MTI805 – Compréhension de l'image

Projet

Détection, classification et tracking de joueurs de football

Session	**Hiver 2017**
Groupe	**01**
Numéro du travail	**Projet de session**
Étudiants	**Etienne Dewaele** **Tony Duong**
Courriel des étudiants	**etienne.dewaele.1@ens.etsmtl.ca** **tony.duong.1@ens.etsmtl.ca**
Chargé de cours	**Luc Duong**
	luc.duong@etsmtl.ca

Sommaire

Introduction

Dans le cadre du cours MTI805, nous avons eu l'occasion de réaliser le projet de notre choix afin de mettre en pratique les notions de vision par ordinateur apprises en cours et lors des précédents laboratoires.

Pour notre revue de littérature, nous avons choisi d'étudier l'article "Automatic Player Detection, Labeling and Tracking in Broadcast Soccer Video" que nous avons présenté au reste du groupe. Le but de notre projet était donc de réaliser la détection, la classification et le tracking des joueurs de soccer sur une vidéo de match de football. Nous avons pensé que l'analyse des données issues d'un tel projet pourrait avoir de nombreuses applications utiles dans le domaine sportif comme la mise en place de stratégie d'équipe, l'analyse des performances de joueurs, ou encore l'aide à l'arbitrage. C'est ce qui nous a poussé à nous intéresser à ce sujet.

Ce rapport présente notre implémentation d'un algorithme tentant d'atteindre cet objectif. Nous parlerons des méthodes que nous avons utilisés pour chaque partie du projet, tout en expliquant nos divergences avec l'article étudié. Nous présenterons et analyserons les résultats obtenus pour conclure sur les améliorations possibles de notre algorithme.

Vue d'ensemble du projet

Objectifs

Pour ce projet, nous nous sommes fixés plusieurs objectifs. Les détails algorithmiques et méthodiques seront explicités dans les prochaines sections.

Tout d'abord, la première étape à réaliser a été de segmenter une image issue d'un match de football diffusé à la télévision. En effet, seule la partie du terrain nous intéresse.

Puis, une fois le terrain segmenté, la tâche suivante consisterait à détecter les joueurs et l'arbitre sur le terrain, toutes équipes confondues. Cette tâche est primordiale pour les deux objectifs suivants.

En effet, après avoir détecté les joueurs, nous souhaitons les classifier selon leurs équipes. L'arbitre serait classifié en tant que classe « Arbitre ».

Enfin, le suivi des joueurs constitue notre dernière tâche afin de ne pas se reposer uniquement sur la détection de joueurs. Ce suivi permettrait idéalement de connaitre la trajectoire de chaque joueur.

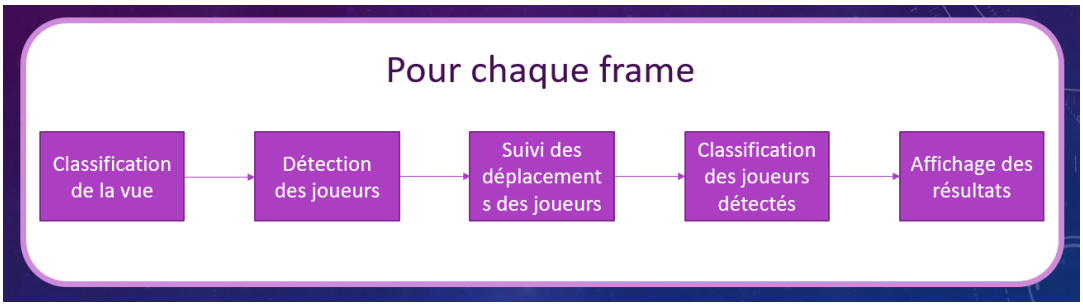
Environnement de travail

L'application du projet a été développé sous l'IDE Visual Studio dans le langage C++. La librairie OpenCV a bien sûr été utilisé.

