Sorbonne Université

Master 2 Automatique, Robotique

Parcours Systèmes Avancés et Robotique

RAPPORT DE STAGE

Développement de protocoles de Contrôle en compliance robustes d'un Bras-robot collaboratif Kuka lbr iiwa 14 pour essais Biomécaniques.

Rédigé par : Tir Abd Elhafid

(Mars à août 2023)

Tuteur de stage : M. Bertrand FRECHEDE

Enseignant référent : M. Faiz BEN AMAR

Entreprise d'accueil : laboratoire de biomécanique et mécanique des chocs (LBMC)

Remerciement :

Je tiens tout d'abord à exprimer ma profonde gratitude envers mon département de formation du Master Automatique robotique pour m'avoir offert l'opportunité d'acquérir des connaissances et des compétences qui ont été essentielles pour la réussite de mon stage. Les enseignements et les expériences acquises au sein du département ont jeté les bases solides sur lesquelles j'ai pu m'appuyer pour aborder avec confiance les défis techniques et les tâches de recherche qui m'ont été confiés.

Je tiens également à adresser mes sincères remerciements à mon tuteur de stage, M. Bertrand FRECHEDE, pour son soutien, son encadrement et son expertise tout au long de mon stage. Sa vision éclairée, ses conseils judicieux et sa passion pour la recherche ont été une source d'inspiration constante. Sa disponibilité pour répondre à mes questions et sa volonté de partager ses connaissances ont grandement contribué à ma croissance en tant que jeune professionnel dans le domaine de la biomécanique et de la robotique.

La combinaison de l'environnement académique et du mentorat bienveillant de M. FRECHEDE a créé un cadre exceptionnel qui m'a permis d'atteindre les objectifs de mon stage avec succès. Je suis reconnaissant pour cette expérience enrichissante et le soutien inestimable que j'ai reçu tout au long de mon parcours.

Résumé :

Mon stage s'est déroulé au sein du Laboratoire de Biomécanique et Mécanique des Chocs (LBMC), une unité de recherche conjointe entre l'IFSTTAR et l'Université Claude Bernard Lyon 1. Pendant 6 mois, j'ai contribué à l'axe de recherche "Le corps réparé : l'implant dans son environnement". L'objectif de mon stage était d'améliorer le contrôle en compliance d'un bras-robot collaboratif Kuka lbr iiwa 14 R820, En utilisant un capteur de force 6 axes monté sur le bras robot, j'ai développé des outils et des méthodes pour asservir la cinématique du robot en boucle fermée.

Mes tâches englobaient la continuation d'une étude bibliographique amorcée par des précédents stagiaires afin d'identifier les meilleures pratiques. En parallèle, j'ai travaillé sur le développement d'algorithmes ainsi que sur l'implémentation de scripts de contrôle. Les résultats de mon travail ont contribué de manière significative à l'avancement de l'étude sur le contrôle du robot. Plus spécifiquement, mes efforts ont permis d'obtenir des résultats positifs dans le domaine de l'asservissement en compliance, ouvrant ainsi la voie à des améliorations ultérieures. Une discussion approfondie à ce sujet est présentée dans la suite du rapport.

En tant qu'étudiant en dernière année de Master, avec une spécialisation en automatique-robotique, j'ai pu utiliser mes compétences en programmation, notamment en utilisant Matlab/Simulink et en réalisant des simulations de robots. Mon bagage de connaissances en robots manipulateurs et en mécanique m'a également été bénéfique pour mener à bien ces missions.

Ce stage m'a offert une opportunité unique de travailler au cœur de la recherche en biomécanique et en robotique, et j'ai pu contribuer de manière significative à l'amélioration des capacités du bras-robot Kuka lbr iiwa 14 en termes de contrôle en compliance.

