Amélioration de l'algorithme d'interception pour un robot badminton et adaptation de sa stratégie à son adversaire humain

Ing. G. Van Hoeke
Dr F. De Bruyne
ECAM – Bruxelles

Dr B. Depraetere
Dr S. Vandenplas
FMTC - Leuven

Dans le but de démontrer ses compétences ainsi que les possibilités offertes par la mécatronique, FMTC (Flander's Mechatronics Technology Centre) a développé un robot badminton : JADA. Cet article présente la logique implémentée sous forme de code dans le robot afin de rendre possible l'interception du volant. Nous veillerons à ce que celle-ci soit implémentée de manière optimale et robuste à toutes les améliorations et changements continuellement appliqués au robot.

Mots-clefs : Robot, Badminton, Algorithme, Matlab, Mécatronique.

In order to show its competences and the possibilities offered by mechatronics, FMTC (Flander's Mechatronics Technology Centre) *has developped a badminton robot : JADA. This paper focuses on the logic implemented in the robot through a code to make the interception of the shuttlecock possible. We will manage to implement it in an optimal way, robust to all the improvements and changes continously applied on the robot.*

Keywords : Robot, Badminton, Algorithm, Matlab, Mechatronics.

1. Introduction

1.1. Présentation de l'infrastructure

En 2008, avec le support du gouvernement flamand et de la K.U.L (« Katholieke Universiteit Leuven »), FMTC crée le premier robot badminton au monde : JADA. L'infrastructure alors développée est présentée dans la figure ci-dessous.

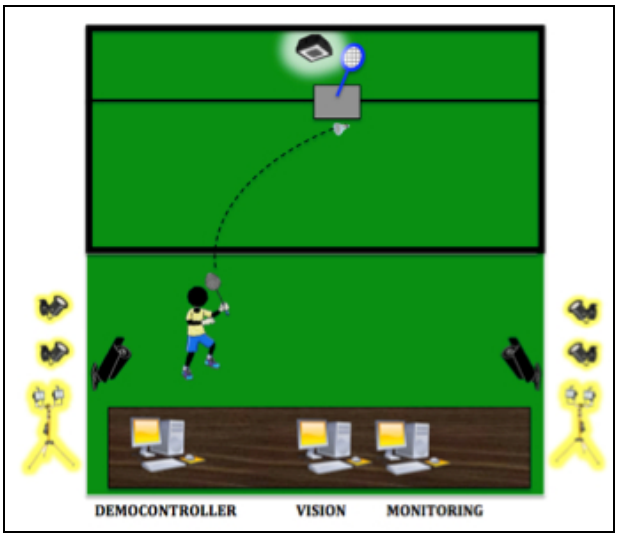

Figure 1 : Infrastructure développée

- JADA se situe dans une cage de sécurité impénétrable par le joueur lorsque le jeu est enclenché.
- Des caméras, aidées par des conditions spécifiques d'éclairage, traquent la trajectoire du volant.
- Le « Vision PC » fait tourner des modèles LabVIEW en temps réel.
- Le «Monitoring PC» affiche une interface de supervision LabVIEW des éléments hardware en fonctionnement.
- Le « Demoncontroller PC » compile en temps réel les codes C pour l'estimation de trajectoire du volant ainsi que pour l'interception. Il envoie des consignes à une plateforme de contrôle Triphase.

Tous les développements exposés dans cet article ont donc été encodés sous Matlab pour être compilés en code C lui-même finalement compilé en temps réel vers un niveau encore plus bas dans le « Democontroller PC ».

1.2. Notations

Le système de référence (dénommé « World Coordinates axes system » ou « WC axes system ») est fixé comme suit.

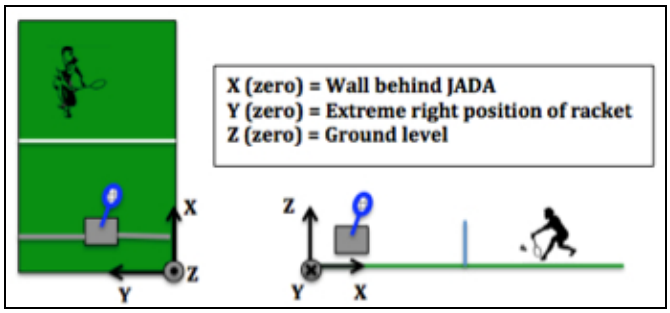

Figure 2 : WC axes system

Les notations relatives au robot lui-même sont décrites ci-dessous.

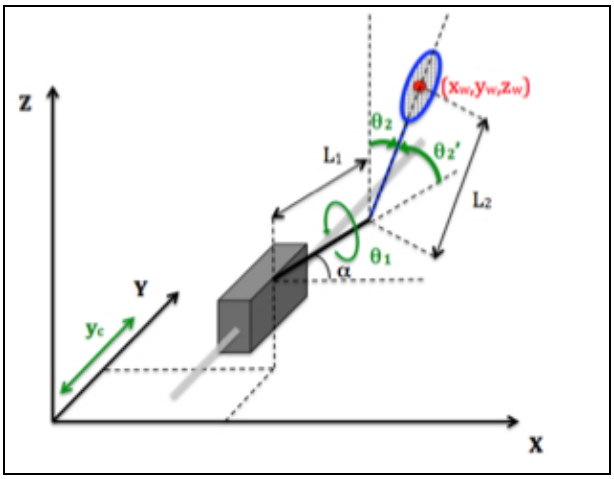

Figure 3 : Notations relatives au robot

Le robot possède 3 degrés de liberté (les « Robot Coordinates » ou « RC ») :

- Translation le long du rail (y_c)
- Rotation de la raquette autours du bras (θ_1)
- Angle de frappe (θ_2)

L_1, L_2 et α sont des paramètres de construction fixes.

