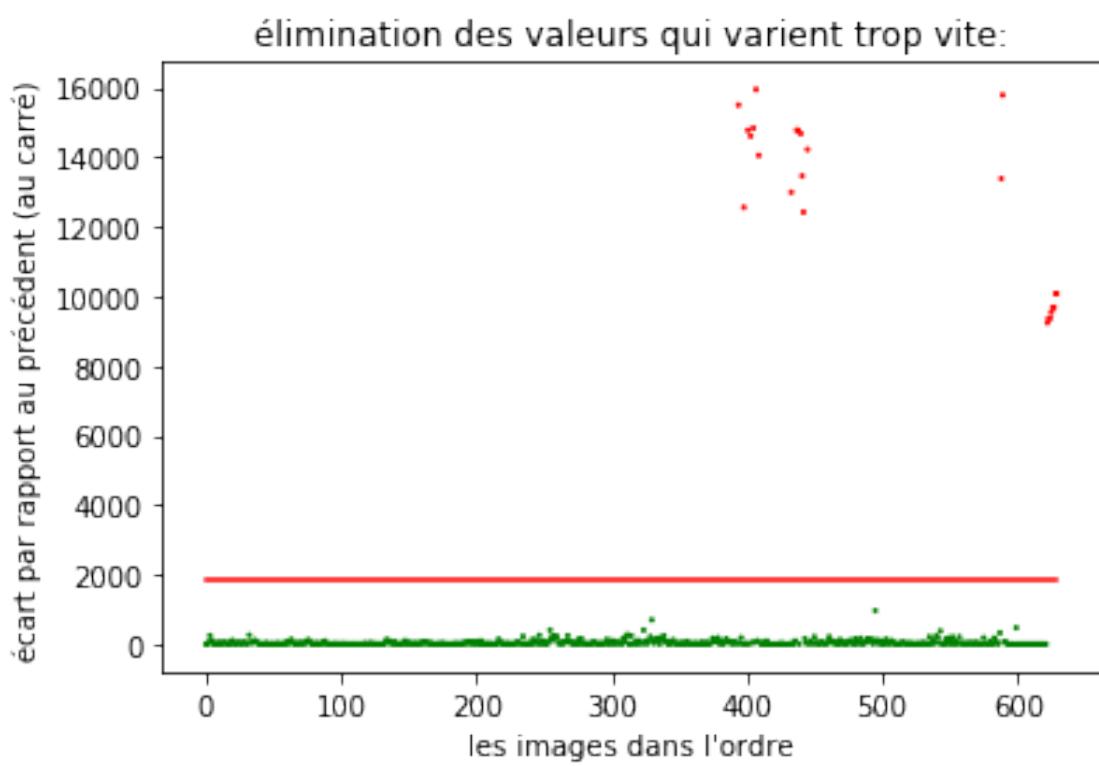
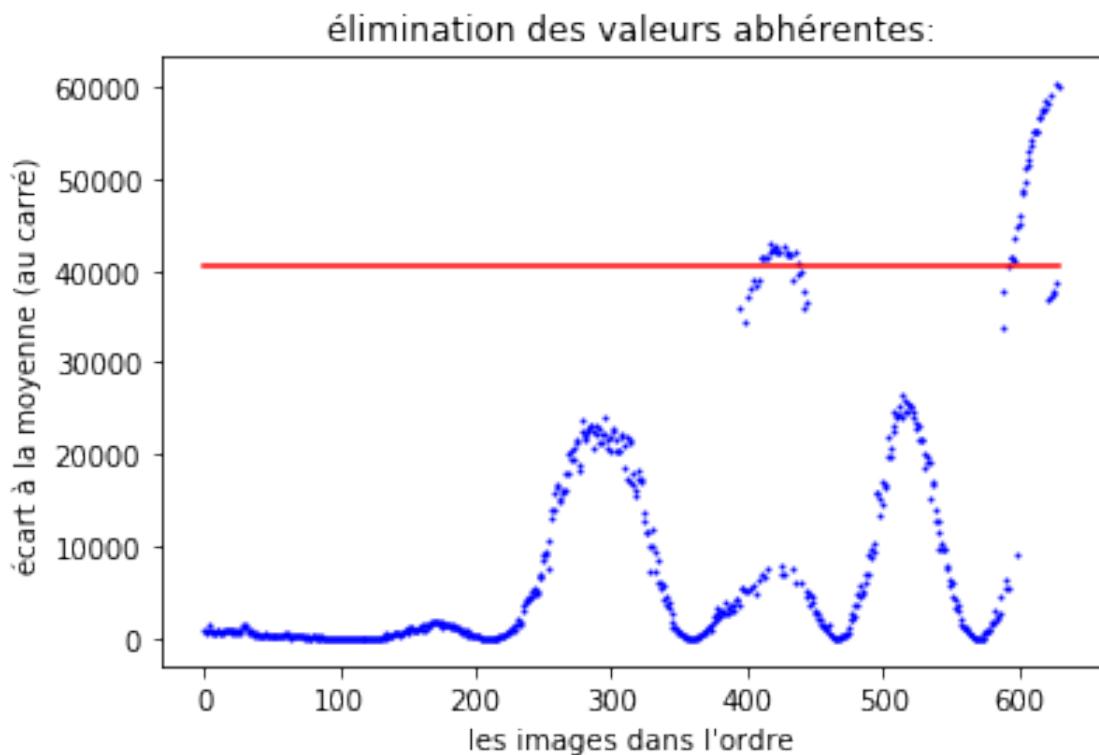
Ensuite, la variation des homographies est censée être continue. Nous éliminons donc les homographies où la variation est jugée trop rapide. Il est d'ailleurs assez facile d'identifier graphiquement les points aux valeurs aberrantes, comme on peut le voir dans la figure 6 :

