# Génération de trajectoires et évitement des singularités pour un robot manipulateur 3RRR plan

**CMI Mécanique – 3ème année**
2025-2026

**Marion GEHIN, Artsiom HERASIMENKA et Igor GIRARD–KHADIR**



*Professeurs :*
**Faïz BEN AMAR**
**Hugo CORREAS**

**Département d'ingénierie mécanique**
**Sorbonne Université**

# 1 Remerciements

Nous souhaitons remercier Sorbonne Université pour l'opportunité qui nous a été donnée de réaliser ce projet au sein de ses locaux et de nous immerger dans un environnement scientifique stimulant. Nous exprimons également notre reconnaissance envers le laboratoire ISIR (Institut des Systèmes Intelligents et de Robotique), qui nous a permis d'accéder au matériel et aux ressources nécessaires à la conception, à la simulation et à la mise en œuvre du robot parallèle 3RRR. Pour finir, nous remercions l'ensemble du personnel du laboratoire pour leur accueil et pour l'environnement de travail mis à notre disposition.

## Table des matières

# 2 Résumé

## 2.1 Résumé en français

En travaillant sur le projet 3RRR, nous avons réalisé une série d'étapes systématiques allant de l'analyse des sources à la mise en œuvre d'une chaîne logicielle complète. Nous avons d'abord rassemblé et étudié tous les éléments disponibles : codes MATLAB, pilotes Dynamixel et scripts Python existants (dont une version de tracé en forme de trèfle). À partir de cette base nous avons défini la partie mathématique essentielle - en particulier la cinématique inverse - et la manière d'exploiter les positions initiales des servomoteurs pour lancer les calculs.

Par prudence, nous avons d'abord développé une couche de simulation (NumPy + Matplotlib) pour valider la géométrie, repérer les zones inatteignables et éviter tout risque matériel. Ensuite nous avons implémenté progressivement le code de pilotage : calibration manuelle des offsets, conversion angle - ticks, et envoi synchrone des objectifs aux servos. Nous avons privilégié un réglage manuel et contrôlé des vitesses et trajectoires plutôt qu'une optimisation entièrement automatique.

Ce travail nous a permis de traduire et d'unifier des algorithmes MATLAB en Python, d'acquérir des bonnes pratiques de sécurité et de contrôle, et de constituer une base logicielle exploitable pour de futures expérimentations.

## 2.2 Summary in english

Working on the 3RRR project, we carried out a systematic sequence of steps from source analysis to a complete software implementation. First we gathered and studied all available materials : MATLAB scripts, Dynamixel drivers and existing Python code (including a clover-shaped path example). From this foundation we defined the core mathematical part - notably the inverse kinematics - and how to use the servomotors' initial positions to start the computations.

Out of caution, we first developed a simulation layer (NumPy/Matplotlib) to validate the geometry, detect unreachable regions and avoid risking hardware. We then progressively implemented the control code : manual calibration of offsets, angle - ticks conversion, and synchronized goal writes to the servos. We deliberately favoured controlled, iterative tuning of speeds and trajectories rather than fully automatic optimization.

This work allowed us to translate and consolidate MATLAB algorithms into robust Python, to adopt safety and control best practices, and to build a software foundation suitable for further experiments.

# 3 Introduction

Dans le cadre de notre stage au laboratoire ISIR, nous avons travaillé sur le projet « Génération de trajectoires et évitement des singularités pour un robot manipulateur 3RRR ». Ce travail nous donnait deux missions pricipales : d'une part comprendre et formaliser la cinématique d'un robot parallèle 3RRR, d'autre part développer une chaîne logicielle fiable permettant de passer de la modélisation mathématique à un pilotage réel et sécurisé des servomoteurs.

Les premières semaines ont été consacrées à la préparation expérimentale et administrative nécessaire à toute intégration en laboratoire : prise de contact avec les encadrants, accès aux locaux et aux bancs expérimentaux, mise en place des outils de travail (création d'un dépôt GitHub pour le projet, gestion des versions et sauvegarde des sources). Sur le plan technique, nous avons installé et vérifié les outils et pilotes indispensables : environnement Python, bibliothèques scientifiques (NumPy, Matplotlib), l'interface et le SDK Dynamixel pour la communication avec les servomoteurs, ainsi que les scripts MATLAB fournis en appui pédagogique. Des tests initiaux ont permis de s'assurer du bon fonctionnement des liaisons série et de la détection des actionneurs avant toute expérimentation sur le robot.

Sur le plan scientifique et logiciel, le projet se compose de plusieurs étapes progressives. Nous avons d'abord réalisé un inventaire et une analyse des codes et documents existants (MATLAB, exemples Python, drivers Dynamixel) afin d'en extraire les modèles mathématiques essentielles : géométrie de la plateforme, modèle 2R de chaque bras, formules d'inverse et de directe cinématique, et critères de singularité. À partir de cette base théorique nous avons choisi une stratégie prudente : valider entièrement la chaîne mathématique par simulation avant toute mise en charge du prototype. La phase de simulation (NumPy/Matplotlib) permettait de générer et visualiser des trajectoires, d'identifier les régions inatteignables et de repérer les configurations proches des singularités sans risque matériel.