2. Algorithme d'interception

2.1. Concept

Lorsqu'un être humain joue au badminton, il exécute instinctivement 2 actions :

- Localisation du volant sur le terrain
- Choix du meilleur coup à faire

Il n'y a pas de raison qu'il en soit autrement pour un robot ! De fait, 2 fichiers distincts ont été développés sous Matlab pour l'estimation de la trajectoire du volant d'une part et l'algorithme d'interception d'autre part. Nous aborderons ici l'étape de l'interception.

2.2. Rôle et place dans le process

Le déroulement des actions appliquées à chaque période d'échantillonnage peut se schématiser comme suit :

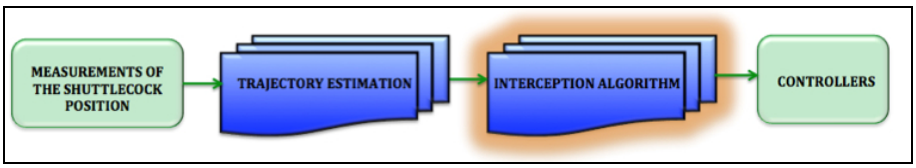

Figure 4 : Rôle et place de l'algorithme d'interception dans le process

1°) La position du volant est mesurée en temps réel à l'aide de 2 caméras et d'algorithmes de triangulation et de reconnaissance de forme.
2°) Une estimation de la trajectoire totale est prédite sur base d'une mémorisation des anciennes mesures confrontées à un modèle mathématique pour la trajectoire du volant en dynamique (filtre de Kalman).
3°) La trajectoire estimée est un ensemble de points exprimés en « WC » qui constitue l'entrée de l'algorithme d'interception. A ce stade, il s'agit de choisir le meilleur point à intercepter sur la trajectoire et de traduire ses coordonnées de « WC » vers « RC ».
4°) L'algorithme d'interception envoie 3 consignes aux régulateurs :

- Position de frappe en « RC » $[y_c;\theta_1;\theta_2]$
- Vitesse de frappe $[v_{hit}]$
- Moment de la frappe $[t_{hit}]$

2.3. Conversions « WC » ⇔ « RC »

Le principal rôle de l'algorithme d'interception réside donc en la traduction des coordonnées [x,y,z] du volant par rapport au repère des « World Coordinates » sur le terrain vers les « Robot Coordinates » [y_c;θ_1;θ_2] compréhensibles par JADA. Le moment de la frappe est intrinsèquement lié au point [x,y,z] de la trajectoire que l'on désire intercepter et le choix de la vitesse de frappe sera discuté plus tard (section 4.2).

Méthodologie

Le point d'interception est considéré comme étant le centre de la raquette exprimé par [x_w;y_w;z_w]. Pour trouver le lien entre les « WC » et les « RC », il suffit alors d'effectuer un changement de base de xyz vers x'y'z' via les paramètres du robot. Dans x'y'z', les coordonnées de ce point sont invariablement les suivantes : [x' = 0 ; y' = 0 ; z' = L_2].

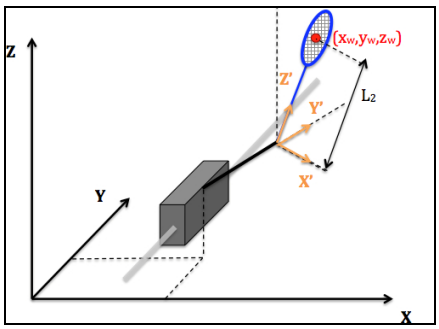

Figure 5 : De la base xyz à la base x'y'z'

Ce changement de base est appliqué par le calcul matriciel suivant :

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = T * \begin{bmatrix} 0 \\ 0 \\ L_2 \\ 1 \end{bmatrix}$$

Où T est la matrice de transformation totale obtenue par la succession des déplacements simples (translations et rotations) à travers les paramètres du robot. Le calcul simplifié[1] permet de trouver les résultats de la page suivante.

[1] Voir «Improve the interception algorithm of a badminton robot and adapt its strategy to its human opponent » G. Van Hoeke pour les calculs complets non simplifiés.

RC ➔ WC

Soit $[x_h,y_h,z_h]$ le centre du rail, on trouve les coordonnées « WC » du centre de la raquette exprimées en fonction des paramètres du robot :

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} x_h + L_2 * sin\alpha * cos\theta_1 * sin\theta_2{}' + (L_1 + L_2 * cos\theta_2{}') * cos\alpha \\ y_h + y_c - L_2 * sin\theta_1 * sin\theta_2{}' \\ z_h - L_2 * cos\alpha * cos\theta_1 * sin\theta_2{}' + (L_1 + L_2 * cos\theta_2{}') * sin\alpha \\ 1 \end{bmatrix}$$

Pour des RC $[y_c,\theta_1,\theta_2]$ fixées, la solution dans ce sens est unique.

WC ➔ RC

Le système d'équations trouvé précédemment peut être utilisé pour isoler les « RC » et les exprimer en fonction des « WC », ce sont ces équations qui fournissent la « position de frappe » de l'algorithme d'interception.

$$cos\theta_2{}' = \frac{(x_w - x_h) * cos\alpha + (z_w - z_h) * sin\alpha - L_1}{L_2}$$

$$cos\theta_1 = \frac{(x_w - x_h) * sin\alpha - (z_w - z_h) * cos\alpha - L_1}{L_2 * sin\theta_2{}'}$$

$$y_c = y_w - y_h + L_2 * sin\theta_1 * sin\theta_2{}'$$

Dans ce sens, il y a plusieurs solutions possibles puisque deux angles égaux mais de signe contraire donnent lieu à un même cosinus. Etant donné l'aspect trivial de la considération d'un angle θ_2' (angle de frappe) négatif, nous comprendrons que la multiplicité des solutions provient surtout de θ_1 (angle de rotation). Physiquement, cet aspect mathématique peut se comprendre par le fait que le robot peut frapper le volant « par la gauche » ou « par la droite » comme l'indique la figure ci-dessous.

