ENSEIRB-ENSC

Parcours Robotique et Apprentissage

Projet Robotique

travail réalisé par

Kloé Bonnet et Guillaume Lauga

Rapport de projet : Prise en compte des contraintes intrinsèques et extrinsèques lors de la téléopération de robots : application au robot Reachy

---

Année universitaire 2024 - 2025

Encadrement

M.	Rémi FABRE	Ingénieur en robotique à Pollen Robotics
M.	Vincent PADOIS	Chercheur à Inria - Auctus

Table des matières

Table des figures

Introduction

Reachy2 est un robot anthropomorphe conçu par la startup *Pollen Robotics* [46] mettant à disposition son code en *open source* [42] ainsi qu'une documentation de celui-ci [43]. Ce robot peut notamment être contrôlé en téléopération à l'aide d'un casque VR (Réalité Virtuelle, en anglais *Virtual Reality*). L'opérateur contrôle les bras et pinces de Reachy2 à l'aide de manettes et le robot doit alors réaliser les consignes données de façon optimale tout en respectant des contraintes intrinsèques et extrinsèques. A l'image des bras humains, ils possèdent sept articulations, offrant ainsi sept degrés de liberté. L'espace de contrôle du robot (*joint space*) est ainsi de dimension 7, tandis que l'espace de travail (*task space*) est de dimension 6 : le bras est donc "redondant". Il peut ainsi exister plusieurs solutions, voire une infinité de solutions, qui permettent d'atteindre une pose souhaitée [41].

En téléopération, la continuité de la loi de commande doit être toujours être garantie à tout instant. Cependant, un décalage peut parfois se produire entre l'intention de l'opérateur et l'action réellement effectuée par Reachy2. Un léger mouvement de l'opérateur peut entraîner un mouvement disproportionné du robot. Ce phénomène est particulièrement observable à proximité des positions de singularité où les capacités de contrôle du robot sont limitées.
Si pendant un temps, la résolution de la cinématique inverse du robot était effectuée de manière numérique en utilisant une méthode itérative, elle est actuellement résolue de façon analytique [41]. Cette nouvelle approche permet une solution plus rapide et plus précise dans les mouvements du robot. La gestion des limites articulaires est aussi implémentée avec une solution approximative pour les poses, lorsque les contraintes physiques du robot sont proches de leurs limites. Cependant, la question de "Que doit faire le robot dans ces situations ?" est à résoudre.
En effet, bien qu'une implémentation ait été réalisée pour gérer ces cas, elle manque de robustesse et de flexibilité, demeurant ainsi une problématique ouverte.

L'objectif de ce document est donc d'étudier les alternatives de l'approche de la cinématique inverse mise en œuvre par *Pollen Robotics* dans le cadre de l'architecture du robot Reachy2. Au regard de ce travail, plusieurs alternatives sont proposées pour résoudre le problème de la startup.

Ce rapport présente dans un premier temps les divers problèmes rencontrés lors de la téléopération d'un robot anthropomorphe ainsi que leurs causes. Dans un second temps, il aborde les problèmes liés à la cinématique en robotique, en présentant notamment diverses solutions pour résoudre les problèmes de cinématique inverse, leurs expressions mathématiques, ainsi que des solveursexistants pour une résolution en programmation. Puis, une troisième partie présente les méthodes d'implémentation de contraintes et les différents moyens de les résoudre. Enfin, ce document propose un ensemble de solutions adaptées aux problèmes spécifiques de *Pollen Robotics*.

1. Problème lié à la téléopération

Cette première partie définie les termes du domaine de la téléopération. Elle présente ensuite des exemples de représentation visuelle pour mieux comprendre les solutions parfois multiples. Et enfin, les limites des solutions en cinématique inverse, dont les phénomènes de singularité et les contraintes physiques des robots, sont abordés.

1.1. Définitions

La téléopération signifie “Faire un travail à distance” ou “contrôler un véhicule ou un système externe à distance” [32]. Elle “vise à transférer les compétences et la dextérité de manipulation humaine sur une certaine distance (arbitraire) et à une certaine échelle (arbitraire) vers un lieu de travail éloigné” [51]. La téléopération fait intervenir un “opérateur” qui est la personne qui va contrôler et diriger le système et ses actions, et le “téléopérateur” qui est le système dirigé comme on peut le voir sur la Figure 1.1. L’opérateur peut envoyer les données par l’intermédiaire de manettes [46], d’une interface [7] ou par le suivi en temps réel de capteurs portés par celui-ci [40]. Les mouvements de l’opérateur sont alors retransmis au téléopérateur par l’intermédiaire de signaux directs permettant ainsi une rétroaction en temps réel. On parle alors de commande en boucle fermée. Enfin, la “téléprésence” est l’incarnation du téléopérateur par l’opérateur, cela est possible si un grande quantité d’informations lui sont transmises (vision, son, force, ...) [32]

Il existe divers systèmes téléopérés, que ce soit des robots humanoïdes [46, 6] , des avatars virtuels représentant des humains [18] ou des animaux [30] . Des exemples sont présentés sur les Figures 1.2 et 1.3

Afin de réaliser la consigne donnée par l’opérateur, il faut pouvoir réaliser la traduction entre le vecteur position et orientation de la pose de l’opérateur et le vecteur des angles articulaires du système téléopéré. Cette transition entre ces deux vecteurs est réalisée par la cinématique directe, pour passer des valeurs articulaires à la pose de l’organe terminal, et par la cinématique inverse, pour passer de la pose de l’organe terminal aux valeurs articulaires [35].
La cinématique directe est un calcul géométrique et/ou trigonométrique et ne pose

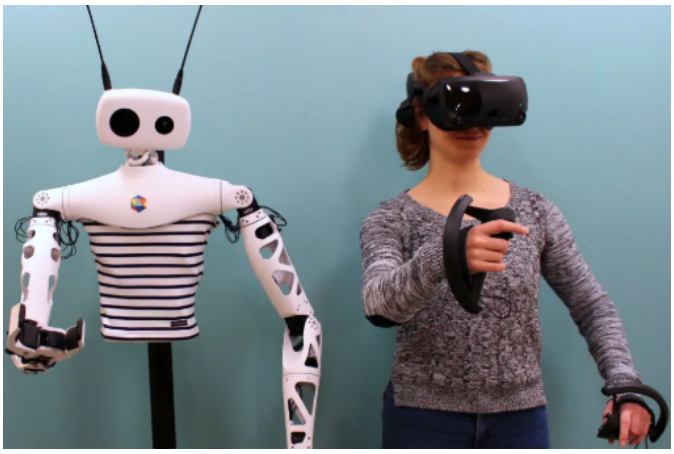

FIGURE 1.1. – Téléopérateur et opérateur [46]

