Robots footballeurs : Commande par optimisation de trajectoire polynomiale

Gabin LEMBREZ – avec la relecture de Javier OJEDA

Culture Sciences de l'Ingénieur

Édité le 22/10/2021

école normale supérieure paris—saclay

Élève de l'ENS Paris-Saclay, Gabin Lemprez, au cours de sa première année en Sciences pour l'Ingénieur (année SAPHIRE), a réalisé ce projet dans le cadre de l'enseignement sur les méthodes d'identification paramétrique et d'optimisation.

La commande de robots peut être envisagée sous de nombreux angles. Une des méthodes consiste à reconnaître, lorsque c'est possible, un problème d'optimisation. Dans le cas de robots footballeurs, nous allons détailler une façon d'envisager ce problème sur la gestion des déplacements élémentaires. En effet, à tout instant lors d'un match, un robot cherche à atteindre une position de consigne fournie par le coach tout en évitant les collisions avec les autres joueurs sur le terrain.

Il s'agit donc de transformer un problème d'évitement en problème d'optimisation. À cette fin nous devrons définir un modèle puis construire le critère sur la base duquel sera conduite l'optimisation. Enfin, la méthode développée sera confrontée à une méthode classique d'évitement.

1 – Introduction

La *RoboCup* est une compétition opposant deux équipes de robots dans un match de foot. Chaque équipe commande ses robots via un ordinateur appelé « coach » qui élabore la stratégie suivie ainsi que les trajectoires correspondantes.

Figure 1 : Robots footballeurs sur le terrain

Le projet d'optimisation traité dans cette ressource concerne la gestion des déplacements de robots footballeurs dans le cadre d'une compétition au format de la *RoboCup*. La démarche suivie consiste en la modélisation d'un phénomène : dans notre cas, un déplacement de robot ; puis en son

optimisation au sens d'un critère que nous définirons par la suite, au moyen des algorithmes étudiés en cours (descente de gradient, méthode de Newton, Gauss-Newton, …).

Tous les programmes mentionnés sont écrits dans le langage *python* et le simulateur utilisé pour piloter les robots en temps réel est le logiciel *grSim*.

2 – Contexte

2.1 - Loi de commande des robots

Dans cet article, nous allons chercher à créer la trajectoire qu'un robot devra suivre au cours du temps. Or, dans le logiciel utilisé, les robots sont commandés en vitesse. Il s'agit d'être capable de définir à chaque pas de la simulation un vecteur vitesse. Seuls la direction et le sens de ce vecteur comportent un intérêt pour notre étude. En conséquence, la norme du vecteur sera toujours la même quelle que soit la méthode de calcul de trajectoire utilisée : la vitesse est proportionnelle à la distance entre le robot et l'objectif, avec une saturation.

$$V = max\left(V_{sat}, P \times \sqrt{\left(x_{objectif} - x\right)^2 + \left(y_{objectif} - y\right)^2}\right)$$

Les valeurs V_{sat} et P ont été ajustées pour assurer que le robot se déplace le plus souvent possible à sa vitesse maximale tout en arrivant sur la balle avec une vitesse suffisamment faible pour pouvoir l'intercepter sans qu'elle rebondisse.

2.2 - Méthode concurrente

2.2.1 - Méthode des champs potentiels

Dans l'objectif de quantifier les performances de la méthode de calcul développée, il convient de définir une méthode de référence pour calculer la trajectoire d'un robot. Nous retiendrons la méthode qui a été la plus largement employée par les élèves : la méthode des champs de potentiels. L'objectif est un point attractif et chaque obstacle se voit attribuer un champ répulsif. Les différents champs sont sommés et le robot se « laisse tomber » dans le champ global ainsi obtenu. Cette méthode est simple à mettre en place, mais elle comporte des défauts. En effet, dès que l'objectif est dans une zone avec un fort champ répulsif, le robot peut être très ralenti, voire stoppé. En outre, les robots de deux équipes adverses peuvent se bloquer l'un l'autre dans une situation d'équilibre avec la balle au milieu.