Workflow de l'application

Voici un framework du système tel qu'on l'avait établi lors de la revue de littérature. La seule étape omise pour notre projet est celle de la classification de la vue. Nous remarquons bien sur la vidéo qu'il y a différents types de vues : bords de terrain, centre, zoom-in sur les joueurs, vues des présentateurs, public....

Dans l'article qu'on a étudié, chaque type de vue était classifié. Dans notre projet, nous avons préféré tester notre algorithme uniquement sur des vues de jeu pour ne pas avoir à gérer toute cette partie de classification des vues qui est moins intéressante que la détection des joueurs en elle-même.



I. Segmentation du terrain

Avant de directement aller à l'étape de détection des joueurs, une étape de segmentation du terrain a été nécessaire. En effet, un système de détection n'est jamais parfait et cette étape bien que simple, permettrait lors de la détection de limiter le nombre d'erreurs de détection.

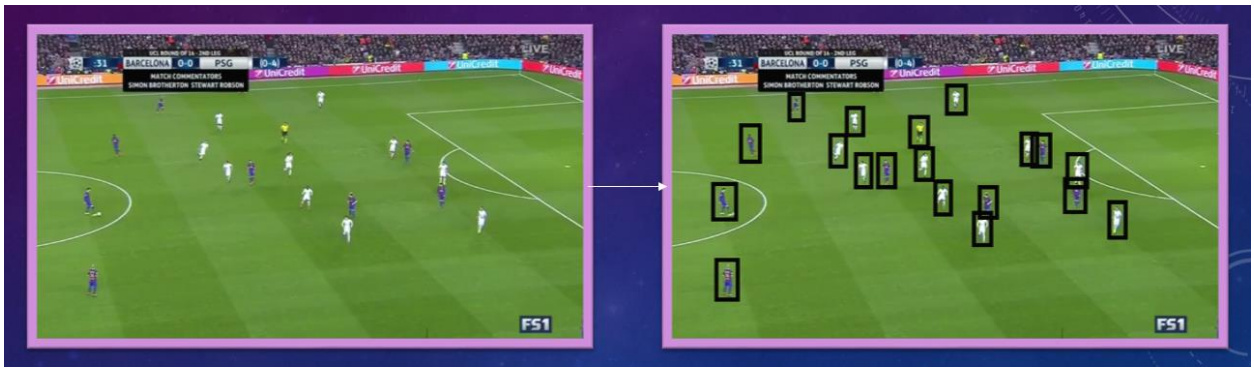
Les résultats attendus

Nous souhaitons donc conserver seulement la partie du terrain, celle qui nous intéresse le plus pour la détection des joueurs et de l'arbitre. Celle-ci se démarque par la couleur dominante verte qui s'y trouve. Pour seulement conserver les pixels d'une couleur, OpenCV fournit la fonction *inRange()* qui permet justement de générer une image binaire de même dimension que l'image originale où un pixel blanc à la position (*x,y*) signifie que le pixel de l'image originale à la position (*x,y*) correspond à la couleur voulue (entre deux bornes de couleurs HSV). Mais afin de conserver également les joueurs dans cette zone, il a été nécessaire d'utiliser les fonctions d'érosion (*erode*) et de dilatation (*dilate*) de OpenCV. Ci-dessous, nous pouvons voir l'image originale, le masque généré par couleur dominante et l'image résultante après application du masque sur l'image originale.

II. Détection de joueurs

Maintenant, il est nécessaire avant toute chose de détection les joueurs et l'arbitre sur le terrain.

Les résultats attendus



Méthodes et techniques employées

Pour la détection, nous avons choisi de suivre la méthode proposée par l'article qui est une analogie à la méthode de détection proposée par Viola-Jones. Nous avons donc décidé d'entraîner notre propre classifieur en suivant la méthode de « Cascade Classifier Training » proposée par OpenCV.