Table des matières

Table des figures :

1. Introduction

Les robots manipulateurs en série jouent un rôle central dans la révolution technologique actuelle, se déployant de plus en plus dans divers environnements et applications. Leur influence s'étend à des secteurs variés tels que l'industrie manufacturière, l'automobile, la santé et l'assistance, l'aérospatiale, etc. Dans cette optique, il est impératif de mener des études approfondies pour améliorer ces mécanismes et les intégrer de manière optimale dans tous les aspects de notre société.

L'interaction entre la robotique et la biomécanique est particulièrement significative. Un exemple concret illustrant cette convergence est son application dans le domaine chirurgical. Dans ce contexte, les robots permettent aux chirurgiens de réaliser des interventions directes sur les patients en collaboration avec des robots ou par le biais de la télé-opération [Horgan (2001)] , ce qui leur permet notamment de réduire le tremblement et d'augmenter la précision. Le concept de compliance robotique occupe une place centrale dans cette dynamique. Son objectif est de permettre aux robots de s'adapter naturellement à leur environnement et d'interagir de manière plus fluide avec les êtres humains. En favorisant une réaction flexible aux forces extérieures et aux contraintes, la compliance robotique contribue de manière significative à améliorer la sécurité et l'efficacité des interactions entre humains et robots. Cette approche s'enracine dans les besoins de la biomécanique et stimule le développement de solutions robotiques adaptées aux défis de la santé et de la réhabilitation.

1.1 Contexte et Besoin

Le projet de stage se déroule au Laboratoire de Biomécanique et Mécanique des Chocs (LBMC), un groupe de recherche collaboratif entre l'IFSTTAR (Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux) et l'Université Claude Bernard Lyon 1, qui est reconnue pour son expertise en transports, santé et biomécanique. Le LBMC se consacre à des recherches innovantes pour rendre les transports plus sûrs et plus confortables, et contribue également aux avancées médicales futures. L'un des domaines d'étude du LBMC, nommé "le corps réparé : l'implant dans son environnement", rassemble des cliniciens et des chercheurs en biomécanique. Leur objectif est d'appuyer les besoins scientifiques de l'industrie des implants en examinant comment ces dispositifs fonctionnent dans le corps humain, en se concentrant sur des sujets tels que leur intégration, leur ancrage et leur performance fonctionnelle.
Dans ce contexte, émerge la nécessité de créer la possibilité d'appliquer des sollicitations mécaniques complexes et « physiologiques » sur des segments biologiques in-vitro, visant à reproduire les contraintes similaires à celles que le corps humain expérimente. Ces sollicitations mécaniques sont appliquées lors d'essais visant par exemple à évaluer la performance d'implants ou de prothèses, de manière comparative à des segments sains ou lésés. Historiquement, diverses approches ont été tentées pour générer de telles sollicitations mécaniques. Par exemple, l'utilisation de systèmes à câble et poulie Figure 1, ainsi que des dispositifs de traction-compression Figure 2, a été explorée. Au sein du laboratoire, une machine de traction-compression existe également Figure 3, mais son montage complexe nécessite une configuration spécifique pour chaque protocole d'essai et pour chaque segment/articulation. Ces différentes méthodes ont leurs avantages et inconvénients respectifs. L'application principale visée par les développements de mon stage était celle de protocoles sur segments rachidiens constitués de plusieurs vertèbres, le comportement intervertébral étant décrit comme passivement instable (la stabilité dynamique étant assurée activement par les muscles) et assez fortement non-linéaire (i.e. pour ce qui concerne les comportements moment=f(angle) généralement décrits dans les plans anatomiques classiques).

