Goal Invincible

Rapport du projet

Michael Bleuez, Xavier Bouclé, Juan Daniel Gomez, Kilian Mac Donald

Introduction

Le but de ce projet est de réaliser un goal invincible pour le babyfoot. Ce goal devra bouger de façon autonome grâce à un servomoteur commandé par une raspberry Pi qui détectera la balle grâce à une caméra. La raspberry Pi devra calculer la trajectoire de la balle, et si la balle rentre dans les cages, déplacer le goal afin qu'il intercepte celle-ci.

1. Fonctionnement général du programme

Le programme général fonctionne selon une boucle acquisition-traitement-action.
Après une phase d'initialisation, où sont notamment repérés les poteaux du goal, le programme rentre dans cette boucle jusqu'à la fin du flux vidéo/caméra d'entrée.

Acquisition: Reconnaissance des positions du goal et de la balle (d'une couleur précise) par reconnaissance d'image (via OpenCV).

Traitement: Filtrage des positions (du centre de la balle) obtenues, prédiction de la position d'interception. Conversion de cette position en valeur PWM de commande du servo.

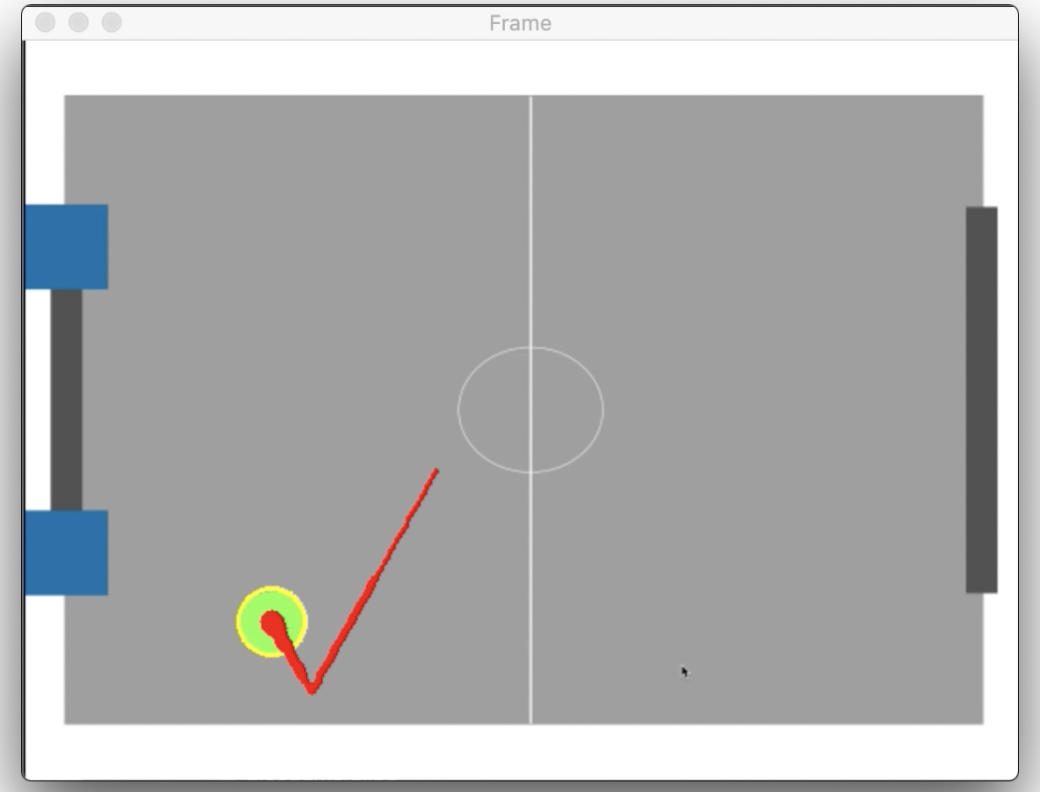
Action: Affichage de la position à atteindre, ou envoi de l'ordre au servomoteur via le module (API python) correspondant sur Raspberry Pi. (partie simple et non discutée: l'API suffit à nos besoins)

2. Reconnaissance d’image: détection de la balle et du but

Cette partie est la base du projet. Pour arriver à construire un goal invincible il faut connaître la position de la balle à chaque instant, et utiliser cette information pour prédire sa trajectoire.
La reconnaissance d’image est découpée elle-même en trois parties : la détection de la balle, celle de la cage et l’intégration du code au programme.