Une fois la simulation jugée satisfaisante, l'étape suivante a été la calibration et la préparation du pilotage réel : définition d'une procédure de calibration des offsets mécaniques pour chaque Dynamixel, conversion angulaire (angles - ticks), et mise en place d'un envoi synchronisé des commandes. Conscients des risques liés aux mouvements non maîtrisés, nous avons privilégié un réglage itératif et manuel des vitesses et des profils de commande plutôt qu'une automatisation complète : chaque paramètre était d'abord testé en simulation, puis expérimenté progressivement sur la plateforme avec des limites de vitesse et des arrêts d'urgence.

Les objectifs concrets de ce stage étaient donc : (i) formaliser le modèle cinématique du 3RRR et définir des critères de non-atteignabilité et de singularité ; (ii) implémenter ces formules dans une bibliothèque Python réutilisable ; (iii) concevoir une procédure de calibration fiable et reproductible ; (iv) réaliser l'interface de pilotage avec les servomoteurs Dynamixel et vérifier la continuité simulation - expérimentation. Ce travail a abouti à une base logicielle modulaire (modules de cinématique, calibration, contrôleur)

et à une méthodologie expérimentale sécurisée, ouvrant la voie à des expérimentations plus avancées en planification de trajectoires et optimisation.

## 4 Modèle géométrique du robot plan 3RRR

Dans le cadre de ce stage de recherche à l'ISIR nous avions étudié le système plan d'un robot 3RRR composé de trois bras de longueur $L$ et de deux liaisons pivots définie par les angles $\alpha$ et $\beta$. L'objectif est donc d'étudier le comportement géométrique du robot pour pouvoir le modéliser et analyser ses propriétés cinématiques.

Le but est donc de contrôler le déplacement du point $G$, le centre du triangle formé par nos 3 bras, pour se faire on modifiera nos angles $\alpha_i$

Nous avons décidés de résoudre un système de cinématique inverse, on va suivre un chemin de point $G$ connu et déterminer nos angles $\alpha_i$ correspondant à chaque pas de temps. Nos bras sont tous de longueur $L$ et les cotés de notre triangle équilatéral de longueur $h$.

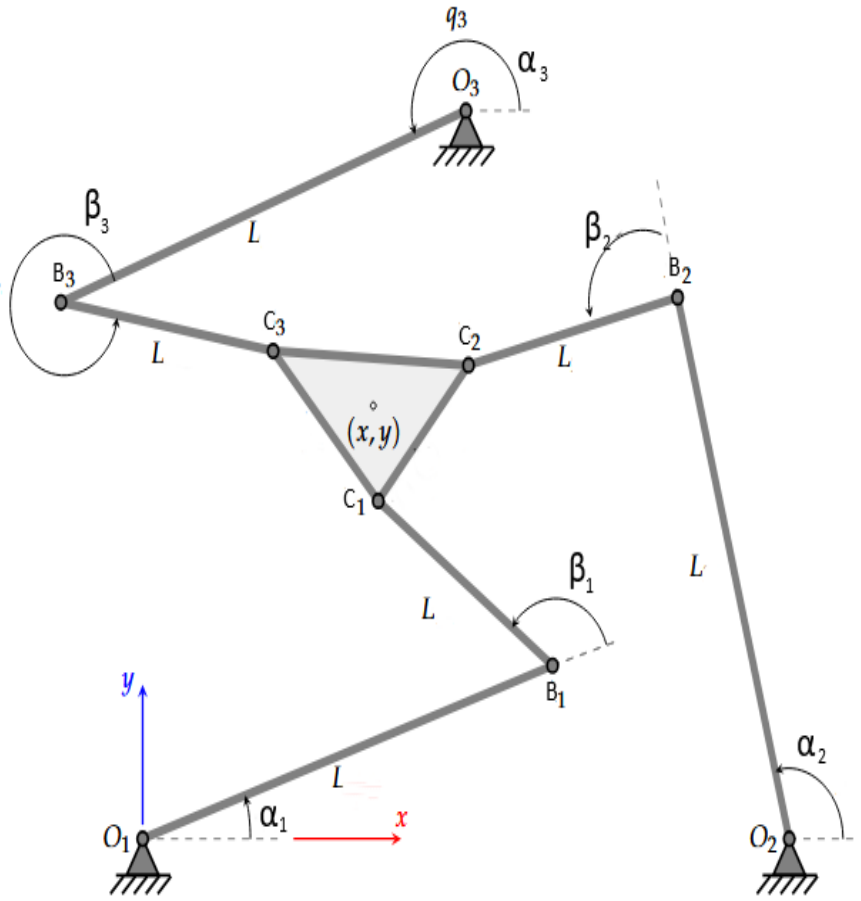


FIG. 1 – Schéma robot 3RRR plan

## 4.1 Cinématique inverse du robot

Le but d'un problème cinématique inverse est de déterminer les angles $(\alpha\,\beta)$ à partir d'une position $(x, y)$

Voici une figure nous permettant de mieux visualiser le problème 2RR :