Enfin nous appliquons un filtre gaussien sur les homographies calculées pour obtenir un résultat plus lisse.

Le résultat obtenu est très satisfaisant. Cependant, beaucoup d'améliorations sont envisageables pour cette partie du projet. Il y a d'abord un certain nombre de paramètres à optimiser. Ensuite les lignes du terrains sont considérées comme si elles n'avaient pas d'épaisseur, or, pour obtenir des résultats plus précis, il faudrait chercher à distinguer le bord extérieur et le bord intérieur. Nous pourrions aussi traquer plus de points (cf partie 3D) ce qui augmenterait la précision pour estimer les homographies. Nous avons décidé de ne pas nous attarder plus que ça sur cette partie car c'est un 'faux problème' étant donné qu'il n'y a pas de raisons de s'imposer des caméras qui bougent, et le résultat sur la séquence d'images obtenu est suffisant pour pouvoir poursuivre nos recherches.

2.3 Description du code :

Le fichier README.md donne des explications sur comment faire marcher nos algorithmes. Les fonctions définies dans le fichier lines.py permettent de détecter les lignes dans les images, puis de sélectionner celles qui nous intéressent, de calculer leurs intersections et enfin d'en déduire l'homographie qui relie deux images. Le script warp_images.py calcule les homographies d'une série d'images par rapport à une image de référence. Le traitement des valeurs abhérantes et le lissage sont ensuite effectués. Enfin la rectification de l'image est effectuée et sauvegardée en mémoire.



3 Détection de trajectoire avec une seule prise de vue

3.1 Première étape : localiser la balle dans l'image

3.1.1 Application de l'algorithme GraphCuts

La méthode retenue pour localiser la balle dans l'image est d'utiliser l'algorithme du GraphCuts, vu en cours. On définit donc l'intérieur comme étant la balle avec sa couleur, l'extérieur comme étant le reste de l'image, à cet effet on pointe la couleur du terrain.

Dans un premier temps, on s'aperçoit que l'algorithme de base ne donne pas de résultats satisfaisants, en effet, on s'aperçoit que beaucoup de pixels autres que la balle sont classifiés comme faisant partie de la balle. Pour résoudre ce problème, on décide de dilater le coût D_p pour les noeuds/pixels qui seront classés à l'intérieur :

$$D_p(ext) = \|color - color_{ext}\| \quad (1)$$

$$D_p(int) = k \|color - color_{int}\| \quad (2)$$

On cherche alors de manière empirique un compromis : une trop faible dilatation nous empêche de discriminer uniquement la balle, une trop forte dilatation risque de nous faire rater la balle comme on peut le voir avec les trois images ci dessous (5). Empiriquement on décide une valeur de dilation de 2.8.

3.1.2 Définir la position de la balle

A ce point là, on dispose donc pour chaque image du résultat d'un GraphCut que l'on peut représenter visuellement : les pixels noirs sont ceux classés comme étant à l'extérieur de la balle, les pixels blancs sont ceux classés comme étant à l'intérieur de la balle par l'algorithme.

