1 Introduction

L'obtention de la meilleure trajectoire possible lors des différentes épreuves (sprint et rotation) peut être vue comme un problème d'optimisation mono-objectif sous contraintes. En effet, par exemple, dans le cas de l'épreuve du Sprint, il s'agit d'optimiser la vitesse de déplacement du robot selon une direction tout en s'assurant la validité de la trajectoire ainsi qu'une bonne stabilité du robot. Afin de résoudre au mieux ce problème, l'équipe rouge a fait le choix d'utiliser un algorithme génétique. Ce rapport détaille ainsi les principaux aspects menant au programme d'optimisation. Notamment le calcul du modèle géométrique direct (MGD) et du modèle géométrique indirect (MGI) de manière analytique. La manière dont le robot a été modélisé, puis, enfin comment cette modélisation permet d'obtenir la fonction d'aptitude (fonction permettant d'évaluer les performances de chaque trajectoire), permettant *in fine* l'optimisation de la trajectoire via l'algorithme génétique.

2 Calcul du MGD et MGI

Afin d'être en mesure de contrôler le robot, il est d'abord nécessaire de déterminer le MGD et le MGI d'une patte. Ici, la géométrie d'une patte étant assez simple et son nombre d'articulations étant suffisamment faible (Figure 1) alors le calcul analytique est faisable, et comme il a le mérite d'être plus précis et rapide que le calcul par Jacobienne. C'est la méthode par laquelle nous avons déterminé le MGI.

2.1 Calcul du MGD

Afin de calculer le MGD il est d'abord nécessaire de paramétrer la patte du robot. Pour ce faire, on modélise la patte comme sur la Figure 1.

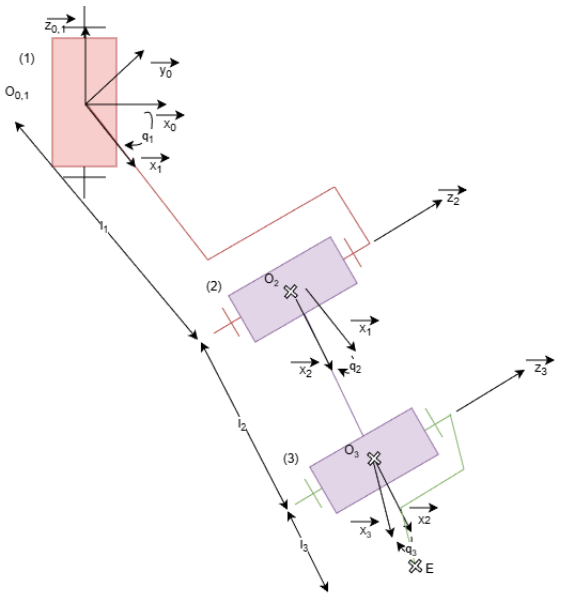

FIGURE 1 – Schéma cinématique et d'une patte du robot avec ces caractéristique géométriques

A partir de ce schéma cinématique Figure 1, on détermine le paramétrage selon Khalil Kleinfinger (Table 1).

Solide	α_i	d_i	θ_i	r_i
1	0	0	q_1	0
2	$-\frac{\pi}{2}$	l_1	q_2	0
3	0	l_2	q_3	0

TABLE 1 – Paramétrage selon Khalil Kleinfinger d'une patte

A partir de ce paramétrage, l'on peut en déduire les matrices de transformation homogène associées à 2 solide successifs. Et par multiplication de ces matrices, la matrice de transformation homogène ${}^0T_3$ entre le repère 0 et 3. Et alors, avec ${}^iX$ la position de l'effecteur dans le repère i, on a :

$$ {}^0X = {}^0T_3^3X $$

et, dans le cas présent :

$$ {}^0X = \begin{pmatrix} (l_1 + l_2cos(q_2) + l_3cos(q_2+q_3))cos(q_1) \\ (l_1 + l_2cos(q_2) + l_3cos(q_2+q_3))sin(q_1) \\ -l_2sin(q_2) - l_3sin(q_2+q_3) \\ 1 \end{pmatrix} $$