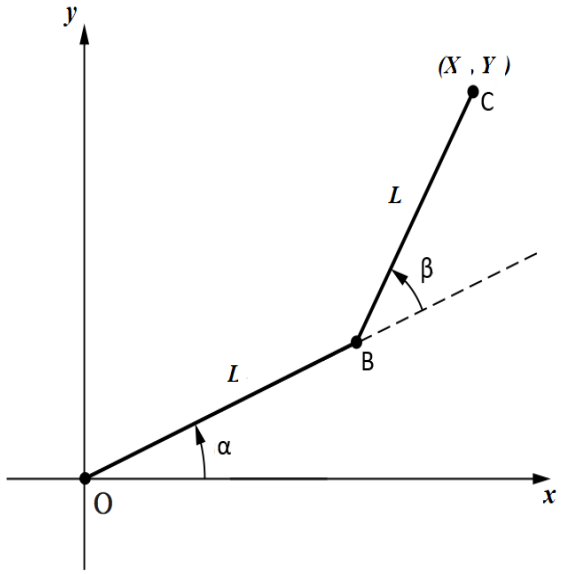


FIG. 2 – Schéma robot 2RR plan

On obtient donc le système d'équations suivant :

$$\begin{cases} x = L\cos\alpha + L\cos(\alpha+\beta) \\ y = L\sin\alpha + L\sin(\alpha+\beta) \end{cases} \tag{1}$$

En appliquant le théorème de Pythagore on obtient $r = x^2 + y^2$, $r$ étant la distance entre O et C $(x, y)$.

Pour déterminer l'angle $\beta$ nous allons utiliser la loi des cosinus. On appelle loi des cosinus l'égalité suivante, valable dans tout triangle ABC, qui relie la longueur des côtés en utilisant le cosinus d'un des angles du triangle $a^2 = b^2 + c^2 - 2b \cdot c\cos(\widehat{A})$.

Avec $\widehat{A}$ l'angle du point A et $a, b, c$ les cotés opposés à ces angles.

Nous visualisons le triangle de notre robot 2RR pour la loi des cosinus de cette manière là :

Appliqué à notre, triangle la loi des cosinus nous donne :

$$r^2 = L^2 + L^2 - 2L^2\cos(\theta)$$
$$r^2 = 2L^2 - 2L^2\cos(\theta)$$
$$\implies cos(\theta) = \frac{2L^2 - r^2}{2L^2}$$

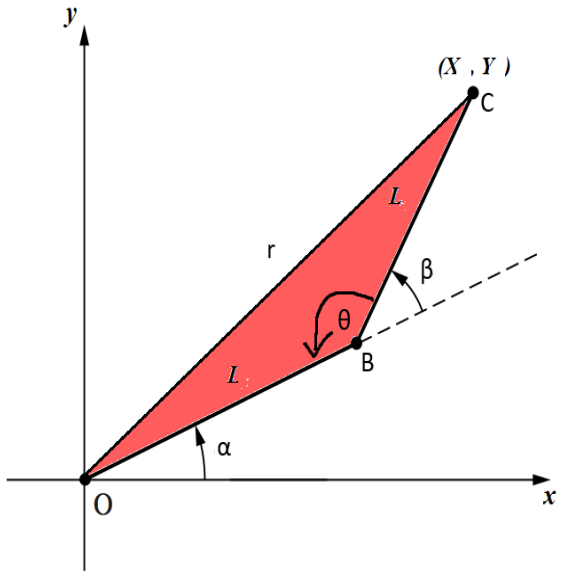


FIG. 3 – Schéma robot 2RR plan, mise en évidence loi des cosinus

Vu que $\beta = \pi - \theta$ on obtient :

$$cos(\beta) = -cos(\theta) = \frac{r^2 - 2L^2}{2L^2}$$

Donc

$$\boxed{\beta = arccos(\frac{r^2 - 2L^2}{2L^2})}$$

Désormais nous introduisons un nouveau triangle, le triangle rectangle 'bleu' sur notre figure. Nous rajoutons aussi les angles $a$ et $b$.

Ce qui nous permet d'obtenir la relation

$$\alpha = b - a$$

Nous voyons naturellement que

$$b = arctan(\frac{y}{x})$$

et de même manière que

$$tan(a) = \frac{Lsin(\beta)}{L + Lcos(\beta)}$$

$$\implies a = arctan(\frac{Lsin(\beta)}{L + Lcos(\beta)})$$

Donc nous avons $\alpha = b - a$ comme mentionné au dessus, de cela découle

$$\boxed{\alpha = arctan(\frac{y}{x}) - arctan(\frac{Lsin(\beta)}{L + Lcos(\beta)})}$$