La deuxième étape est donc d'attribuer une coordonnée à la balle pour chaque image. Deux idées sont développées :

1. On cherche le barycentre des pixels blancs. Evidemment dans ce cas de figure, le bruit et les pixels blancs isolés gachent complètement la mesure. On cherche donc pour améliorer le résultat le barycentre des pixels blancs entourés eux même de 4 pixels blancs, ce qui donne un résultat bien plus convaincant quoiqu'il fasse disparaître la balle dans certains cas où celle-ci est trop "discrète".
2. La deuxième méthode est d'appliquer un filtre gaussien assez serré (on considère seulement des régions de 5 pixels de diamètre), puis de chercher le point le plus "blanc" pour le définir comme étant la position de la balle. Empiriquement, on voit qu'il arrive quand le signal relatif à la balle est trop diffus qu'un bruit puisse être plus "blanc" et soit détecté comme étant la balle, débouchant sur une mesure totalement fausse. On établit donc empiriquement un seuil en dessous duquel on ne renvoie pas de résultat : pour un niveau de blanc plus bas que 180.

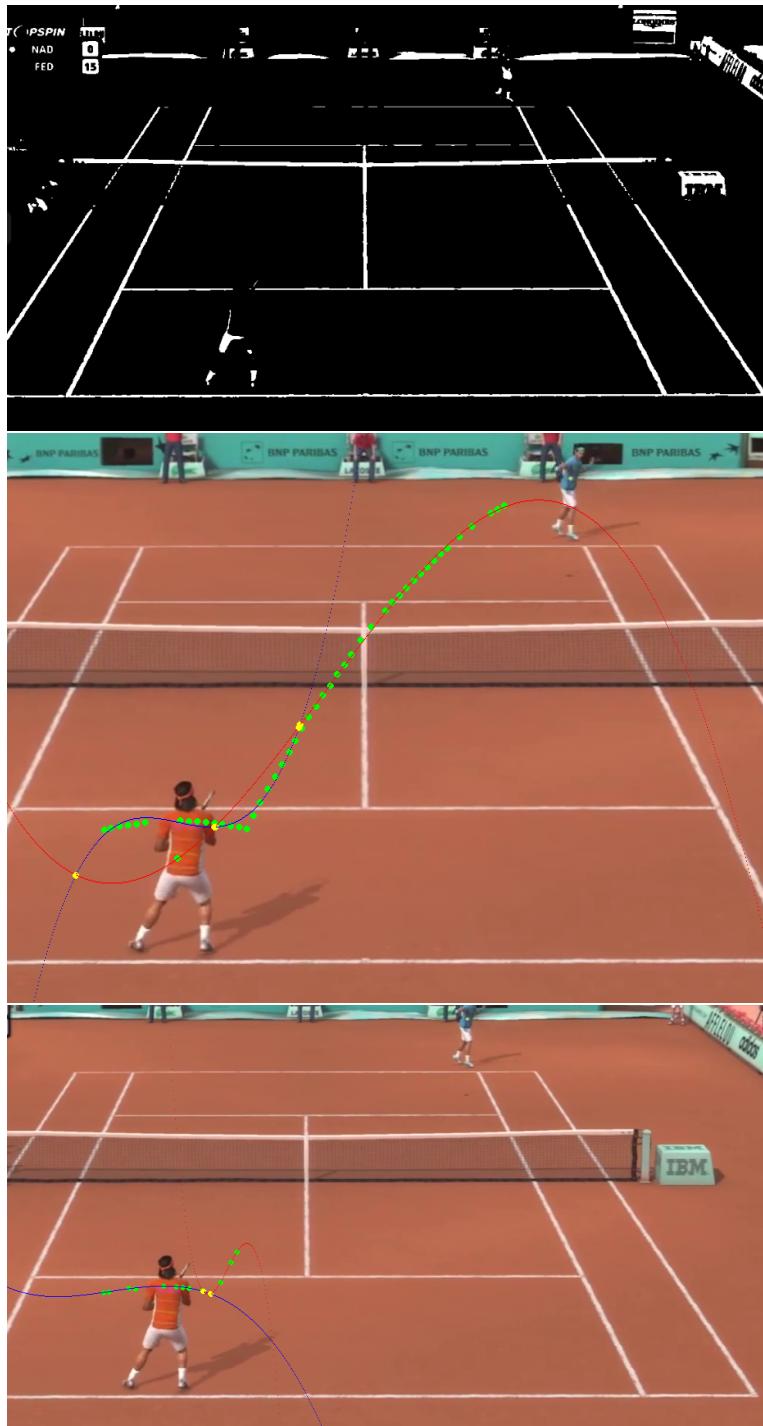


FIGURE 5 – Pour $k=1$, on met les pixels discriminés comme appartenant à la balle pour une image.
Pour $k=2.8$ et $k=7$, les positions de balles repérés sur 60 images

3.2 Deuxième étape : Reconstruire la trajectoire de la balle

3.2.1 Choix du modèle de la trajectoire

De rapides études physiques le prouvent, selon la manière de prendre en compte les frottements et autres forces "mineures", des polynômes de degré 2 ou 3 permettent d'excellentes approximations de la trajectoire d'une balle en mouvement libre. Ainsi, on choisit de ne pas essayer de comprendre le rebond intrinsèquement mais plutôt de modéliser la trajectoire avant le rebond et la trajectoire après le rebond par deux polynômes différents. Le rebond est alors détecté à l'intersection. Considérant la qualité des mesures et la qualité de l'ajustement de la courbe aux mesures, on peut considérer que l'approximation de la trajectoire à deux polynomes de degré 2 ou 3 est suffisante car elle excède la résolution des images que nous avons à notre disposition.