2.2.2 - Spécificités du champ retenu

Le champ répulsif que nous utiliserons par la suite, représenté figure 2, présente les caractéristiques suivantes :

- Intensité de la répulsion en $1/r$ et n'intervient qu'à une distance limitée de l'obstacle
- Le champ est « tournant ». Notons $\mathrm{R} = \sqrt{\mathrm{x}^2 + \mathrm{y}^2}$ et Posons $\tau = 1 - \mathrm{e}^{-\mathrm{pR}}$. On peut définir les vecteurs du champ répulsif par un angle θ donné par $\theta = \arg(\mathrm{u} + \mathrm{iv})$, où $\mathrm{u} = \frac{(1-\tau)\mathrm{x}+\tau\mathrm{y}}{\mathrm{R}}$ et $\mathrm{v} = \frac{(1-\tau)\mathrm{y}-\tau\mathrm{x}}{\mathrm{R}}$. Le coefficient p détermine à quel point le champ tourne.
- Le sens de rotation du champ répulsif dépend du demi-plan par lequel arrive le robot.

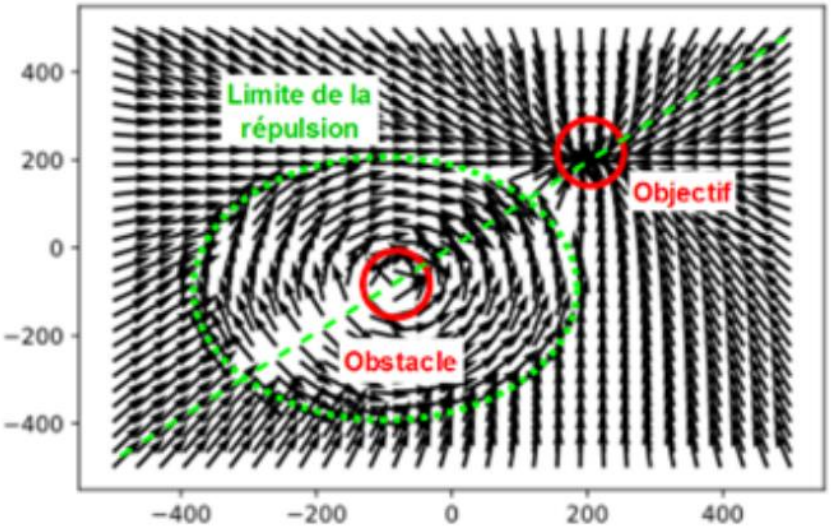

Figure 2 : Allure du champ obtenu

3 – Cas statique

Pour commencer, cherchons comment aborder le problème dans un cas simple. À un instant t donné, le plan (x,y) du simulateur est pris en photo. La configuration observée est un cas générique simple schématisé en figure 3. On se place dans le repère ayant le joueur pour origine.

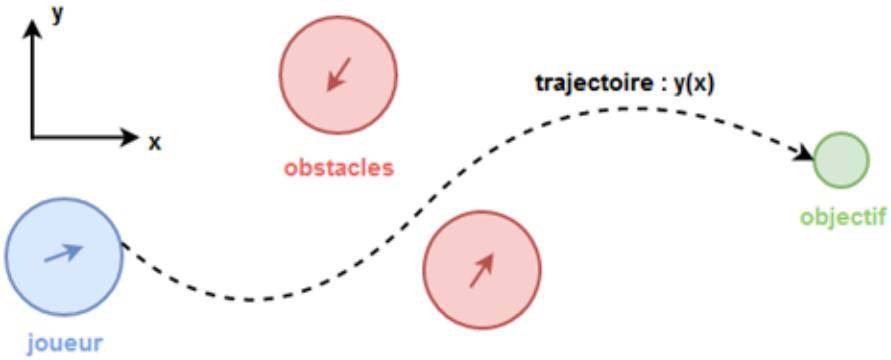

Figure 3 : Situation étudiée

3.1 – Définition du modèle

Le modèle retenu pour y est un polynôme en x. La trajectoire doit passer par le joueur et atteindre l'objectif, ce qui impose deux conditions limites. Après avoir discrétisé l'axe des abscisses, il vient :

$$y_m(x_i, p) = \sum_{n=0}^{N} p_n x_i^n = p^T x_i$$

Par ailleurs, les conditions aux limites imposent :