2.1 Détection de la balle:
Pour arriver à détecter la balle, nous avons choisi d’utiliser un méthode de détection de couleur. Dans notre cas, la couleur verte/jaune similaire à celle des balles de tennis. Nous avons adapté un code déjà fait (pyimagesearch.com / Ball tracking with OpenCV [1]) qui utilise principalement la librairie OpenCV de Python.
Tout d’abord nous avions conçu le code pour utiliser la webcam connectée à la raspberry, mais nous avons ensuite également ajouté la possibilité de fournir au programme une vidéo déjà réalisée pour tester la détection, en raison de l’impossibilité de tester avec le prototype à cause du confinement.
Nous avons d’abord déterminé les deux bornes de la couleur jaune/verte dans l'espace de couleur HSV. Cela sera la plage de couleur détectable. Pour chaque image de la vidéo nous cherchons des objets de couleur dans l’intervalle en appliquant un masque sur l’image.
Ensuite nous pouvons trouver les surfaces qui ont été découvertes par le masque et sauvegarder le centre de la plus grande surface seulement. Cette surface devrait normalement correspondre à notre balle dans l’image. Enfin, nous récupérons ce qui nous intéresse le plus : les coordonnées du centre de cette surface.
En plus, nous avons ajouté une aide visuelle pour mieux vérifier le fonctionnement du programme. Cette aide visuelle dessine le cercle autour de la balle et sauvegarde les centres de chaque image dans une collection (deque) pour dessiner aussi la trajectoire de la balle.
Il est à remarquer que le script marche parfaitement uniquement que s’il n’y a qu’un seul objet vert détecté. Dans le cas où il y en a plusieurs, le programme essaiera de choisir le contour le plus grand en supposant que la balle va être toujours l'objet jaune-vert avec la plus grande surface dans le babyfoot.

Le programme actuel fonctionne bien avec la webcam et nos vidéos tests. Voici le résultat d’une simulation.

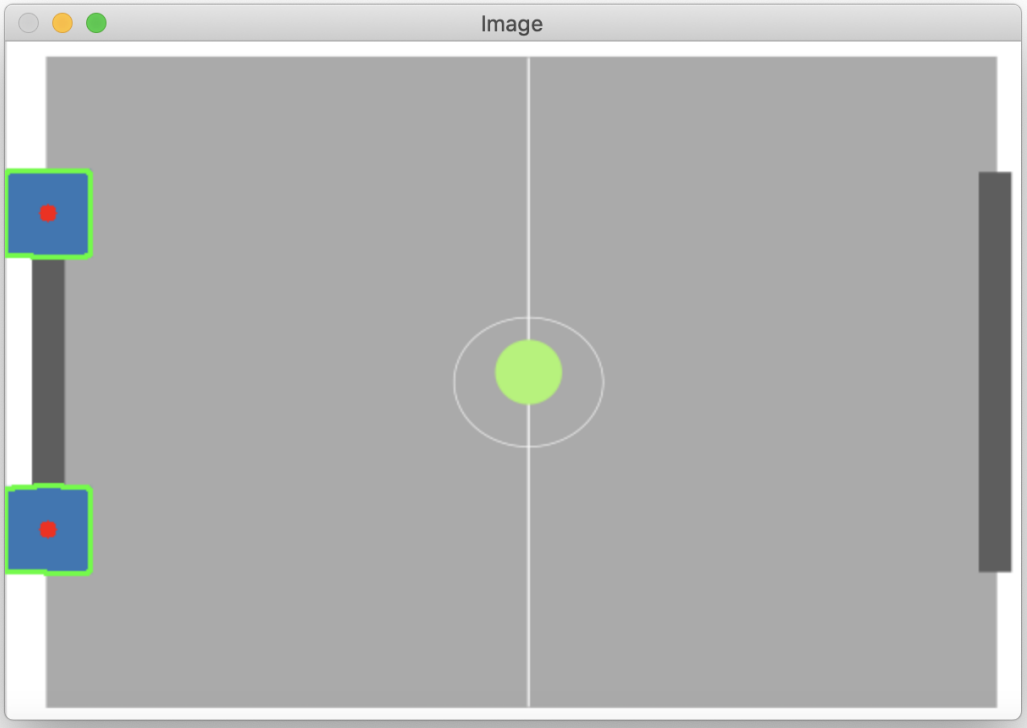


2.2 Détection des cages :

Pour la détection de la cage, nous avons utilisé le même principe de détection par couleur, nous avons donc repris plusieures parties du code de la détection de balle pour cette partie. L'objectif est de trouver la position des poteaux de la cage au début de la vidéo pour améliorer la précision en éliminant les erreurs provoquées par le placement de la webcam. Nous plaçons deux points bleus (un à chaque poteau) puis effectuons la reconnaissance une seule fois pour ne pas surcharger la Raspberry Pi. Les principales différences avec l'algorithme utilisé pour la balle sont que la couleur utilisée est le bleu au lieu du vert, que l'algorithme n'est utilisé que sur la première image de la vidéo, et enfin, que l'algorithme doit trouver deux points au lieu d'un seul.