Pour réaliser ce classifieur, il nous fallait évidemment un jeu de données composé d'images négatives et positives de joueurs. Nous nous sommes basés sur le match PSG – Barcelone du 8 Mars 2017 en 854x480 pixels et nous avons extrait manuellement 260 images positives et 520 images négatives. Nos précédents tests avec un vidéo de moins bonne qualité et avec un nombre moins élevé d'images étaient peu concluants. En effet, beaucoup d'erreurs de détection (faux positifs et absence de détection) se produisaient.

Notre méthode de choix des positifs consistait à extraire un nombre égal d'échantillons de joueurs de chaque équipe (soit 120 par équipe), et un total de 20 échantillons (ou *sample*) d'arbitres. Nous avons tenté de prendre des poses variées des joueurs dans différents cas de figures (occlusions avec les lignes du terrain par exemple).

Nous nous sommes basé sur la documentation d'OpenCV pour connaître la marche à suivre pour l'entrainement :
http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html.

Exemples d'images positives et négatives extraites

Une fois le classifieur généré (sous forme de fichiers XML), il peut enfin être utilisé pour la détection.

Résultats

Nous avons fait le test sur une séquence de 10 secondes tiré aléatoirement du match. Les résultats obtenus sont les suivants.

Frame	Nombre de joueurs détectés	Total de joueurs presents sur l'image	Détectés cumulatifs
1	5	17	5
2	8	17	9
3	7	17	9
4	6	17	9
5	9	17	10
6	6	17	11
7	10	16	11
8	10	16	11
9	8	16	11
10	9	16	11
11	8	16	12
12	7	16	12
13	10	17	12
14	8	17	12
15	9	17	12

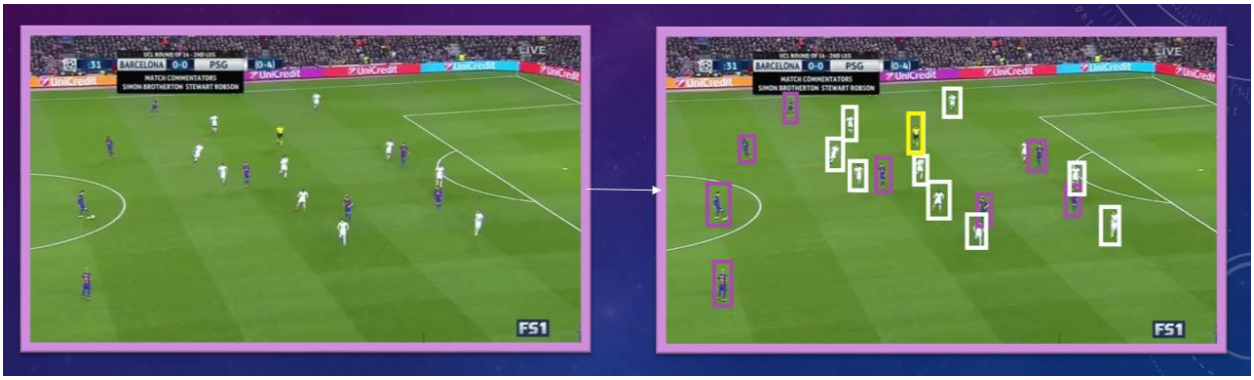
16	10	17	14
17	10	17	14
18	10	17	14
19	**11**	17	14
20	6	17	14
21	7	17	14
22	7	17	15
23	8	17	16
24	6	16	16
25	7	16	16

Comme nous pouvons le remarquer, le système ne détecte jamais tous les joueurs d'un même frame. Néanmoins, si l'on observe la colonne des « détectés cumulatifs », nous remarquons qu'en l'espace d'une seconde (ou 25 frames), l'ensemble des joueurs est détecté. Par la suite de ce rapport, nous vous expliquerons comment on a tiré avantage de ce trait pour conserver la détection des joueurs.

III. Classification

L'étape de classification arrive donc après celle de détection. L'objectif est de classifier les joueurs détectés.