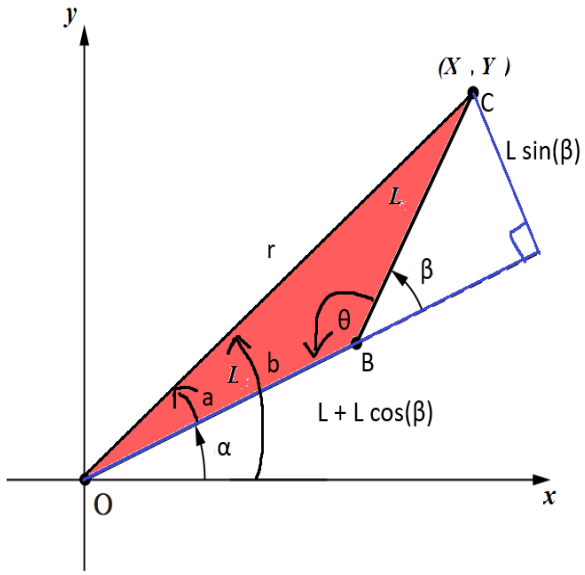


FIG. 4 – Schéma robot 2RR plan, mise en évidence loi des cosinus

## 4.2 Calcul des singularités

Le modèle cinématique d'un robot parallèle 2RR relie les vitesses des actionneurs $q_i$ au torseur des vitesses de l'effecteur $\dot{X}$ exprimé dans un repère local $R_c$. Finalement on obtient l'équation suivante :

$$\boxed{J_x \dot{X} = J_q \dot{q}}$$

Avec $\dot{X} = [G_x, G_y]^T$ et $\dot{q} = [\alpha, \beta]^T$, nous définissons les matrices Jacobiennes :

$$\mathbf{J_q} = \frac{\partial \mathbf{f}}{\partial \mathbf{q}}$$
$$\mathbf{J_X} = \frac{\partial \mathbf{f}}{\partial \mathbf{X}}$$

À partir de ces équations nous pouvons déterminer les différentes singularités.

La singularité série apparaît dès lors que $det(J_q) = 0$ et la singularité parallèle quand $det(J_x) = 0$.

### 4.2.1 Singularité série

Nous posons donc un problème 2RR plan ;

$$\begin{cases} G_x = L\cos\alpha + L\cos(\alpha+\beta) \\ G_y = L\sin\alpha + L\sin(\alpha+\beta) \end{cases} \tag{2}$$

Dans la suite de cette démonstration nous écrirons que $cos(\alpha) = C_1$ et $sin(\beta) = S_2$ etc...

Donc

$$J_q = \frac{\partial \mathbf{f}}{\partial \mathbf{q}} = \begin{pmatrix} -L\sin(\alpha) - L\sin(\alpha+\beta) & -L\sin(\alpha+\beta) \\ L\cos(\alpha) + L\cos(\alpha+\beta) & L\cos(\alpha+\beta) \end{pmatrix}$$

Qui se simplifie en

$$J_q = \begin{pmatrix} -LS_1 - LS_{12} & -LS_{12} \\ LC_1 + LC_{12} & LC_{12} \end{pmatrix}$$

Calculons donc le $det(J_q)$ ;

$$\begin{aligned} det(J_q) &= (-LS_1 - LS_{12})L_{12} + L_{12}(LC_1 + LC_{12} \\ &= L^2(-C_{12}S_1 - C_{12}S_{12} + S_{12}C_1 + S_{12}C_{12}) \\ &= L^2(S_{12}C_1 - C_{12}S_1) \\ &= L^2(sin(\alpha+\beta)cos(\alpha) - cos(\alpha+\beta)sin(\alpha) \end{aligned}$$

On remarque une identité trigonométrique connue, on pose $a = \alpha + \beta$ et $b = \alpha$. L'équation devient alors :

$$= L^2(sin(a)cos(b) - cos(a)sin(b)$$

$$= L^2(sin(a-b) = L^2(sin(\alpha+\beta-\alpha) = L^2(sin\beta)$$

$$\Longrightarrow \boxed{det(J_q) = L^2(sin\beta)}$$

Donc le déterminant de $J_q$ est nul si et seulement si $\boxed{\beta = 0 \text{ ou } \pi}$

### 4.2.2 Singularité parallèle

Pour un robot 3RRR plan, il est plus aisé de comprendre la singularité parallèle en termes de géométrie, que de calculer

$$J_x = \frac{\partial \mathbf{f}}{\partial \mathbf{X}}$$

En effet la singularité parallèle se produit lorsque les trois points d'articulation intermédiaires $B_i$ sont alignés.

Les trois points sont alignés si et seulement si les vecteurs $\overrightarrow{B_1B_2}$ et $\overrightarrow{B_1B_3}$ sont colinéaires.

$$\Longrightarrow \boxed{det(\overrightarrow{B_1B_2}, \overrightarrow{B_1B_3}) = 0}$$

## 4.3 Calcul de la distance entre les points $C_i$ et $G(x, y)$

Chaque point $C_i$ est le bout d'un bras de notre robot 3RRR, mais c'est aussi un point de notre triangle équilatéral de longueur $h$. Au centre de ce triangle se trouve le point $G(x, y$, qui est donc l'endroit ou se situe notre crayon, et donc notre tracé.