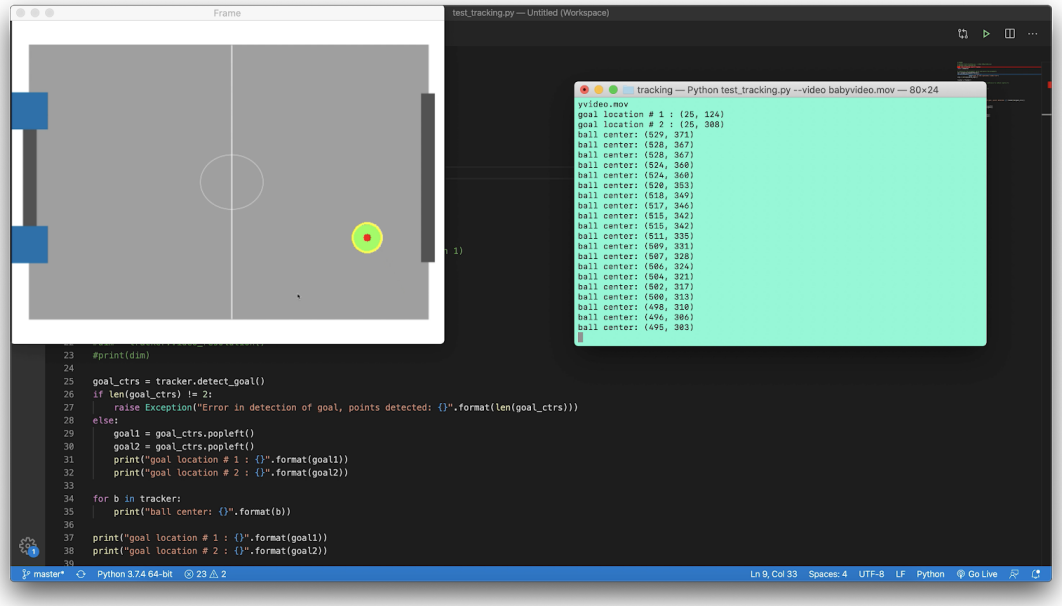
Cet algorithme fonctionne aussi bien avec la webcam (des carrés bleus sont collés au sommet des poteaux pour les identifier) qu'avec une vidéo/photo fournie. Afin de garder l'ordre des poteaux identique entre différents lancements du programme, nous assumons que la caméra est un minimum alignée avec le terrain, ce qui nous permet alors de trier les poteaux par proximité au coin (0,0) de la vidéo sans ambiguïté. L'ordre est inversable à la main si besoin (par ex si la caméra est montée à l'envers).

Voici le résultat de la simulation. Nous voyons bien les deux poteaux ainsi que leur centre.



2.3 L'intégration:
Pour intégrer les deux parties au reste du programme, nous avons créé une classe Tracker avec différentes méthodes avec les codes déjà faits. De cette manière, nous avons réussi à factoriser le code qui est en grande partie identique pour les deux parties (poteaux/balle) et nous aboutissons ainsi à un système de détection complet.
Le code factorisé correspond au calcul du masque avec comme seule différence la couleur utilisée (bleu ou vert) et aussi à celui d'initialisation de la webcam et préparation de la vidéo et de l'image.
Les parties différentes correspondent au calcul des centres selon besoin.

Pour les tests, nous créons une instance de la classe Tracker dans le fichier principal de test. Avec cette instance nous pouvons accéder aux méthodes de détection. D'abord, nous faisons la détection du but avec une méthode qui renvoie une liste des deux points. Le programme continue seulement si la collection contient deux points, sinon il lance un erreur de détection du but. Ensuite, nous utilisons un itérateur dans la classe Tracker pour calculer le centre de la balle à chaque frame qui renvoie les centres trouvés. De cette manière nous trouvons le centre à chaque frame et affichons tous les résultats. Voici le résultat de la simulation.



Les coordonnées de la balle en pixels sont affichées à chaque instant de la vidéo dans le terminal à droite.

3. Prédire la trajectoire de la balle

Considérons que nous connaissions toutes les positions de la balle* jusqu'au moment présent (pour chaque image). Il est possible d'en déduire la trajectoire (direction) de la balle et de calculer l'intersection de cette trajectoire avec la ligne du goal.

