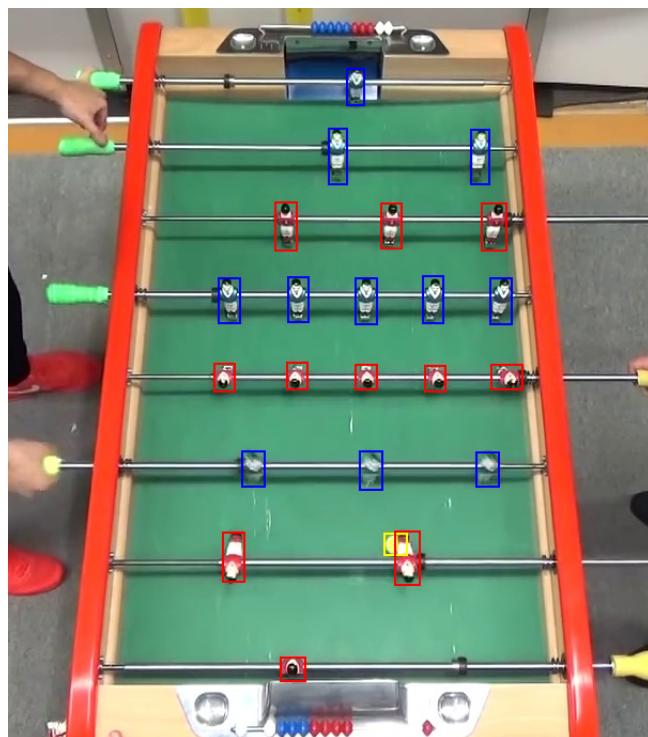


# Projet 3A InfoNum : Extraction des statistiques en temps réel de matchs de baby-foot

Alexandre Gautier, Guillaume Dugat et Vivien Conti

Avril 2023



*Client : Clément Vaes  
Encadrant école : Céline Hudelot*

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>État de l'art</b>	<b>3</b>
2.1	Projets similaires . . . . .	3
2.2	Modèles d'IA de détection d'objets . . . . .	4
2.2.1	Two-stage-detectors VS One-stage-detectors . . . . .	4
2.2.2	Techniques basées sur des modèles transformers . . . . .	8
<b>3</b>	<b>Structure Générale</b>	<b>10</b>
3.1	Phase d'entraînement . . . . .	11
3.2	Phase d'inférence . . . . .	11
<b>4</b>	<b>Implémentation</b>	<b>11</b>
4.1	Description de notre problématique . . . . .	11
4.2	Reconnaissance du terrain . . . . .	12
4.3	Reconnaissance de la balle . . . . .	16
4.4	Reconnaissance des joueurs . . . . .	17
4.5	De l'image 2D au terrain 3D . . . . .	20
4.6	Implémentation des statistiques . . . . .	22
4.7	Présentation de la pipeline . . . . .	25
<b>5</b>	<b>Conclusion</b>	<b>26</b>

# 1 Introduction

Les domaines du Deep Learning et de la Computer Vision ont émergé très rapidement un peu avant les années 1990 et notamment grâce à Yann Lecun dans [8] où il montre la capacité d'un réseau de neurones à reconnaître les chiffres écrits à la main. Depuis, le secteur de la computer vision est en pleine expansion, car il permet de répondre à un grand nombre de problématiques intéressantes et souvent complexes.

Par exemple, si l'on s'intéresse à la retransmission de rencontres sportives à la télévision, les statistiques de cette rencontre sont calculées par des algorithmes. En effet, considéré comme long et fastidieux, ce travail a été automatisé permettant aussi des calculs plus précis. Le projet s'inscrit ainsi dans cette même problématique où notre but est d'implémenter un algorithme permettant de calculer et d'afficher les statistiques d'un match de babyfoot en temps réel.

Pour cela, nous résumerons tout d'abord l'état de l'art des projets similaires au nôtre qui nous ont guidés et inspirés ; puis nous présenterons un état de l'art des modèles de détection d'objets en Intelligence Artificielle. Ensuite, nous expliquerons notre démarche et justifierons les techniques de détection que nous avons utilisées en accord avec l'état de l'art, mais également en accord avec notre problématique d'avoir un algorithme qui tourne en temps réel. Cette contrainte importante va beaucoup conditionner nos choix dans l'intégralité du projet. Enfin, nous discuterons des statistiques implémentées.

## 2 État de l'art

### 2.1 Projets similaires

Plusieurs projets sur l'extraction de statistiques de matchs de babyfoot existent déjà, et nous avons compilé et résumé leurs principales caractéristiques dans le tableau suivant 1. Il est important de noter que parmi ces projets, ceux qui n'impliquent pas de babyfoot modifié pour l'occasion (avec par exemple un terrain en verre pour installer une caméra en dessous), ne sont pas des projets aboutis. Notre projet est donc tout à fait pertinent.

Projets	TableSoccerCV	Foosballytics	Bambach Lee	Stryker	Foosball	Automated Foosball Table
Description	Repository Github	Article	Article Balle, joueurs (position et rotation)	Article	Repository Github Balles, terrain, joueurs (position et rotation)	Rapport de projet balle
Détection	Balle, terrain	Balle, terrain		Balle		
Statistiques	Score, vitesse balle, heatmap	Score	-	Score, vitesse balle, possession	Score	Vitesse balle, prédiction balle
FPS Caméra	-	60	-	-	-	120
FPS Processing	-	56	27	-	-	13
Hardware	-	-	GPU	Raspberry Pi	-	CPU
Modèles	Détection couleur, marqueurs	Détection couleur	Détection couleur	-	Détection couleur, marqueurs	Détection couleur
Software	Python, OpenCV	Python, OpenCV	Matlab, CV toolbox	Python, OpenCV	C++, OpenCV	-

TABLE 1 – Comparaison de projets similaires

## 2.2 Modèles d'IA de détection d'objets

En Computer Vision, on retrouve dans la littérature différentes méthodes pour faire de la détection d'objet (Figure 2). Les deux plus connues sont notamment celles basées sur les two-stage-detectors (comme les R-CNN [9]) et les one-stage-detectors (comme YOLO par exemple [14]). Ces deux types de détecteurs ne fonctionnent pas de la même manière malgré certaines similarités et nous allons donc les mettre en regard pour les comparer et les comprendre.

Cependant, depuis les années 2020, de nouvelles techniques ont émergé en se basant sur des modèles transformers et nous essaierons donc ici de proposer un aperçu de ce type de technique en se basant notamment sur l'analyse du fonctionnement de l'algorithme DETR développé par Facebook AI [2].

### 2.2.1 Two-stage-detectors VS One-stage-detectors

Les détecteurs two-stage-detectors et one-stage-detectors ont tous les deux pour but d'identifier des zones sur une image puis d'identifier si un objet est présent dans cette zone pour enfin le classifier. Les two-stage-detectors réalisent ces tâches en deux étapes tandis que les one-stage-detectors peuvent les réaliser en une seule étape. Cette différence est illustrée sur la Figure 3.

Tout d'abord, les two stage detectors utilisent un backbone, c'est à dire un réseau pré-entraîné pour la reconnaissance d'objets, pour extraire les différentes features caractéristiques de l'image. On remarque bien que la principale différence réside dans l'inclusion d'un réseau permettant d'identifier des régions d'intérêt dans les two-stage-detectors. Cette détection est réalisée à partir d'un Region Proposal Network (RPN). Le but du RPN est donc de trouver

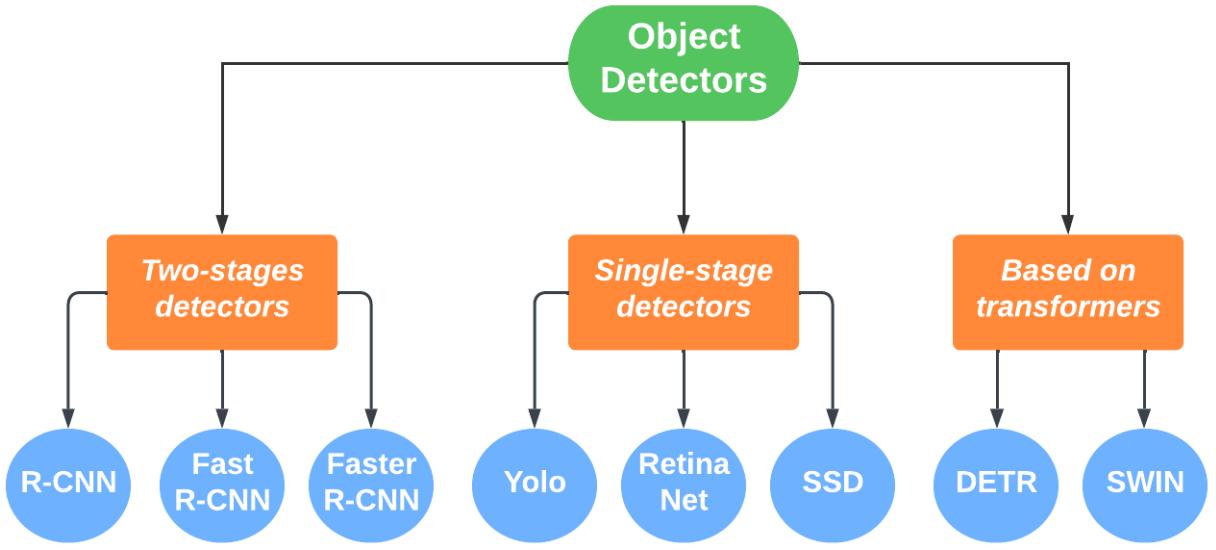


FIGURE 2 – Schéma des différents modèles d’IA de détection d’objet sur des images [9], [4], [15], [14], [11], [12], [2], [13]