Nous allons donc ici calculer la distance entre $C_i$ et $G(x, y)$, voici une figure pour commencer :

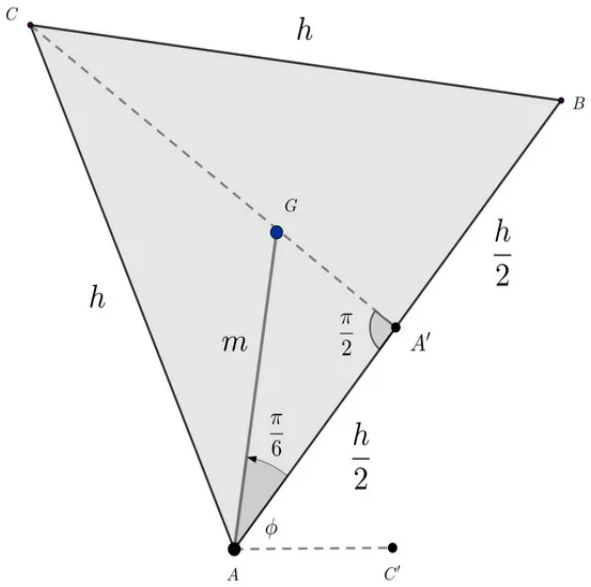


FIG. 5 – Triangle équilatéral avec $C_i = (A, B, C)$

Nous posons donc la distance $|AG| = m$ et le point $A'$ se situant entre $A$ et $B$. Nous créons donc un nouveau triangle, le triangle équilatéral $\Delta AGA'$.

On a donc que

$$cos(\frac{\pi}{6}) = \frac{AA'}{AG}$$

$$\implies AG = \frac{h}{2}\frac{1}{cos(\frac{\pi}{6})} = \frac{h}{2}\frac{2}{\sqrt{3}}$$

$$\implies \boxed{AG = \frac{h}{\sqrt{3}}}$$

Avec $h$ les cotés de notre triangle équilatéral.

# 5 Développement du code de pilotage du robot 3RRR

## 5.1 Simulation numérique du robot parallèle 3RRR

Afin de valider la cinématique du robot avant tout test sur le prototype réel, une simulation numérique a été développée sous Python à l'aide des bibliothèques NumPy et Matplotlib. Cette phase permet de contrôler la cohérence des équations, d'observer les mouvements générés, et de détecter les zones inatteignables sans risquer d'endommager le robot.

La simulation numérique développée en Python a permis de reproduire le fonctionnement du robot et d'observer le mouvement des trois bras ainsi que la trajectoire de la plateforme. À partir des paramètres géométriques et du point central $G$ , elle vérifie si la position est atteignable et visualise la configuration des bras, tout en mettant en évidence d'éventuelles singularités ou limites mécaniques.
Ce travail préliminaire joue un rôle important, notamment dans le cadre d'un projet où les servomoteurs sont coûteux et potentiellement fragiles.Un mauvais calcul pourrait entraîner un blocage mécanique, une surtension dans les articulations ou même la casse du robot. La simulation sert donc à tester des mouvements rapidement, sans aucune conséquence matérielle, ce qui en fait un outil d'expérimentation libre et sans risque.

Le fonctionnement du code de simulation repose sur plusieurs modules : la définition géométrique du robot, le calcul cinématique inverse, la reconstruction cinématique directe ainsi que la génération et l'animation d'une trajectoire. Chaque étape est calculée en continu image par image, ce qui permet de visualiser le robot.

### 5.1.1 Initialisation des paramètres géométriques

Dans un premier temps, le programme définit l'ensemble des caractéristiques physiques du robot :

— La longueur des bras $L$
— La taille d'un côté du triangle de la plateforme $h$
— Le rayon du cercle moteur $R$
— Les positions angulaires des trois servomoteurs qui sont espacés de 120°

À partir de ces informations, les coordonnées des points fixes $O_i$ sont calculées grâce à une simple projection trigonométrique :

$$O_x = R\cos(\theta_i), \quad O_y = R\sin(\theta_i)$$

Ces points servent d'origine à chacun des bras articulés et ne changent jamais durant la simulation.

### 5.1.2 Cinématique inverse : calcul des angles moteurs

Pour chaque nouvelle position cible $(x, y)$ du point central $G$, la simulation génère la position des trois points $C_i$ de la plateforme mobile. Ces points sont définis en décalant le centre $G$ selon la géométrie d'un triangle équilatéral tourné à -30°. Ils représentent la zone où les bras viennent se fixer.

$$C_i(x, y) = G(x, y) + \text{rotation et décalage associé au triangle}$$

Une fois ces points connus, la cinématique inverse cherche à déterminer les angles $\alpha_i$ permettant de relier chaque moteur $O_i$ à son point $C_i$ via deux segments de longueur fixe $L$. Le problème revient à résoudre un triangle dont trois longueurs sont connues :

$$r_i^2 = (x_C - x_O)^2 + (y_C - y_O)^2$$

Le cosinus de l'angle interne est ensuite obtenu par la loi des cosinus :

$$\cos(\beta_i) = \frac{r_i^2 - 2L^2}{2L^2}$$

Si $|\cos(\beta_i)| > 1$, cela signifie que le point est géométriquement inaccessible : le bras n'a pas la longueur suffisante pour atteindre la position. Dans ce cas, la simulation stoppe l'animation à cet instant et signale un point hors zone de travail. Lorsque le calcul est valide, l'angle d'entrée moteur est déterminé par :

$$\alpha_i = \arctan 2(dy, dx) - \arctan 2\big(L\sin(\beta_i),\ L + L\cos(\beta_i)\big)$$

Ces trois angles constituent le cœur de la commande robotique, car ce sont les mêmes valeurs qui seront envoyées plus tard aux servomoteurs réels.