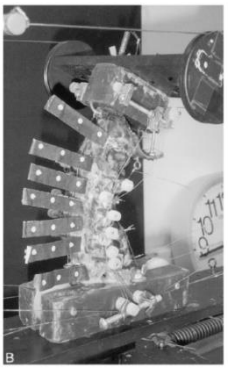


Figure 1 : Panjabi 1995

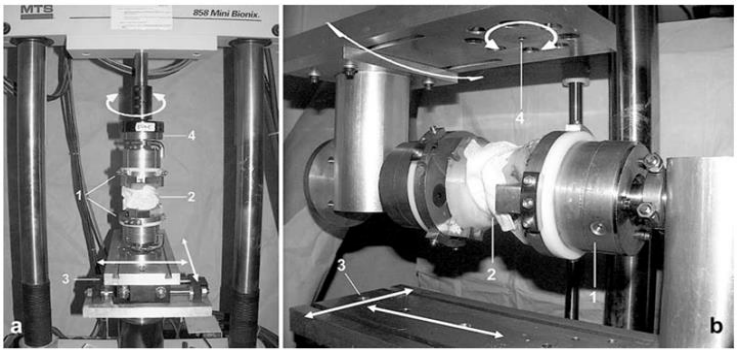


Figure 2 : Sen 2005

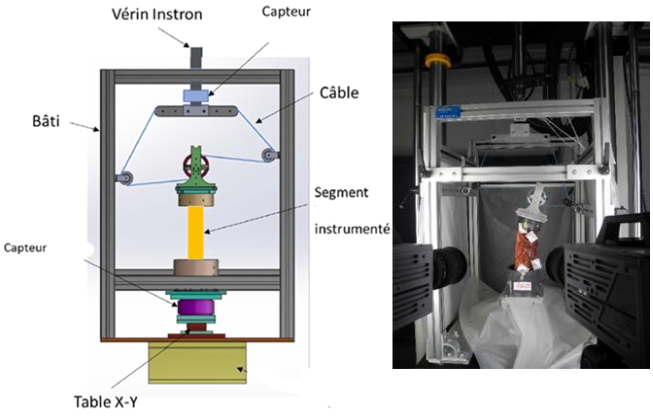


Figure 3 : Machine traction compression LBMC

Le principal défi de ce projet résidait dans la nécessité de prendre en compte cette instabilité et les grandes amplitudes de rotation propres au cou qui en résultent Figure 4. Les articulations intervertébrales de la colonne cervicale peuvent être mécaniquement décrites par des liaisons cinématiques de type rotule Figure 5, deux degrés de liberté particuliers (la rotation axiale et l'inflexion latérale – lateral bending dans la figure) étant néanmoins couplés cinématiquement. Ainsi, lorsque l'humain incline latéralement sa tête pour rapprocher son oreille gauche de son l'épaule gauche, cela entraîne également une rotation axiale de la tête par rapport au tronc. Par conséquent, reproduire fidèlement ces mouvements complexes tout en respectant ces liaisons cinématiques, sans endommager les structures osseuses principalement à l'origines de ces couplages (facettes articulaires postérieures constituées de petites surfaces de contact ostéo-cartilagineuses présentes sur chaque vertèbre) représente un défi important lors de la création de sollicitations mécaniques adaptées à la colonne cervicale, i.e. compatibles avec l'existence de ces liaisons.

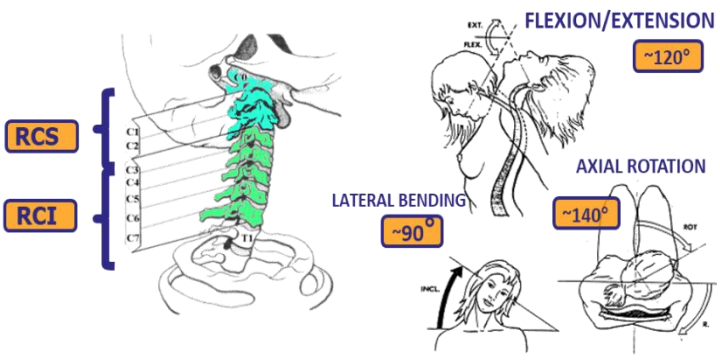


Figure 4 : Watier 97 - Source Kapandji 86

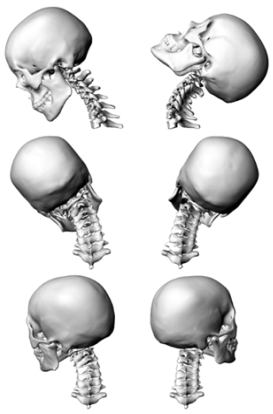


Figure 5 : Zhou 2020

1.2 Existant et Problématique

Les problématiques abordées dans ce contexte se répartissent comme suit :