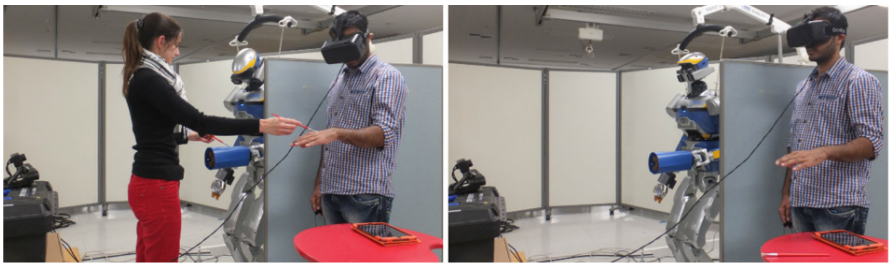

FIGURE 1.2. – Robot humanoïde, plusieurs mains et pinces sont testées [6]

FIGURE 1.3. – Avatars du tigre, de l'araignée, de la chauve-souris et de l'humain testées [30]

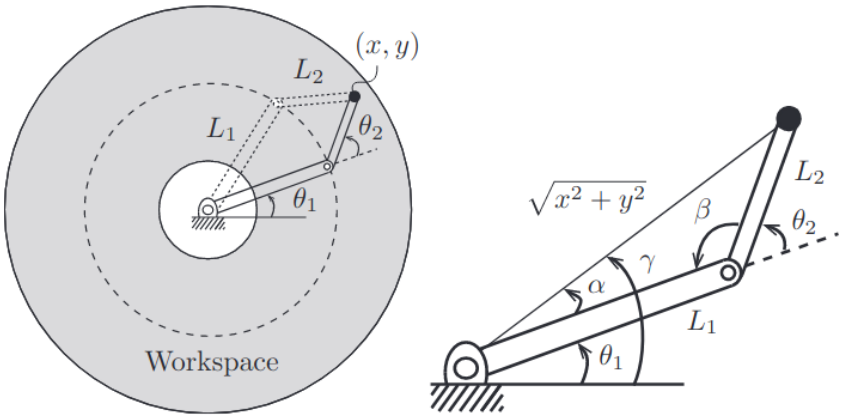

FIGURE 1.4. – Schéma du problème de cinématique inverse pour un robot plan 2R [35]

pas de grand problème. La cinématique inverse est cependant beaucoup plus difficile à calculer et pose plusieurs difficultés qui complexifient la téléopération. Ces différentes cinématiques sont davantage détaillées dans le Chapitre 2 suivant.

1.2. Solutions multiples

À l'inverse de la cinématique directe, un problème de cinématique inverse peut avoir une, plusieurs, voire aucune solution. Prenons par exemple un robot plan 2R (c'est-à-dire, avec deux joints rotatifs ou rotoïdes) présenté sur la Figure 1.4. On cherche les valeurs des angles articulaires q_1 et q_2 (notés θ_1 et θ_2 sur le schéma) permettant d'atteindre une position donnée (x, y) [35, 4]. Si la position cible est en dehors de la couronne circulaire définie par la portée des bras, il n'existe aucune solution. Si elle se situe sur la bordure de la couronne, il n'y a qu'une seule solution. En revanche, si elle est à l'intérieur de la couronne circulaire, deux solutions sont possibles.

Dans le cas d'un problème en trois dimensions avec un robot 7R, on peut représenter l'ensemble des configurations articulaires possibles dans une sphère [42]. Pour une pose de l'effecteur terminal située dans cette sphère, l'ensemble des solutions se trouve sur le cercle formé par l'intersection des deux sphères correspondant aux segments du bras robotique, comme illustré sur la Figure 1.5. Il convient de noter que cette intersection se réduit à un point unique lorsque le robot atteint une pose de singularité. L'effecteur terminal se trouve alors sur la surface de la sphère.

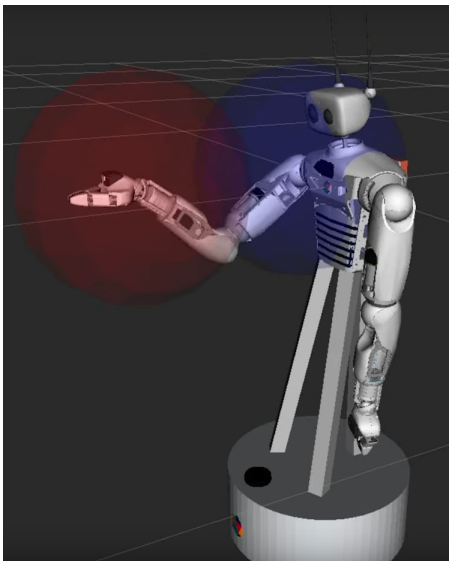

FIGURE 1.5. – Schéma du problème de cinématique inverse pour un robot 7R [42]

1.3. Butées articulaires et singularités

D'un autre côté, il existe des poses où le robot perd de la mobilité. Une des causes est une simple butée articulaire où l'une des articulations est aux limites de son possible.

Une autre cause provient de l'effecteur terminal du robot perdant la capacité de se déplacer instantanément dans une ou plusieurs directions. Une telle posture est appelée **singularité cinématique,** ou simplement **singularité**, et se traduit mathématiquement à travers la jacobienne $J(q)$ ne parvenant pas à avoir un rang maximal [35] et peut être détecté par l'utilisation la décomposition en valeur simple, plus connue sous le nom de *Singular Value Decomposition* (*SVD*) [45]. Des comportements atypiques, dû par exemple a des divisions par zéro ou une valeur très petite, peuvent alors être observés à l'approche de ces singularités.

Un autre problème majeur de la cinématique inverse est qu'elle permet de calculer les articulations nécessaires pour une pose donnée, sans garantir au préalable que cette pose soit atteignable par le robot. Cette vérification est réalisée par la suite en regardant si chaque consigne articulaire est possible. Dans le cas où une des consignes articulaires n'est pas réalisable, il faut trouver une solution alternative car, dans le cas contraire, les moteurs tenteront de bouger une articulation qui est dans une butée, pouvant alors créer des dommages matériels et/ou physiques [25].

2. Résolution de la Cinématique Inverse : Approches et Limitations

Cette partie présente les problèmes de cinématique en robotique et en particulier ceux de la cinématique inverse. Les détails de certaines méthodes sont développés ainsi que l'écriture de contraintes. Des solveurs pour une résolution en programmation sont aussi évoqués.

2.1. Écriture d'un problème de cinématique

Avant d'expliquer le principe de la cinématique directe et de la cinématique inverse, il est important de comprendre la différence entre les différents espaces comme présenté sur la Figure 2.1.

Dans le *joint space* (espace des joints), les positions des articulations sont représentées par les coordonnées intrinsèques du vecteur $\mathbf{q} = [q_1, \ldots, q_n], \mathbf{q} \in \mathbb{R}^n$, où n est le nombre d'articulations du robot. Dans le *task space* (espace de travail), ou *cartesian space* (espace cartésien), les coordonnées extrinsèques $\mathbf{x} \in \mathbb{R}^m$ représentent la position et l'orientation, aussi appelée « pose », de l'effecteur terminal [31].
Les transformations entre ces deux espaces permettent la planification de trajectoires et la commande du robot. Elles sont permises grâce à la cinématique directe et la cinématique inverse.