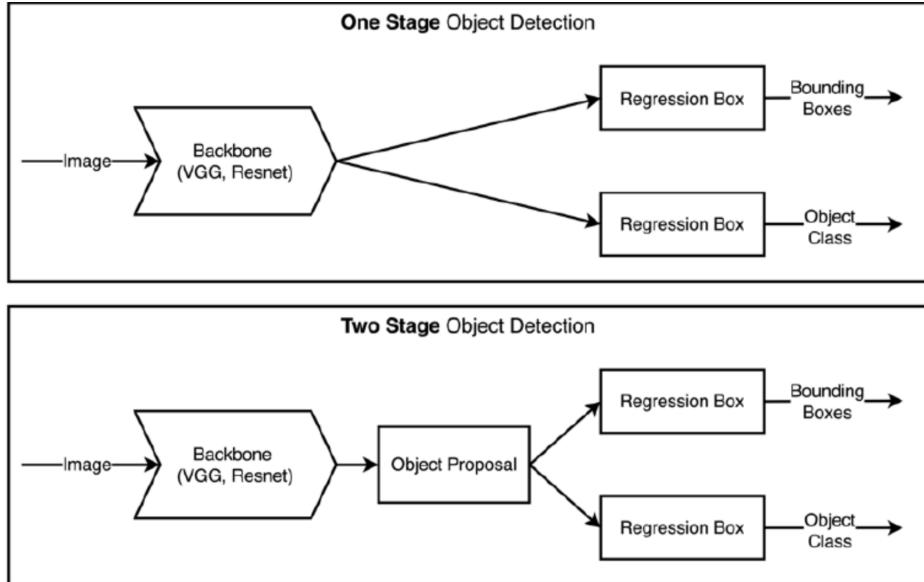


FIGURE 3 – Schéma des différentes étapes pour un one stage detector et un two stage detector

des régions (sous forme de bounding boxes) qui contiennent un objet. Pour entraîner ce réseau la loss qui a été minimisée est la suivante :

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Avec  $i$  étant l’indice de la bounding box et  $p_i$  la probabilité qu'il y ait un objet sur dans

la bounding box  $i$ . La probabilité  $p_i^*$  correspond à la vérité terrain et vaut 1 si un objet est présent et 0 sinon. Le terme  $t_i$  est un vecteur de dimension 4 représentant les coordonnées de la bounding box et  $t_i^*$  correspond aux coordonnées de la box de vérité terrain. Finalement on obtient une probabilité de présence d'objet sur chacune des régions d'intérêt et les coordonnées de la bounding box. Enfin, grâce à un classifieur il est possible d'identifier la classe de l'objet appartenant à la région d'intérêt. Cette méthode de détection d'objet est notamment caractéristique des algorithmes tels que R-CNN [9] et est résumée sur la Figure 4.

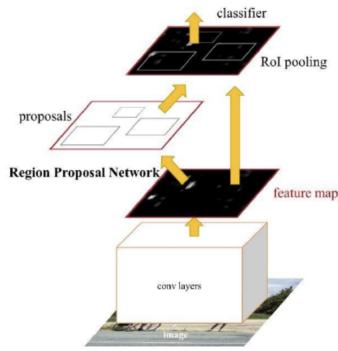


FIGURE 4 – Architecture d'un two stage detector

Au contraire, les one-stage-detectors réalisent l'ensemble du processus de détection dans un seul réseau, sans nécessiter de réseau d'identification de régions d'intérêt. C'est à dire que la position des bounding boxes et les classes des objets détectés sont calculées en même temps. L'idée est de découper l'image en une grille de cellules de taille  $N \times N$  et de prédire sur chaque cellule si un objet est présent ainsi que sa position et sa classe.

Tout d'abord, la structure de l'algorithme se base sur un backbone (identiquement aux two stage detectors). A partir des feature maps extraites par les couches convolutives du backbone, une nouvelle couche convulsive est apprise pour repérer quelle cellule possède le centre d'un objet. Cette grille sera responsable de produire la bounding box correspondante à l'objet. On cherche ensuite à calculer la probabilité de présence d'un objet sur la cellule, la classe à laquelle appartient l'objet et les coordonnées de la bounding box l'englobant (représentées par quatre réels pour identifier le centre, la longueur et la largeur). Une nouvelle couche convulsive doit donc être apprise pour identifier chacun des différents paramètres cités. Comme nous pouvons le voir sur la Figure 5, la sortie de cette couche, pour chacune des cellules de l'image, sera la superposition de  $5 + C$  feature maps avec  $C$  le nombre de labels

possible.

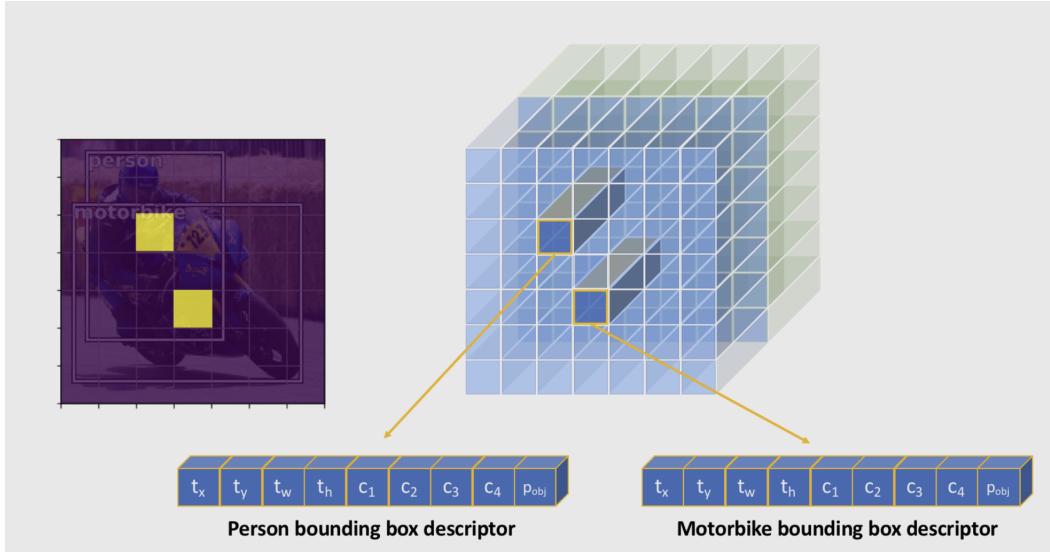


FIGURE 5 – Schéma de la structure d’ensemble

Ainsi, il est possible d’identifier une bounding box par cellule. Évidemment, il est aussi possible d’identifier plusieurs bounding boxes par cellule puisque plusieurs objets peuvent appartenir à la même cellule. Pour ce faire, il suffit de choisir le nombre  $B$  de bounding boxes que l’on souhaite détecter puis modifier les paramètres de la couche convulsive pour obtenir une sortie de taille  $B(5 + C)$ . Plusieurs bounding boxes étant calculées par cellule, il n’est pas rare d’obtenir des bounding boxes différentes pour un même objet. C’est ici qu’il est utile d’avoir pré-déterminé la grille possédant le centre de l’objet. D’autres méthodes sont également possibles. Par exemple, il est naturel de vouloir sélectionner la meilleure bounding box parmi toutes celles trouvées. Une technique classique pour résoudre ce problème est la non-maximum suppression. Elle a pour but de conserver la bounding box avec le taux de confiance le plus élevé parmi les bounding boxes qui s’intersectent.