1.2.1 Imposer une sollicitation en couple pur

Dans ce contexte, l'équipe de recherche du laboratoire a acquis en 2019 un robot manipulateur de type Kuka LBR iiwa 14, pour lequel elle a élaboré un protocole d'essai visant à appliquer des sollicitations de manière physiologique. Ce protocole concerne la sollicitation d'une pièce en contrôlant les forces appliquées suivant les six degrés de liberté. Les forces sont mesurées à partir des capteurs de couple intégrés dans chaque articulation du robot et sont ensuite ramenées à l'effecteur, où elles sont évaluées pour obtenir les efforts extérieurs à la bride. Un exemple de test couramment exécuté est présenté dans la Figure 6 ci-dessous : il s'agit de solliciter un segment vertébral en inflexion latérale en appliquant un couple pur (couple de forces) autour d'un axe spécifique relié au robot (axe initialement perpendiculaire à l'image dans le cas de la Figure 6.

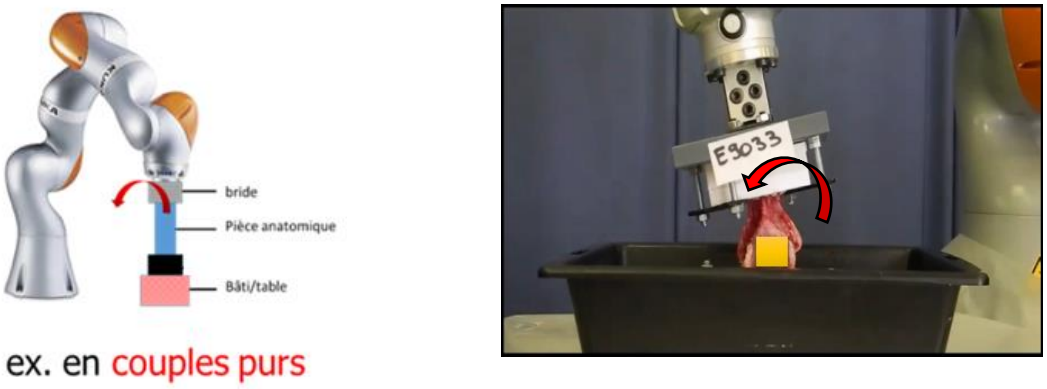


Figure 6 : Illustration du principe du protocole, source : « document interne Laboratoire de Biomécanique et Mécanique des Chocs »

1.2.2 Contrôler la précision

L'implémentation était effectuée en Java via l'interface utilisateur Sunrise Workbench fournie par Kuka mais, en raison de difficultés d'utilisation des méthodes de contrôle en compliance disponibles, une solution fonctionnelle mais non idéale avait été implémentée avant mon début de stage. Concrètement, une rotation « utile » autour d'un CIR (Centre Instantané de Rotation) préalablement identifié était imposée par petits incréments, des petites translations et rotations étaient ensuite appliquées en boucle sur les cinq degrés de liberté restants afin d'éliminer toutes les composantes du torseur d'efforts généré, à l'exception donc au final du moment autour de l'axe utile. L'objectif principal de cette sollicitation était de reproduire de manière optimale les sollicitations mécaniques à la fois représentatives des conditions physiologiques et de condition d'essais similaires de la littérature en vue de possibles comparaisons, en présence ou non d'un implant ou d'une prothèse. Cette approche, bien que fonctionnelle, ne permettait pas une correction précise des moments à l'intérieur de la plage de -0,2 à 0,2 N.m, comme illustré dans la Figure 7 ci-dessous. Cette figure représente l'évolution des efforts parasites résiduels lors d'une flexion/extension (première colonne), inflexion latérale (deuxième colonnes) et rotation axiale (troisième colonne). Ces évolutions traduisent que les forces et les moments sont bien corrigés dès qu'ils atteignent les bornes de $\pm 2N$, respectivement $\pm 0{,}2N.m$. On remarque aussi des petits pics de force qui apparaissent ponctuellement aux changements de mouvement (ex. en repartant en extension après une flexion).