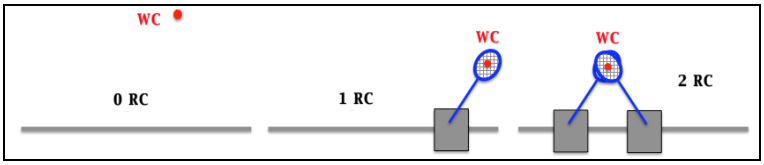

Figure 6 : Maximum 2 solutions possibles lors de la conversion WC ➔ RC

3. Développement d'un code fonctionnel

Avant de considérer toute optimisation des performances de visée ou même une adaptation de la stratégie de jeu, il s'agit de construire une base simple et fonctionnelle.

3.1. Séquence logique de base

La base du code a été pensée comme présentée ci-dessous.

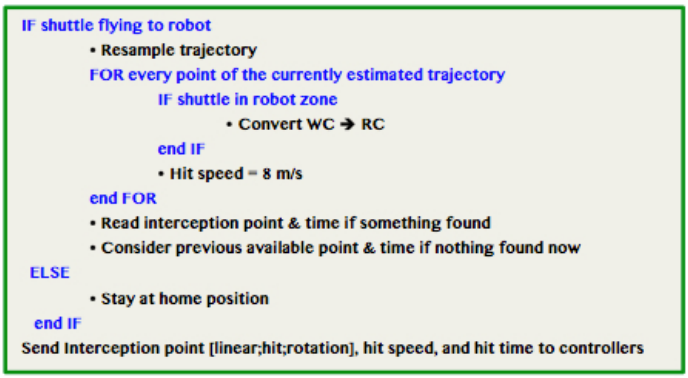

Figure 7 : Séquence logique de base

- Si aucun volant en déplacement vers le robot n'est détecté, le robot reste au centre du rail. Sinon, le cœur de l'algorithme est lancé.
- Une interpolation est opérée pour maximiser le nombre de points considérés sur la trajectoire estimée et ainsi augmenter la résolution des possibilités.
- Tous les points de la trajectoire rééchantillonnée qui appartiennent à la zone atteignable par le robot sont traduits en « RC ». Aucun choix n'est appliqué pour l'instant et le point choisi n'est autre que le dernier de la trajectoire puisqu'il écrase tous les précédents.
- La vitesse de frappe est considérée comme constante pour l'instant. Nous considérerons une exploitation intelligente de ce paramètre plus loin (section 4.2).
- L'algorithme est appelé à chaque période d'échantillonnage et fournit ainsi des consignes de position, temps et vitesse de frappe continuellement mises à jour.

3.2. Simulation

Avant de tester l'algorithme sur le process réel, il convient de vérifier en simulation que tout se passe comme prévu. Pour ce faire, un jeu entre 2 personnes a été enregistré afin de fournir une entrée à notre code. Nous

allons donc simplement vérifier le choix de positionnement de l'algorithme sur un jeu déjà existant sans pour autant influencer le retour du volant.

La figure ci-dessous présente le résultat de cette simulation.

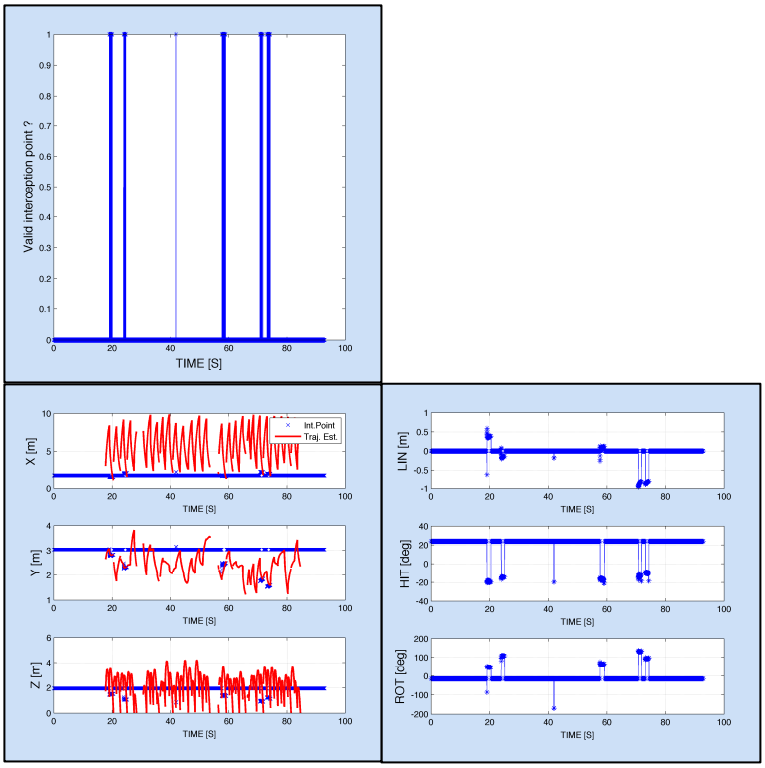

Figure 8 : Simulation de la séquence logique de base

- Cadre supérieur gauche : Un point d'interception faisable a-t-il été détecté ? (1 = oui ; 0 = non)
- Cadre inférieur gauche : La trajectoire du volant est tracée en rouge et les coordonnées du point d'interception choisi par le robot sont marquées par les croix bleues (« WC »).
- Cadre inférieur droit : traduction des points d'interception en « RC ».