2.2 Calcul du MGI

On souhaite maintenant déterminer les paramètres q_1, q_2 et q_3 à partir des coordonnées x_E, y_E, z_E du point E dans 0. Pour cela remarquons d'abord que :

$$ tan(q_1) = \frac{y_E}{x_E} \; donc \; q_1 = arctan2(y_E, x_E) $$

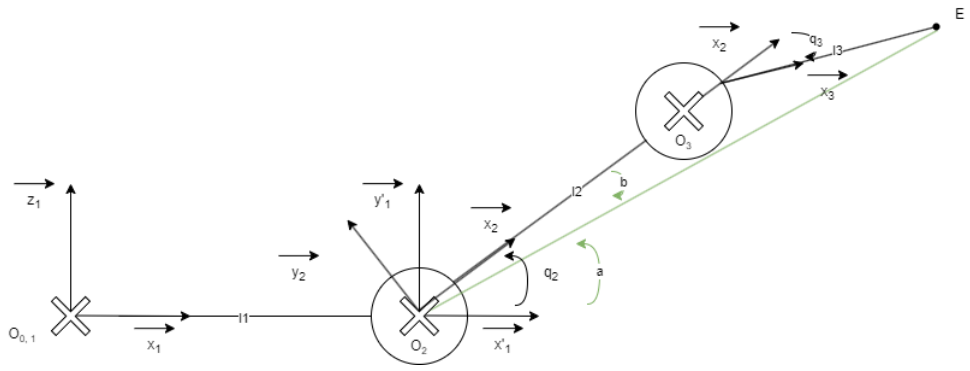

FIGURE 2 – Projection selon le plan $(\vec{x_1}, \vec{z_1})$ du modèle cinématique de la patte

Alors, on a, d'après la Figure 2 :

$$x = cos(q_2)l_2 + cos(q_2 + q_3)l_3 \ (1)$$

$$z = sin(q_2)l_2 + sin(q_2 + q_3)l_3 \ (2)$$

Et, avec $(1)^2 + (2)^2$ il vient :

$$q_3 = \pm arcos(\frac{x^2 + z^2 - l_3^2 - l_2^2}{2l_2l_3})$$

De plus, toujours d'après Figure 2 :

$$tan(a) = \frac{z}{x} \ et \ tan(b) = \frac{l_3 sin(q_3)}{l_2 + l_3 cos(q_3)}$$

Or, $q_2 = a - b$, d'où finalement, de par l'orientation de l'angle b :

$$q_2 = arctan2(z, x) + arctan2(l_3 sin(q_3), l_2 + l_3 cos(q_3))$$

Notons de plus que, si ces expressions nécessitent de connaitre ${}^{1'}X$ afin de se placer dans (1'). Ce dernier est calculable via l'expression de q_1 calculé en premier lieu (via la matrice de transformation homogène ${}^0T_{1'} = {}^0T_1^1T_{1'}$). Ainsi, nous avons déterminé l'expression de q_1 puis celles de q_2 et q_3. Ce qui réalise le MGI.
Néanmoins, lors de l'intégration dans un programme informatique il est important de noter peut y avoir des problèmes d'offsets et d'orientation. Ainsi, il convient de tester le MGI avec une patte en position (q_1,q_2,q_3) = (0, 0, 0). Afin d'annuler ces offsets possibles. Puis, il faut vérifier les signes de (q_1,q_2,q_3) donnés par le MGI, ce qui se fait en positionnant une patte à des coordonnées angulaire quelconques (via le MGD) et en regardant la différence entre les coordonnés articulaires entrées et celles renvoyées. C'est ce que nous avons fait pour notre programme.

3 Modélisation du robot

La réalisation du MGI et MGD nous permet d'ouvrir la voie à la modélisation cinématique du robot. L'objectif étant de permettre de simuler le déplacement de chacune des articulations du robot en fonction du temps lors de le suivi d'une trajectoire et de fournir le centre de gravité et positionnement des effecteurs durant toute la durée de la réalisation de la trajectoire. Cette modélisation est réalisée sous Python en programmation orientée objet en utilisant la méthode de simulation pas à pas (time-stepping). Afin d'aborder les différents aspects de la modélisation nous verrons la modélisation d'une patte avec ses moteurs, puis du robot dans son entièreté. Afin, nous aborderons la problématique du suivi de trajectoire et du calcul du centre de gravité.