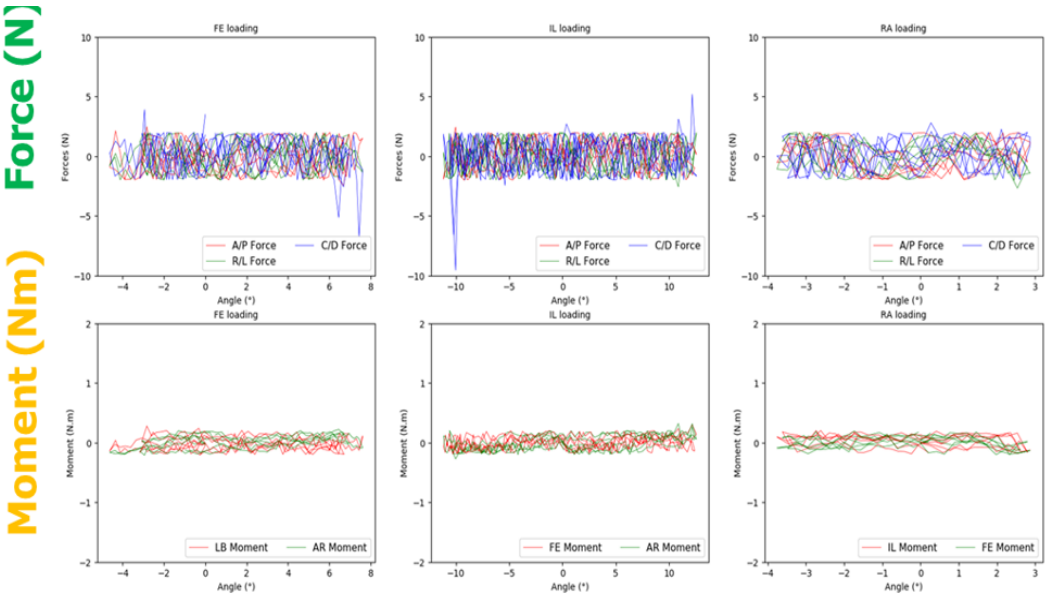


Figure 7 : évaluation des efforts parasites résiduels (précision contrôlée à 2N et 0.2Nm), source : « document interne Laboratoire de Biomécanique et Mécanique des Chocs »

1.2.3 Contrôler la robustesse et la vitesse :

Toujours en prenant l'exemple de l'inflexion latérale, bien que cela ne soit pas immédiatement évident, ce mouvement présente une interaction complexe. Comme mentionné (1.1), lorsque nous induisons un mouvement d'inflexion latérale, cela entraîne simultanément une rotation axiale au-delà d'une zone qualifiée de zone neutre. Cette zone neutre Figure 8, qui s'étend sur une plage d'environ +/- 0,2 N.m au niveau d'une unité fonctionnelle vertébrale, ajoute une couche de complexité à cette situation. Dans cette zone, le seul disque intervertébral est responsable de la raideur intervertébrale. Il s'agit d'un tissu mou se comportant comme une liaison mécanique, avec des valeurs très faibles (et donc des amplitudes de rotation relative importantes) de raideur quel que soit l'axe de rotation. Une problématique clé en début de stage était que mes superviseurs avaient réussi à quantifier (cette information n'étant pas fournie par le constructeur) que la capacité du robot à mesurer avec précision les moments extérieurs appliqués était à peu près du même ordre de grandeur que la zone neutre elle-même, Autrement dit, le protocole ne pouvait dans tous les cas pas permettre de corriger avec l'exactitude requise les moments (hors moment utile où on applique le couple pur) à l'intérieur de la plage de -0,2 à 0,2 N.m, ce qui représente un défi pour la robustesse et la précision du système de mesure.