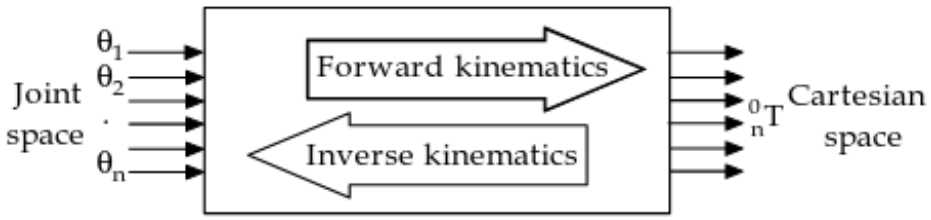

FIGURE 2.1. – Comparaison entre la cinématique directe (*Foward kinematics - FK*) et la cinématique inverse (*Inverse kinematics - IK*) [31]

La cinématique directe (*Forward Kinematics - FK*) permet, avec un ensemble de positions articulaires $\mathbf{q}$, de trouver une position et une orientation d'effecteur terminal unique. [31].
Avec le vecteur des angles articulaires $\mathbf{q}$ et le vecteur de la pose de l'effecteur terminal $\mathbf{x}$ définis précédemment, la fonction de cinématique directe peut alors s'écrire ainsi [16], [35] :

$$\mathbf{x} = f(\mathbf{q}) \tag{2.1}$$

Un problème de cinématique inverse (*Inverse Kinematics - IK*) a pour objectif de déterminer un ensemble d'angles d'articulations permettant de réaliser la configuration attendue de l'effecteur [35]. On peut écrire la fonction de cinématique inverse ainsi :

$$\mathbf{q} = f^{-1}(\mathbf{x}) \tag{2.2}$$

Ce problème peut devenir difficile à résoudre car la solution n'est pas toujours unique. Il peut exister plusieurs solutions ou alors aucune [35].
La suite de ce document se concentre davantage sur les problèmes de cinématique inverse.

2.2. Méthodes de résolutions de cinématique inverse et leurs limites

Plusieurs travaux précédents présentent une liste et explication de méthodes de résolution de problèmes de cinématique inverse [35, 5, 4]. Le tableau 2.1 décrit la différence entre les méthodes de résolution **analytique** et celles **numérique** [35, 44, 31, 29, 33]. Les avantages et inconvénients de chacune des méthodes y sont présentés ainsi que des exemples dans le Tableau 2.2.

Il existe deux principales approches pour la **méthode analytique** [31]. La première est la **résolution géométrique**, qui s'applique généralement aux structures simples. Elle repose sur une analyse directe des relations spatiales (cercles, sphères, etc) pour déterminer les configurations articulaires. La seconde est la **résolution algébrique**, utilisée pour des cas plus complexes, où des équations algébriques, issues des relations cinématiques, sont résolues pour obtenir les solutions. Dans ces deux approches permises par des calculs formels, les données d'entrée incluent la pose désirée de l'effecteur terminal $\mathbf{x_d}$ ainsi que les longueurs des segments de la structure. En sortie, on détermine alors les variables articulaires $\mathbf{q_d}$, définissant la configuration requise pour atteindre $\mathbf{x}_d$.
Plusieurs approches basées sur des **méthodes numériques** sont couramment

utilisées pour résoudre les problèmes de cinématique inverse, notamment des **méthodes itératives** (Newton-Raphson, méthode de la bissection, méthode de la sécante, descente de gradient), des **méthodes utilisant les dérivées** (jacobienne transposée, pseudo-inverse jacobienne), des **méthodes hybrides** combinant plusieurs approches (moindres carrés amortis (DLS), algorithme de cinématique inverse en boucle fermée (CLIK)) ou encore des **méthodes d'optimisation** (LP, QP, SOCP, SDP) [5, 4]. Ces grandes familles de méthodes partagent des paramètres d'entrée communs : une configuration initiale $\mathbf{q}_0$, une pose cible désirée $\mathbf{x}_d$, une fonction $f(\mathbf{q})$ décrivant la relation entre les configurations articulaires et la pose, et, pour certaines, la jacobienne $J(\mathbf{q})$. En sortie, elles fournissent une configuration articulaire approximative $\mathbf{q}_d$ et une erreur résiduelle, souvent mesurée par la distance entre la pose atteinte et la pose cible.

	Méthodes analytique	**Méthodes Itératives**
Avantages	- Solution exacte, sans approximation - Pas de dépendance aux conditions initiales	- Flexibilité : Applicable à une large gamme de robot - Adaptabilité : peut intégrer des contraintes d'environnement - Gestion de la redondance avec des critères d'optimisations
Inconvénients	- Complexe pour certains systèmes - Rigide, n'est pas adapté dans des conditions changeantes - Ne comprend pas le calcul de la dérivé	- La solution est une estimation - Le temps de calcul peut être long - Dépend des conditions initiales

TABLE 2.1. – Méthodes de résolution de cinématique inverse

2.3. Détails des méthodes de cinématique inverse et solveurs

Cette partie explicite l'écriture de diverses méthodes de résolution de cinématique inverse. Elle présente aussi des solveurs permettant la résolution de ces problèmes d'un point de vue programmation. Enfin, l'écriture de contraintes est présentée.

Catégorie	Méthodes analytiques	Méthodes numérique
Exemples	- Méthode géométrique [31] - Méthode algébrique [31]	- Newton-Raphson [35, 44] - Méthode de la bissection [49] - Méthode de la sécante [55] - Descente de gradient [35, 11, 25] - Jacobienne transposée [5, 4] - Pseudo-inverse jacobienne [35] - Moindres carrés amortis (DLS) [5, 4] - Algorithme de cinématique inverse en boucle fermée (CLIK) [14, 54] - Méthodes d'optimisation (LP [8, 17], QP [25, 52, 22, 9], SOCP [28, 37], SDP [28, 53], CP [23], ...)

TABLE 2.2. – Comparaison des méthodes analytiques et itératives pour la résolution de problèmes de cinématique inverse.

2.3.1. Méthodes analytique

Cette méthode consiste à trouver la ou les solutions exactes du problème sous la forme d'expressions algébriques à l'aide d'équations mathématiques et de la géométrie du robot utilisée. Par exemple, des résolutions de bras 6 DOF (*Degree Of Freedom*) [29] et 7 DOF [33], utilisant la méthode analytique ont été réalisées sur des bras anthropomorphiques. Cette méthode est utilisée pour sa précision et son efficacité [29]. Cependant, cette méthode mène à plusieurs solutions, 16 pour la cinématique inverse de la chaîne 7R, notre cas d'étude [24].

Pour utiliser ces méthodes de résolution analytique en programmation, il est possible de s'aider de bibliothèques pour le calcul symbolique comme avec **SymPy** en Python [39] ou encore **SageMath** [50] une alternative *open-source* des logiciels **Maple** [38], **Magma** [20] ou **Mathematica** [56].