Résultats attendus



Méthodes et techniques employées

Ce problème posé correspond parfaitement à un problème de *clustering*. En effet, il s'agit de regrouper des données similaires entre elles (ici, les joueurs de même équipe). Les caractéristiques déterminées sont très simples : les couleurs. En effet, le système va classifier les joueurs selon leur couleur. L'espace de couleur considéré dans notre cas est l'espace RGB.

La méthode de *clustering* utilisé est la méthode *EM* (ou *Expectation-Maximisation*).

Notre méthode d'extraction des caractéristiques de couleur diverge quelque peu avec celle proposée de l'article étudié. En effet, comme dans la méthode proposée par l'article, on enlève l'arrière-plan par soustraction de couleur dominante (qui est le vert). Il nous reste donc seulement théoriquement que les pixels correspondant au joueur à traiter.

L'article proposait de segmenter encore une fois le joueur pour ne garder que la partie du haut du corps. Les pixels restants ne correspondraient donc qu'à cette partie de l'image du joueur détecté.

Dans notre cas, nous faisons simplement une moyenne de couleur RGB de tous les pixels du joueurs (donc maillot + short + chaussures normalement), les couleurs dominantes étant celles du maillot et du short, les valeurs moyennes de couleur RGB des joueurs d'une même équipe devraient être similaires. Les caractéristiques de notre vecteur d'entrée dans l'algorithme EM sont alors les trois valeurs R, G, B moyennes.

Il y a eu tout d'abord une phase d'entrainement avec les mêmes images de joueurs que l'on avait extraites manuelles pour la détection. A l'issue de cet entrainement en ressort un classifieur permettant de classifier chaque personne sur le terrain en trois classes : Equipe A, Equipe B ou Arbitre.

Résultats

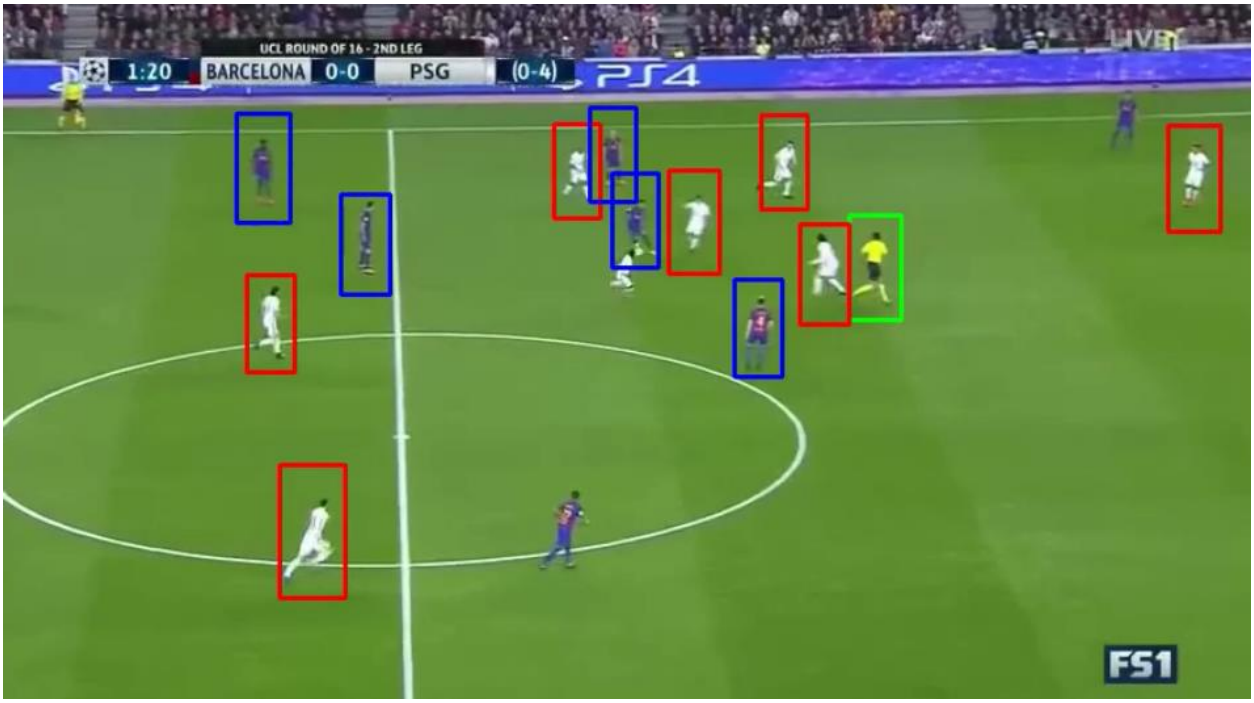
D’après le tableau suivant, nous pouvons remarquer un haut taux de précision autour de 90%. Les résultats suivants proviennent de la même séquence qui a été utilisée dans la partie précédente « Détection ».