### 5.1.3 Cinématique directe : reconstruire les positions intermédiaires

Une fois les angles calculés, le programme détermine la position des points $B_i$ (coudes) par :

$$B_x = O_x + L\cos(\alpha_i), \qquad B_y = O_y + L\sin(\alpha_i)$$

Cette étape sert à tracer visuellement les bras dans l'animation. On obtient ainsi la chaîne complète :

$$O_i \to B_i \to C_i$$

Elle permet également de vérifier la cohérence du modèle : si la cinématique directe ne reconstruit pas la plateforme correctement, cela signifie que l'IK n'est pas fiable ou que le robot est proche d'une singularité.

### 5.1.4 Génération d'une trajectoire

Nous allons prendre l'exemple de notre simulation pour un cercle. Nous avons pris la décision de simuler un cercle de rayon de 2 cm.

```
i = 0
while(i < 125):
targetx = 0.00 + 0.02 * np.cos(i * 0.05)
targety = 0.00 + 0.02 * np.sin(i * 0.05)
targetpoints.append((targetx, targety))
i += 1
```

La variable $i$ joue le rôle d'un paramètre angulaire progressif : à chaque tour de boucle, elle augmente et génère un nouveau point sur le cercle. Les coordonnées calculées sont ensuite stockées dans la liste `targetpoints`, ce qui constitue l'ensemble des positions que le robot devra atteindre dans l'ordre. Ainsi, en parcourant la liste point après point, la plateforme décrit progressivement la trajectoire circulaire complète.
Chaque point est introduit dans le solveur IK puis affiché successivement, ce qui permet d'observer la plateforme se déplacer comme si le robot était réel. Le code nous fournit une interface dynamique dont nous parlerons dans la section suivante.

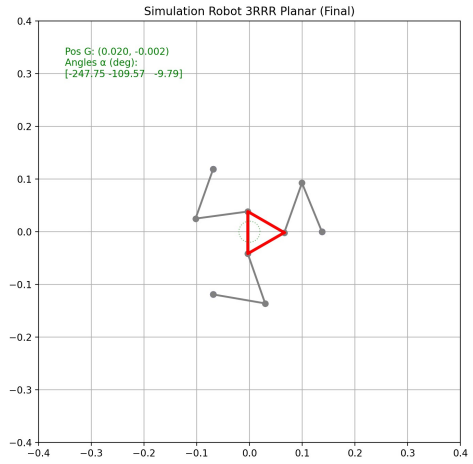


FIG. 6 – Interface simulation cercle

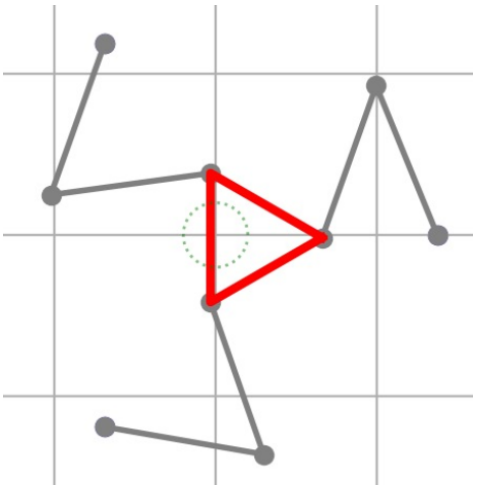

FIG. 7 – Vue zoom

Ces deux images représentent l'interface réalisée pour un cercle, ce sont des captures d'écran de l'animation. A gauche, la vue d'ensemble et à droite, une vue zoomée.

De même, voici le résultat obtenu pour un carré :

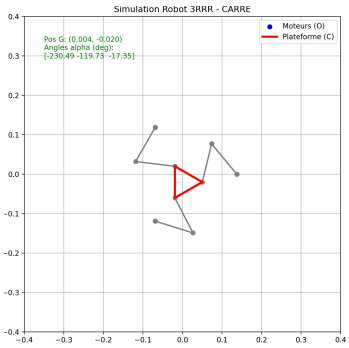


FIG. 8 – Carré 1 simulation

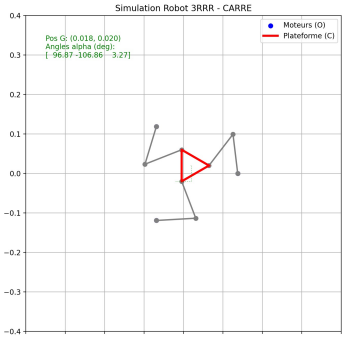


FIG. 9 – Carré 2 simulation

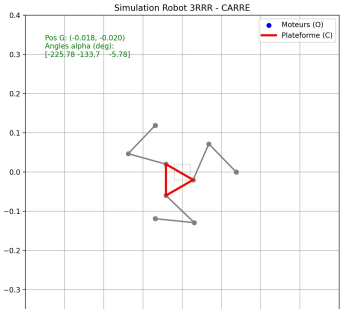


FIG. 10 – Carré 3 simulation

### 5.1.5 Animation et visualisation dynamique

Matplotlib met à jour l'affichage image par image via FuncAnimation.
À chaque cycle cela nous permet de :

- Récupérer un point cible
- Calculer les angles moteurs
- Reconstruire la cinématique
- Mettre à jour les segments graphiques
- Stocker la position $G$ pour tracer son mouvement

Le résultat final est une animation fluide où le mécanisme se déplace en temps réel. La trajectoire finale apparaît en vert, ce qui permet une comparaison immédiate entre mouvement demandé et mouvement obtenu.
L'interface graphique générée avec Matplotlib affiche simultanément la base fixe du robot avec les trois moteurs, les trois bras articulés représentés dynamiquement, la plateforme mobile sous forme de triangle équilatéral et la trajectoire du point central $G$, affichée en vert au fur et à mesure.