Le calcul de ce taux de confiance dépend évidemment de la métrique utilisée. De nombreux algorithmes (comme YOLO par exemple) utilisent l’Intersection Over Union (IoU). L’idée est intuitive et consiste à calculer le rapport entre la surface représentée par l’intersection de la bounding box prédite et la vérité terrain et la surface représentée par leur union (Figure 6). Ainsi le taux de confiance pour chaque bounding box est défini par :

$$taux\_confiance = P(Class_i) \times IOU_{pred}^{truth}$$

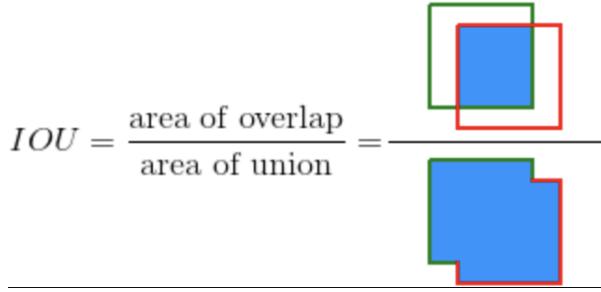


FIGURE 6 – Définition de l’IoU

### 2.2.2 Techniques basées sur des modèles transformers

Malgré les très bonnes performances des modèles présentés précédemment, de nouvelles techniques ont notamment émergées dans les années 2020 et sont basées sur l’utilisation de modèles transformers. Nous détaillerons cette partie en nous concentrant sur l’exemple de l’algorithme DETR [2]. Les modèles transformers sont beaucoup utilisés dans le domaine du NLP du fait de leur structure récurrente permettant de prendre en compte le contexte dans une phrase par exemple. Cependant leur application dans le domaine de la Computer Vision était relativement limitée. En revanche, ils permettent aujourd’hui d’obtenir des performances similaires voire meilleures sur la détection de certains types d’objets.

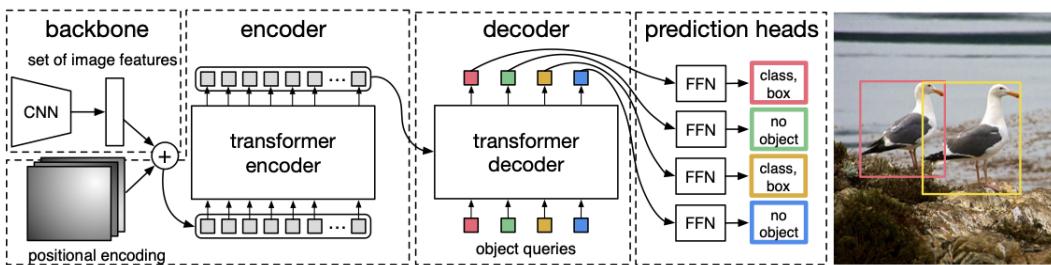


FIGURE 7 – Architecture DETR

L’architecture de DETR est constituée de trois composants principaux (Figure 7) ; un backbone, un transformer encodeur décodeur et un Feed Forward Network (FNN) pour effectuer la prédiction finale.

Le backbone a le même rôle que dans les modèles two-stage-detectors et one-stage-detectors. C’est-à-dire qu’il est entraîné à la reconnaissance d’objet, mais permet aussi d’obtenir une représentation de l’image en dimension plus faible. En effet, si l’image  $x_{img} \in \mathbb{R}^{3 \times H_0 \times W_0}$  alors le backbone génère une carte d’activation  $f$  de taille  $f \in \mathbb{R}^{C \times H \times W}$ . Dans le cas de l’algorithme

DETR on a  $C = 2048$ ,  $H = \frac{H_0}{32}$  et  $W = \frac{W_0}{32}$ .

L'encodeur permet, à l'aide d'une convolution de taille  $1 \times 1$ , de réduire la dimension  $C$  des channels en une dimension plus petite  $d$ . La carte d'activation  $z_0$  en résultant est  $z_0 \in \mathbb{R}^{d \times H \times W}$ . De plus, chaque couche de l'encodeur est composée d'un module de self-attention et d'un réseau FFN. L'encodeur transforme alors l'input en une séquence de  $N$  embeddings de taille  $d$ , c'est ce que l'on appelle les "object queries". Cela signifie que chacun des embeddings correspond à une "object query" et qu'il sera responsable de la détection d'un objet sur l'image.

Le décodeur est aussi composé de modules de self-attention et a pour but de décoder cette séquence de  $N$  embeddings de taille  $d$ . La particularité du décodeur est qu'il est capable de décoder les  $N$  embeddings en parallèle. Les  $N$  "object queries" sont donc transformées en embeddings par le décodeur puis utilisées par le FNN pour établir la prédiction finale. Celle-ci est donc composée de  $N$  prédictions puisque l'on a une prédiction par "object query".

La prédiction finale du FNN est réalisée par un perceptron à 3 couches avec une fonction d'activation ReLU ainsi qu'une couche de projection linéaire. Le FFN prédit les coordonnées normalisées du centre, la hauteur et la largeur de la bounding box. La couche de projection linéaire, tant qu'à elle, prédit la classe de l'objet à l'aide d'une fonction softmax. Pour pouvoir détecter l'ensemble des objets présents sur une image,  $N$  est généralement beaucoup plus grand que le nombre réel d'objets dans une image. De plus, une classe spéciale ayant pour label "no object" est utilisée pour représenter le fait qu'aucun objet ne soit détecté à l'intérieur d'une bounding box.

La fonction de loss utilisée dans un DETR est particulière puisqu'elle produit un matching biparti optimal entre les objets prédits et la vérité terrain. On note  $y$  l'ensemble des objets de la vérité terrain et  $\hat{y}$  l'ensemble des  $N$  prédictions.  $N$  étant plus grand que le nombre d'objets à détecter, l'ensemble de vérité terrain est augmenté par des labels "no object" (ou  $\emptyset$ ) pour être également de taille  $N$ . Pour trouver le matching biparti optimal, il faut chercher quelle est la permutation  $\hat{\sigma}$  des  $N$  éléments qui a le coût le plus faible :

$$\hat{\sigma} = \arg \min_{\sigma} \sum_{i=1}^N \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$$

Le coût de matching  $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$  prend non seulement en compte la prédition de la classe de l'objet mais aussi la prédition de la bounding box. L'ensemble des éléments de l'ensemble de vérité terrain sont caractérisés par  $y_i = (c_i, b_i)$  où  $c_i$  représente la classe cible (qui peut être  $\emptyset$ ) et  $b_i \in [0, 1]^4$  un vecteur définissant les coordonnées de la bounding box (coordonnées du centre, largeur et hauteur). En définissant  $\hat{p}_{\sigma_i}(c_i)$  comme la probabilité que l'élément  $\hat{y}_{\sigma(i)}$  soit de classe  $c_i$  et  $\hat{b}_{\sigma(i)}$  comme la bounding box prédictive pour l'élément  $\hat{y}_{\sigma(i)}$  nous pouvons écrire le coût de matching comme :

$$\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{1}_{\{c_i \neq \emptyset\}} \hat{p}_{\sigma_i}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$$

Le calcul du coût total du matching de toutes les paires se fait grâce au matching optimal  $\hat{\sigma}(i)$  déterminé précédemment et à la *Hungarian* loss définit ainsi :

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^N -\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\hat{\sigma}(i)})$$

Enfin, pour évaluer les performances des bounding boxes prédictives, l'algorithme DETR utilise la  $\mathcal{L}_{box}$  qui est une combinaison de la loss  $l_1$  et de la Intersection over Union loss ( $\mathcal{L}_{iou}$ ). Cela permet de prédire directement les bounding boxes sans avoir de supposition sur leurs tailles ou formes. On peut donc écrire :

$$\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{iou} \mathcal{L}_{iou}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L1} \left\| b_i - \hat{b}_{\sigma(i)} \right\|_1$$

### 3 Structure Générale

Pour la réalisation de ce projet, nous avons tout d'abord identifié une pipeline pour organiser notre avancement. Tout d'abord, il a fallu récolter des données pour pouvoir entraîner nos algorithmes. Pour cela, nous avons choisi à court terme d'utiliser des vidéos Youtube de joueurs de babyfoot professionnels. Pour compléter cette récolte de données, nous avons acheté une caméra permettant de filmer des parties et les utiliser en tant que données d'entraînement. Par ailleurs la caméra sera aussi utilisée pour filmer en temps réel le match de