Les erreurs provenaient des occlusions qui survenaient lorsqu’une partie de l’arbitre était dans l’image du joueur considéré par exemple.

Utiliser l’espace de couleur HSV aurait sûrement été une meilleure option et aurait produit de meilleurs résultats.

Correct	Total	% Precision
4	4	1
7	8	0.875
8	8	1
10	10	1
9	10	0.9
10	12	0.83333333
11	12	0.91666667
13	13	1
13	13	1
13	14	0.92857143
12	12	1
9	10	0.9

Dans l’image ci-dessous, nous pouvons remarquer les résultats affichés sur une frame.

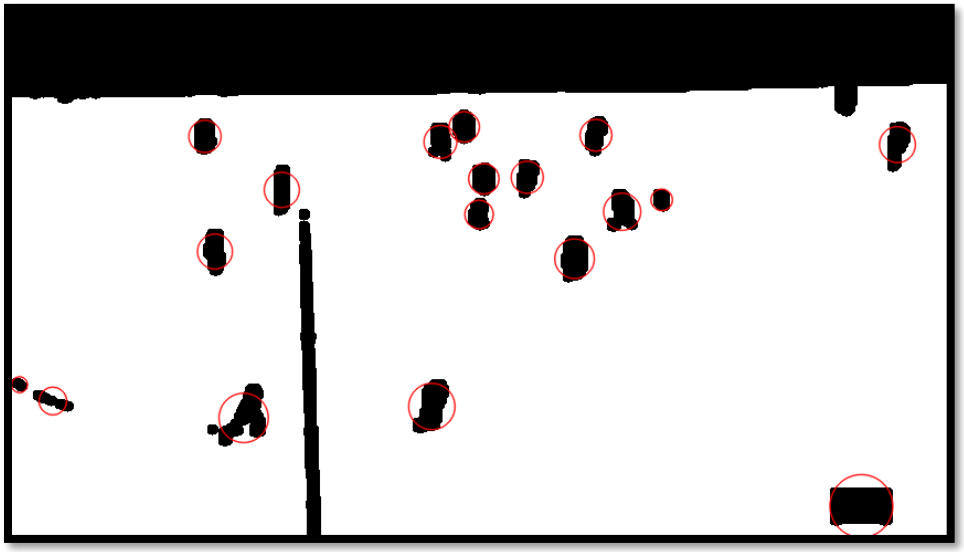

UCL ROUND OF 16 - 2ND LEG
1:20
BARCELONA
0-0
PSG
(0-4)
PS4
FS1

IV. Suivi des joueurs (tracking)

Nous avons vu précédemment que notre algorithme de détection ne parvenait pas à détecter tous les joueurs dans une même frame. En effet, à cause de la variation des poses surtout, le système peut ne pas pouvoir détecter un joueur entre deux frames.

Nous avons alors pensé à un algorithme permettant de conserver cette détection. Si un joueur a été détecté auparavant, il sera conservé en tant que « détecté » par le système. Pour cela, nous avons utilisé les « blobs ». Comme pour la section « Segmentation », nous allons utiliser la fonction *inRange()* pour ne garder que les pixels de teinte verte sans utiliser les fonctions d'érosion et de dilatation.