Interprétation du résultat : La trajectoire est bel et bien interceptée sur ses derniers points estimés comme prévu provisoirement dans notre séquence de base. La pertinence de la conversion de coordonnées a été vérifiée en s'assurant que les « RC » repassant par le fichier de conversion « RC ➔ WC » ramenaient bien aux « WC » initiaux de la trajectoire estimée.

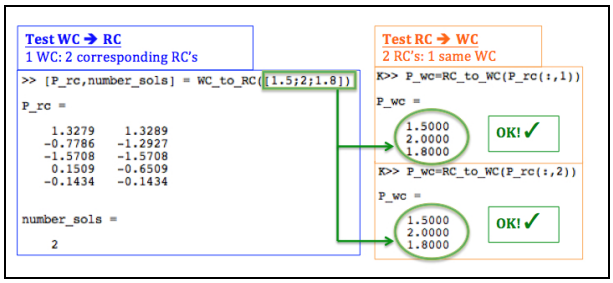

Figure 9 : Test de pertinence de la conversion RC ⇔ WC

3.3. Test sur process

Une fois que la simulation a permis de supposer une implémentation correcte de la théorie dans le code Matlab, ce dernier a été compilé en code C afin de tourner en temps réel sur le «Democontroller PC».

Il en est ressorti le résultat de la figure 10 :

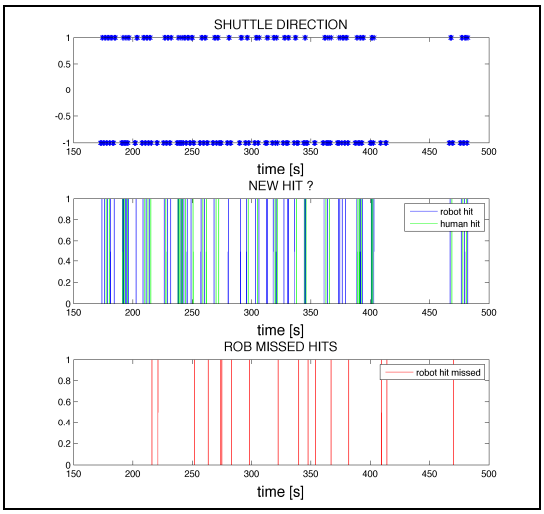

Figure 10 : Test de la séquence logique de base sur le process réel

Interprétation du résultat : Le premier graphe indique la direction du volant (négatif de l'humain vers le robot) Le deuxième graphe signale un nouveau coup à chaque changement brutal de direction. Le troisième graphe détecte tout coup manqué par une perte du signal de direction. 17 coups ont été manqués sur 76 volants envoyés au robot. Cependant, une analyse plus

approfondie a permis de trouver des causes externes à l'algorithme d'interception.

1°) 15 coups ont été ratés car les trajectoires correspondantes ne croisaient pas la zone d'action de JADA.

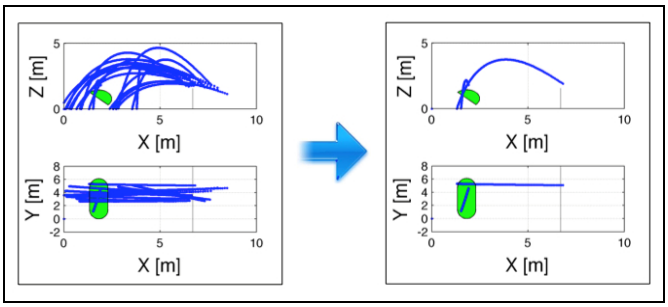

Figure 11 : 15 trajectoires en dehors du range de JADA (vert)

2°) 1 coup a été raté car le robot n'a pas eu assez de temps pour atteindre la position de consigne (frappe en t=273.6s alors qu'une trajectoire est toujours disponible). S'en est suivi un rebond également raté (visible sur la figure ci-dessus et par le faible laps de temps de 0.161s entre les 2 coups ratés).

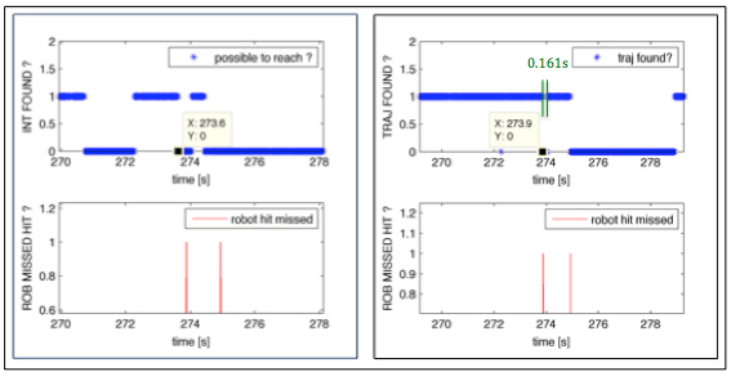

Figure 12 : 1 coup raté par manque de temps et 1 rebond 0.161s plus tard

Rien n'a donc permis de remettre en cause la séquence logique de base.

3.4. Design d'un filtre de sécurité

Afin de minimiser les risques de casse pour la raquette, un filtre de sécurité a été mis en place pour limiter les variations brusques de consigne de

position (rotation et translation), les régulateurs suivant déjà la consigne (variable) aussi vite que possible.

Filtrage sur le temps de montée

Les forces potentiellement dangereuses se développant suite à des accélérations inadmissibles, le filtrage sur le temps de montée s'assimilera simplement à une limitation d'accélération.