Les rebonds ou déviations sont considérés comme imprévisibles (donc nous ne prédisons pas le changement de trajectoire induit). Il est assez intuitif de positionner le goal aussi rapidement que possible sur l'intersection de sa ligne de déplacement et de la trajectoire de la balle. (voir 3.4)

*La liste des points précédents contient les positions (bruitées) détectées et non pas les positions corrigées, pour perdre le moins d'information possible.

3.1 Retard

Qu'est-ce que le retard? Pour simplifier admettons que la balle change de trajectoire. Le retard est le **temps caractéristique** qu'il faut au goal pour se positionner sur la prochaine position d'interception.

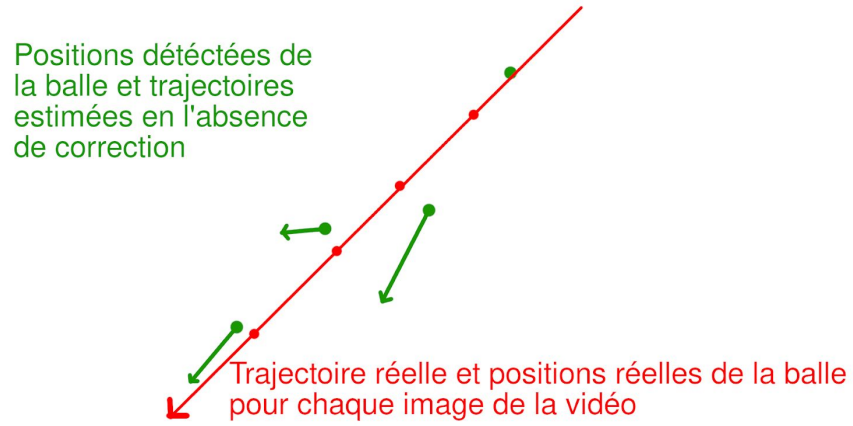
Le retard mécanique induit par le servomoteur et l'inertie du système n'est pas pris en compte dans cette partie, nous estimons simplement que pour une commande donnée, le servomoteur la suivra du mieux qu'il peut. Si, le retard du côté du programme étant minimisé, le servomoteur est quand même trop lent, c'est qu'il n'est pas adapté : il nous faudra utiliser un servomoteur plus rapide/performant.

C’est le retard ‘logiciel’ qui nous intéresse dans cette partie. Ce retard (lié au temps de calcul mais surtout à la nécessité de réduire le bruit en entrée) sera minimisé au possible. Pour une raison évoquée dans le N.B.,, le retard logiciel (Ri) sera donné en ‘images’, nous pouvons toujours retrouver le retard temporel (Rt) par
Rt = Ri / F
où F est la fréquence de rafraîchissement de la vidéo d’entrée.

Nota Bene: On remarque que, tant que la puissance de calcul est suffisante pour finir le cycle détection-prédiction-ordre avant l’arrivée de l’image suivante, le système est indépendant du temps. Ainsi, une balle deux fois plus rapide mais filmée par une caméra à un taux de rafraîchissement deux fois plus grand donnera exactement les mêmes résultats après l’étape de détection dans les deux cas.

Les deux causes principales du retard sont l’élimination du bruit en entrée et l’amélioration de la précision de la ligne de trajectoire.

3.2 Bruit et vitesse
La caméra du système réel étant loin d’être d’excellente qualité, et opérant dans des conditions de stabilité et d’éclairage non optimales, il est normal que les positions détectées sur la vidéo ne correspondent pas à la positions réelle de la balle.
Une caméra qui tremble pourrait donner l’impression que la balle zigzague : alors, bien que le mouvement global aille vers le but, la vitesse ‘instantanée’ de la balle calculée à partir de deux images aurait une direction complètement arbitraire.



C’est le rôle de la partie prédiction de filtrer cette entrée pour amortir le bruit sur cette liste de positions qui s’agrandit à chaque fois qu’une nouvelle image est fournie au système.

À noter que la clarté de l’image (résolution, absence de flou de mouvement) diminue le bruit perçu, mais moins que l’on pourrait s’y attendre.

3.3 Minimisation du bruit
Cette partie a nécessité plusieures itérations pour parvenir à un résultat satisfaisant.

Vu l'inertie de la balle, il est adapté de faire passer un filtre passe-bas sur les positions connues de la balle : nous appliquons le filtre sur la liste des abscisses et celle des ordonnées des positions.