2.3.2. Newton-Raphson

La méthode de Newton permet de trouver la solution d'un ensemble d'équations non linéaires de façon itérative à partir d'une valeur initiale de $\mathbf{q_0}$. Celle-ci est souvent utilisée dans les problèmes de cinématique inverse, étant très rapide et efficace comparée à d'autres méthodes comme la **méthode de la bissection** [49], qui divise un intervalle en deux pour isoler une racine, ou encore **la méthode de la sécante** [55], qui utilise une approximation linéaire entre deux points pour converger vers la solution. [35, 44] .

On réalise un développement de Taylor de $f(\mathbf{q})$ à $\mathbf{q_0}$ au premier ordre :

$$f(\mathbf{q}) \simeq f(\mathbf{q_0}) - f'(\mathbf{q})(\mathbf{q} - \mathbf{q_0}) \tag{2.3}$$

Pour trouver le zéro d'approximation de cette fonction, on calcule l'intersection de la tangente avec l'axe des abscisses :

$$0 \simeq f(\mathbf{q_0}) - f'(\mathbf{q})(\mathbf{q} - \mathbf{q_0}) \tag{2.4}$$

De cette façon, on obtient alors $\mathbf{q_1}$. On continue ensuite de manière itérative pour se rapprocher de 0 avec la suite suivante construite par récurrence :

$$\mathbf{q_{k+1}} = \mathbf{q_k} - \frac{f(\mathbf{q_k})}{f'(\mathbf{q_k})} \tag{2.5}$$

On continue de manière itérative jusqu'à ce que le critère ϵ d'arrêt soit satisfait, avec une valeur définie par l'utilisateur :

$$\left| \frac{\mathbf{q_k} - \mathbf{q_{k+1}}}{\mathbf{q_k}} \right| \leq \epsilon \tag{2.6}$$

Cependant, pour être utilisée, cette méthode nécessite de pouvoir calculer la dérivée de f. Si la dérivée est seulement estimée avec la pente entre deux points de la fonction, cela revient à utiliser la méthode de la sécante. Enfin, la méthode de Newton-Raphson peut parfois diverger, c'est pourquoi il est important de rajouter un nombre maximum d'itérations comme limite.

Des outils comme les bibliothèques **NumPy** et **SciPy** de Python [10] ou encore **CasADi** [3], un outil *open-source* pour l'optimisation non linéaire et la différenciation algorithmique, peuvent aider à la résolution d'un point de vue programmation.

2.3.3. Méthodes de la Jacobienne inverse

Il existe plusieurs méthodes de résolution numérique utilisant la jacobienne inverse [35, 5, 4]. En dérivant l'équation de la cinématique directe 2.1, définie précédemment, par rapport au temps, on obtient :

$$\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}} \tag{2.7}$$

avec $J(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} \in \mathbb{R}^{m \times n}$, la matrice jacobienne de $f(\mathbf{q})$ qui relie les vitesses articulaires $\dot{\mathbf{q}}$ aux vitesses cartésiennes $\dot{\mathbf{x}}$.

En réalisant le développement de Taylor de $f(\mathbf{q_d})$ à $\mathbf{q_d}$ au premier ordre de façon

similaire à l'Équation 2.3 , avec $\mathbf{x_d} = f(\mathbf{q_d})$ et $\mathbf{\Delta q} = (\mathbf{q_d} - \mathbf{q_0})$, on peut alors écrire :

$$J(\mathbf{q_0})\mathbf{\Delta q} = \mathbf{x_d} - f(\mathbf{q_0}) \tag{2.8}$$

Puis, en assumant que $J(\mathbf{q_0})$ est carrée et inversible, il est alors possible de résoudre $\mathbf{\Delta q}$ comme :

$$\mathbf{\Delta q} = J(\mathbf{q_0})^{-1}(\mathbf{x_d} - f(\mathbf{q_0})) \tag{2.9}$$

Si la cinématique directe est linéaire dans **q** , c'est-à-dire que les termes d'ordre supérieur dans l'équation 2.3 sont nuls, alors la nouvelle estimation $\mathbf{q_1} = \mathbf{q_0} + \mathbf{\Delta q}$ satisfait exactement $\mathbf{x_d} = f(\mathbf{q_1})$. Si la cinématique directe n'est pas linéaire dans **q**, ce qui est généralement le cas, la nouvelle estimation $\mathbf{q_1}$ devrait tout de même être plus proche de la solution que $\mathbf{q_0}$. On répète ensuite le processus, produisant une séquence $\{\mathbf{q_0}, \mathbf{q_1}, ...\}$ convergeant vers $\mathbf{q_d}$ [35].

Cependant, si la jacobienne $J(\mathbf{q_0})$ n'est pas carrée ou inversible, ce qui est généralement le cas en pratique, il est courant d'utilisé une approximation de celle-ci en remplaçant $J(\mathbf{q_0})^{-1}$ par **la pseudo-inverse de Moore–Penrose** $J^{\dagger}(\mathbf{q})$. La pseudo-inverse de Moore-Penrose a l'avantage qu'elle minimise la norme du vecteur **q**, réduisant ainsi le risque de solutions multiples. Dans le cas d'une jacobienne de rang plein ($n > m$ ou $n < m$), la pseudo-inverse est calculée ainsi :

$$J^{\dagger} = J^T(JJ^T)^{-1} \quad \text{si } n > m \,(\text{appelée inverse à droite de la Jacobienne } JJ^{\dagger} = I) \tag{2.10}$$

$$J^{\dagger} = (J^TJ)^{-1}J^T \quad \text{si } n < m \,(\text{appelée l'inverse à gauche de la Jacobienne } J^{\dagger}J = I) \tag{2.11}$$

Ainsi, l'équation 2.9 devient :

$$\mathbf{\Delta q} = J^{\dagger}(\mathbf{q_0})(\mathbf{x_d} - f(\mathbf{q_0})) \tag{2.12}$$

On utilise alors l'algorithme itératif de Newton–Raphson pour trouver $\mathbf{q_d}$:

(a) Initialisation **:** Pour $\mathbf{x_d} \in \mathbb{R}^m$ et une estimation initiale $\mathbf{q_0} \in \mathbb{R}^n$, $i = 0$.
(b) On pose $e = \mathbf{x_d} - f(\mathbf{q_i})$. Tant que $\|e\| > \epsilon$ avec ϵ petit :
— On défini $\mathbf{q_{i+1}} = \mathbf{q_i} + J^{\dagger}(\mathbf{q_i})e$.
— Incrémenter i.

Cependant, cette équation ne résout pas les problèmes de singularité éventuels. Une alternative est donc d'utiliser la méthode *Damped least squares (DLS)* ou Moindres carrés pondérés amortis, aussi connue sous le nom de **l'algorithme de**

Levenberg–Marquardt [5, 4]. Elle s'exprime alors comme :

$$\mathbf{\Delta q} = J^T(JJ^T + \lambda^2 I)^{-1} \tag{2.13}$$

avec λ, est un facteur de régularisation constant non nul dans $\mathbb{R}$ choisi de façon à stabiliser $\mathbf{\Delta q}$.
Par la suite, il est possible d'étendre la méthode *DLS* en utilisant la décomposition en valeur simple, plus connue sous le nom de *Singular Value Decomposition* (*SVD*), de cette façon :

$$J^T\left(JJ^T + \lambda^2 I\right)^{-1} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \mathbf{v}_i \mathbf{u}_i^T \tag{2.14}$$

D'un point de vue programmation, plusieurs approches sont alors possibles.