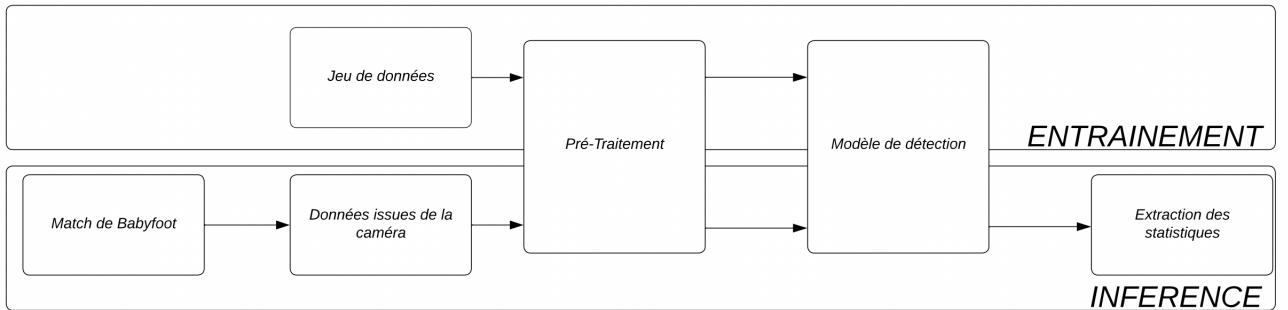


FIGURE 8 – Schéma de la structure d'ensemble

babyfoot et ainsi donner les statistiques de ce dernier.

### 3.1 Phase d'entraînement

Pour la phase d'entraînement, les images récoltées sont pré-traitées pour orienter correctement le terrain de babyfoot. Cela permet de faciliter la tâche d'apprentissage du modèle et permet à notre jeu de données d'être assez réduit. Suite à cette étape, nous pouvons entraîner notre modèle de détection afin d'obtenir en sortie la position de chaque joueur.

### 3.2 Phase d'inférence

La phase d'inférence est utilisée pour extraire les statistiques en temps réel d'un match. Les données de ce dernier sont filmées par la caméra et pré-traitées de la même manière que durant la phase d'entraînement. Le modèle de détection est ensuite appliqué et nous permet alors de calculer les statistiques du match grâce à l'extraction de la positions des joueurs et de la balle. Enfin, l'ensemble de ces statistiques sont affichées sur un leaderboard pour pouvoir suivre l'évolution du match en temps réel.

## 4 Implémentation

### 4.1 Description de notre problématique

Avant de pouvoir calculer les statistiques d'un match de babyfoot nous avons besoin de reconnaître et détecter l'ensemble des objets d'intérêt sur le babyfoot. Ces derniers sont

divisés en trois catégories : le terrain, les joueurs et la balle. De plus, nous allons devoir distinguer les joueurs appartenant à une équipe des joueurs qui appartiennent à l'équipe adverse. Nous parlerons des avantages et inconvénients de chaque méthode de détection pour chacune des trois application. Il est utile de rappeler que la finalité du projet étant de pouvoir suivre en temps réel la position des joueurs et de la balle, le temps d'inférence va être un critère très important à nos yeux, d'autant plus que la balle pouvant se déplacer très vite (jusqu'à 40 m/s) une fréquence d'échantillonnage élevée est requise.

## 4.2 Reconnaissance du terrain

**Introduction** Les images dont nous disposons ne sont pas limitées au terrain et une partie entière de l'image n'est donc pas intéressante pour notre problème, voire pire : peut dégrader les performances de notre détection d'objets. En effet, pour la reconnaissance de balle par exemple, certaines poignées introduisent un bruit dans nos prédictions en étant d'une couleur similaire à la balle. Réduire notre image au terrain et donc finalement à la seule partie de notre image qui nous intéresse permet ainsi d'avoir des prédictions plus fiables, mais aussi plus rapides car appliquées sur des images de taille réduite.

**Choix de la méthode** Pour ce problème de segmentation du terrain, plusieurs approches nous étaient disponibles. Les approches d'apprentissages, performantes pour cette tâche, demandent de labelliser des données, et ne sont pas rapidement adaptable à un changement de domaine. Une approche de segmentation par croissance de régions [1] aurait demandé de placer soit même les germes (il en aurait fallut plusieurs sachant que le terrain est traversé par les barres de maintien des joueurs), et la balle comme les joueurs étant mobiles, il aurait été compliqué de toute façon de placer des germes initiaux correctes pour chaque image. Une approche par clustering [16] aurait quand à elle très bien marché, se serait adaptée facilement au changement de domaine mais cette approche est trop lente par rapport aux performances que nous recherchons.

La méthode que nous avons finalement employé pour la détection du terrain se base sur un filtre passe bande de couleur : nous passons tout d'abord l'image en HSV (Teinte Saturation Valeur) puis filtrons selon la couleur du terrain du babyfoot. Cette approche est extrêmement

rapide et a l'avantage d'être très facile à adapter si l'environnement change.

**Description de la méthode choisie** L'étape de filtrage consiste à garder tous les pixels de l'image dont les valeurs HSV sont comprises entre deux bornes. Ce passage en HSV nous permet notamment d'avoir un filtrage plus robuste aux conditions de luminosité. Pour définir les deux bornes de notre étape de filtrage, nous avons tout d'abord pris une image de notre dataset, crée rapidement un masque du terrain. A partir de ce masque, nous avons pu constituer une liste de valeurs HSV ainsi que leur "classe" : terrain ou autre. Cette liste nous a permis d'entrainer un arbre de décision volontairement peu profond qui a pu nous donner de bonnes bases pour les bornes, que nous avons ensuite amélioré par quelques essais. Un arbre de décision réalise une partition de l'espace d'entrée (ici nos trois valeurs HSV) dans le but de classifier au mieux l'appartenance au terrain d'un pixel selon sa position dans l'espace d'entrée. Lors de l'entraînement, chaque noeud de l'arbre de décision minimise le Gini Index (une mesure de pureté des données)

$$Gini = 1 - \sum_{i=1}^{n_{classes}} p_i^2$$

avec  $p_i$  le rapport entre nombre d'instances de classe i et le nombre d'instances totales dans le noeud. Entrainer un arbre peu profond nous permet d'avoir une séparation de l'espace de couleurs HSV explicable et implémentable avec un filtre passe bande. Une fois ce filtrage effectué, nous appliquons une étape d'érosion/dilatation qui permet d'éliminer le bruit. L'érosion/dilatation d'une image à niveaux de gris  $f : \mathbb{Z}^2 \rightarrow \mathbb{R}$  par un noyau binaire  $N \subseteq \mathbb{Z}^2$  est notée  $f \oplus N$ . Après érosion, la nouvelle valeur d'un pixel est donnée par la valeur minimale (maximale pour dilatation) des pixels sous le noyau.

$$\forall x, (f \oplus N)(x) = \min_{p \in N} f(x + p)$$

Il est alors important de considérer un noyau circulaire pour cette étape car l'orientation du terrain pouvant être quelconque, cette étape doit être invariante à la rotation. La figure 9 décrit ces premières transformations.

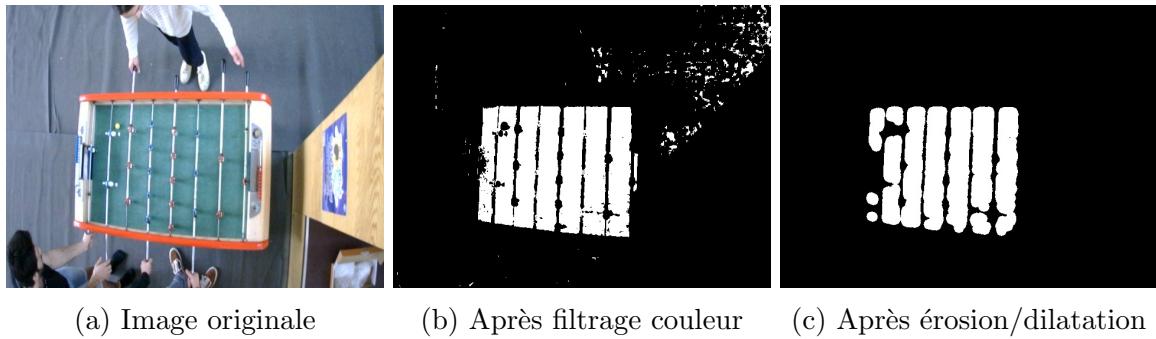


FIGURE 9 – Premières étapes de l'extraction de terrain

**Correction de l'orientation** La prochaine étape consiste en la correction de l'orientation du terrain. En effet pour la suite il sera plus facile de considérer une orientation constante. Pour la détection des joueurs par exemple, cela facilite grandement la tâche s'ils ont toujours la même orientation, et cela évite une étape de data augmentation de notre base de données et participe donc à réduire le temps d'entraînement et donne de meilleures performances.