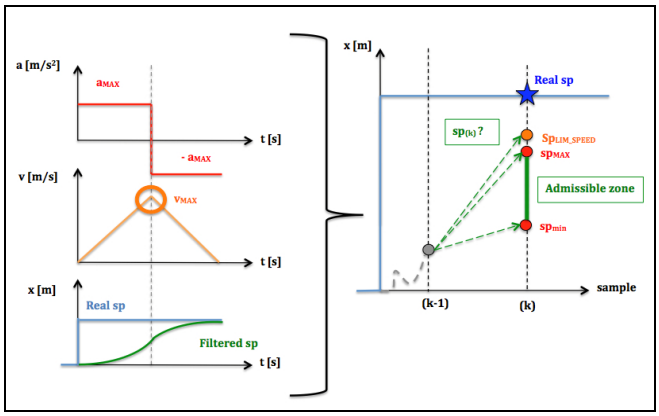

Figure 13 : Filtrage sur le temps de montée

La consigne filtrée $sp_{(k)MAX/min}$ varie suivant une accélération/décélération maximale, dans le domaine discret cette limitation peut être exprimée par :

$$sp_{(k)_{MAX/min}} = (+/-a_{MAX}) * sign(sp_{(k)} - sp_{(k-1)})(T_s)^2 - 2 * sp_{(k-1)} + sp_{(k-2)}$$

La notion de vitesse maximale est utilisée comme pivot pour décider de l'accélération/décélération qui devra être appliquée à l'échantillon suivant. Il s'agit de la vitesse stockée à l'échantillon k telle que la décélération maximale ne suffit plus pour freiner jusqu'à la consigne non filtrée sans dépassement. Tant que la vitesse maximale n'est pas atteinte, la position de consigne $sp_{(k)MAX/min}$ tend vers la consigne non-filtrée $sp_{(k)}$ avec l'accélération maximale. Une fois que cette vitesse est atteinte, la consigne filtrée freine avec la décélération maximale autorisée.

Filtrage à proximité de la consigne non-filtrée

Etant donnée l'importance relative du bruit tout près de la consigne non-filtrée, la méthode du temps de montée doit être abandonnée au-delà d'un certain seuil (fonction des oscillations jugées acceptables). Au-delà de ce

seuil, le filtrage est opéré par un premier ordre aux coefficients arbitrairement fixés pour obtenir une dynamique jugée assez lente :
$SP_{filtered} = 0.9*SP_{(k-1)} + 0.1* SP_{(k)}$.

Résultat en simulation :

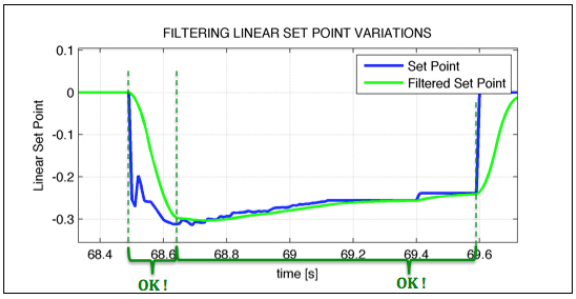

Figure 14 : Résultat du filtre en simulation

Résultat en temps réel :

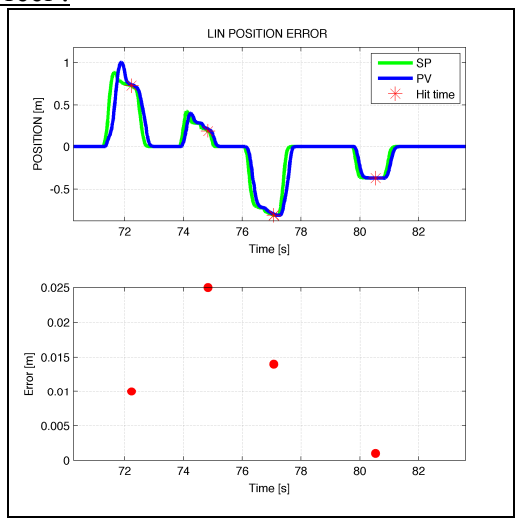

Figure 15 : Résultat du filtre en temps réel (erreur au moment de la frappe)

Il y a naturellement une différence entre la consigne filtrée (SP en vert) et la valeur du process (PV en bleu) que les régulateurs arrivent à suivre. De fait, les accélérations maximales fixées ne sont pas forcément respectées. Ce préfiltrage au niveau des consignes est donc un premier pas mais il y a également un travail à fournir au niveau du contrôle.

4. Amélioration des performances de visée

Dans la rubrique précédente, chaque point de la trajectoire du volant était considéré tour à tour en écrasant le précédent et le tout dernier point de la trajectoire était alors défini comme le point d'interception. C'est à ce niveau qu'un choix peut être fait et nous disposons de 3 possibilités :

- Un point défini en « WC » peut donner lieu à maximum 2 positions pour JADA.
- Pour un unique point d'interception, le volant peut être frappé plus ou moins fort et donc envoyé plus ou moins loin.
- Sur la trajectoire incidente, il existe toute une grappe de points atteignables par le robot. Il est donc possible de choisir le point jugé le plus favorable.

Ces 3 sources de choix ont été exploitées de telle sorte que le volant atterrisse le plus près possible d'une cible définie du côté de l'adversaire.

Fixons dès à présent certaines notations indispensables pour les considérations qui suivent : on définit le vecteur de frappe ainsi que ses angles ϕ_1 et ϕ_2 comme présenté dans la figure 16.