Pinocchio est une bibliothèque *open source* efficace pour calculer la dynamique de corps articulés, comme un bras robotique [12]. Il est alors possible d'utiliser un **algorithme de cinématique inverse en boucle fermée** (*closed-loop inverse kinematics* - CLIK), une approximation numérique permettant de résoudre des problèmes de cinématique inverse, principalement la redondance au niveau de la vitesse. Cette dernière est détaillée dans les travaux suivants [14, 54]. Pour améliorer la stabilité numérique et éviter les problèmes de singularité, une pseudo-inverse régularisée (ou *damped pseudo-inverse*) est utilisée [13]. D'autres outils cités précédemment comme les bibliothèques Python **NumPy** et **SciPy** ou encore **CasADi** peuvent aider à l'écriture d'un programme de résolution.

3. Implémentation de contrainte et leur résolution

Cette partie montre les deux grandes méthodes d'implémentation de contraintes dans le calcul de la cinématique inverse. Que ce soit par l'utilisation du noyau de la jacobienne ou en posant le problème comme un problème d'optimisation. Un ensemble de méthodes d'optimisation est ensuite listé ainsi que des bibliothèques informatiques.

3.1. Importance de l'implémentation des contraintes

Les robots, comme les humains, n'ont pas des capacités illimitées. Ils vont se heurter à des problèmes comme des poses inatteignables par exemple (Chapitre 1). Ainsi, des contraintes intrinsèques au robot comme

$$\mathbf{q_{min}} \leq \mathbf{q} \leq \mathbf{q_{max}} \tag{3.1}$$

$$\dot{\mathbf{q}}_{\mathbf{min}} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\mathbf{max}} \tag{3.2}$$

doivent être absolument respectées.
Mais d'autres contraintes peuvent aussi être appliquées au robot dû à sa morphologie, à l'environnement de travail, aux tâches qu'il doit réaliser, etc...[27, 26, 14] Ces contraintes sont importantes à implémenter pour la sécurité du robot, mais aussi pour celle des opérateurs ou encore de son environnement proche.
La résolution du problème doit alors trouver la solution aux tâches du robot comme celle du déplacement $x = f^{-1}(q)$ tout en respectant les contraintes qui lui sont imposées.

3.2. Méthodes d'implémentation des contraintes

Il existe alors deux grandes familles permettant d'appliquer des contraintes : *L'explicit inversion methods* (méthodes d'inversion explicites) et la *Constrained optimization*

methods (méthodes d'optimisation contraintes) [25].

3.2.1. *Explicit inversion methods*

L'explicit inversion methods se base sur le calcul explicite de la jacobienne (2.7). Dans le cas où le robot est redondant par rapport à la tâche proposée, il est possible d'utiliser ces degrés supplémentaires pour appliquer une tâche secondaire que le robot devra résoudre, sans que cela n'interfère avec la première tâche, sous la forme de l'Équation (3.3) [34] :

$$\dot{\mathbf{q}}^{\mathbf{c}} = J^{\dagger}(\mathbf{q})\dot{\mathbf{x}}_{\mathbf{d}} + \left(I_n - J^{\dagger}(\mathbf{q})J(\mathbf{q})\right)\dot{\mathbf{q}}_{\mathbf{0}} \tag{3.3}$$

où $\left(I - J^{\dagger}(\mathbf{q})J(\mathbf{q})\right)$ est un projecteur sur le noyau de la pseudo-inverse jacobienne avec $I_n \in \mathbb{R}^{n\times n}$ la matrice identité de taille n. Les degrés de liberté redondants du robot deviennent ainsi des degrés de liberté disponibles que celui-ci pourra utiliser afin de satisfaire les contraintes.
Il est possible de généraliser cette méthode sur plusieurs tâches et sous-tâches[47, 27]. Le problème de résolution peut être posé comme étant l'Équation (3.4) :

$$\begin{pmatrix} \dot{x_{d,1}} \\ \dot{x_{d,2}} \\ \dots \\ \dot{x_{d,n}} \end{pmatrix} = \begin{pmatrix} J_1 \\ J_2 \\ \dots \\ J_n \end{pmatrix} \dot{\mathbf{q}} \tag{3.4}$$

La résolution de ce problème est réalisée par une méthode itérative. La solution de la prochaine tâche correspond à la solution de la tâche précédente ajustée par la solution de la prochaine tâche projetée sur le noyau de la jacobienne de la tâche précédente (3.5)

$$\dot{\mathbf{q}}^{\mathbf{c}}_{\mathbf{i}} = \dot{\mathbf{q}}^{\mathbf{c}}_{\mathbf{i-1}} + (J_i(I_n - J^{\dagger}_{i-1}J_{i-1}))^{\dagger}(\dot{\mathbf{x}}_{\mathbf{d,i}} - J_i\dot{\mathbf{q}}_{\mathbf{i-1}}) \tag{3.5}$$

avec pour initialisation $\dot{\mathbf{q}}^{\mathbf{c}}_{\mathbf{1}} = J^{\dagger}_1\dot{\mathbf{x}}_{\mathbf{1}}$
Cette méthode a l'avantage de facilement hiérarchiser les tâches dans un système où les tâches inférieures ne peuvent pas interférer avec les tâches supérieures. Cela permet ainsi de créer des contraintes et des tâches qui seront hiérarchisées selon leur importance.

Cependant, elle a pour désavantage de toujours prioriser la tâche supérieure. Dans le

cas où il y a plus de contraintes que de degrés de liberté, le noyau de la jacobienne sera nul et certaines consignes de contraintes ne seront pas projetées dans la jacobienne de la tâche principale et ces consignes ne seront donc pas appliquées [25]. L'utilisation d'une telle méthode mettrait en danger l'intégrité du robot.

Afin d'assurer le respect des contraintes, une alternative à cette méthode est de passer les contraintes en tant que tâche principale et la consigne de mouvement en tant que tâche secondaire [36]. Il se peut alors que le mouvement ne soit pas réalisé s'il existe trop de contraintes. En effet, les degrés de liberté du robot seraient utilisés en priorité au respect des contraintes, laissant un nombre insuffisant de degrés de liberté disponibles pour accomplir la tâche de mouvement [25].