Pour corriger l'orientation, nous utilisons une méthode qui consiste à contenir notre terrain récemment extrait dans un rectangle d'aire minimale. L'orientation de ce rectangle nous donnera l'orientation du terrain. Nous avions également considéré récupérer cette orientation avec une PCA, mais cette méthode s'est révélée bien plus rapide, donc préférable pour notre application. La correction de l'orientation du terrain est illustrée en figure 10.

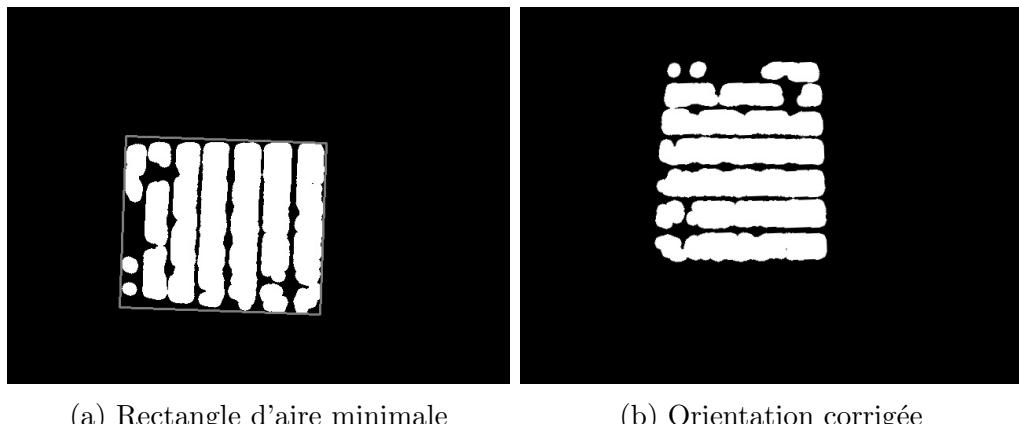


FIGURE 10 – Correction de l'orientation

**Extraction des coins** Il ne reste donc plus qu'à récupérer les 4 coins de notre terrain. Pour cela, sachant que notre terrain est un rectangle déformé par la perspective (nous ne pouvons donc pas juste prendre les 4 coins du rectangle d'aire minimale), il ne reste plus qu'à corriger l'orientation de notre terrain extrait, et simplement en extraire les quatres

points extrêmaux du trapèze que constitue le terrain. Pour cela, notre terrain étant à présent réorienté, nous prenons tout simplement l'ordonnée du pixel activé le plus bas de notre masque avec orientation corrigée, puis nous considérons une bande de 25 pixels de haut à partir de cette ordonnée. Les abscisses du pixel activé le plus à gauche et le plus à droite dans cette bande nous donnent les coordonnées des deux coins inférieurs du terrain. Il est important de considérer une certaine marge représentée par notre bande de 25 pixels car les coins du terrains passent souvent moins bien l'étape de filtrage car ils sont souvent moins bien atteints par l'éclairage. Nous obtenons ainsi un trapèze donnant les quatres coins du terrain en répétant l'opération pour les coins supérieurs, comme visible sur la figure 11.

**Obtention de l'image rognée** Une fois les quatres coins de notre terrain extraits, nous pouvons rogner notre image d'entrée de façon rectangulaire pour avoir notre terrain et masquer les pixels de l'image qui ne font pas partie de notre terrain (utile au vu de l'angle de la caméra). Prendre une certaine marge est également important pour détecter les joueurs qui ne sont pas sur le même plan que le terrain, et qui peuvent donc en dépasser à cause de l'angle de la caméra. C'est cette nouvelle image qui servira d'entrée aux étapes suivantes. Le terrain ne bougeant peu, inutile de faire la détection à chaque image, sachant qu'actuellement (supposant un terrain aligné verticalement), la détection prend environ 4ms pour une image en 640\*480, ce qui est compatible avec nos critères de rapidité en effectuant par exemple la détection du terrain toutes les secondes.



(a) Extraction des coins

(b) Détection du terrain

(c) Image rognée, marge 25px

FIGURE 11 – Extraction du terrain

### 4.3 Reconnaissance de la balle

**Choix de la méthode** Pour la reconnaissance de la balle, nous procédons selon la même méthode que pour le terrain : nous utilisons un filtre passe bande sur la couleur. Les balles utilisées étant de couleur jaune se démarquant bien avec le reste des objets présents sur notre image réduite par la détection du terrain, la détection est peu bruitée et nous pouvons donc prendre un noyau de taille très réduite pour l'étape d'érosion/dilatation. Nous avons choisi d'utiliser une détection par couleur plutôt que d'utiliser un algorithme de deep learning pour trois raisons principales. La première étant la difficulté de reconnaître la balle lorsque celle-ci se déplace rapidement. En effet, avec une vidéo filmée à 25 images par seconde par exemple comme celles téléchargées sur youtube, les artéfacts visuels sont fréquents avec une balle qui peut atteindre les 40m/s. Les images présentant ces artéfacts restant rares comparées aux images présentant une balle immobile (en phase de possession), il est difficile pour un réseau de la détecter dans ces cas là. De plus, la balle est très souvent partiellement occultée par un joueur ou une barre, rendant sa détection difficile. Les approches de flot optiques [5] auraient pu fonctionner, mais le temps d'inférence d'un réseau neuronal profond étant bien plus important que de simples traitements d'images, et cette dernière approche ne demandant aucun entraînement donc aucune annotation manuelle, elle nous est largement préférable. Notre approche prend moins d'une milliseconde par image, la rendant très efficace.

**Cas de detections multiples** Dans le cas où plusieurs objets seraient détectés, un score de rondeur de chaque forme est alors calculé, permettant de retourner la forme la plus ronde. Pour obtenir un tel score, la méthode que nous employons consiste à calculer l'aire de la forme (donnée par le nombre de pixels qui la constituent). Ainsi nous pouvons en déduire le rayon d'un cercle parfait ayant cette aire. Notre score de rondeur correspond à la proportion de notre forme se situant dans ce cercle. Cette partie nous servait surtout lorsque nous n'avions pas encore obtenu le rognage selon le terrain, car sur les vidéos que nous avons téléchargés sur youtube, certains joueurs avaient des poignées de la même couleur que la balle.

**Résultats** Pour obtenir des indicateurs de performances sur la reconnaissance de balle, nous avons constitué un petit jeu de données de 100 images issues de nos parties de babyfoot,

et avons annoté la position de la balle sur ces images. Nous avons ainsi pu obtenir quelques indicateurs de performances de notre détecteur de balle. Parmi les 100 images, la balle n'était pas présente du tout sur 12 d'entre elles, et occultées sur d'autres mais nous les avons quand même annotées dans ce cas là. Sur ces 12 images, une balle n'a été faussement détectée que sur 1 seule image. Sur les 88 images présentant une balle, la balle n'a pas été détectée sur 14, et sur les 74 où la balle a été détectée, 100% des prédictions sont correctes. Une prédition est jugée correcte lorsque la distance entre le centre de la prédition et le centre réel de la balle est inférieure à 10 pixels. Nous travaillons ici avec des images en 1920\*1080, et la balle fait environ 20 pixels de diamètre. La distance moyenne entre la prédition et la réalité est de 1.9 pixels sur ces 74 images. Nous avons donc d'un point de vue détection de la présence de la balle une précision de  $74/75 \approx 0.987$  et un rappel de  $74/88 \approx 0.841$ . Il est intéressant ici de voir qu'avoir une précision proche de 1 est bien plus intéressant que d'avoir un rappel proche de 1, car il est préférable d'éviter de fausses prédictions plutôt que d'essayer de trouver la balle sur toutes les images. En effet, lorsque la balle est occultée, nous pouvons inférer sa position en connaissant sa position et sa vitesse à des moments antérieurs.

## 4.4 Reconnaissance des joueurs

**Choix du modèle** L'état de l'art des méthodes de détection en deep learning effectué plus haut, nous a permis de comprendre réellement notre besoin et ce que nous devions chercher précisement. Les performances que nous cherchions en premier lieu ont du être nuancées par la rapidité d'exécution qui nous était nécessaire. Après avoir exploré quelques méthodes, nous nous sommes appuyé sur un tableau comparatif [**Fig 12**].