3.3.1 Moyenne des positions

La première version de ce filtre, qui faisait une moyenne pondérée des dernières positions pour corriger le bruit sur la dernière image, avait le défaut majeur d'introduire un retard proportionnel au nombre de positions précédentes prises en compte. Or, un grand nombre de celles-ci étaient nécessaire pour corriger le bruit important présent dans nos vidéos de test (même avec une vidéo créée par logiciel, le module de détection créant des imperfections).

Le retard mesuré, en excès de 40 images si nous souhaitions avoir un bruit un tant soit peu minimisé, était bien trop important pour envisager de garder cette méthode.

3.3.2 Coupure FFT

Le principe de cette deuxième méthode est de transformer la liste des points (liste des abscisses, liste des ordonnées) en listes de fréquences. Les fréquences supérieures à une fréquence de coupure choisie sont annulées, et les listes de fréquences sont reconverties en listes de coordonnées, puis la dernière (plus récente) est extraite et constitue la position qui sera considérée.

La fréquence de coupure est un paramètre variable qu'il conviendra d'ajuster en fonction de la rapidité souhaitée et la stabilité de la position d'intersection prédite (un compromis inévitable).

<u>Avantages du filtrage par FFT:</u>

- Meilleur temps caractéristique (retard) à réduction de bruit équivalente

<u>Défauts du filtrage par FFT:</u>

- Phénomène d'interférence embêtant si la fréquence de coupure s'approche de 1 (la fréquence de coupure approchant la fréquence d'échantillonnage normalisée) et ralentissant le système en *augmentant* la fréquence de coupure à partir d'un certain seuil*
- Oscillations en cas de 'téléportation' de la balle**

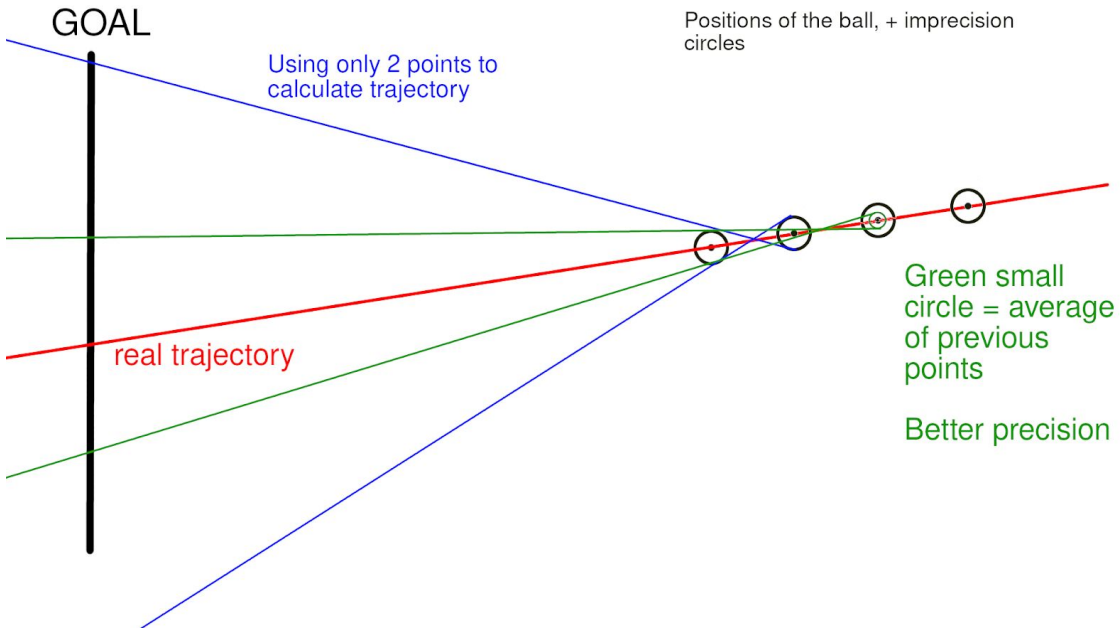
*en pratique, cela signifie que nous n'aurons jamais de bons résultats (bruit totalement filtré) avec un retard de moins de 3-4 images (2 images de retard => fréquence de coupure = 0.5, trop proche de 1, donc créant un début d'interférence). Mais il fallait s'y attendre.

** cela signifie qu'il faut que le programme se réinitialise à chaque remise en jeu

3.4 Prédiction de l'intersection

Nous considérons cette fois en mémoire une liste des points *filtrés*. À partir de deux points seulement, il est possible de calculer la direction du mouvement, mais c'est très imprécis, voir schéma ci-dessous.