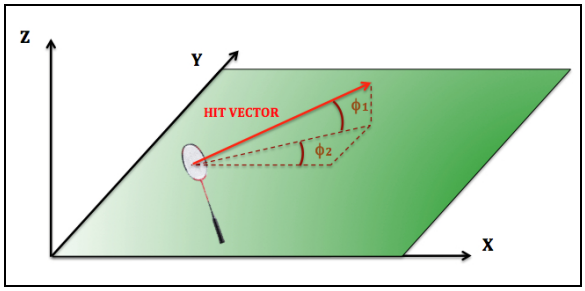

Figure 16 : Vecteur de frappe et angles ϕ_1, ϕ_2

4.1. Optimisation de la conversion WC ➔ RC

Comme l'illustre la figure 6, il peut arriver que 2 configurations du robot permettent d'intercepter un même point en « WC ». Dès lors, l'optimisation s'avère être relativement simple : la configuration du robot à choisir est celle pour laquelle la projection du vecteur de frappe sur le plan xy se rapproche le plus possible de la direction pointant vers la cible.

Le critère est donc : min(abs($\phi_{2possible} - \phi_{2desired}$)).

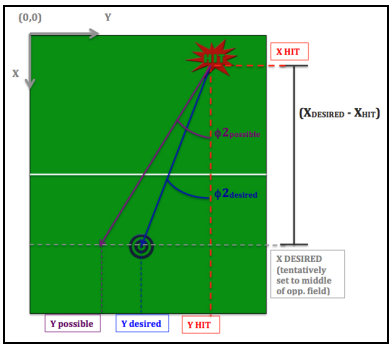

Figure 17 : Optimisation de WC ➔ RC

4.2. Optimisation de la vitesse de frappe

L'étape précédente permettait de se rapprocher grossièrement de la cible. Toujours pour un même point d'interception en « WC », il est possible de se rapprocher encore plus de la cible en jouant sur la vitesse de frappe.

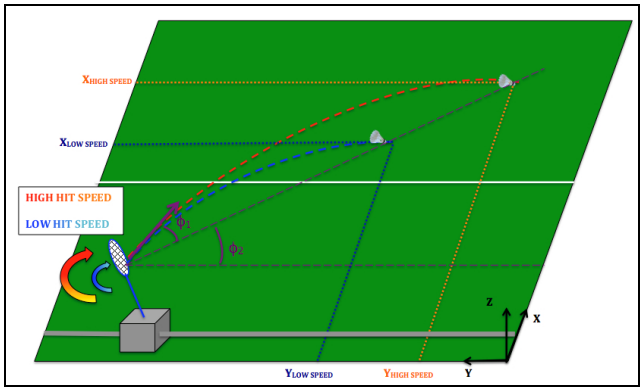

Figure 18 : Optimisation de la vitesse de frappe

Le critère est donc : min(distance entre cible et point d'atterrissage) et la vitesse de frappe est donc un moyen de faire tendre le point d'atterrissage estimé vers la cible définie en x et y sur la partie de terrain de l'adversaire.

Une estimation de la trajectoire selon les paramètres de la frappe effectuée est donc nécessaire. Cependant, au vu du temps de calcul considérable que cela impliquerait, il n'est pas envisageable de faire tourner l'algorithme d'estimation de la trajectoire pour chaque coup envisagé par le robot. C'est

pourquoi la décision s'est portée sur une quantification des différentes combinaisons de valeurs au travers d'une Look-Up Table.

Cette Look-Up Table a été générée offline en faisant tourner les calculs d'estimation de trajectoire pour plusieurs positions du robot. Pour chaque position, plusieurs vitesses de frappe ont été envisagées et ont mené à un point d'atterrissage propre stocké en mémoire.

En temps réel, l'algorithme fait alors appel à cette Look-Up Table en assimilant le point en cours au point le plus proche enregistré et sélectionne la vitesse de frappe menant au point d'atterrissage le plus proche de la cible.

Le choix de la Look-Up Table a pour unique but de minimiser la charge de calcul en temps réel. Il ne change en rien la philosophie d'optimisation de la vitesse de frappe.

4.3. Optimisation du choix pour le point d'interception

Jusqu'ici, les optimisations considérées étaient concentrées sur un point en « WC » à la fois. Or, chanceux que nous sommes, il y a généralement toute une grappe de points que JADA est en mesure d'intercepter. Dès lors, une fois que les 2 optimisations décrites ci-dessus ont été opérées sur chaque point, il est encore possible de choisir le meilleur point à intercepter sur cette portion de trajectoire incidente !

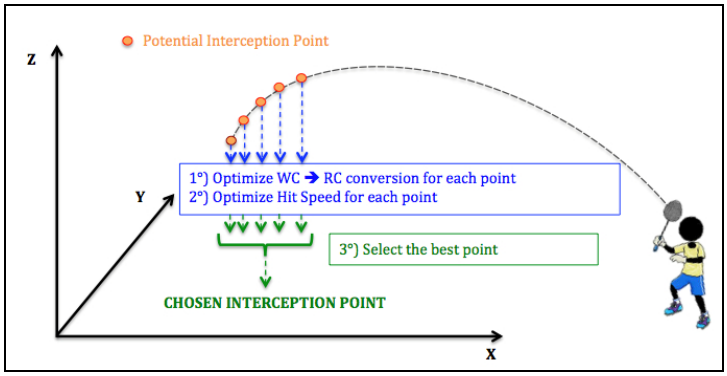

Figure 19 : Optimisation du choix pour le point d'interception

Le critère est le même qu'au point précédent.

4.4. Résultats en simulation

La figure 20 illustre le choix d'un point d'interception parmi tous les points à la portée de JADA. Le point choisi (petit cercle vert) mène bien au point d'atterrissage possible (croix rouges) le plus proche de la cible (grand cercle bleu).