Ce tableau nous a permis de discriminer Yolov5 [7], une architecture obtenant de bonnes performances et ayant un temps d'inférence très réduit. Notre problème n'étant réduit qu'à deux classes (joueurs rouges et joueurs bleus) avec une variabilité assez faible dans les images, le problème de détection ne nécessite pas forcément l'algorithme avec les meilleures performances. Ainsi, nous avons encore pu nous restreindre et utiliser la version small de Yolov5 : Yolov5s (implémentée sur le github <https://github.com/ultralytics/yolov5>) choisie donc à la fois pour ses performances et pour sa rapidité d'inférence. Le modèle étant pré-entraîné sur COCO [10], un jeu de données challengeant pour la reconnaissance d'objet,

Name	Year	Dataset	mAP	Inference rate (fps)
<b>R-CNN</b>	2014	Pascal VOC	66%	0.02
<b>Fast R-CNN</b>	2015	Pascal VOC	68.80%	0.5
<b>Faster R-CNN</b>	2016	COCO	78.90%	7
<b>YOLOv1</b>	2016	Pascal VOC	63.40%	45
<b>YOLOv2</b>	2016	Pascal VOC	78.60%	67
<b>SSD</b>	2016	Pascal VOC	74.30%	59
<b>RetinaNet</b>	2018	COCO	61.10%	90
<b>YOLOv3</b>	2018	COCO	44.30%	95.2
<b>YOLOv4</b>	2020	COCO	65.70%	62
<b>YOLOv5</b>	2021	COCO	56.40%	140
<b>YOLOR</b>	2021	COCO	74.30%	30
<b>YOLOX</b>	2021	COCO	51.20%	57.8

FIGURE 12 – Comparaison des différents modèles pour la détection d’objet [10] [3]

il suffit de le fine-tune pour l’adapter à notre problème de détection à deux classes (joueurs rouges et bleus). En effet, ayant déjà appris à extraire de nombreuses features pertinentes d’une image, il doit déjà être en capacité de reconnaître les éléments clefs d’un joueur sur une image. Il ne lui reste alors plus qu’à apprendre comment combiner ces features pour segmenter les joueurs. Sachant que nous avons un nombre de joueurs invariants à chaque image (sauf occlusions), nous pouvons également spécifier le nombre de détection maximales que nous souhaitons, et Yolo ne retourne alors que ses prédictions en lesquelles il est le plus confiant. Voulant détecter 22 joueurs (11 par côtés), nous avons fixé cette limite à 22.

**Fine tuning** Pour fine tuner Yolo, n’ayant pas encore de caméra à notre disposition, nous avons téléchargé deux vidéos de tournois de babyfoot disponibles sur Youtube, puis extrait et annoté environ 150 images à la main. L’annotation des joueurs étant très fastidieuse (22 joueurs par images), nous nous sommes contentés de ces 150 images pour entraîner une première fois Yolo sur cette base de données (divisée en 120 images pour l’entraînement, 15 pour la validation et 15 pour le test). Ce premier entraînement n’a pas donné de bonnes performances car 150 images n’est pas assez pour montrer les joueurs avec une assez grande variabilité, mais ce n’était pas le but. Avec cette première itération, nous avons pu demander

à Yolo d'estimer la position des joueurs sur 400 nouvelles images. Bien que ses estimations étaient alors loin d'être parfaite, il ne suffisait alors plus que de les corriger, et non de les reprendre de zéro, ce qui fut très rapide. Nous avons donc pu entraîner Yolo une nouvelle fois sur notre nouvelle base de 550 images, et ainsi obtenir un modèle avec de bonnes performances.

Lorsque nous avons reçu la caméra, nous avons alors pu enregistrer nos parties sur les babyfoot de l'école. Evidemment, la caméra, l'angle de vue et les babyfoots étant différents, notre modèle n'obtenait pas des prédictions assez fiables. Il fallut alors fine-tune notre modèle une dernière fois. Pour cela, nous avons procédé de la même manière que précédemment, et avons ajouté 100 images issues de nos propres parties, où nous en avons profité pour placer les joueurs selon des positions compliquant la tâche de détection. Après un rapide fine tuning, notre modèle s'est montré parfaitement adapté à la détection de joueurs sur nos enregistrements, et ce même lors de mouvements rapides ou de tours sur eux-même.

**Résultats** Lorsque nous parlons des performances de Yolo, nous nous référons à une métrique principalement : la mAP50. La mAP (mean Average Precision [6]) est définie comme telle :

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$$

sachant qu'une prédiction est considérée correcte (True Positive) si l'IoU (Intersection over Union) entre la prédiction et la vérité est au dessus d'un certain seuil, valant 0.5 pour la mAP50, et est considérée fausse (False Positive) sinon. L'architecture que nous utilisons étant performante dans la tâche de détection, il sera également intéressant de regarder la mAP50-95 qui est une moyenne de la mAP pour 11 valeurs de seuils régulièrement espacés entre 0.5 et 0.95. Pour le deuxième entraînement par exemple, nous avions lancé le fine-tuning sur 150 epochs et avons gardé l'itération ayant obtenue les meilleures performances sur le jeu de validation. Ce modèle obtenait sur le jeu de test une mAP50 de 0.995, une mAP50-95 de 0.799 pour une précision de 0.999 et un recall de 0.998, et nous avons pu ré-obtenir des métriques similaires après le dernier fine-tuning sur le jeu de données complété par les images issues de nos parties. Les métriques finales obtenus sont donc de 0.995 pour la mAP50, 0.785 pour la mAP50-95, 0.999 pour la précision et 0.998 pour le rappel.

**Intégration à la pipeline** Lors de l'appel de Yolo dans notre pipeline d'extraction de statistiques, nous nous sommes rendus compte que l'inférence image par image avec Yolo n'était pas très rapide, et qu'il fallait mieux les grouper par batch si nous voulions avoir un temps de traitement par image le plus réduit possible. Après tests, nous nous sommes rendus compte que sur nos machines (une NVIDIA GTX-1650) une batch size de 50 donnait un temps de traitement par image optimal et permettait à la pipeline de détection du terrain, de la balle et des joueurs de traiter plus de 40 images par secondes.

Une vidéo de présentation de cette pipeline d'extraction du terrain, de la balle et des joueurs est disponible en suivant ce lien ([cliquez ici](#)). Cette vidéo a été tourné à Centrale-Supelec par nous-même, et n'a pas été utilisée lors de l'entraînement de Yolo. Par ailleurs, elle a même été tournée un autre jour avec des conditions d'éclairage différentes. Les bandes noires visibles sur le côté sont le résultat de la correction de l'orientation, car nous avons voulu garder un format d'image de sortie 16/9 lors de l'affichage (ceci n'impactant pas les étapes suivantes du projet, car les images elles-mêmes ne sont pas prises en entrée des éléments suivant de la pipeline).

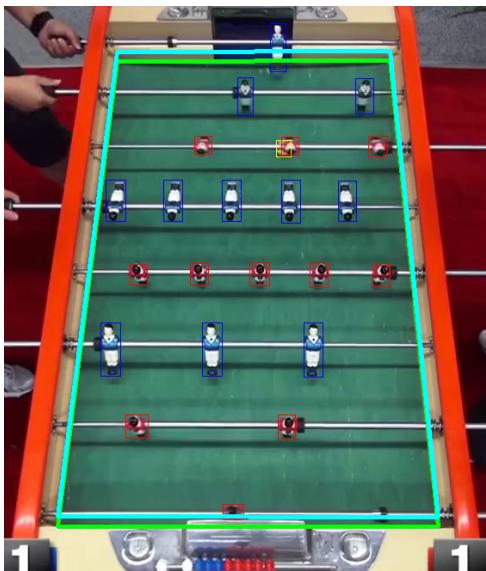
## 4.5 De l'image 2D au terrain 3D

Afin de pouvoir extraire des statistiques d'un match de baby-foot, il nous faut pouvoir raisonner sur le monde 3D du terrain. Cela nécessite donc de pouvoir reconstruire ce dernier à partir des images obtenues par la caméra et notamment de prendre en compte la projection induite par la caméra.

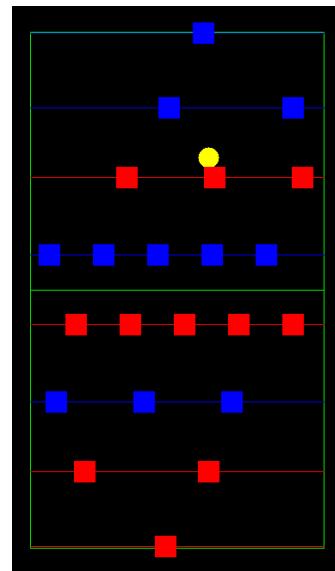
La projection du monde 3D dans un plan 2D se décrit à l'aide d'un modèle de caméra. Par simplicité, nous avons choisi celui du modèle sténopé (ou trou d'épingle), qui modélise une caméra par une projection perspective. Celui-ci se caractérise par quatre référentiels et trois transformations. Celle du monde 3D au repère caméra permet de prendre en compte les déplacements de la caméra. Celle du repère caméra au repère image tient compte de la distance focale. Enfin, celle du repère image au repère pixel permet de prendre en compte les pixels. L'étape permettant de connaître les paramètres de ces transformations s'appelle la calibration d'une caméra : cette étape est cruciale pour obtenir le terrain en 3D. Différentes approches permettent de mener à bien cet objectif; avec une seule caméra (et donc un seul

point de vue), celle s'imposant naturellement était la calibration avec un objet 3D connu, à savoir le babyfoot.