- $y_m(0, p) = 0$
- $y_m(x_{objectif}, p) = y_{objectif}$

D'où le modèle complet :

$$y_m(x_i, p) = \frac{y_{objectif}}{x_{objectif}} x_i \sum_{n=0}^{N} p_n \left(x_i^n + x_{objectif}^{n-1} x_i\right)$$

Par la suite, le degré du polynôme sera limité à 3. En effet, plus le degré du modèle est élevé, plus la trajectoire est libre d'onduler. Or, l'objectif est d'amener le robot à sa destination rapidement. Les ondulations sont donc des parasites. Le degré 3 permet des trajectoires en forme de « C » et de « S », jugées suffisantes pour cette utilisation.

$$y_m(x_i, p) = p_1\left(x_i^3 - x_b^2 x_i\right) + p_0\left(x_i^2 - x_b x_i\right) + \frac{y_b}{x_b} x_i$$

L'optimisation porte sur les deux coefficients restants du modèle, appelés paramètres :

$$\mathrm{p} = (\mathrm{p}_0 \quad p_1)^{\mathrm{T}}$$

3.2 - Définition du critère

Afin de résoudre un problème d'optimisation, il est nécessaire de définir quel critère est utilisé pour comparer des solutions entre elles. Le critère est une fonction des paramètres p dont le minimum sera cherché.

Pour le joueur, l'objectif de cette optimisation est d'atteindre l'objectif le plus rapidement possible, tout en évitant au maximum le contact avec les obstacles. Pour représenter ces deux contraintes, il convient de définir deux grandeurs :

- La longueur du trajet, qui doit être minimisée. Pour la construire, la trajectoire est approchée par une fonction linéaire par morceaux dont les longueurs des segments sont sommées.

$$L(p) = \sum_{i=1}^{M-1} \sqrt{(x_{i+1} - x_i)^2 + \left(y_m(x_{i+1}, p) - y_m(x_i, p)\right)^2}$$

- La plus petite distance entre le joueur et un obstacle, qui doit être maximisée.

$$D_{min}(p) = \min_{i \in \{1 \ldots M\}} \left(\sqrt{(x_{obstacle} - x_i)^2 + (y_{obstacle} - y_i)^2}\right)$$

Soit $\beta \in R$. Ce scalaire représente l'importance accordée à l'évitement d'obstacle lors de la résolution. Plus β est important, plus le robot doit passer loin des obstacles, au détriment de la longueur du trajet. Ce coefficient est fixé manuellement pour avoir un évitement satisfaisant selon les règles en vigueur dans la compétition. Le critère est la fonction suivante, à minimiser :

$$c(p) = L(p) + \frac{\beta}{D_{min}(p)}$$

3.3 - Résolution par descente de gradient

Pour résoudre le problème d'optimisation, il convient de choisir l'algorithme utilisé pour minimiser la fonction c. L'algorithme de descente de gradient est la solution la plus simple à implémenter, mais pas la plus performante. Toutefois, dans la situation étudiée ici, cette méthode est suffisante. Dans un premier temps, nous nous contentons de l'algorithme de descente de gradient à pas fixe :

- Le pas est fixé à 0.5 ;
- Le nombre d'itérations est limité à 1000 ;
- La norme du gradient est jugée suffisamment petite lorsqu'elle est inférieure à 10^{-3}. Ce critère sera appelé « précision » ;

- La différence entre deux valeurs consécutives du gradient doit rester supérieure à 10^{-3}. Ce critère sera appelé « stagnation ».