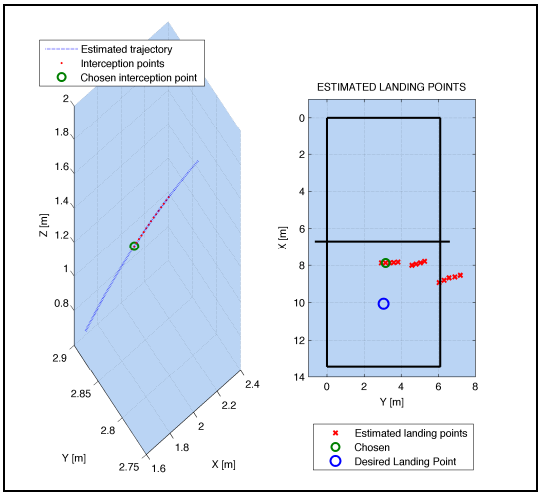

Figure 20 : Choix du point d'interception sur une trajectoire incidente

La figure 21 affiche un jeu entier simulé (le même qu'à la figure 8 mais décalé en x de manière à rendre plus de points atteignables).

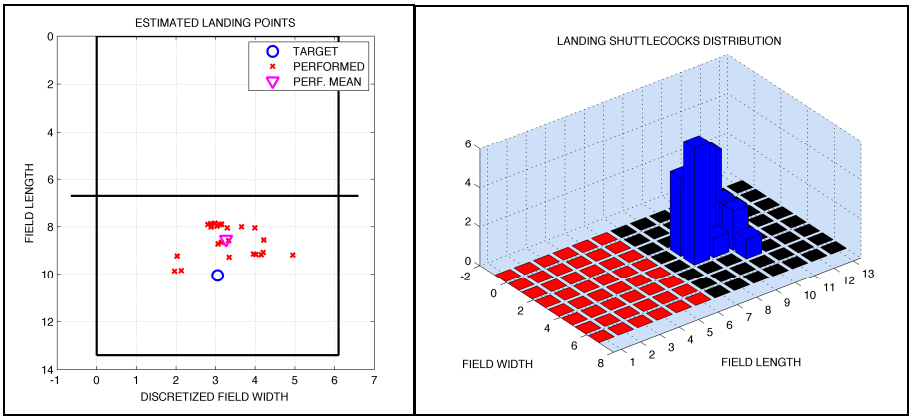

Figure 21 : Simulation de l'amélioration des performances de visée

Malgré les 3 degrés de liberté fortement contraignants du robot, une nette influence de la tendance des points d'atterrissage a pu être obtenue. Des résultats encourageants ont été obtenus sur le process réel mais seuls les résultats de la simulation étaient disponibles lors de la rédaction de cet article.

5. Adaptation de la stratégie à l'adversaire humain

Maintenant que la tendance d'atterrissage des volants peut être influencée, il ne reste plus qu'à adapter la cible choisie par l'algorithme en fonction de nos desiderata. 3 modes de jeu ont été implémentés dans le code.

5.1. Mode 1 : Le joueur choisit une cible

Ce mode n'inclut rien de spécifique par rapport à ce qui a été décrit précédemment si ce n'est que la cible peut être choisie librement par le joueur au cours du jeu.

5.2. Mode 2 : Le joueur choisit un niveau de difficulté

Le joueur décide lui-même du niveau de difficulté du jeu (easy ou insane). L'algorithme enregistre les coordonnées x et y de la dernière frappe du joueur et adapte son tir en fonction du choix encodé.

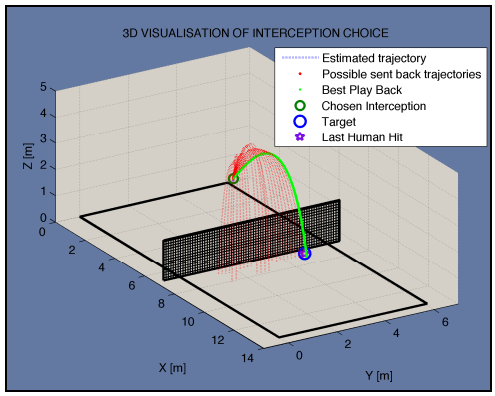

Figure 22 : Easy level (cible = dernier tir du joueur)

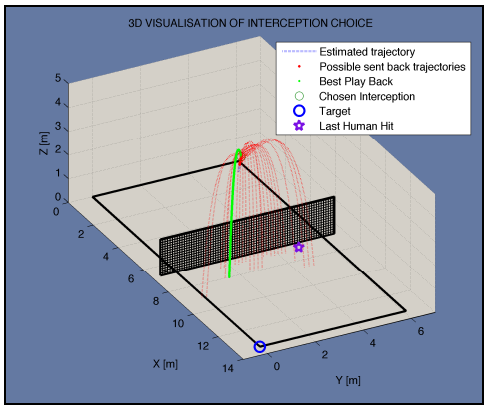

Figure 23 : Insane level (cible = opposé du dernier tir du joueur)

Le choix du mode est graduel et exposé au joueur via une jauge réglable :

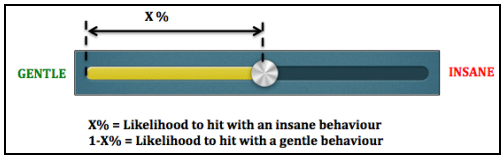

Figure 24 : Jauge de niveau réglable

5.3. Mode 3 : Le robot s'adapte au niveau du joueur

Il s'agit ici d'un mode entièrement automatique au cours duquel l'algorithme garde en mémoire les 5 derniers coups du joueur et juge de sa virtuosité suivant certains critères laissés à l'appréciation de l'utilisateur. La position du curseur sur la jauge de niveau est alors automatiquement ajustée aux performances de l'adversaire.