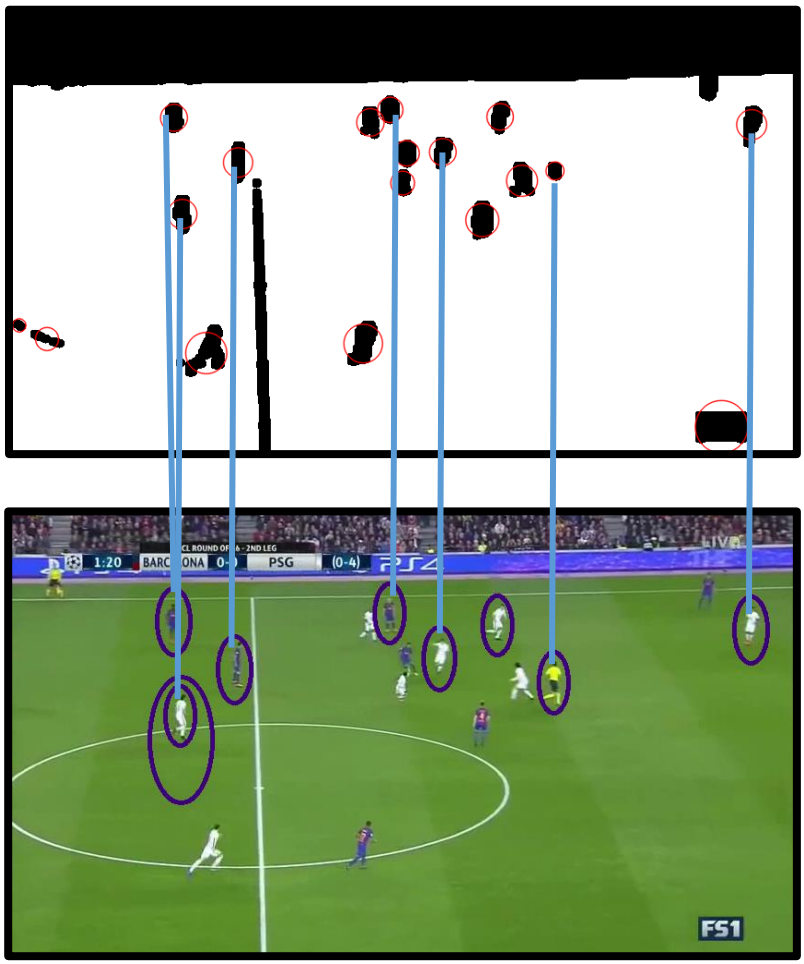
Des blobs vont naturellement se former et OpenCV fournit une fonction permettant de détecter des blobs comme ceux-ci. Dans l'image suivante, il est possible de voir les blobs détectés et jugés intéressantes pour le tracking entourés d'un cercle rouge.



Pour chaque joueur détecté, un blob lui est relié selon la distance qui les sépare (inférieur à un seuil déterminé). Nous obtenons donc à l'issue de cette phase de couplage des couples (blob, joueur détecté).

A l'itération suivante (frame suivant donc), pour combler la détection manquante de certains joueurs, les couples (blob, joueur détecté) du frame précédent sera utilisé. Plus précisément, si un joueur n'est plus détecté au frame suivant, il est tout de même présent dans les couples (blob, joueur détecté). De ce fait, avec l'image des blobs du frame suivant où chaque blob représente potentiellement un joueur, il est possible d'associer chaque blob du frame précédent à un blob du frame suivant (sauf si le joueur est sorti du champ de vision). Alors le couple (blob, joueur) est mis à jour avec les nouvelles positions du blob.

Nous partons de la supposition qu'un blob par joueur sera toujours présent. En effet, nous pouvons remarquer que certains blobs ne correspondent absolument pas à des joueurs comme le rectangle en bas à droite. Mais ce n'est pas dérangeant car aucun joueur ne sera détecté dans cette zone. De plus, le seuil déterminé est très petit pour éviter les mauvaises correspondances entre les blobs et les joueurs détectés.