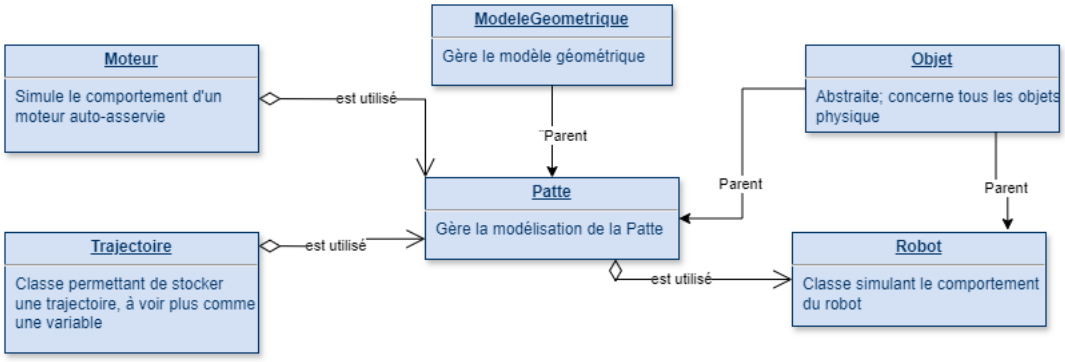

FIGURE 3 – Architecture générale des fichiers du programme servant à la modélisation

3.1 Modélisation d'une patte et des moteurs

3.1.1 Modélisation des moteurs

La modélisation des moteurs permet principalement d'obtenir des déplacements cohérents dans le temps par rapport au comportement réel du robot. Pour ce faire, l'on fait l'hypothèse que chacun des moteurs se déplacent à une vitesse constante v_m et qu'il a une précision infinie. De plus, on suppose qu'il possède une position angulaire minimum q_{min} et maximum q_{max}. Dès lors, avec $q_m(t)$ et $q_{min} < q_v < q_{max}$ les positions angulaires actuelle et voulue. L'on a :

$$q(t+dt) = \begin{cases} q_v \ si \ v_m dt > |q_v - q(t)| \\ q(t) + v_m dt \ sinon \ si \ q_v - q(t) > 0 \\ q(t) - v_m dt \ sinon \end{cases}$$

Ce qui permet de modéliser le déplacement temporel des articulations.

3.1.2 Modélisation des pattes

La modélisation d'une patte est avant tout un modèle géométrique de celle-ci (contenant notamment le MGD et MGI) ainsi la classe Patte est fille de ModèleGéométrique (Figure 3). Néanmoins, la classe Patte intègre 3 moteurs permettant de simuler le déplacement temporel de chaque articulations. De fait, lorsque la patte reçoit un ordre de déplacement à une coordonnée opérationnelle donnée, elle la traduit en coordonnées articulaires via le MGI. Puis, elle fixe chaque variable q_v de ces moteurs selon ces coordonnées articulaires. Lesquelles vont traduire le déplacement dans le domaine temporelle. De plus, notons qu'une méthode permet d'afficher la patte dans une position angulaire donnée, elle est héritée de ModèleGéométrique. Enfin, des autres méthodes importantes est celle permettant de suivre une trajectoire donné (plus sur cela plus tard) et celles qui permettent d'obtenir la position de l'effecteur (on reviendra dessus aussi).

3.2 Modélisation du robot

La classe Robot a pour rôle de gérer la synchronisation entre les pattes, de permettre l'affichage du robot (en entier), de calculer le centre de gravité et de calculer la fonction d'aptitude. Plus généralement elle centralise la prise de décision et certains calculs.

3.2.1 Calcul de position dans le référentiel du robot

Nous avons établi précédemment comme objectif de connaître le positionnement des effecteurs. Car cela dessert la fonction d'aptitude. Néanmoins, la fonction d'aptitude a besoin de connaître la position des effecteurs dans le repère du robot (Figure 4). Or, jusque là la position des effecteurs est donné dans le repère local de chaque patte. Pour adresser ce problème, chaque patte contient une matrice de transformation homogène permettant de passer du repère local au repère du robot. Celle-ci est calculée notamment à partir des paramètres α, r_1 et r_2 (Figure 4) au moment de l'initialisation par la classe Robot. Pour en revenir à ce que nous disions section 3.1.2 concernant les deux méthodes des pattes calculant la position de l'effecteur. L'une donne la position de l'effecteur dans le repère de la patte et l'autre dans le repère du robot.
Notons que le repère du robot n'est pas le repère lié au sol. Il à l'avantage d'être pratique pour y décrire les mouvements du robot et de plus, cela ne posera aucun problème de calculer les différentes caractéristiques de la trajectoires (vitesse, ...) à partir de ce repère.