Cette représentation permet d'évaluer immédiatement la fluidité du mouvement. Les variations des angles articulaires peuvent être observées visuellement, mais également sous forme numérique grâce à l'affichage en continu dans la fenêtre de simulation. Il est donc possible de vérifier si les déplacements sont cohérents, progressifs, et surtout si le mécanisme n'atteint pas des configurations extrêmes proches des singularités. La simulation a également révélé des zones de travail plus sensibles où les bras se rapprochent fortement d'un alignement. Ces cas représentent des configurations singulières

dans lesquelles le contrôle du robot devient instable ou imprécis. Le programme permet d'identifier ces régions sans aucune manipulation physique, offrant ainsi un moyen d'ajuster la géométrie ou la trajectoire avant de commander les servomoteurs.

La simulation a donc constitué une étape indispensable pour la suite du projet. Elle a permis de valider la cohérence du modèle mathématique, de visualiser la cinématique complète du mécanisme, et de générer des trajectoires autorisées, fluides et exploitables. Grâce à elle, les premiers tests réalisés ensuite sur les servomoteurs Dynamixel ont pu être lancés avec un risque réduit et une confiance accrue dans le fonctionnement du système. Ce travail prépare directement l'implémentation sur robot réel, car les mêmes angles issus de la simulation sont ensuite envoyés physiquement aux moteurs. On peut ainsi passer de la modélisation numérique au pilotage matériel sans reconception supplémentaire, ce qui constitue un gain considérable en temps et en fiabilité.
En annexe se trouve des captures d'écran de l'interface de simulation pour un cercle ainsi que pour un carré.

## 5.2 Pilotage réel du robot 3RRR à l'aide des servomoteurs Dynamixel

### 5.2.1 Servomoteurs Dynamixel et logiciel Dynamixel Wizard 2.0

Pour la mise en œuvre réelle du robot 3RRR, les calculs effectués dans la simulation doivent ensuite être convertis en commandes envoyées aux actionneurs. Dans notre projet, les moteurs utilisés sont des Dynamixel MX-28. Ils sont capables de mesurer, contrôler et communiquer des informations, notamment leur vitesse, position, angle. Chaque moteur Dynamixel reçoit un angle issu de la cinématique inverse calculée précédemment. Cet angle est ensuite transformé en une valeur exploitable par les servomoteurs, afin de reproduire fidèlement la position calculée en simulation.

Avant de pouvoir utiliser les moteurs sur le robot, il est nécessaire de les paramétrer. Pour cela, nous avons utilisé Dynamixel Wizard 2.0, un logiciel fourni par ROBOTIS. Cet outil permet de détecter automatiquement les moteurs connectés, de modifier leurs identifiants de communication, d'ajuster leurs paramètres de fonctionnement (notamment la vitesse) et de vérifier leur comportement.

### 5.2.2 Structure et fonctionnement du code

Le programme Python développé permet de piloter en temps réel les trois servomoteurs Dynamixel du robot 3RRR à partir des coordonnées souhaitées du point mobile $G(x, y)$. Il a pour rôle de convertir une position cartésienne en angles articulaires via la cinématique inverse, puis de transmettre ces valeurs sous forme de ticks exploitables par les moteurs.

Après avoir défini l'adresse mémoire des paramètres Dynamixel (position, couple, vitesse), la communication est ouverte sur le port série choisi avec un baudrate de 1 million de bits par seconde.
Les trois moteurs, identifiés via les ID [1,2,3] sont ensuite activés (Torque Enable) et configurés à une vitesse définie à l'aide des fonctions :

```
enable-torque(dxl-id)
set-speed(dxl-id, VITESSE)
move-motor(dxl-id, tick)
```

Tous les moteurs sont d'abord envoyés à une même orientation grâce à une fonction `initialisation(110°)` avant le début du mouvement, afin de démarrer dans une configuration stable.

Les servomoteurs Dynamixel interprètent une position non pas en degrés, mais en unités internes (ticks). Un tour complet correspond à 4096 unités, ce qui impose une

conversion systématique des angles calculés. Le programme intègre donc une fonction de transformation :`ticks = (angle-deg / 360) * 4096`.
Ainsi, un angle de 90° correspond à 1024 ticks, 180° à 2048 etc.

Cependant, même avec une conversion correcte, le mouvement réel du robot reste inexact si l'on ne tient pas compte des décalages mécaniques entre la position réelle des moteurs et la position théorique. Durant le montage, les servomoteurs ne sont jamais alignés parfaitement avec l'orientation qui devrait correspondre à l'angle zéro de la simulation. Pour corriger cette différence, un offset propre à chaque moteur a été mesuré à l'aide du logiciel Dynamixel Wizard 2.0 et d'un autre code à partir de la position d'initialisation à 110°. Une fois ces offsets déterminés, ils sont ajoutés systématiquement aux angles calculés avant leur conversion en ticks `tick = degres-to-moteur(angle-a[i] + OFFSET[i])`, garantissant ainsi que la position envoyée corresponde réellement à celle prévue en simulation.