3.2.2 Méthode d'établissement des coefficients des polynomes

Pour établir les coefficients, nous nous sommes appuyés sur les fonctions de la librairie "Eigen" (<http://eigen.tuxfamily.org>) pour le calcul matriciel. Concrètement, nous avons réalisé une régression polynomiale, sur l'ensemble des points $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$ de la trajectoire que l'on a pu isoler. Pour ce faire, on résout l'équation matricielle suivante :

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & & & \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \quad (3)$$

On affirme sans plus de démonstration que $\beta = (X^T X)^{-1} X^T Y$ est solution. Pour obtenir deux polynomes distincts, on choisit donc, comme on peut le voir dans l'algorithme de séparer en deux groupes distincts les points trouvés (ceux avant une certaine image et ceux après une certaine image) et de construire pour chacun le polynôme. En testant pour toutes les séparations et en calculant le résidu des moindres carrés pour chaque séparation, on arrive à trouver le meilleur couple de polynomes possible pour la trajectoire.

3.3 Amélioration de la précision par gestion des outliers

On a pu voir que les méthodes utilisées pour trouver la trajectoire de la balle n'excluait pas la possibilité de trouver des mesures aberrantes pour la balle. Deux cas sont possibles : soit la mesure n'est pas gênante, soit c'est un outlier. On implémente donc en correctif une méthode simple pour supprimer les outliers : si la somme des distances d'un point aux deux points qui le précédent et aux deux points qui le suivent est trop importante, on ne le prend pas en compte dans notre regression.

On trouvera ci-contre le résultat de nos mesures pour deux séries d'images. [6]