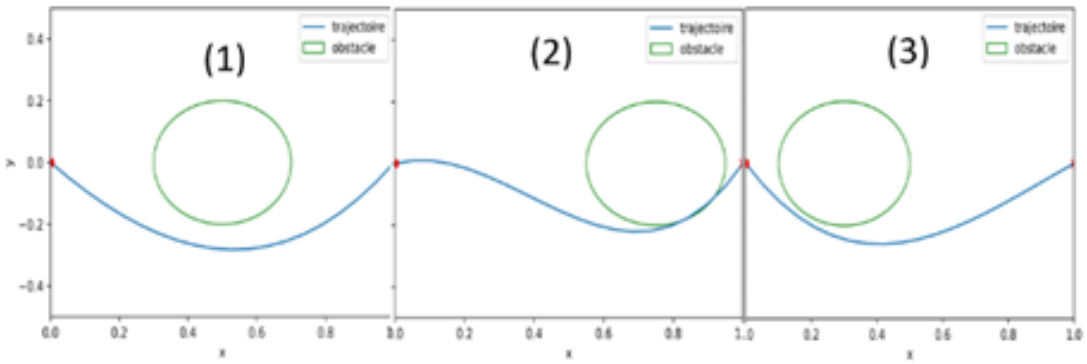

Figure 4 : Trajectoires obtenues

Dans un premier temps, l'algorithme est testé sur un scénario simple : le robot, son objectif et l'obstacle sont alignés, et seule la position de l'obstacle varie. En essai statique, il s'agit des configurations les plus défavorables pour un obstacle unique. Les trajectoires obtenues à l'aide de l'algorithme de descente de gradient sont affichées figure 4. Les durées sont mesurées au moyen de la méthode *perf_counter()* de la bibliothèque *time* de python et listées dans le tableau 1.

Essai	Nombre d'itérations	Temps d'exécution	Critère responsable de l'arrêt
1	63	0.23s	Stagnation
2	29	0.05s	Précision
3	114	0.20s	Stagnation

Tableau 1 : Rapidité du calcul des trajectoires

4 – Modèle en temps réel

Il s'agit maintenant de prévoir une trajectoire que le robot va suivre en temps réel, avec tous les éléments (obstacle, objectif) qui peuvent se déplacer. Nous devons concevoir un programme qui reçoit et fournit des informations en continu. L'idée est que le robot va régulièrement calculer la trajectoire optimale à suivre, suivre la tangente à l'origine de la courbe correspondante pendant quelques temps, puis réitérer avec des informations mises à jour.

Ainsi, le calcul des paramètres du modèle doit être fait à chaque pas de la simulation. Or, ces calculs doivent être faits pour tous les robots de l'équipe, en parallèle avec d'autres opérations que fait déjà le programme (localiser les robots, discrétiser le terrain, déterminer la stratégie, …). La principale contrainte est donc de minimiser le temps d'exécution de l'optimisation.

1 - Minimisation du nombre de points

À chaque pas de la simulation, le robot est piloté en vitesse. Il suffit donc de pouvoir construire un vecteur vitesse à l'origine du repère. Pour ce faire, il n'est pas nécessaire d'avoir un grand nombre de points sur l'ensemble du trajet. Nous limiterons à 10 le nombre de points dans les essais suivants ($M = 10$).

2 - Algorithme de backtracking

L'objectif de cet algorithme est d'optimiser le pas à chaque itération de la descente de gradient, afin d'éviter qu'il prenne une valeur trop faible. A chaque itération, l'algorithme suivant est exécuté :

```
alpha = 1
#alpha est le pas de la descente
tau = 0.5 #coefficient variable
l = 0 #itération
while c(x+alpha*d) >= c(x) :
    #c est la fonction critère
    alpha = tau*alpha
    l += 1
return l,alpha
```

3 - Effet sur les essais précédents

La campagne d'essais est répétée en changeant uniquement le nombre de points et l'optimisation du pas. Globalement, le temps d'exécution diminue de près de moitié (voir tableau 2). Pour une équipe d'un seul robot, on peut espérer réaliser entre 4 et 6 calculs de trajectoire par seconde sur le simulateur *grSim*. Avec de telles performances, on peut passer à une utilisation en temps réel de cette méthode de calcul de trajectoire.

Essai	Nombre d'itérations	Temps d'exécution	Critère responsable de l'arrêt
1	4	0.007s	Stagnation
2	42	0.06s	Stagnation
3	98	0.12s	Stagnation

Tableau 2 : Rapidité d'exécution après améliorations

5 – Comparaison des méthodes de calcul de trajectoire

Comparons les trajectoires obtenues par la méthode des champs potentiels et l'optimisation de trajectoire polynomiale.