Nous avons utilisé les quatre coins du terrain (d'où la nécessité de la reconnaissance du terrain, voir ci-dessus) comme points connus de l'objet 3D pour déduire la géométrie de l'image. Par soucis de simplification du monde 3D, nous avons considéré deux plans distincts : celui contenant la balle et celui contenant les joueurs (voir figure 13a). En effet, la caméra n'étant pas placée parfaitement au-dessus du terrain de manière perpendiculaire (notamment avec les vidéos obtenues sur Internet), ces deux plans apparaissent légèrement décalés sur l'image bien qu'ils soient parallèles dans le monde réel. Ils apparaissent également sous la forme d'un quadrilatère quelconque et non d'un rectangle, ce qui justifie bien la nécessité d'une transformation non-triviale. Nous avons donc ramené le problème de reconstruction 3D à celui de trouver les coordonnées 2D de la balle et des joueurs dans leur plan respectif, pour ainsi obtenir une figure comme celle présentée figure 13b.



(a) Image caméra, avec le plan de la balle (en vert clair) et celui des joueurs (en cyan).



(b) Représentation 2D obtenue après reconstruction 3D

FIGURE 13 – Processus de reconstruction 3D

Pour le plan de la balle, nous avons considéré qu'il était confondu avec celui des quatre coins du terrain. Pour obtenir la transformation à appliquer pour obtenir les coordonnées de la balle pour la représentation 2D, nous avons alors utilisé la fonction `PerspectiveTransform()` d'OpenCV avec en entrée le quadrilatère formé par les quatre coins du terrain. Nous avons utilisé ce même principe pour le plan des joueurs, mais avec des points légèrement décalés

d'une hauteur  $h$  par rapport aux coins du terrain. Ce  $h$  est obtenu de manière expérimentale en tenant compte de la géométrie du quadrilatère.

En outre, nous avons appliqué un post-traitement afin de lisser les positions obtenues et associer à chaque joueur une position (parmi `goal`, `defense`, `middle`, `attack`). En effet, la détection des joueurs ne renvoie qu'une liste de joueurs qui ne contient pas ces positions. Ensuite, les coordonnées des joueurs (milieu des bounding box) ne coïncident pas avec la barre, car la partie inférieure des joueurs est plus longue que la partie supérieure. Enfin, l'espace entre des joueurs à une même position peut varier dans la détection à cause du bruit alors qu'il est fixe dans la réalité. C'est pourquoi nous avons utilisé la coordonnée selon l'axe dans la longueur du terrain pour discriminer les positions, puis avons calculé la moyenne de la coordonnée selon l'autre axe pour des joueurs de même position afin de replacer parfaitement les joueurs sur la barre avec un espacement fixe.

## 4.6 Implémentation des statistiques

Nous avons déterminé deux types de données à renvoyer à l'utilisateur et à afficher : des statistiques globales qui concernent la partie entière, et des événements, qui se produisent à un instant particulier. Ces derniers sont basés sur des variables internes calculées en amont.

**Variables internes** Tout d'abord, ces éléments nécessitent un pré-calcul de variables internes à chaque frame, qui permettront par la suite le calcul des statistiques globales et événements. Ces variables internes sont :

- **Vecteur déplacement de la balle** : différence de positions de la balle entre deux frames,  $d_{t_i}^{balle} = x_{t_i}^{balle} - x_{t_{i-1}}^{balle}$
- **Vecteur vitesse de la balle** : vitesse de la balle entre deux frames,  $v_{t_i}^{balle} = \frac{x_{t_i}^{balle} - x_{t_{i-1}}^{balle}}{t_i - t_{i-1}}$
- **Vecteur accélération de la balle** : accélération de la balle entre deux frames,  $a_{t_i}^{balle} = \frac{v_{t_i}^{balle} - v_{t_{i-1}}^{balle}}{t_i - t_{i-1}}$
- **Distances joueurs - balle** : distance avec la balle pour chaque joueur,  $D_{t_i} = \{||x_{t_i}^{balle} - x_{t_i}^j||, j \in \mathcal{Joueurs}\}$
- **Joueur le plus proche de la balle** :  $j_{t_i}^{closest} = \arg \min_j D_{t_i}$
- **Distance du joueur le plus proche avec la balle** :  $d_{t_i}^{closest} = \min_j D_{t_i}$

- **Angle déplacement de la balle** : angle entre le vecteur déplacement de la balle et le vecteur  $\vec{u} = (1, 0)$  dans le sens de la largeur du terrain,  $\alpha_{t_i} = \angle(d_{t_i}^{balle}, \vec{u})$
- **Angle de déviation de la balle** : angle entre deux vecteurs déplacement de la balle pour deux frames successives,  $\beta_{t_i} = \angle(d_{t_i}^{balle}, d_{t_{i-1}}^{balle}) = \alpha_{t_i} - \alpha_{t_{i-1}}$
- **Possession** : couleur de l'équipe qui a touché la dernière la balle,  $p_{t_i} \in \{red, blue\}$  (en considérant que toucher la balle équivaut à avoir  $\beta_{t_i} < 0$  et que c'est le joueur le plus proche,  $j_{t_i}^{closest}$ , qui l'a touché)

Dans certains cas, la balle n'est pas détectée (car par exemple elle est absente du terrain, cachée ou trop rapide) ou les joueurs ne le sont pas. Nous avons choisi de gérer cela avec des **None**, car certaines variables ne sont alors pas définies.

**Événements** Ensuite, à partir de ces variables, nous avons déduit deux événements à reconnaître et à enregistrer :

- **Tirs** : Un des joueurs effectue un tir
  - *Condition d'activation* : changement de joueur le plus proche et acceleration de la balle au dessus d'un certain seuil ( $j_{t_i}^{closest} \neq j_{t_{i-1}}^{closest} \cap \|a_{t_i}^{balle}\| > A_{seuil\ tir}$ )
  - *Statistiques associées* : temps associé  $t_i$ , joueur ayant tiré  $j_{t_i}^{closest}$ , vitesse de tir  $\|v_{t_i}^{balle}\|$
- **Buts** : Un but est marqué
  - *Condition d'activation* : la balle disparaît pendant les  $N$  dernières frames (nous avons choisi  $N = 5$ ) et, soit la balle était dans un but à la dernière frame où elle était présente, ou soit sa prédiction y était  $(\forall i' \in [i - N + 1, i], x_{t_{i'}}^{balle} = \text{None}) \cap (x_{t_{i-N}}^{balle} \in \mathcal{G} \cup x_{t_{i-N}}^{balle} + d_{t_{i-N}}^{balle} \in \mathcal{G})$  où  $\mathcal{G} = \mathcal{G}_{Goal\ red} \cup \mathcal{G}_{Goal\ blue}$  est la zone de but de l'équipe rouge ou de l'équipe bleue
  - *Statistiques associées* : temps associé  $t_{i-N}$ , couleur de l'équipe ayant marqué  $c \in \{red, blue\}$

Ces règles de reconnaissance des événements sont très arbitraires et d'autres règles auraient été possibles, pour prendre en compte des situations précises où nos règles actuelles ne fonctionnent pas forcément correctement. Par exemple un rebond proche d'un joueur peut induire une accélération suffisante pour être reconnue comme un tir dans certains cas ; mais pour

prendre en compte ceci il faudrait détecter si la balle est proche du bord et utiliser une règle spécifique : ces situations spécifiques peuvent rapidement complexifier les règles. Nous avons ainsi fait le choix de conserver des règles simples et ne pas trop aller dans le détail, afin de conserver un comportement explicable et parce que le travail de spécification de règles aurait une valeur ajoutée relativement faible pour un temps d'implémentation non-négligeable. De même, d'autres événements à reconnaître peuvent aisément être ajoutés dans une perspective de continuation du projet, car les variables internes ne sont volontairement pas spécifiques à la reconnaissance des tirs et des buts, mais contiennent au contraire une grande partie de l'information d'un match de babyfoot.