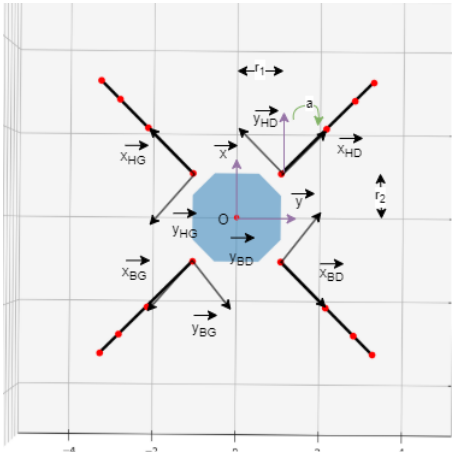

FIGURE 4 – Représentation du repère du robot (O, $\vec{x}$, $\vec{y}$) et des repères liés à chaque patte

3.2.2 Calcul du centre de gravité

On suppose que les masses autres que celles du corps du robot (partie bleu Figure 4) et des moteurs (points rouges Figure 4 sauf ceux situés aux extrémités des pattes) sont négligeables. On suppose les centres de gravité des moteurs à l'endroit de la modélisation géométrique de leur articulation (points rouges Figure 4). Enfin, on suppose que le centre de gravité du corps est O. Sous ces hypothèses, avec $\vec{OG_c}$ le centre de gravité du corps et m_c sa masse. En numérotant les moteurs de 1 à 12 et notant $\vec{OG_i}$ le centre de gravité du i-ème moteur et m_M sa masse. On a, un centre de gravité :

$$\vec{OG} = \frac{\vec{OG_c} + m_M \sum_{i=1}^{12} \vec{OG_i}}{12 m_M + m_c}$$

3.3 Suivi de trajectoire

Le suivi de la trajectoire peut se décomposer en trois problématique. Celle du stockage de la trajectoire, du suivi de la trajectoire par une patte et enfin de la synchronisation.

3.3.1 Stockage de la trajectoire

Le stockage de la trajectoire s'effectue via la classe Trajectoire. Cette classe assure donc le stockage de

la trajectoire et permet si besoin de l'afficher. Le stockage de la trajectoire est réalisé par le stockage des coordonnées opérationnelles de chaque point de passage pour une patte, de plus à chaque point de passage l'on associe le booléen RP. Ce booléen vaut $True$ si lors de la trajectoire la patte doit être en RP et $False$ sinon (l'utilité de ce booléen est expliciter ci-après). Ainsi, la classe Trajectoire stocke 2 listes, une pour stocker les coordonnés opérationnelles de chaque points de passage et une pour stocker les valeurs de RP correspondant à chaque point.

3.3.2 Suivie d'une trajectoire par une patte

Nous avons évoqué dans la section 3.1.2 que la classe Patte possède une méthode pour effectuer un suivi de trajectoire. Il est temps d'en parler plus longuement. Son fonctionnement de base est assez simple. En effet, chaque Patte possède une variable de la classe Trajectoire stockant notamment la trajectoire suivie, nommée $Straj$, et une variable donnant permettant de connaître le rendu dans la trajectoire $posTraj = [RP, pos, RPposs, TN]$. Avec $RP = True$ si la patte est en RP et $RP = False$ sinon. L'entier pos est le numéro du point de passage suivi actuellement. A chaque fois que le points de passage est atteint (i.e la distance liée au produit scalaire usuelle entre l'emplacement actuel et l'emplacement voulue est suffisamment faible), l'on incrémente pos et la position suivie (modulo $len(Straj)$ pour que la trajectoire se répète).

3.3.3 La problématique de la synchronisation

Nous n'avons pas encore évoqué à quoi servent $RPposs$ et TN de $posTraj$. Car, ils servent à la synchronisation des pattes, nécessaire pour modéliser fidèlement le déplacement du robot lors d'une trajectoire. Afin d'illustrer la problématique de la synchronisation des pattes, considérons une trajectoire comme sur la Figure 5. Considérons deux pattes notées $PatteA$ et $PatteB$.