V. Améliorations possibles

Nous avons évalué un certain nombre de pistes d'améliorations pour notre algorithme qu'on aurait pu mettre en place avec plus de temps.

Segmentation

Il existe un problème au niveau de la segmentation. En effet, les joueurs se situant proche de la limite supérieure du terrain voient leur corps se « couper en deux », après application du masque comme vu dans la section « Segmentation ». Par conséquent, à l'étape suivante, ces joueurs « coupés partiellement » ne seront pas détectés. Une solution possible pour remédier à ce problème de segmentation serait d'augmenter la surface du masque pour pouvoir couvrir les joueurs entièrement.

Détection

On remarque bien que les résultats sont meilleurs sur les vues larges.

De plus, on aurait pu gagner en performance en ayant un plus grand nombre d'images ou des images plus variées pour couvrir un maximum de situations dans notre jeu de données.

Classification

On aurait pu utiliser l'espace de couleur HSV au lieu de RGB pour la classification. En effet, contrairement à RGB, HSV sépare la luminance et la chrominance. En vision par ordinateur, cette séparation est très pratique pour plusieurs raisons comme la robustesse face aux changements de luminosité. Dans notre situation, en utilisant RGB, l'application a su classifier de manière efficace car les variations en termes de luminance étaient très faibles. Les couleurs des maillot d'une même équipe se ressemblaient beaucoup.

Tracking

Notre algorithme de tracking s'est basé sur la détection de blobs, nous aurions pu cependant utiliser d'autres méthodes plus performantes et reconnues. Parmi celle-ci, nous pensons à la méthode par filtre de particules, ou encore la méthode du filtre de Kalman qui prend en compte des éléments comme le vecteur vitesse, la direction, et permet de générer une prévision relativement probable.

Pour l'affichage du tracé, il n'a pas été possible du fait que le mouvement de la caméra n'a pas été pris en compte. Nous aurions alors pu détecter le mouvement de la caméra et en déduire un vecteur de translation pour connaître le mouvement réel du joueur et sa trajectoire dans la vidéo.

Une des améliorations possibles pour notre programme aurait aussi porté sur la performance. En effet, même si les résultats sont là, le temps de traitement d'une image est de plusieurs secondes ce qui ne permet pas la vision du résultat instantané. Si on voulait avoir une

vision fluide des résultats, il faudrait traiter la vidéo en amont ce qui est déplaisant. Améliorer la gestion de la mémoire dans notre programme et trouver un algorithme plus performant permettrait de réduire ce temps de traitement.

Conclusion

Nous pouvons affirmer que notre application est fonctionnelle et que nos objectifs en détection, classification et tracking ont été atteints avec des résultats acceptables, même si l'algorithme a de nombreuses pistes d'amélioration possibles. La performance est en effet à revoir, il faudrait donc améliorer la gestion de la mémoire, mettre en place tracking plus robuste aux occlusions, et éventuellement prendre en compte le mouvement de la caméra. D'autres méthodes plus efficaces existent notamment pour le tracking et la classification, nous estimons donc que notre implémentation du problème n'est pas forcément la meilleure dans toutes les parties du projet.

Nous avons trouvé ce dernier intéressant car il nous a permis de nous intéresser au sujet de notre choix ce qui était vraiment motivant. Nous avons appris en faisant des erreurs et en cherchant diverses méthodes d'implémentations du problème. Notre utilisation de la librairie OpenCV nous a permis d'acquérir des connaissances sur des techniques d'apprentissage automatique comme SVM ou K-Means. Nous avons également découvert comment réaliser nos propres classifieurs et concevoir notre propre algorithme de suivi des joueurs.

Ce projet pourrait éventuellement être un point de départ pour diverses applications dans d'autres domaines sportifs similaires en termes de vue en étant adapté convenablement, on pense notamment au rugby ou au football américain.