**Statistiques globales** Puis, nous mettons à jour des variables qui résument une partie du match de babyfoot, que nous calculons à partir des variables internes et des événements :

- **Score** : Nombre de buts marqués par l'équipe rouge et par l'équipe bleu, mis à jour après chaque détection de l'événement but
- **Durée**
- **Distance totale parcourue par la balle** :  $D_{tot}^{balle} = \sum_i ||d_{t_i}^{balle}||$
- **Vitesse maximale de la balle** :  $v_{max}^{balle} = \max_i ||v_{t_i}^{balle}||$
- **Possession** : Pourcentage de possession de l'équipe rouge et de l'équipe bleue (lorsque la balle est en mouvement, la possession revient à l'équipe du joueur qui l'a touchée en dernier, comme défini précédemment par la variable interne  $p_{t_i}$ ),  $P_c = \frac{n_c}{n}$  où  $c \in \{red, blue\}$  est la couleur de l'équipe,  $n_c$  est le nombre de frame où  $p_{t_i} = c$  et  $n$  est le nombre de total de frames

Même remarque que précédemment, il est possible de rajouter facilement de nouvelles statistiques globales en se basant sur les variables internes et les événements.

**Affichage** Enfin, ces données - variables internes, événements et statistiques - sont sauvegardées au format json et sont affichées en fin de partie. Notamment, nous avons implémenté une fonction permettant de tracer un graphique qui récapitule une partie de ces données, comme celui présent figure 14. Il présente l'évolution frame par frame de la vitesse de la balle  $||v_{t_i}^{balle}||$ , de l'accélération de la balle  $||a_{t_i}^{balle}||$  et de l'angle de déviation du déplacement de la balle  $\beta_{t_i}$ , les événements détectés (sous forme de point sur l'axe des abscisses), ainsi que la

possession (background rouge ou bleu du graphique). Ce graphique n'est pas nécessairement destiné aux joueurs de la partie ; il vise plutôt à vérifier le bon fonctionnement des règles et à aider à en implémenter de nouvelles.

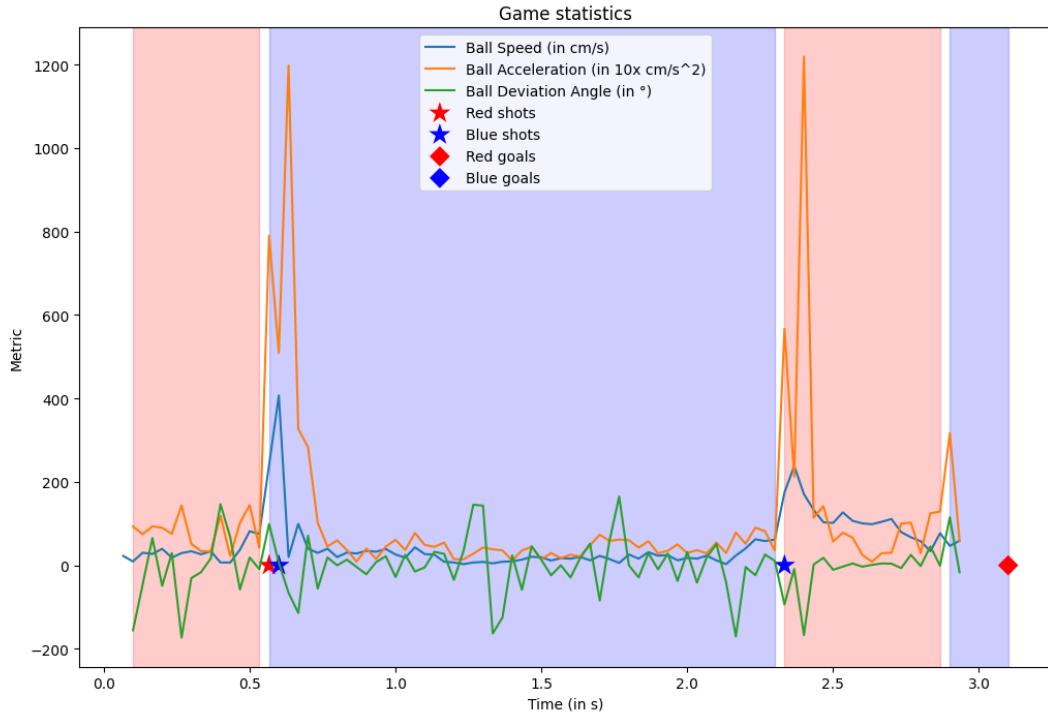


FIGURE 14 – Graphique affiché en fin de partie et représentant certaines données de la partie de babyfoot

## 4.7 Présentation de la pipeline

Nous utilisons la caméra branchée en USB sur notre ordinateur (voir photo de notre montage 15). Pour la pipeline, nous commençons par créer une instance de chaque détecteur, et chargeons Yolo sur le GPU pour la détection des joueurs. Une fois ceci effectué, nous créons un générateur qui à chaque itération sur ce dernier, capture et renvoie une image de la caméra, ainsi que le temps auquel cette image a été capturée (utile pour la suite). Les images sont ensuite stockées dans un buffer afin d'effectuer une inférence par batch, plus efficace pour la détection des joueurs. On obtient alors pour chaque image la détection du terrain, de la balle et des joueurs. Tout ceci est effectué dans un fichier `tracking_pipeline.py`. Ensuite a lieu l'étape de reconstruction 3D, dans laquelle on obtient les positions de la balle et des

joueurs dans un plan 2D afin d'éliminer les effets dus à l'inclinaison de la caméra et placer correctement les joueurs sur la barre. Cela est implémenté dans un fichier `game_state.py`. Enfin, les statistiques sont calculées dans `stats_extraction.py` puis l'ensemble (image, représentation 2D et stats) est affiché dans `animate.py`. Tous nos codes sont accessibles sur notre github : [https://github.com/Alex-experiments/babyfoot\\_CV](https://github.com/Alex-experiments/babyfoot_CV).

## 5 Conclusion

Ce projet nous a permis de nous renseigner sur l'état de l'art en matière de détection d'objets, tout en appliquant également des méthodes plus anciennes enseignées durant nos cours. Le regard critique que nous avons du avoir sur l'état de l'art au vu de notre contrainte très importante de temps d'inférence s'est révélé très formateur. De plus, nous avons souvent été confrontés à des problématiques nouvelles comme la constitution d'un jeu de données ou obtenir des performances satisfaisantes en temps réel. En effet, nous nous sommes rendus compte qu'utiliser l'état de l'art avec les meilleures performances ne suffisait pas pour répondre à la problématique.

Nous sommes heureux d'avoir mené à bien notre projet en répondant aux contraintes auxquelles nous étions soumis et travaillons actuellement sur un démonstrateur (une interface graphique basée sur streamlit) afin de pouvoir en proposer une démonstration lors d'un tournoi de babyfoot organisé à CentraleSupelec en juin, avec Headmind Partners en partenaire.



FIGURE 15 – Exemple de montage du dispositif

## Références

- [1] A. MAARIR, I. AGNAOU, B. B. (2014). Evaluation de quelques méthodes de segmentation - application aux caractères tifinagh. *TICAM*.
- [2] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *Computer Vision–ECCV 2020 : 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer.
- [3] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2) :303–338.
- [4] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2015). Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1) :142–158.
- [5] Horn, B. K. and Schunck, B. G. (1981). Determining optical flow. *Artificial intelligence*, 17(1-3) :185–203.
- [6] Hui, J. (2018). map (mean average precision) for object detection.
- [7] Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, Lorna, Yifu), Wong, C., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, UnglvKitDe, Sonck, V., tkianai, yxNONG, Skalski, P., Hogan, A., Nair, D., Strobel, M., and Jain, M. (2022). ultralytics/yolov5 : v7.0 - YOLOv5 SOTA Realtime Instance Segmentation.
- [8] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2.
- [9] Li, Z., Peng, C., Yu, G., Zhang, X., Deng, Y., and Sun, J. (2017). Light-head r-cnn : In defense of two-stage object detector. *arXiv preprint arXiv :1711.07264*.

- [10] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Doll'a r, P., and Zitnick, C. L. (2014). Microsoft COCO : common objects in context. *CoRR*, abs/1405.0312.
- [11] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.
- [12] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd : Single shot multibox detector. In *Computer Vision–ECCV 2016 : 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer.
- [13] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer : Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022.
- [14] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once : Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.
- [15] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn : Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.
- [16] Xu Ji, Joao F. Henriques, A. V. (2019). Invariant information clustering for unsupervised image classification and segmentation. *arXiv :1807.06653*.