Une fois cette étape résolue, l'exécution du mouvement peut être automatisée. Lorsque l'utilisateur demande au robot d'atteindre un point $(x, y)$, la fonction `mouvement-vers()` calcule d'abord les 3 angles moteurs grâce à la cinématique inverse. Elle vérifie ensuite que la position est physiquement réalisable et qu'elle ne conduit pas le robot vers une singularité, c'est-à-dire une configuration dans laquelle le mouvement devient impossible ou dangereux. Si la position est valide, les angles corrigés par les offsets sont convertis en ticks puis transmis individuellement aux trois servomoteurs.

Le robot se déplace ainsi point après point le long d'une trajectoire définie dans le programme. Un temps d'attente `time.sleep(0.8)` est volontairement imposé entre chaque déplacement afin de laisser aux moteurs le temps de converger vers leur objectif, ce qui permet d'obtenir un mouvement fluide et évite les contraintes inutiles. Un système d'arrêt d'urgence a également été intégré :

```
if key == keyboard.Key.space:
stop-program = True
```

Une simple pression sur la barre d'espace interrompt immédiatement l'exécution du code et coupe le couple des moteurs afin de préserver le matériel en cas de calcul incorrect ou de mouvement inattendu.

Au final, ce code permet de faire la passerelle entre notre simulation vue précédemment et la version physique du robot. La simulation génère les positions souhaitées, les fonctions de pilotage traduisent ces données en commandes réelles, et les servomoteurs reproduisent les mouvements calculés. Cette structure assure une continuité entre théorie et expérimentation.

# 6 Conclusion

Ce travail nous a permis d'aborder l'étude complète d'un robot parallèle 3RRR, depuis sa modélisation géométrique jusqu'à son pilotage réel à l'aide de servomoteurs Dynamixel. Dans un premier temps, nous avons établi les équations de cinématique inverse nécessaires au calcul des angles moteurs en fonction d'une position souhaitée du point mobile. Cette étape mathématique a constitué la base du projet, puisqu'elle permet de relier directement l'espace cartésien de la trajectoire à l'espace articulaire des moteurs. Sans cette formulation, le robot n'aurait aucune information pour se déplacer, ce qui montre le rôle central des mathématiques dans la conversion d'un mouvement théorique à un mouvement réel.

Une fois le modèle géométrique établi, nous avons développé une simulation afin de visualiser et valider le comportement du robot avant toute manipulation physique. La simulation s'est révélée être un outil indispensable pour analyser la faisabilité des trajectoires, détecter les zones de singularités et observer l'évolution des trois bras en mouvement. Cette étape nous a offert un environnement d'expérimentation sûr, dans lequel il est possible de tester un grand nombre de positions sans risquer d'endommager le matériel. Elle a servi de pont entre la théorie et la pratique, en permettant de vérifier que les équations obtenues conduisent bien à un mouvement cohérent du mécanisme.

Enfin, nous avons mis en œuvre le pilotage réel du système à l'aide des servomoteurs Dynamixel MX-28. Les angles calculés ont été convertis en valeurs internes comprises entre 0 et 4096 ticks, puis corrigés par un offset propre à chaque moteur afin d'aligner la référence théorique avec la position mécanique réelle. Le contrôle a été implémenté en Python, en intégrant la gestion des positions, des vitesses, de la sécurité ainsi qu'un arrêt d'urgence manuel. Le robot a ainsi pu reproduire physiquement les trajectoires testées en simulation, démontrant la validité de notre chaîne de calcul complète.

Ce projet nous a donc permis de comprendre et de maîtriser toutes les étapes nécessaires au fonctionnement d'un robot parallèle 3RRR : modélisation mathématique, vérification par simulation, puis commande réelle par actionneurs. Il illustre parfaitement comment un modèle théorique peut se transformer en mouvement tangible, et souligne l'importance du lien entre science, algorithmique et expérimentation physique.

# 7 Bibliographie

## Références

[1] 'Direct and Inverse Kinematics of a 3RRR Symmetric Planar Robot : An Alternative of Active Joints' https://www.mdpi.com/2073-8994/16/5/590#symmetry-16-00590-f002 *Consulté le 1 novembre 2025*

[2] 'Condition nécessaire et suffisante pour que trois points du plan soient alignés' https://uel.unisciel.fr/mathematiques/determinant1/determinant1_ch02/co/sexercer_ch2_12.html *Consulté le 29 novembre 2025*

[3] 'Formule d'al-Kashi, loi des cosinus' https://www.bibmath.net/dico/index.php?action=affiche&quoi=./a/alkashi.html *Consulté le 17 novembre 2025*

[4] 'Kinematic Modelling of a 3RRR Planar Parallel Robot Using Genetic Algorithms and Neural Networks' https://oa.upm.es/84964/1/10108582.pdf *Consulté le 3 novembre 2025*