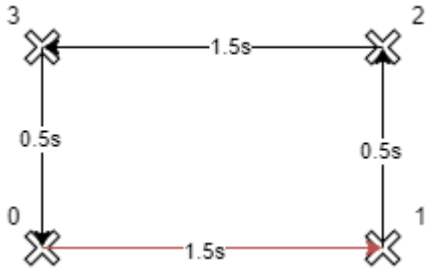

FIGURE 5 – Exemple de trajectoire d'une patte avec les temps de trajets entre chaque point, la flèche rouge représente la partie en RP

Supposons que leurs positions initiales soient toutes deux 0. Alors, elles arriveront au point 1 en même temps. Mais, afin que le robot reste debout, il est nécessaire qu'une des deux pattes reste au sol, donc en 1. Donc, par exemple, $PatteA$ reste en 1 tandis que $PatteB$ continue vers 2. Alors, avant que $PatteA$ puisse bouger, il est nécessaire qu'elle attende que $PatteB$ revienne en 0. Ainsi, il est nécessaire de synchroniser le déplacement des pattes entre elles. Concrètement, une manière d'effectuer la synchronisation est de non pas faire une trajectoire qui commande chaque patte indépendamment, mais faire une trajectoire qui commande toutes les pattes en même temps. Ainsi, au lieu d'avoir une trajectoire contenant une position à atteindre pour chaque point de passage, la trajectoire contiendrait 4 positions à atteindre (une pour chaque patte) pour chaque point de passage de la trajectoire. Néanmoins, comme l'on ne passe au point de passage suivant que lorsque toutes les pattes ont atteint leurs positions à atteindre cela peut considérablement ralentir le robot. Par exemple, dans notre cas, si $PatteA$ commence au point 0 et $PatteB$ au point 1. Alors, que $PatteB$ aurait largement pu dépasser le point 2 dans sa poursuite de la trajectoire pendant que $PatteA$ effectue le trajet de 0 à 1. De par la méthode exposée, elle est obligée d'attendre que $PatteA$ arrive au point 1 avant de continuer son chemin. Ainsi, la trajectoire met 5s à être effectuée contre 4s dans le meilleur des cas (en prenant en compte qu'une patte doit rester au sol).

3.3.4 Solution retenu pour la synchronisation

Ainsi, nous avons plutôt retenu une autre méthode pour synchroniser les pattes du robot. Dans cette méthode chaque patte est pilotée indépendamment (par la classe Patte via la méthode exposée en 3.3.2). Néanmoins, les pattes s'échangent des signaux via la classe Robot pour savoir communiquer l'état de leurs trajectoire et savoir si quand passer d'un état à un autre. Plus précisément, $RPposs$ et TN de $posTraj$ sont des booléens. $RPposs$ vaut $True$ si la patte est prête à passer de l'état RP à l'état hors-sol (noté TN par la suite). Et, TN vaut lui $True$ si la patte lorsque la patte passe de l'état TN à RP. Ainsi, lorsque la patte a atteint la dernière position en RP, avant de passer en TN, elle vérifie que le $RPposs = True$, et lorsqu'elle passe de l'état TN à RP et on fixe $TN = True$. Ces variables propres à la classe Patte permettent via la classe Robot de communiquer l'état de la trajectoire au global. Ainsi, pour reprendre l'exemple de $PatteA$ et $PatteB$. La valeur de $RPposs$ de $PatteA$ est fixée, via la classe Robot, à celle de TN de $PatteB$ et inversement. Ici, comme nous avons 4 pattes, elles sont divisées en 2 groupes (les pattes en diagonales étant dans le même groupe). Avec la valeur de $RPposs$ des pattes d'un groupe fixée à la valeur minimum de TN des pattes de l'autre groupe. De cette manière, l'on s'assure qu'un groupe soit en mode TN et l'autre en mode RP. Concernant la position de départ. Un groupe commence à la première position du RP (0 sur la Figure 4 par exemple), l'autre commence à la dernière position du RP (1 pour la Figure 4) avec la valeur $RPposs = True$, ce groupe sera dit déphasé. Cette configuration de départ permet au robot de ne pas

être bloqué au début. Car dans cette configuration de synchronisation afin qu'il n'y ait pas de blocage (un groupe reste indéfiniment en RP par exemple) il faut que un groupe soit en TN tandis que l'autre soit en RP, ce qui est immédiatement le cas après l'initialisation du robot (le groupe déphasé passant en TN). De plus, cette configuration permet une bonne stabilité (4 pattes au sol) avant le début du déplacement.