Il est possible de réaliser des fonctions d'activation de contrainte afin de réduire le nombre de contraintes devant être respectées par le robot [25, 26, 27].
Les méthodes basées sur ce principe consistent en un premier calcul, sans prise en compte de contraintes, du mouvement. Dans le cas où une butée est rencontrée, un nouveau calcul de cinématique inverse est réalisé en prenant l'articulation en butée comme une constante du calcul. Ce processus se répète jusqu'à trouver une solution ne rencontrant aucune butée [26, 27]. Cela permet ainsi de se rapprocher le plus possible de la pose désirée, sans pour autant demander au robot une consigne articulaire excédant ses capacités. Ce type de méthode ne peut cependant prendre en compte uniquement les contraintes articulaires.

3.2.2. *Constrained Convex Optimization Methods*

La deuxième méthode se prénomme la *Constrained Convex Optimization Methods* [25] Le problème peut aussi être exprimé comme une résolution d'un problème d'optimisation pour minimiser la norme de $J(\mathbf{q})\dot{\mathbf{q}} - \dot{\mathbf{x}}$ tout en satisfaisant les contraintes exprimées par les équations (3.1), (3.2) ou toutes autres contraintes que l'on souhaiterait appliquer au robot. On peut représenter l'équation générale sous la forme (3.6) :

$$\begin{aligned} minimiser \quad & f(\mathbf{y}) \\ avec \quad & h(\mathbf{y}) \leq 0 \\ & g(\mathbf{y}) = 0 \end{aligned} \tag{3.6}$$

où $f(\mathbf{y}) \in \mathbb{R}^n \rightarrow \mathbb{R}$ est une fonction objectif qui doit être minimisée selon les variables d'optimisation $\mathbf{y} = (y_1, y_2, ..., y_n) \in \mathbb{R}^n$. L'optimisation peut être influencée par des

contraintes d'inégalité ou d'égalité exprimées comme une fonction des variables d'optimisation. Les contraintes d'inégalité sont exprimées dans $h(\mathbf{y}) \in \mathbb{R}^n \rightarrow \mathbb{R}^{n_{ci}}$ où n_{ci} est le nombre de contraintes d'inégalité et de même pour les contraintes d'égalité avec $g(\mathbf{y}) \in \mathbb{R}^n \rightarrow \mathbb{R}^{n_{ce}}$ où n_{ce} est le nombre de contraintes d'égalité.

Une solution optimale au problème d'optimisation possède la norme de la fonction objectif minimale [27].
Il est possible d'ajuster cette solution avec un polynôme de Lagrange pour centrer la norme autour de points spécifiques [26].

Dans la forme générale, la fonction objectif peut être écrite comme $f(\mathbf{y}) = ||E\mathbf{y} - f||_2^2$ avec $||.||_2$ la norme Euclidienne. Dans le cas spécifique de notre problème (2.7), on a $E = J$ et $f = \dot{\mathbf{x}}$. La fonction est convexe et peut être écrite dans sa forme quadratique :

$$f(\mathbf{y}) = ||E\mathbf{y} - f||_2^2 = (E\mathbf{y} - f)^T(E\mathbf{y} - f) = \mathbf{y}^T E^T E\mathbf{y} - 2f^T E\mathbf{y} + f^T f \tag{3.7}$$

Le problème d'optimisation est donc un problème quadratique à plusieurs variables. Il est donc possible de l'écrire sous la forme quadratique :

$$\mathbf{y}^T E^T E\mathbf{y} - 2f^T E\mathbf{y} + f^T f = \frac{1}{2}\mathbf{y}^T H\mathbf{y} + \mathbf{g}^T\mathbf{y} + r \tag{3.8}$$

avec $H \in \mathbb{R}^{n \times n}$ et $\mathbf{g} \in \mathbb{R}^n$ respectivement la matrice Hessienne constituée des dérivés partielles de second degrés, le vecteur gradient de la fonction objectif et r un scalaire.

L'optimisation linéaire quadratique est une branche de l'optimisation pour laquelle la fonction objectif est quadratique et la fonction contrainte est affine [25]. Ces méthodes cherchent la solution optimale $\mathbf{y}^{opt}$, au problème suivant :

$$\begin{aligned} \mathbf{y}^{opt} = \quad & \arg\min_{y} \quad \frac{1}{2}\mathbf{y}^T H\mathbf{y} + \mathbf{g}^T\mathbf{y} + r \\ & s.t. \qquad\quad \mathbf{l}_b \leq A\mathbf{y} \leq \mathbf{u}_b \\ & \qquad\qquad\quad C\mathbf{y} = \mathbf{d} \end{aligned} \tag{3.9}$$

Les fonctions contraintes sont divisées en égalités et inégalités avec A et C correspondant aux matrices correspondantes. $\mathbf{l}_b$ et $\mathbf{u}_b$ sont respectivement les vecteurs des limites inférieures et supérieures des inégalités et $\mathbf{d}$ un vecteur associé aux contraintes d'égalité. Si la fonction objectif est convexe c'est-à-dire que

$$f(\alpha\mathbf{x} + \beta\mathbf{y}) \leq \alpha f(\mathbf{x}) + \beta f(\mathbf{y}) \tag{3.10}$$

pour tout $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ et pour tout $\alpha, \beta \in \mathbb{R}^+$ avec $\alpha + \beta = 1$, un minimum local de la fonction est aussi son minimum global. Cette propriété simplifie le problème de résolution et réduit le temps de calcul.
La résolution de problème d'optimisation quadratique nécessite l'utilisation d'une méthode d'optimisation quadratique aussi appelée méthode *QP* pour *Quadratic programming*.

3.3. Méthodes d'optimisation et solveurs

Les méthodes d'optimisation suivantes, présentées sur la Figure 3.1, sont largement utilisées dans les problèmes d'optimisation convexe et non linéaire. D'abord, la **Programmation Linéaire (*Linear programming - LP*)** [8, 17] consiste à minimiser une fonction linéaire sous des contraintes linéaires, souvent exprimées comme $A\mathbf{x} \leq \mathbf{b}$ avec $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^n$ et A une matrice.
Puis, la **Optimisation quadratique (*Quadratic programming - QP*)** [25, 52, 9], implique une fonction objectif quadratique de la forme $\frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$, avec des contraintes linéaires similaires à celles de la LP, mais avec l'ajout de termes quadratiques dans l'objectif. Il existe par ailleurs certaines approches QP utilisant un réseau de neurones [52, 22].
La **Programmation Conique Second-Ordre (*Second Order Cone Programming - SOCP*)** [28, 37] est un cas particulier où les contraintes sont des cônes de second ordre, définis par des inégalités telles que $\|A_i\mathbf{x} + \mathbf{b}_i\|_2 \leq \mathbf{c}_i^T\mathbf{x} + d_i$.
Pour les problèmes impliquant des matrices, la **Programmation Semi-Définie (*Semidefinite programming - SDP*)** [28, 53] est utilisée, où l'objectif est de minimiser $\text{tr}(C\mathbf{X})$, soumis à des contraintes sur des matrices semi-définies positives.
Enfin, la **Programmation Complémentaire (*Conic optimization - CP*)** [23] est une approche où des variables $\mathbf{x}_i$ et $\mathbf{y}_i$ sont contraintes par des relations de complémentarité $\mathbf{x}_i\mathbf{y}_i = 0$, souvent utilisée pour modéliser des phénomènes de contact ou des systèmes à contraintes discontinues.
A noter que certaines contraintes ne s'écrivent pas comme des cônes de second degré, rendant impossible l'implémentation de méthodes plus spécifique comme SOCP, SDP ou CP.
Plusieurs résolutions de la cinématique inverse différentielle par des tâches pondérées existent en python, notamment la bibliothèque Python **Pink** basée sur **Pinocchio** [11], mais aussi **CVXOPT** [2] ou encore **Quadprog** utilisant l'algorithme d'écrit dans [19]. Il existe ensuite **Pyomo** [21], un *package open source* qui peut être utilisé pour définir des problèmes symboliques généraux, créer des instances de problèmes spécifiques