5.1 - Protocole de comparaison

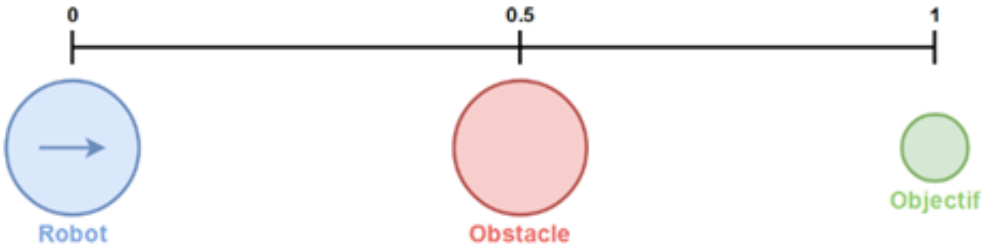

Figure 5 : Scénario de comparaison

Pour comparer les deux méthodes, on joue un scénario fixé représenté en figure 5. Ce scénario est un cas où la méthode des potentiels est peu efficace. Le robot bleu est commandé. L'objectif et le robot obstacle sont immobiles. On filme le déplacement en 15 images par secondes, puis on retrouve la trajectoire ainsi que le temps mis pour faire le parcours avec le logiciel *tracker video analysis*. Les durées présentées dans le tableau 3 sont obtenues comme moyenne sur dix essais.

5.2 - Analyse des résultats

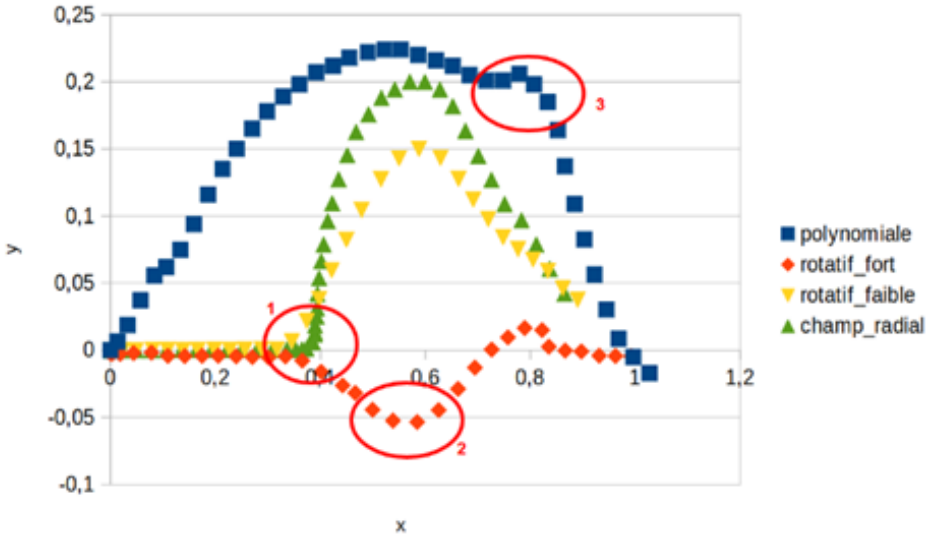

Figure 6 : Comparaison qualitative des trajectoires

La méthode d'optimisation développée permet un évitement plus performant que les champs tournants, au prix d'une rapidité légèrement inférieure. En effet, sur la figure 6, la zone 2 représente un endroit où le robot est entré en contact avec l'obstacle. Le défaut des champs radiaux visible en zone 1 (ralentissement voire arrêt en entrant de front dans la zone de répulsion) est totalement éliminé. Visuellement, les trajectoires données par les champs de vecteurs paraissent plus « naturelles ». En effet, la trajectoire issue de l'optimisation est une fonction continue polynomiale par morceaux. En conséquence, la dérivée est discontinue d'un pas de temps au suivant, ce qui donne les singularités comme mis en évidence en zone 3. Ces changements de direction brutaux font ralentir le robot. Par la suite, l'objectif sera d'éliminer ce défaut.