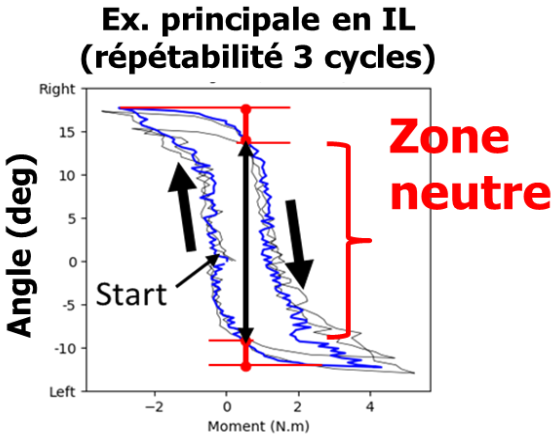


Figure 8 : la plage de zone neutre (cas Inflexion latéral), source : « document interne Laboratoire de Biomécanique et Mécanique des Chocs »

En résumé, ces trois problématiques soulignent l'importance de développer des méthodes de sollicitation et de mesure précises pour reproduire fidèlement les conditions physiologiques et de garantir la fiabilité et la performance des dispositifs de test utilisés. Cependant, afin d'améliorer la précision des mesures, l'équipe de recherche a opté pour une alternative : l'utilisation d'un capteur de forces externe plus précis. Ce capteur a été installé à l'extrémité du robot à mon début de stage, offrant ainsi une solution permettant de relever le défi de la mesure précise des moments et forces extérieurs appliqués.

1.3 Objectifs de stage

L'objectif principal du stage consistait à effectuer une transition du contrôle en Java vers l'utilisation de MATLAB et Simulink, en vue d'améliorer à la fois la vitesse et la précision du contrôle en compliance du robot manipulateur. Cela impliquait la réalisation des étapes intermédiaires suivantes :

- Tâche (1) : Explorer en profondeur la documentation technique du robot Kuka LBR iiwa 14, comprendre son fonctionnement et acquérir les compétences pour son utilisation. En parallèle, réaliser une analyse bibliographique portant sur diverses approches de contrôle en compliance.

- Tâche (2) : Analyser en détail les scripts Java des protocoles existants afin de développer une compréhension approfondie du sujet et proposer des améliorations potentielles pour les implémentations existantes.

- Tâche (3) : Me familiariser avec le capteur externe à six axes et créer des scripts dans Matlab pour réaliser une lecture en temps réel des mesures, en définissant les divers paramètres tels que la fréquence d'acquisition et la durée des mesures, etc…

- Tâche (4) : Me familiariser avec le logiciel de simulation du robot « CoppeliaSim » et explorer les diverses fonctions et outils pour simuler les solutions implémentées de manière virtuelle avant de passer à la mise en pratique sur le robot réel.

- Tâche (5) : Acquérir une bonne compréhension de l'environnement KUKA Sunrise Workbench afin d'établir une communication efficace avec le robot et pouvoir maintenir tous les aspects liés à la sécurité via l'interface Matlab.

- Tâche (6) : Prendre en main, explorer et sélectionner les méthodes et les outils les plus pertinents pour un contrôle en admittance. En particulier, cela impliquait la mise en place de l'interface avec le capteur de force à six axes. Trouver un schéma adapté aux tâches spécifiques avec les entrées/sorties requises. Opter pour la méthode de commande qui répond aux spécifications en termes d'entrées/sorties et choisir l'outil le mieux adapté pour l'implémentation.