Heureusement, il est possible de faire une moyenne d'un certain nombre de points précédents, de sorte à obtenir comme point précédent un point mieux 'aligné' à la vraie trajectoire de la balle. Nous gardons toujours comme point 'avant' le point filtré seul, pour réduire le retard et conserver l'information d'un changement brusque de trajectoire au mieux. Ce système introduit toutefois un retard supplémentaire, à donc utiliser avec modération.
En pratique la réduction de bruit est importante puisque l'imprécision se multiplie avec la distance au goal, et se multiplie avec l'inverse de la distance entre les points qui créent la ligne, mais une moyenne sur les 3 positions précédentes éloigne suffisamment les points pour que la ligne de trajectoire soit significativement plus précise.



(intervalles de confiance (exagérés) pour deux méthodes de prédiction de la trajectoire)

Les gains en précisions diminuent au fur et à mesure que l'on prend plus de points, il n'est donc pas intéressant d'augmenter plus la valeur de cette constante, d'autant plus que cela augmenterait notre retard.
On calcule ensuite le point d'intersection de la ligne du goal et de cette nouvelle ligne de trajectoire puis sa position par rapport au goal est calculée et renvoyée à la boucle principal.

Nota bene: l'imprécision augmente avec la proximité des points, donc quand la balle est lente. Dans ce cas, le gardien aura donc plus de temps pour se repositionner (de façon plus précise) au fur et à mesure que la balle se rapproche du but. Ce phénomène n'est donc pas néfaste à l'interception.

3.5 Performances théoriques

Bien sûr les performances dépendent de l'intensité du bruit en entrée et de la précision souhaitée, mais pour nos vidéos échantillons (voir partie 5), nous avons réussi à maintenir les variations (imprécisions) de la commande de position inférieures à la largeur du gardien de but, tout en gardant un retard de l'ordre de 10 images.

Pour différentes fréquences de rafraîchissement de la vidéo, si nous restons conservateurs, cela signifie* :

30 FPS => 0.33 sec

60 FPS => 0.16 sec (l'algorithme prédit le point d'intersection pour une trajectoire rectiligne plus vite qu'un humain)

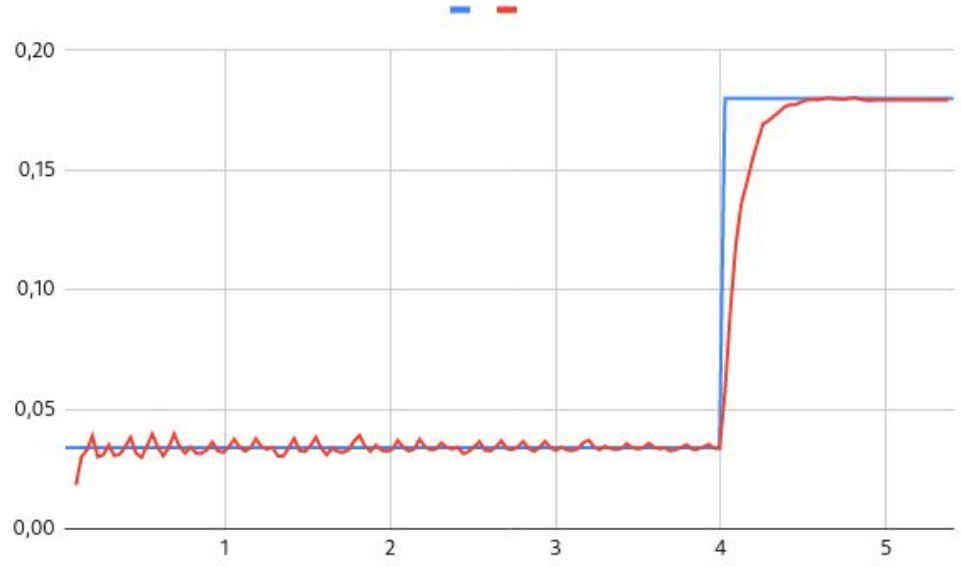
120 FPS => 0.08 sec

Bien sûr, les capacités de prédiction d'un humain, notamment quand des déviations sont en jeu (sans parler de la prévision du mouvement de l'adversaire) sont nettement supérieures à celle de la machine… pour l'instant.

Ci-dessous le graphe de la commande en position (rouge) pour une balle qui change de trajectoire brusquement (intersection change selon la fonction bleue) en fonction du temps (vidéo à 30fps pour calculer l'échelle temporelle)

On remarque l'amélioration de la précision lorsque la balle s'approche du goal (avant le changement de trajectoire)

commande de position (cm)



temps (secondes)

*Cela n'inclut bien sûr pas le retard mécanique induit par le servomoteur. Pour référence, un servo de haute performance produit un temps de réponse à 60° de 0.2 sec [2], et notre mécanisme a besoin d'un débattement de 90°.