Méthode utilisée	Succès de l'évitement	Durée du parcours
Optimisation	Oui	2.55s
Champ radial	Oui	3.48s
Champ rotatif faible	Oui	2.7s
Champ rotatif fort	Non (partiel)	2.14s

Tableau 3 : Comparaison des performances des méthodes

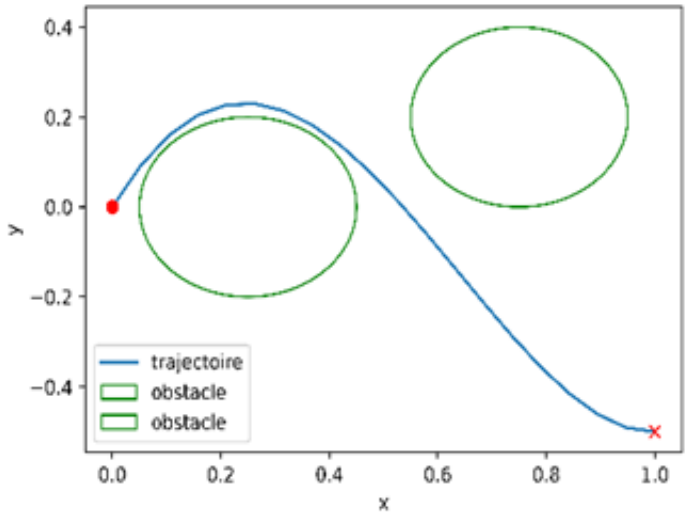

Figure 7 : Exemple de trajectoire pour deux obstacles

6 – Améliorations

6.1 - Économie de calculs

L'algorithme d'optimisation est exécuté plusieurs fois par seconde. En conséquence, le résultat de l'optimisation varie peu d'un pas de temps au suivant. Ainsi, en appelant k l'indice de l'itération, $p_k \approx p_{k+1}$. En conséquence, pour initialiser la descente de gradient à l'itération $k+1$, on utilise le résultat de l'itération k. Cette manipulation rend le calcul plus rapide, et adoucit la courbe au niveau des raccordements.

6.2 - Anticipation des mouvements

Lorsque l'obstacle est en mouvement, si le robot cherche à esquiver dans la direction de ce mouvement, une collision peut survenir. Pour éviter ce phénomène, le robot doit chercher à éviter une approximation de la future position de l'obstacle, plutôt que sa potion actuelle. Or, si on mémorise les positions successives des obstacles, on peut déterminer leur vecteur vitesse. Dès lors, on peut utiliser une approximation de leur future position donnée par ce vecteur.

$$\begin{cases} v_x^{(k)} = \dfrac{x^{(k+1)} - x^{(k)}}{dt} \\ v_y^{(k)} = \dfrac{y^{(k+1)} - y^{(k)}}{dt} \end{cases} \Rightarrow \begin{cases} x^{(k+1)} = x^{(k)} + v_x^{(k)} dt \\ y^{(k+1)} = y^{(k)} + v_y^{(k)} dt \end{cases}$$

Cette méthode n'est pas totalement satisfaisante : il arrive que le robot cherche à esquiver une future position d'un obstacle en s'impliquant encore plus dans la collision (dans le cas où l'obstacle tourne et où le robot esquive dans la direction du sens de rotation, on se rend compte que le résultat est pire). Pour comparer cette méthode au cas sans anticipation, on cherche à se placer dans des scénarios complexes (tous les éléments sont en mouvement) qui ne sont pas facilement reproductibles. On observe qu'un robot qui anticipe valide plus de situations qu'un robot qui se contente d'éviter la position actuelle de l'obstacle, mais ce résultat est à nuancer (ce protocole de tests est très qualitatif). D'autre part, on choisit la quantité $\mathrm{dt} = \mathrm{cte} = 0.1\mathrm{s}$ en considérant l'écart temporel entre deux mesures de position en moyenne, car cette quantité n'est pas constante dans la réalité. Parmi les évolutions possibles de cet algorithme, un choix plus précis de ce paramètre est prometteur, par exemple à partir de mesures de *time.perf_counter()* judicieusement placées.