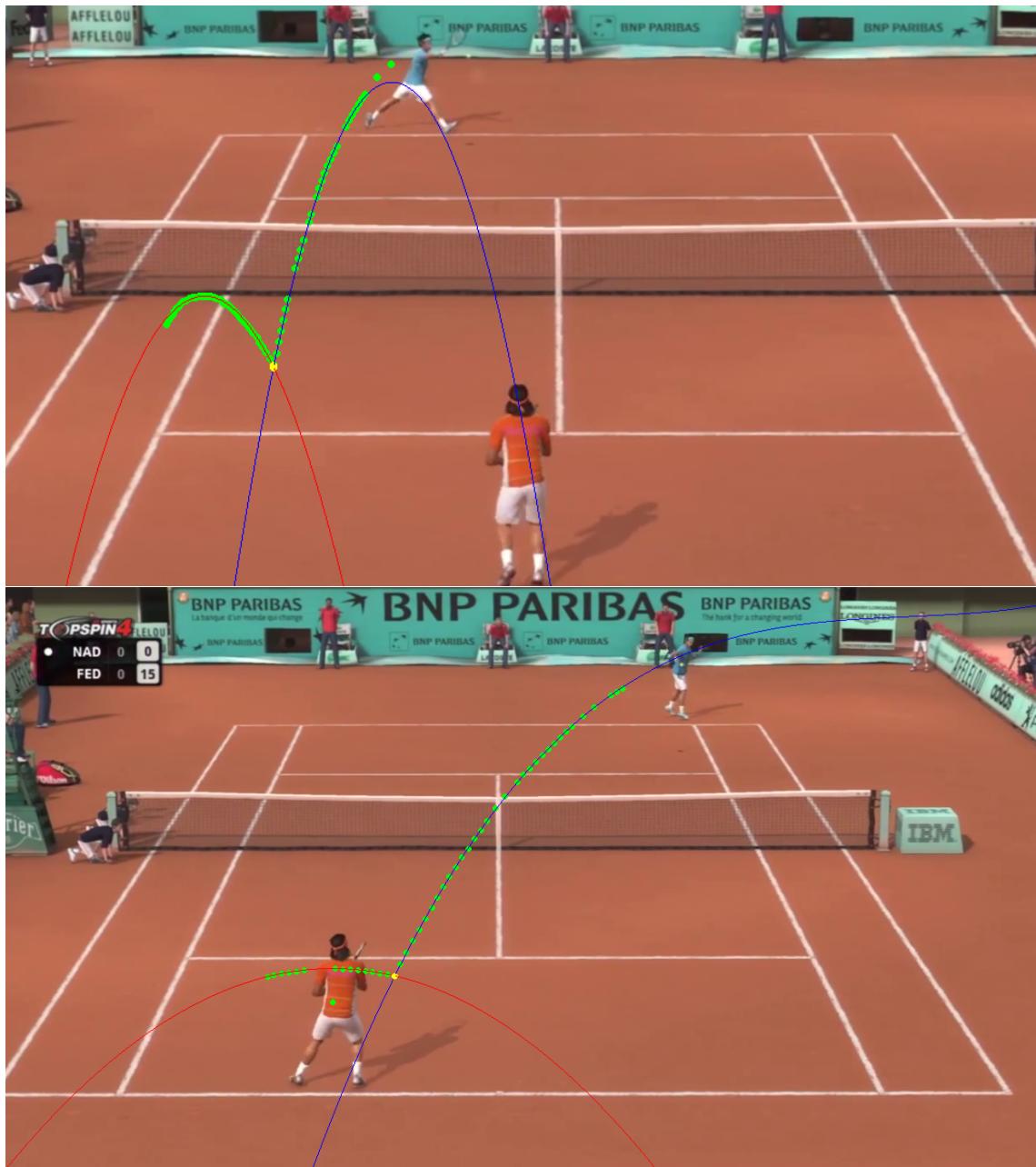


FIGURE 6 – En haut, un exemple avec un polynome de degré 2. En bas, un exemple avec un polynome de degré 3.

4 Travail en 3D :

4.1 Données

L'obtention de données satisfaisantes (la vidéo d'un match de tennis de deux points de vues différents) est difficile. Nous avons donc décidé d'utiliser une fois de plus des images de jeux vidéos. Encore une fois nous avons le problème de la caméra qui bouge ce qui va nous compliquer la tâche.

4.2 Paramètres des caméras

Encore une fois, nous sommes confrontés à un problème dû aux données. En effet, les ingénieurs de HawkEye Technology connaissent les positions des caméras ainsi que leurs caractéristiques intrinsèques. Il est donc possible pour eux, si ils détectent la balle, de connaître sa position dans l'espace par triangulation. Dans notre cas de caméras de caractéristiques inconnues et mobiles, nous allons devoir deviner ces caractéristiques. Une fois de plus, nous prenons avantage de la spécificité du problème, en effet, nous allons profiter de la connaissance à priori des mesures du terrain de tennis.

A l'aide d'un détecteur de lignes, nous allons repérer 8 coins du terrain. Cela permet d'avoir 8 points dont on connaît la position dans l'espace tridimensionnel. Suite à cela, il devient possible d'estimer la matrice de projection si on néglige les distorsions. Il faut donc trouver la matrice (3x4) qui projette le plus fidèlement les points 3D dans l'image. On peut pour cela définir une fonction de coût que l'on va chercher à minimiser.

4.3 Stéréovision

Nous avons donc à disposition deux images de la même scène, et nous connaissons les matrices de projection de chacune des caméras. On détecte alors la balle dans les deux images. Ainsi nous avons deux équations $m1 = P1 M$ et $m2 = P2 M$. Et l'on cherche le point M qui résout ces deux équations. En théorie il n'y qu'une solution mais dans le cas de plus que deux points de vues, il n'y a généralement pas de solutions. Il faut alors se donner une fonction de coût que l'on va chercher à minimiser.

4.4 Description du code :

Le fichier `lines_3d.py` définit des fonctions qui permettent de détecter les 8 coins du terrain.

Le fichier `3d_main.py` utilise des méthodes OpenCv pour calculer les matrices de projections des deux caméras puis différents tests sont menés. La méthode `calibrateCamera` d'OpenCv permet d'estimer directement la matrice de projection et les coefficients de distorsion de la caméra à partir d'un jeu de coordonnées. On mène alors différents tests pour vérifier les résultats.

- D'une part, on peut vérifier que la variable `rms` reste faible (entre 0 et 1), ce qui signifie que l'erreur de reprojection des points est faible.
- les coefficients de distorsion sont faibles
- on teste la reprojection d'un point pour vérifier
- enfin on teste la triangulation sur les coins du terrains. Les résultats sont approximatifs mais restent très cohérents.

Pourtant, la triangulation de la balle de tennis donne un résultat aberrant. Plusieurs facteurs peuvent l'expliquer :

- le manque de précision dans la détection de coins
- le trop faible nombre de points pour estimer la matrice de projection
- le fait que l'on néglige les coefficients de distorsions ?