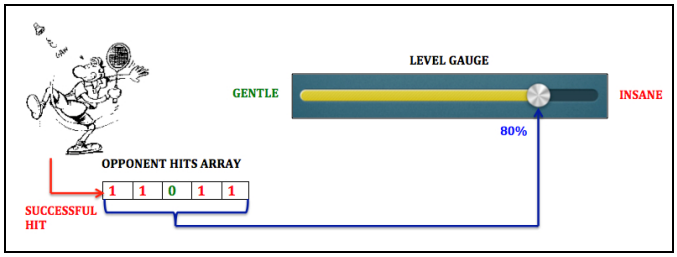

Figure 25 : Mode automatique

Voici le résultat du mode automatique soumis à la simulation de la figure 8 :

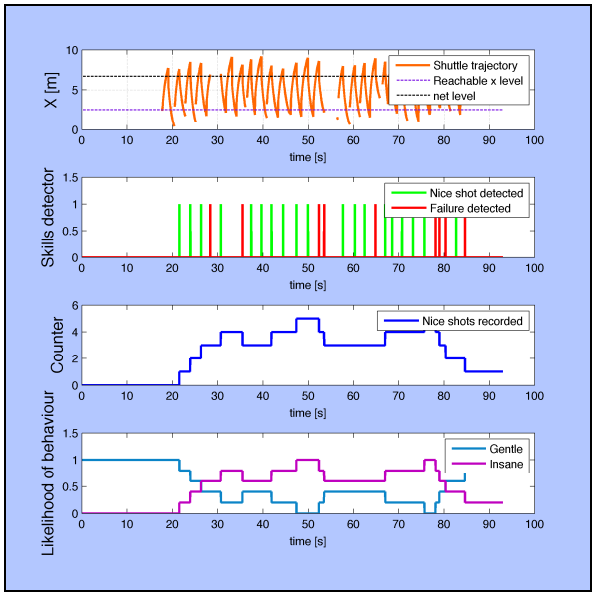

Figure 26 : Mode automatique sous simulation

6. Aperçu final du code

- Open the look-up table
- Define the level of the human's skills if the auto mode is on

IF shuttle flying to robot
- if new hit from human, then determine a new desired landing point
- resample trajectory

FOR every point of the currently estimated trajectory
IF shuttle in robot zone
- define ϕ_2 & kick-off distance from JADA to target
- Convert World Coordinates to Robot Coordinates

(if 2 possibilities, choose the one with ϕ_2 closest to desired ϕ_2)

IF ϕ_1 and ϕ_2 aiming in a consistent direction
- read hit speed in Look-up table
- score the point according to absolute distance from desired landing point

end IF

end IF
- remember the interception point with highest score

end FOR
- Read interception point if something found
- Take previous interception point if nothing found now and a previous value is available

ELSE
- Stay at home position

end IF
- Filter rotation and linear set point
- Send Interception point [$linear_{filtered}$;hit;$rotation_{filtered}$], hit speed, and hit_time to controllers
- Send assessed and expected landing points and the estimated level of the human player

Figure 27 : Aperçu final du code

- Ouverture de la Look-Up Table et définition du niveau si mode auto.
- Si aucun volant en déplacement vers le robot n'est détecté, le robot reste au centre du rail. Sinon, le cœur de l'algorithme est lancé.
- Définition d'une cible en accord avec le mode en cours si un nouveau coup du joueur a été détecté.
- Une interpolation est opérée pour maximiser le nombre de points considérés sur la trajectoire estimée.
- Tous les points de la trajectoire rééchantillonnée qui appartiennent à la zone atteignable par le robot sont traduits en «RC» avec optimisation. Si le point considéré est tel que JADA vise dans une direction non-triviale, lire la meilleure vitesse de frappe dans la Look-Up Table et garder ce point en mémoire seulement si on se rapproche plus de la cible qu'avec le point précédent.
- Si un point a été trouvé, le prendre en compte. Sinon, prendre en compte le précédent.
- Filtrer les consignes de rotation et de translation.
- Envoyer les consignes aux régulateurs ainsi que l'information des points d'atterrissage prévus et du niveau estimé du joueur.

7. Conclusion

De manière non exhaustive, les principales contributions de ce travail résident en :

- Une trace papier des calculs pour l'algorithme et leur implémentation testée dans un code matlab.
- Une structure globale de code intuitive testée en simulation et sur process réel.
- Une visualisation (indispensable pour un tel projet) des concepts développés et des résultats à travers de nombreuses figures.

Par ailleurs, certaines difficultés ont également été mises en exergue. Principalement exposée dans cet article, nous retiendrons surtout la suivante :

- Notre code ne calcule que des consignes ! Gardons à l'esprit qu'il reste un intermédiaire entre ce que nous calculons et ce que le process reçoit comme information : les régulateurs.

8. Sources

8.1. Sites internet et fichiers en ligne

- www.mathworks.nl
  (consulté régulièrement de Octobre 2013 à Mai 2014)
- https://www.khanacademy.org/math/linearJalgebra/matrix_transformations
  (consulté le 26 Décembre 2013)

8.2. Ouvrages et documents écrits

- A.COOKE, *Computer Simulation of shuttlecock trajectories*, Sports Engineering, 13 May 2002

8.3. Documentation interne à FMTC

- S.GILIJNS, *Knowledge Transfer: Interception Algorithm of Badminton Robot*

  (consulté régulièrement de Octobre 2013 à Mai 2014)

- S.GILIJNS, *"Interception.point.zip" containing both algorithms for old and new robot in m files* (accessed :fluently)

  (consulté régulièrement de Octobre 2013 à Mai 2014)

NB : Ce projet faisant l'objet d'une idée originale interne à FMTC, les sources citées ci-dessous ont simplement servi à fournir les outils techniques nécessaires à son bon développement. Ainsi, la construction de la logique présentée dans ce document n'est tirée d'aucune source externe.