6.3 - Trajectoire définie par une spline

Jusqu'à présent, la trajectoire était polynomiale par morceaux. Pour la « lisser », on peut la voir comme une spline et imposer des conditions sur les dérivées successives en les points de raccordement. Ici, on se contente d'imposer que la dérivée de la trajectoire doit être continue quelle que soit l'abscisse. Pour garder la même structure de programme sachant qu'on impose une nouvelle contrainte, il faut rajouter un paramètre. On augmente donc le degré du modèle à 4. $p_i^{(k)}$ Désigne le $i$ème coefficient du modèle à l'itération k, et $d^{(k)}$ est la dérivée imposée.

$$p_0^{(k)} = 0$$

$$p_1^{(k)} = d^{(k)}$$

$$p_2^{(k)} = \frac{y_b^{(k)}}{\left(x_b^{(k)}\right)^2} - \frac{d^{(k)}}{x_b^{(k)}} - P[0]x_b^{(k)} - P[1]\left(x_b^{(k)}\right)^2$$

$$p_3^{(k)} = P[0]$$

$$p_4^{(k)} = P[1]$$

$$d^{(k)} = \frac{\partial}{\partial x} y_m^{(k)} \left(x_r^{(k+1)}, P^{(k)} \right)$$

Ce nouveau modèle permet une trajectoire plus lisse, mais le calcul est plus long. En effet, imposer la dérivée à l'origine est une contrainte forte qui tend à entrer en conflit avec l'évitement des obstacles. Selon le choix du coefficient β, le temps de calcul pour un pas de la simulation varie entre 0.5s et 1.5s, ce qui est trop lent pour être utilisable au cours d'un vrai match.

6.4 - Symétrisation

Le modèle étudié jusqu'alors présente comme principal avantage sa simplicité. En revanche, ce modèle a un défaut majeur : il n'est pas symétrique. En effet, une trajectoire définie par une équation définie comme une fonction de l'abscisse du robot ne permet pas un déplacement vertical. Une solution simple à ce problème serait de pivoter le repère de 90° selon le quart de plan dans lequel se trouve le robot, ce qui fonctionne sans toutefois corriger le défaut du modèle. Il convient donc de définir un modèle plus complexe qui permet des déplacements sans impact de la direction.

L'idée directrice dans la construction de ce nouveau modèle est la prise en compte du temps. Pour ce faire, considérons $t \in [0, t_{max}]$. La position du robot au cours du temps correspond au point $M(t,P) = (x(t,P),\ y(t,P))$, avec :

$$\begin{cases} x(t,P) &= p_0 + p_1 t + p_2 t^2 \\ y(t,P) &= p_3 + p_4 t + p_5 t^2 \end{cases}$$

Pour ce nouveau modèle, un degré 2 en t pour x et y est nécessaire pour éviter une trajectoire systématiquement linéaire. En revanche, pour permettre plus de virages, il faudrait augmenter le degré du modèle. De nouveau, imposons les conditions aux limites : la trajectoire passe par le robot en $t = 0$ et par l'objectif en $t = t_{max}$.

$$\begin{cases} x(t,P) &= x_{robot} + \left(\frac{x_{objectif} - x_{robot}}{t_{max}} - p_0 t_{max} \right) t + p_0 t^2 \\ y(t,P) &= y_{robot} + \left(\frac{y_{objectif} - y_{robot}}{t_{max}} - p_1 t_{max} \right) t + p_1 t^2 \end{cases}$$

Remarquons que l'optimisation reste bidimensionnelle (seulement deux paramètres). Par ailleurs, le critère établi précédemment est utilisable pour ce nouveau modèle. La figure 8 montre une trajectoire irréalisable avec le modèle précédent et obtenue en 0.065s avec le nouveau modèle.