On peut appliquer la fonction OpenCv "undistort" aux points avant de les passer à la triangulation (et c'est le procédé habituel), cependant, cela conduit à des résultats faux même sur la triangulation des coins du terrain tandis que ceux obtenus sans rectifier la distorsion sont cohérents.

Comme nous avions prévu de négliger les effets de distorsion et que les méthodes OpenCv ne proposent pas cette option, nous avons décidé de réimplémenter nous mêmes la recherche directe de la matrice de projection dans le fichier `3d_main_2.py`. On se donne donc une fonction de coût et cela revient donc à minimiser une fonction à 12 paramètres (la matrice de projection est de taille (3x4)). Une fois que l'on obtient les deux matrices de projections, on cherche alors de la même façon le point de l'espace qui minimise l'écart aux deux projections. Les résultats obtenus sont du

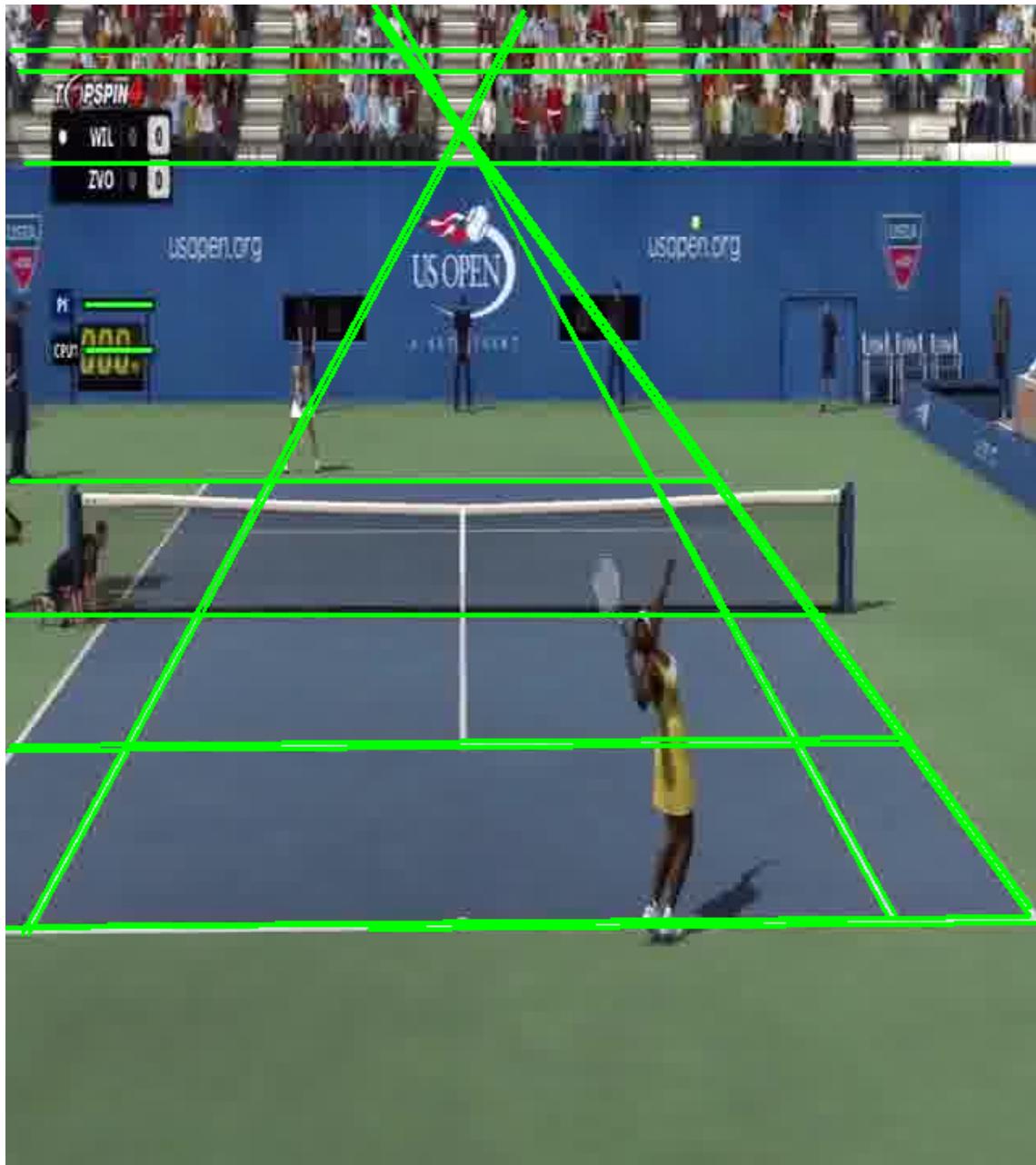


FIGURE 7 – détecteur de lignes

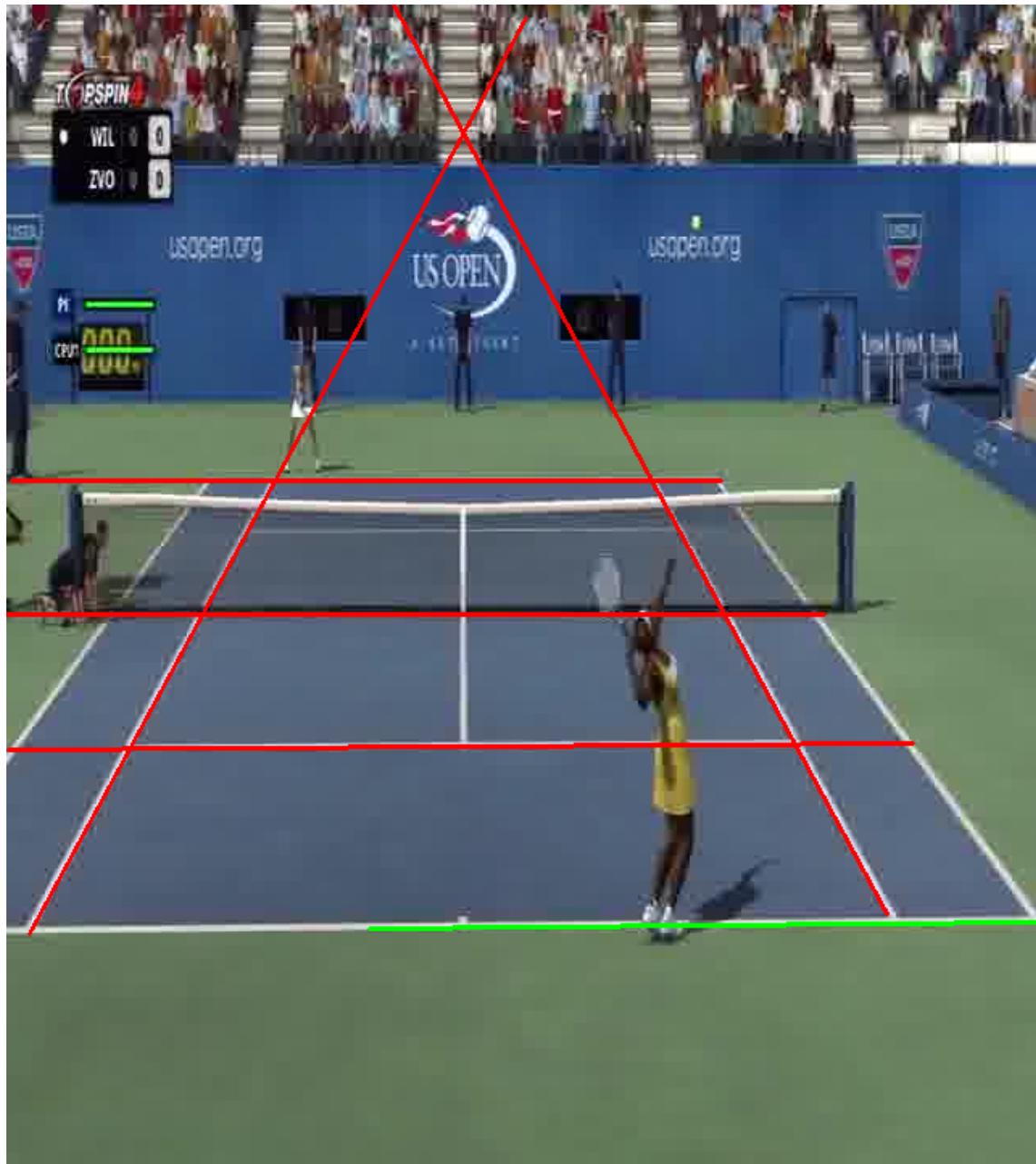


FIGURE 8 – sélection des lignes

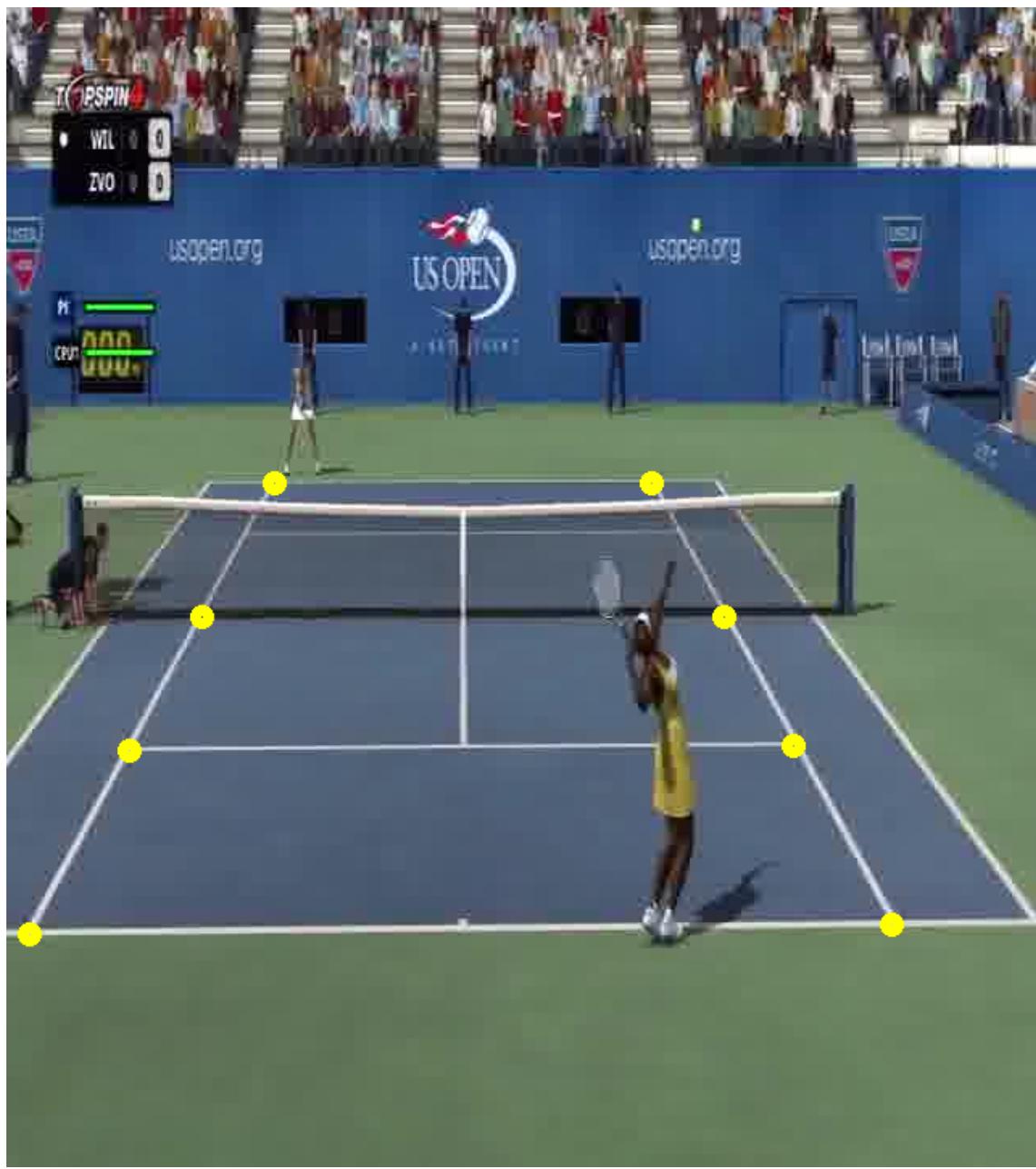


FIGURE 9 – calcul des coordonnées des coins

même ordre que ceux utilisant les méthodes OpenCv : les reprojections et triangulations sur les coins du terrains sont cohérentes, mais la triangulation sur la balle donne des résultats abhérents.

Cette partie du projet n'a donc pas abouti. Dans le cadre d'une suite à ce projet, il faudrait enquêter sur ce problème, par exemple en étudiant des solutions à des problèmes similaires.

5 Conclusion

Nous avons donc montré qu'il est possible de faire un logiciel qui arbitre le tennis avec une seule prise de vue du terrain, avec une précision satisfaisante, et il est probable que les résultats avec une prise de vue immobile soient encore meilleurs. Cela reste cepandant bien inférieure à la précision du système Hawkeye actuel. Enfin nous avons implémenté une méthode pour améliorer la détection de trajectoire grâce à une caméra à deux prises de vues et nous avons proposé des pistes d'exploration pour améliorer ces méthodes.

Bibliographie

Bonnet *Projection de l'espace tridimensionnel*, Université de Lille <http://www-lagis.univ-lille1.fr/bonnet/image/projection.pdf>

Documentation OpenCv https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_recons.html