4. Réalisation du premier prototype

4.1 Délimitation du système.

Le but final de notre projet est que le goal arrête les buts puis renvoie la balle. Nous avons décidé de nous focaliser sur la première étape : l'arrêt. Vu les contraintes dues au Covid-19, nous n'avons pas pu avancer plus loin .

Pour commencer, nous avons limité le système aux acteurs les plus importants : la balle, la cage, le goal et la canne qui le contrôle. Nous avons exclu les murs et les autres joueurs pour avoir un modèle initial simple.

La cage étant la seule entité immobile, elle nous servira de repère. Pour la canne du goal qui possède deux degrés de liberté, nous avons choisi de bloquer la rotation qui n'est pas utile pour l'arrêt et qui n'est pas contrôlée par le programme.

Du côté de notre programme, nous avons une Raspberry Pi qui est reliée à un servomoteur pour contrôler la translation du goal et une caméra pour obtenir la position des différents éléments du système.

4.2 Modélisation du prototype.

Tout d'abord, nous avons mesuré les éléments du babyfoot sur celui présent dans la cafet de l'Ensimag. Comme nous ne modélisons que la partie devant la cage, nous avons décidé de réaliser un modèle taille réelle pour ne pas rajouter de facteur lié au redimensionnement.

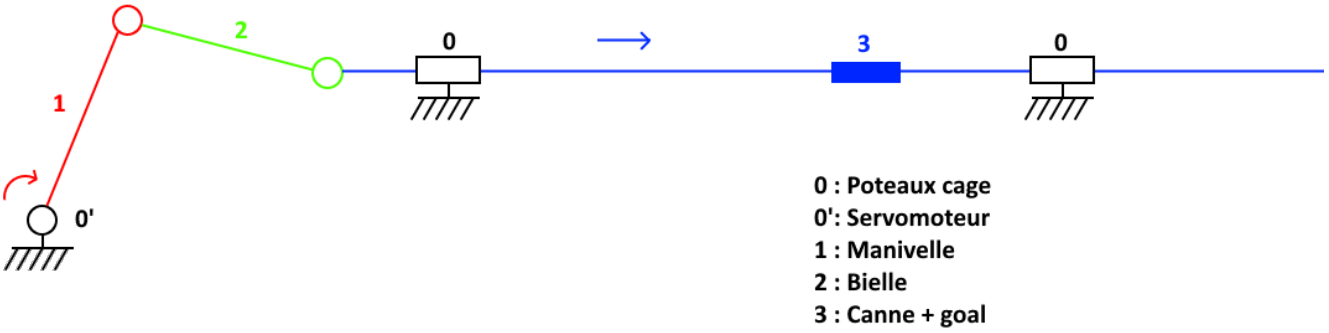
Comme le seul rôle des cages est de servir de repère, nous avons décidé de les fusionner avec les planches soutenant la canne du goal. De plus la différence entre la course du goal et la largeur des cages est de l'ordre de la largeur du goal, nous avons décidé de la négliger. Nous avons donc deux planches verticales distantes d'un peu plus de 20 cm modélisant la cage et servant de support à la canne du goal. Ces planches sont encastrées dans une plaque de bois servant de bâti et leur hauteur est égale à la distance sol-canne du système réel.

Pour le goal, nous avons juste pris une planche en bois de même hauteur et de largeur la base du goal. Nous avons ensuite collé la planche sur la canne avec un pistolet à colle. Pour la canne nous avons d'abord essayé une tige en bois mais à cause des frottements trop importants lors de la translation, nous l'avons changé pour une tige de métal. La liaison entre la canne et son support est une simple encoche guidant la tige créant ainsi une liaison pivot glissant. Pour annuler la rotation, due uniquement à l'impact de la balle, nous avons rajouté une tige derrière les jambes du goal pour limiter la rotation qui pourrait se remonter jusqu'au servomoteur et endommager le prototype.