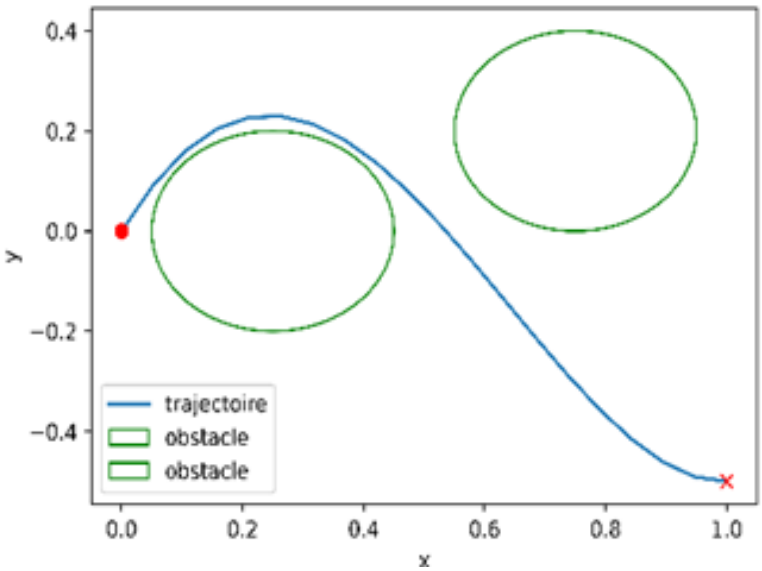

Figure 8 : Exemple de trajectoire verticale

Une vidéo présentant différents essais réalisés avec cette méthode est disponible ici [1] et en annexe [2]. Les essais présentés comportent le protocole de comparaison ainsi que divers tests dynamiques plus complexes à exploiter mais montrant des comportements atteignables avec la méthode développée. Le code python utilisé pour chaque calcul de trajectoire avec un modèle symétrique de degré 2 est également disponible en annexe [2].

7 – Perspectives

- Dans cet article, nous nous sommes contentés de concevoir des trajectoires pour le robot, sans se soucier de sa vitesse. Néanmoins, le nouveau modèle présenté dans la partie précédente permet de prendre en compte le temps et donc d'imposer la vitesse du robot le long de la trajectoire. Pour réaliser cette opération, il est nécessaire de modifier le critère utilisé dans l'optimisation. Ce n'est pas une opération que nous détaillerons ici, toutefois nous pouvons en lister les principaux points clefs.
    - Dans un premier temps, il est nécessaire de définir une quantité de référence, homogène à une durée. Par exemple, le temps mis par le robot pour parcourir la diagonale du terrain à la vitesse maximale autorisée.
    - Ensuite, il convient de construire un nouveau critère inspiré du critère utilisé jusqu'à présent auquel on ajoute des conditions sur les t_i. Dès que le calcul d'un extremum est nécessaire, la quantité utilisée sera la grandeur définie précédemment.
    - Pour finir, considérons un exemple illustrant l'intérêt d'une telle démarche. Pour atteindre au plus vite son objectif, le robot peut suivre deux comportements distincts. Proche des obstacles, le robot doit ralentir, pour minimiser les risques de collisions. Loin des obstacles, le robot cherche à rester le plus longtemps possible à vitesse maximale.
- Le coefficient β contrôle l'importance de l'évitement. Un des problèmes rencontrés est l'indépendance de β vis-à-vis de l'inertie des robots. En effet, plus le robot va vite, plus un changement de direction est difficile. Une amélioration du critère envisageable est un coefficient augmentant avec la vitesse.
- Le critère utilisé ne prend pas en compte les bordures du terrain. En conséquence, le robot peut parfois se bloquer si le vecteur vitesse imposée tend à le faire sortir du terrain. Pour une utilisation plus performante dans le cadre de la compétition, il faut modifier le critère pour pénaliser les trajectoires qui induisent ce phénomène.

8 – Références

[1]: Robots footballeurs : le simulateur grSim, G. Lemprez, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/robots-footballeurs-le-simulateur-grsim

[2]: Robots footballeurs : Commande par optimisation de trajectoire polynomiale, G. Lemprez, J. Ojeda, octobre 2021, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/robots-footballeurs-commande-par-optimisation-de-trajectoire-polynomiale