4 Fonction d'aptitude

La fonction d'aptitude a ici pour but d'évaluer la trajectoire en renvoyant une variable *score*. La modélisation du robot nous permet d'avoir accès à la position des effecteurs ainsi qu'au centre de gravité à tous les intervalles de temps dt. Nous allons voir comment ces informations sont utilisées pour élaborer la fonction d'aptitude de l'épreuve de Sprint et, plus succinctement, de l'épreuve de Rotation. La première chose à faire est d'identifier les valeurs que l'on cherche à optimiser et quelles contraintes fait-on face. Ici, l'on cherche à optimiser la vitesse selon l'axe $\vec{x}$ du robot (Figure 4), c'est pour cela que l'on avance les pattes 2 à 2 du robot. Les contraintes sont ,d'abord, la stabilité, ainsi le centre de gravité du robot doit rester dans le polygone de sustentation à tout instant. En effet, l'absence de gyroscope ou de centrale inertielle nous oblige à être en statique. L'autre contrainte majeure est plus subtile, il faut vérifier que la trajectoire soit valide.

4.1 Contrainte de stabilité

Cherchons à évaluer la stabilité de notre robot. Posons pour cela (d_s) la droite de stabilité qui est la droite passant par les 2 pattes en phase de support (Figure 6). On appel zone de stabilité tous points se situant à une distance maximale donné de la droite de stabilité (d_s). Notons qu'ici le polygone de sustentation, dans la partie intérieur des pattes, est confondue avec une zone de stabilité de paramètre le rayon de l'effecteur r_e. Néanmoins, afin de s'assurer que le centre de gravité est dans le polygone de sustentation. L'on prend une marge de sécurité d_{zs}, on cherche ainsi à positionner le centre de gravité dans la zone de stabilité de paramètre $d_{zs} < r_e$.

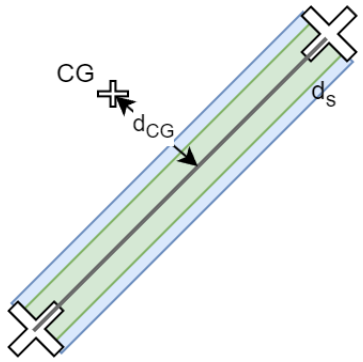

FIGURE 6 – Illustration du concept de droite et zones de stabilité (en vert) ; en bleu l'on trouve le polygone de sustentation et le centre de gravité est noté CG

Ainsi, la contrainte de stabilité se traduit de la manière suivante sur la variable *score* :

$$score = score - \begin{cases} 0 \ si \ d_{CG} < d_{zs} \\ 10 * dt \ si \ d_{zs} < d_{CG} < r_e \\ 20 * dt \ sinon \end{cases}$$

Ces pénalités appliquées en cas de non respect de la stabilité sont plus importantes que les bénéfices qui peuvent être obtenus grâce à la vitesse gagnée, permettant de s'assurer que l'algorithme donnera la priorité à la stabilité.

4.2 Validité de la trajectoire et évaluation de la vitesse

4.2.1 Validité de la trajectoire

Afin de vérifier que la trajectoire est valide, il est nécessaire que les coordonnées opérationnelles puissent être atteinte par le robots. L'on considérera que cette condition est remplie dans la suite (sinon il y a une erreur). De plus, il faut vérifier que les pattes du groupe en mode RP soient les plus basses. Sinon la trajectoire sort du cadre d'analyse possible. Pour cela, l'on vérifie a tous les pas de temps de la simulation que la position de l'effecteur des pattes vérifiant $RP = True$ à une altitude supérieur à la position de l'effecteur. Sinon, on fixe $score = -\infty$.

4.2.2 Evaluation de la vitesse

Pour évaluer la vitesse selon l'axe $\vec{x}$ du robot (Figure 4), on évalue le déplacement que le robot effectue selon l'axe $\vec{x}$ et le temps nécessaire pour réaliser la trajectoire une fois. Pour évaluer la distance parcourue selon $\vec{x}$, l'on additionne les distances parcourues selon $\vec{x}$ entre chaque points de passage de l'effecteur dans la phase de support. Pour évaluer le temps nécessaire à la réalisation de la trajectoire. L'on effectue la simulation pas à pas jusqu'à ce la trajectoire se soit effectuée une fois. Une fois la vitesse v_x ainsi calculée, on fixe :

$$score = score + v_x$$

4.2.3 La fonction d'aptitude de l'épreuve de rotation

Cette fonction est semblable à celle de l'épreuve de sprint à la différence que l'on ne calcule pas la vitesse selon $\vec{x}$ mais la vitesse angulaire de rotation autour de O.