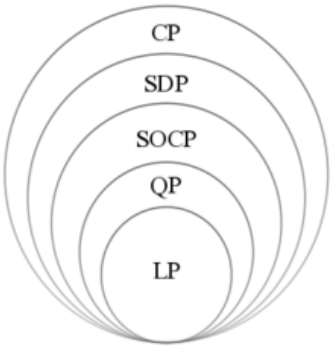

FIGURE 3.1. – Schéma récapitulatif des méthodes d'optimisation

et résoudre ces instances à l'aide de solveurs commerciaux et open source. Puis, **IPOPT** (*Interior Point OPTimizer*) [57], un *open source* pour l'optimisation non linéaire à grande échelle, le solveur **OSQP** (*Operator Splitting Quadratic Program*) [48], un *package* d'optimisation numérique, ou encore **CVXPY** [1, 15], un langage de modélisation intégré à Python pour les problèmes d'optimisation convexe.

Conclusion & perspectives

Cet état de l'art met en lumière le problème récurrent de la cinématique inverse dans des configurations redondantes avec contraintes. Il explore différentes approches, qu'elles soient analytiques ou numériques, tout en détaillant les solveurs disponibles pour leur mise en œuvre dans un code informatique. Il présente notamment l'intégration des contraintes dans les méthodes de résolution itérative.
A la vue de ces recherches et des problèmes rencontrés actuellement par la start-up *Pollen Robotics*, nous élaborons un plan structuré visant à résoudre ces défis et à apporter des solutions concrètes. L'objectif étant, à terme, de tester diverses méthodes de résolutions numériques permettant l'ajout de contraintes.

1. Une première étape consiste à développer une fonction permettant de calculer la cinématique inverse à l'aide de la jacobienne du robot, sans prise en compte des contraintes. Elle vient remplacer la fonction actuellement utilisant une méthode analytique et propose plutôt une approche numérique. Cette fonction sera fortement inspirée des exemples disponibles dans la bibliothèque Pinocchio [13, 12], permettant notamment le calcul de la jacobienne des angles articulaires du robot encore indisponible sur le programme de la start-up. L'utilisation de l'algorithme de Levenberg–Marquardt, avec l'utilisation d'un facteur de régularisation, permet de gérer les problèmes de division par 0. Cela a aussi l'avantage d'utiliser la jacobienne qui sera utilisé dans les méthodes suivantes. Une fois cette fonction implémentée, elle devra être testée afin de valider son bon fonctionnement.
2. La seconde étape consiste à modifier cette dernière implémentation numérique pour y inclure une résolution par la méthode de optimisation quadratique (QP), toujours sans gestion de contraintes. Cela permettra de diminuer le nombre de pose équivalente en définissant un critère de sélection. Il s'agit aussi d'une étape intermédiaire afin de vérifier si la méthode QP fonctionne dans son utilisation basique. Pour cela, des outils issus de la bibliothèque Pink, par exemple, seront exploités [11]. Des tests seront également effectués pour garantir l'efficacité et la robustesse de cette version enrichie.
3. Enfin, des fonctions spécifiques pour la gestion des contraintes sont a ajoutées afin de gérer les problèmes de limites articulaires notamment. Cela nécessitera

d'identifier et de hiérarchiser les problématiques actuelles, avant d'intégrer ces contraintes dans les calculs. Des ajustements supplémentaires seront réalisés en fonction des besoins.

4. Finalement, une évaluation critique des solutions obtenues sera menée, ouvrant ainsi la voie à de futures améliorations et développements.

A. Une annexe

Le Tableau A.1 ci-dessous présente les différentes abréviations utilisées dans ce rapport ainsi qu'une description de ces termes.

Abréviation	Description	Unités
$\mathbf{x}$	Vecteur des positions et orientations de l'effecteur terminal par rapport au repère de base.	$[mm, rad]$
$\mathbf{x}_0$	Pose initiale, exprimée en position et orientation.	$[mm, rad]$
$\mathbf{x}_d$	Pose désirée à atteindre, exprimée en position et orientation.	$[mm, rad]$
$\mathbf{q}$	Vecteur des angles articulaires. Peut représenter des articulations rotoïdes ou prismatiques.	rad (rotoïde), mm (prismatique)
$\mathbf{q}_0$	Configuration articulaire initiale.	rad, mm
$\mathbf{q}_d$	Configuration articulaire désirée, correspondant à une pose cible $\mathbf{x}_d$.	rad, mm
$\mathbf{q}_x$	Coordonnée spécifique du vecteur $\mathbf{q}$. Désigne une composante particulière d'angle ou de déplacement.	/
$\dot{\mathbf{x}}$	Vitesses cartésiennes (translation et rotation).	$[mm/s, rad/s]$
$\dot{\mathbf{q}}$	Vitesses articulaires (dérivées par rapport au temps du vecteur $\mathbf{q}$).	rad/s, mm/s
J	Matrice Jacobienne, décrivant les relations entre les vitesses articulaires et cartésiennes.	/
$J(\mathbf{q})$	Jacobienne évaluée pour une configuration articulaire donnée $\mathbf{q}$.	/
$J^T(\mathbf{q})$	Transposée de la Jacobienne. Utilisée dans certaines méthodes d'optimisation.	/
$J^+(\mathbf{q})$	Pseudo-inverse de la Jacobienne (Moore-Penrose). Calculée pour traiter les redondances ou singularités.	/
σ, u, v	Valeurs et vecteurs propres (contextes SVD). σ : valeurs singulières, u et v : vecteurs propres associés.	/
ε	Erreur, définie comme l'écart entre une valeur cible et une valeur mesurée.	/
Δ	Écart ou variation, souvent utilisée pour indiquer une différence, comme $\Delta\mathbf{q} = \mathbf{q}_d - \mathbf{q}$.	/
E	Fonction d'erreur, exprimant l'écart entre la pose actuelle et la pose cible.	/
$\mathbf{q}^c$	Contraintes sur la configuration articulaire, utilisées pour garantir des limites physiques ou des conditions de sécurité.	rad, mm

TABLE A.1. – Abréviations et descriptions des termes utilisés.

Bibliographie

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. 5(1) :42–60.

[2] Martin S. Andersen, Joachim Dahl, and Lieven Vandenberghe. Solving a quadratic program — CVXOPT.