2. Présentation de l'entreprise

2.1 Présentation générale et historique

Le laboratoire de biomécanique et mécanique des chocs (LBMC) est une unité de recherche mixte entre l'université Gustave Eiffel et l'université de Lyon 1 Claude Bernard.
Crée en 2020, l'université Gustave Eiffel résulte de la fusion entre l'institut français des sciences et technologies des transports, de l'aménagement et des réseaux (IFSTTAR) et l'Université Paris Est-Marne-la-Vallée.
Cette association a permis de mettre en commun leurs expertises et leurs ressources afin de répondre aux enjeux de l'urbanisme, le transport, la mobilité des personnes et des biens. L'université dispose du département « Transport, Santé, Sécurité » (TS2) auquel est rattaché le Laboratoire de Biomécanique et Mécanique des Chocs.
Concernant l'université Claude Bernard Lyon 1, c'est l'un des pôles majeurs de la recherche française. Disposant de plus de 62 unités de recherche, ses équipes sont spécialisées dans les domaines des sciences et technologies, de la santé et des sciences du sport, des domaines qu'un retrouve à travers les thèmes de recherche du LBMC.
Le LBMC est un laboratoire présent sur plusieurs sites comme l'indique le schéma ci-dessous :

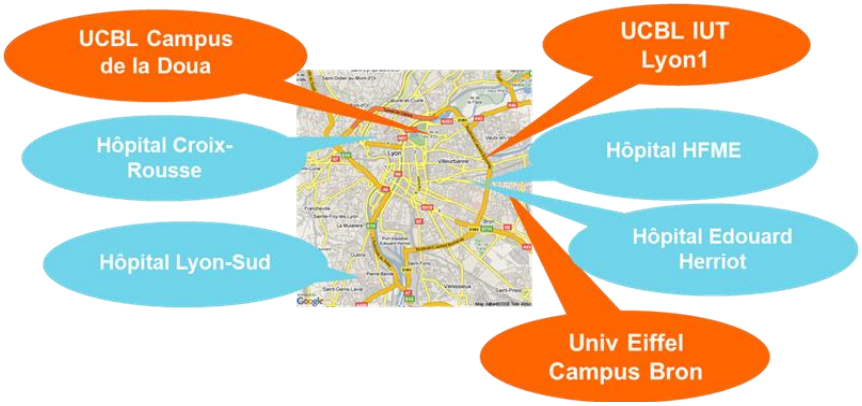


Figure 9 : cartographie des sites du laboratoire LBMC, source : « https://lbmc.univ-gustave-eiffel.fr/contacts/plan-et-acces »

Concernant mon stage il s'est déroulé sur le site de l'université Gustave Eiffel-Campus Bron.

- Le LBMC est ainsi composé de plus de 80 membres (41 permanents, environ 30 doctorants et une dizaine de post-doctorats et CDD) auxquels s'ajoutent une vingtaine de stagiaires accueillis chaque année. Les savoirs faire de ses équipes sont divers : biomécanique des chocs, mécanique des structures, incertitudes, biomécanique des tissus, anatomie et chirurgie, ergonomie physique, biomécanique du mouvement, biomécanique musculosquelettique.

Comme les autres laboratoires présents sur le site de l'université Gustave Eiffel, le LBMC a pour mission de conduire des travaux de recherche finalisée et d'expertise dans les domaines des transports, des infrastructures, des risques naturels et de la ville pour améliorer les conditions de vie de nos concitoyens et plus largement favoriser un développement durable de nos sociétés.

C'est dans cette optique que les équipes du laboratoire travaillent sur deux thématiques : « faciliter les déplacement » à savoir la modélisation humaine dans le contexte de la protection et du confort des passagers dans les transports et la deuxième « maintenir le corps en bonne santé » qui est liée au maintien des capacités fonctionnelles au cours de la vie et à l'intégrité du système musculosquelettique.