5 Algorithme génétique et résultats

Maintenant que la fonction d'aptitude renvoie *score* permettant l'évaluation de la trajectoire. Nous pouvons utiliser cette évaluation dans l'algorithme génétique. Enfin, nous présenterons les résultats obtenus via cette algorithme.

5.1 L'algorithme génétique

L'algorithme génétique étant assez simple et n'étant pas le cœur du sujet nous le présenterons que succinctement. L'algorithme génétique utilisé est un algorithme de sélection par rang via fonction d'aptitude, ne possédant pas de croisement mais faisant systématiquement apparaître des mutations à chaque générations. Les mutations appliquées sont de moins en moins importantes au fil des générations (selon une exponentielle décroissante). L'algorithme génétique applique seulement ces mutations sur les positions des effecteurs dans la trajectoire et non sur le booléen RP, plus précisément ces mutations s'appliquent sur les coordonnées articulaires des effecteurs. Les individus étant alors les coordonnées articulaire des différents points de passage. Cela permet d'éviter d'avoir des trajectoires non valide (position des effecteurs impossible à atteindre).

5.2 Résultats

Les résultats obtenus par l'algorithme génétique pour l'épreuve de Sprint (Figure 7) sont cohérent avec ceux attendus. On observe en effet que la fonction donnant l'évolution des meilleurs trajectoires en fonction des générations est croissante et concave. Le caractère croissant est signe de l'optimisation de la trajectoire et le caractère concave provient des mutations dont l'importance est selon une exponentielle décroissante (cf. 5.1). L'algorithme appliqué à l'épreuve de rotation, on obtient le même type de résultats.

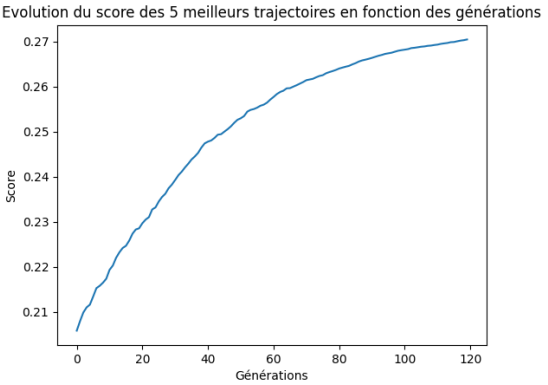

FIGURE 7 – Évolution du score des 5 meilleurs trajectoires en fonctions des générations (90 individus)

6 Limites et conclusion

6.1 Limites

Les limites de notre algorithme et de notre modélisation sont en premier lieu lié aux hypothèses que nous avons réalisées. En effet, dans la section 3.1.1 nous avons par exemple supposé que les moteurs étaient infiniment précis et surtout de vitesse constante. Or, si l'on considère le modèle du premier ordre du moteur, cela est vrai si le temps de déplacement est suffisamment important devant la constante de temps du moteur. L'hypothèse est donc valable pour de suffisamment grand déplacement. L'autre hypothèse majeure sont celles réalisée en section 3.2.2. En supposant que le centre de gravité est au centre du corps, des articulations pour les moteurs et que les autres pièces ont une masse négligeable. Néanmoins, la création d'une zone de stabilité ayant $d_{zs} < r_e$ permet de mitiger les effets possiblement néfastes lié à cette hypothèse.
Une autre limite majeure est le temps de calcul de la méthode. En effet, la simulation étant réalisée sous Python moins rapide que d'autres langages usuels (Java, C/C++ par exemple) et l'optimisation ayant été reléguée par manque de temps, de compétences et car la priorité fût donné à une grande souplesse du code ... Ainsi, la simulation d'une trajectoire prend un temps assez conséquent limitant fortement le nombre de génération et la taille de la population possible et donc l'optimisation réalisée. D'autant que les algorithmes génétiques nécessitent en général beaucoup de temps de calcul pour être efficace. Ainsi, un algorithme par descente de gradient aurait par exemple été la bienvenue car plus efficace. Néanmoins cela fût impossible par manque de temps.

6.2 Conclusion

La réalisation du calcul de trajectoire nous à menés à développer et utiliser un large panel de compétences (calcul de MGD/MGI, modélisation, travail de groupe ...) dans des domaines divers comme l'informatique (programmation, algorithme génétique), la robotique sérielle (calcul du MGD/ MGI, stabilité ...) et plus largement la mécatronique (calcul du centre de gravité, critique des hypothèses sur le déplacement des moteurs ...).