4.3 Réalisation du prototype

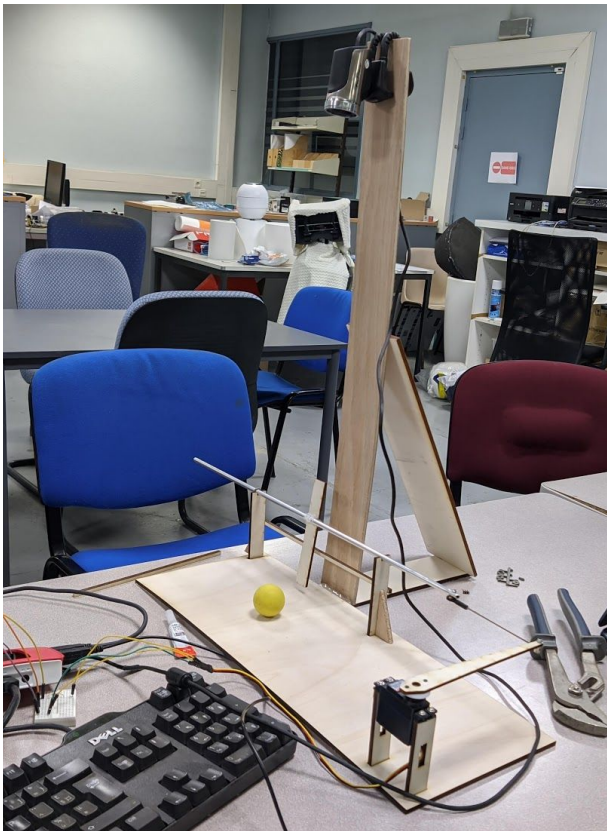
Pour la caméra, nous l'avons fixée sur une structure en bois afin qu'elle ait une vue de dessus sur l'ensemble du modèle.

Pour la réalisation du lien entre le servomoteur et la canne, nous avons opté pour un système bielle-manivelle. On obtient donc le schéma cinématique suivant (vue de dessus):



Possédant déjà une bielle, nous avons dimensionné la manivelle afin qu'un angle de 90° corresponde à une translation de 20 cm. Fixer ce rapport est important car il est utilisé dans le calcul de l'angle donné au servomoteur pour déplacer le goal à la position voulue.
La limite de l'angle à 90°, arbitraire, sert uniquement à ce que l'on ne se retrouvera pas avec une rotation trop importante qui engendrerait différents problèmes (arc-boutement, place importante, lenteur du servo sur de grand débattements, ...).

Nous avons ensuite calculé la position du servomoteur par rapport à la canne pour minimiser l'angle entre la bielle avec la canne afin de ne pas avoir d'arc-boutement. Une fois cette position connue, nous y avons encastré le support du servomoteur. Ce support comporte deux planches verticales permettant d'être au même niveau que la canne.



4.4 Résultats obtenus

Le prototype a été fini peu avant le confinement lié au covid-19 donc un seul test de notre programme a pu être réalisé. Durant le test le prototype n'a eu aucun problème mécanique mais nous avons pu voir que le programme faisait vibrer le goal. Nous avons identifié différentes sources comme les imprécisions de mesure et le traitement des données, problème résolu depuis. Nous n'avons pas pu retester sur le prototype à cause du confinement, à la place nous l'avons testé sur des vidéos basiques modélisant une trajectoire de la balle, voir suite.

5. Processus de validation du programme

Vu les restrictions d'accès au fablab pendant la période de confinement, nous avons mis au point un protocole permettant de vérifier le bon fonctionnement de notre programme

Entrées:
Vidéos faites sous logiciel de montage, à une résolution et qualité importante.
La balle suit des trajectoires rectilignes (éventuellement avec un changement de trajectoire)

Prévision des sorties:
Valeur (entre 0 et 20 cm à partir du 1er poteau) de la position du goal tel qu'il soit sur la trajectoire de la balle au temps donné.

Vérification des sorties:
On lance le programme sur l'entrée et on vérifie qu'à chaque temps, le programme assigne une position au goal qui corresponde à la position prévue.

Certains paramètres du programme, comme la fréquence de coupure pour le filtrage par FFT, sont ajustés avant les tests mais ne varient pas entre ceux-ci.

6. Conclusion

Durant ce projet, nous avons réussi à produire un programme capable de détecter la balle et les cages sur un flux vidéo, d'atténuer les imprécisions de mesure, de prévoir la trajectoire de la balle de manière précise et de commander le servomoteur. Pour valider le fonctionnement de notre algorithme, nous avons utilisé le prototype dans la mesure du possible avec le confinement et nous avons aussi créé des tests logiciels (vidéos sur mesures).

Nous avons réussi à piloter à la main le prototype, qui fonctionnait de façon satisfaisante, notons cependant que la commande envoyée par programme sera bien plus intensive (mécaniquement) que nos essais.

Nous avons eu des résultats montrant le bon fonctionnement de notre programme, il nous ne nous reste donc plus qu'à implémenter notre solution sur le système réel (liaison mécanique à améliorer si besoin et la partie commande à décommenter) pour obtenir un produit final opérationnel.

Ressources:

[1] Repérage de balle avec OpenCV: https://github.com/hemkum/ball-tracking-with-opencv et https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

[2] Specification du servomoteur utilisé: https://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-485hb/product