[3] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. CasADi : a software framework for nonlinear optimization and optimal control. 11(1) :1–36.

[4] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. Inverse kinematics techniques in computer graphics : A survey. 37(6) :35–58.

[5] Andreas Aristidou and Joan Lasenby. Inverse kinematics : a review of existing techniques and introduction of a new fast iterative solver.

[6] Laura Aymerich-Franch, Damien Petit, Gowrishankar Ganesh, and Abderrahmane Kheddar. Non-human looking robot arms induce illusion of embodiment. 9(4) :479–490.

[7] Antal K. Bejczy. Sensors, controls, and man-machine interface for advanced teleoperation. 208(4450) :1327–1335.

[8] Michael J. Best and Klaus Ritter. *Linear programming : active set analysis and computer programs*. Prentice-Hall.

[9] Immanuel M. Bomze. On standard quadratic optimization problems. 13(4) :369–387.

[10] Eli Bressert. *SciPy and NumPy : An Overview for Developers*. "O'Reilly Media, Inc.".

[11] Stéphane Caron, Yann De Mont-Marin, Rohan Budhiraja, Seung Hyeon Bang, Ivan Domrachev, and Simeon Nedelchev. Pink : Python inverse kinematics based on pinocchio. original-date : 2022-02-06T10 :05 :13Z.

[12] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives.

[13] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. pinocchio : Inverse kinematics (clik).

[14] Pasquale Chiacchio, Stefano Chiaverini, Lorenzo Sciavicco, and Bruno Siciliano. Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. 10(4) :410–425.

[15] Steven Diamond and Stephen Boyd. CVXPY : A python-embedded modeling language for convex optimization.

[16] A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 1, pages 298–303 vol.1.

[17] Thomas S Fergusin and M Casquilho. Linear programming, a concise introduction.

[18] Rebecca Fribourg, Evan Blanpied, Ludovic Hoyet, Anatole Lécuyer, and Ferran Argelaguet. Does virtual threat harm VR experience ? : Impact of threat occurrence and repeatability on virtual embodiment and threat response. 100 :125.

[19] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. 27(1) :1–33.

[20] Computational Algebra Group, Sydney School of Mathematics {and} Statistics, and University of Sydney. Magma computational algebra system.

[21] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo — Optimization Modeling in Python*, volume 67 of *Springer Optimization and Its Applications*. Springer International Publishing.

[22] Ahmed A. Hassan, Mohamed El-Habrouk, and Samir Deghedie. Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks : A review. 38(8) :1495–1512.

[23] Didier Henrion and Jérôme Malick. Projection methods in conic optimization.

[24] M. Hiller. Multiloop kinematic chains. In J. Angeles and A. Kecskeméthy, editors, *Kinematics and Dynamics of Multi-Body Systems*, pages 75–165. Springer.

[25] Lucas Joseph. An energetic approach to safety in robotic manipulation.

[26] Oussama Kanoun. Real-time prioritized kinematic control under inequality constraints for redundant manipulators. In *Robotics : Science and Systems VII*. Robotics : Science and Systems Foundation.

[27] Oussama Kanoun, Florent Lamiraux, and Pierre-Brice Wieber. Kinematic control of redundant manipulators : Generalizing the task-priority framework to inequality task. 27(4) :785–792.

[28] Sunyoung Kim and Masakazu Kojima. Exact solutions of some nonconvex quadratic optimization problems via SDP and SOCP relaxations.

[29] Nikos Kofinas, Emmanouil Orfanoudakis, and Michail G. Lagoudakis. Complete analytical inverse kinematics for NAO. In *2013 13th International Conference on Autonomous Robot Systems*, pages 1–6. IEEE.

[30] Andrey Krekhov, Sebastian Cmentowski, and Jens Krüger. The illusion of animal body ownership and its potential for virtual reality games. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. ISSN : 2325-4289.

[31] Serdar Kucuk and Zafer Bingul. Robot kinematics : Forward and inverse kinematics. In Sam Cubero, editor, *Industrial Robotics : Theory, Modelling and Control*. Pro Literatur Verlag, Germany / ARS, Austria.

[32] S Lichiardopol. A survey on teleoperation. page 34.

[33] Weihui Liu, Diansheng Chen, and Jochen Steil. Analytical inverse kinematics solver for anthropomorphic 7-DOF redundant manipulators with human-like configuration constraints. 86(1) :63–79.

[34] Alain Liégois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. 7(12) :868–871. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics.

[35] Kevin M. Lynch and Frank C. Park. *Modern Robotics : Mechanics, Planning, and Control*. Cambridge University Press, 1 edition.

[36] Anthony Maciejewski and Charles Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. 4.

[37] Athanasios Makrodimopoulos and Chris Martin. Limit analysis using large-scale SOCP optimization.

[38] Waterloo Maple. Maple - l'outil essentiel pour les mathématiques et la modélisation - maplesoft.

[39] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller,

Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. SymPy : symbolic computing in python. 3 :e103.

[40] N. Miller, O.C. Jenkins, M. Kallmann, and M.J. Mataric. Motion capture from inertial sensing for untethered humanoid teleoperation. In *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, volume 2, pages 547–565. IEEE.

[41] Pollen Robotics. Robot kinematics explained.

[42] Pollen Robotics. Pollen robotics.

[43] Pollen Robotics. Reachy 2023 documentation.

[44] Akram Saba and ul Ann Qurrat. Newton raphson method. 6 :1748–1752.

[45] Philip N Sabes. Linear algebraic equations, SVD, and the pseudo-inverse.

[46] Pollen Robotics SAS. Reachy, developed by pollen robotics, is an open-source humanoid robot.

[47] Bruno Siciliano and J.-J.E. Slotine. *A general framework for managing multiple tasks in highly redundant robotic systems. In Fifth international conference on advanced robotics (Vol. 2*. Pages : 1216 vol.2.

[48] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP : an operator splitting solver for quadratic programs. 12(4) :637–672.

[49] Pragati Thapliyal and Komal Tomar. Chitra solanki DIT university dehradun, india. 3(8).

[50] The SageMath Developers. SageMath.

[51] Alexander Toet, Irene A. Kuling, Bouke N. Krom, and Jan B. F. van Erp. Toward enhanced teleoperation through embodiment. 7.

[52] Hamid Toshani and Mohammad Farrokhi. Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming : A lyapunov-based approach. 62(6) :766–781.

[53] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. 38(1) :49–95. Publisher : Society for Industrial and Applied Mathematics.

[54] Jingguo Wang, Yangmin Li, and Xinhua Zhao. Inverse kinematics and control of a 7-DOF redundant manipulator based on the closed-loop algorithm. 7(4) :37. Publisher : SAGE Publications.

[55] Philip Wolfe. The secant method for simultaneous nonlinear equations. 2(12) :12–13.

[56] Stephen Wolfram. Wolfram mathematica : le calcul technique et moderne.

[57] Andreas Wächter and Carl Laird. coin-or/ipopt. original-date : 2019-01-28T12 :24 :22Z.