2.2 Thématiques de recherches

Principalement, on distingue deux thèmes de recherche comme nous le montre la Figure 10 :

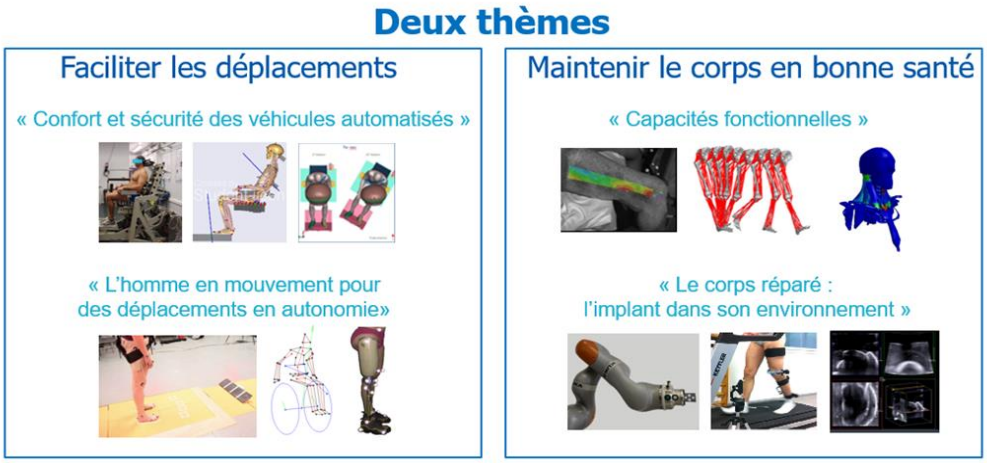


Figure 10 : schéma de l'organisation de la recherche au sein du LBMC, source : « https://lbmc.univ-gustave-eiffel.fr/themes-de-recherches »

- Le corps réparé « l'implant dans son environnement » :

Cet axe représente un défi plus industriel pour le laboratoire où les études s'intéressent à l'évolution réglementaire des implants et leur développement pour les différentes parties du corps (hanche, genou, cheville, épaule…), impression 3D de substituts et implants sur mesure. Il mobilise une équipe de recherche multidisciplinaire composée de cliniciens et de chercheurs en biomécanique dont mon tuteur de stage. L'équipe, se focalise sur l'étude du système musculosquelettique et des capacités fonctionnelles au cours de la vie. C'est dans cette optique, qu'ils ont fait l'acquisition du robot manipulateur comme outil pour les essais et que mon sujet de stage a été créé.

2.3 Les plateformes expérimentales

Dans le but de valider les recherches menées, de nombreux essais expérimentaux sont réalisés au sein du LBMC. Afin de mener à bien ces travaux des moyens d'essais sont mis à disposition sur les différentes plateformes expérimentales illustrés sur le schéma de la figure suivante :

Figure 11 : schéma des différentes plateformes expérimentales du laboratoire, source : « https://lbmc.univ-gustave-eiffel.fr/plateformes »

Mon travail s'est effectué sur le site principal de Bron au sein de la plate-forme expérimentale BioExp.

3. Etude Bibliographique

La convergence entre la robotique et la biomécanique a donné lieu à la robotique biomécanique, explorant les interactions entre les machines et les systèmes biologiques. Ce domaine exploite les principes mécaniques des mouvements humains et des tissus biologiques pour concevoir des systèmes robotiques plus adaptés et en harmonie avec la biologie. Cette symbiose ouvre des perspectives pour la compréhension des mouvements humains, la conception de prothèses naturelles et l'amélioration de la réhabilitation et de la santé humaine.

Pour cette étude, nous avons décidé avec mon tuteur de centrer cette revue bibliographique sur le contrôle des bras robots en réponse à des sollicitations externes. Cette orientation a été prise en considération en raison de son rôle essentiel dans la transition actuelle vers une interface en Matlab/Simulink au sein du LBMC.

3.1 Interactions Humain-Robot

Même si la robotique commerciale commence à changer doucement, elle se compose principalement de bras mécaniques industriels. Bien que ces bras aient amélioré leurs performances et soient devenus moins onéreux au fil des années, ils fonctionnent toujours de la même manière que le tout premier robot, appelé Unimate Figure 12. Ces robots restent généralement confinés dans des espaces clos et effectuent des mouvements préprogrammés par les humains.

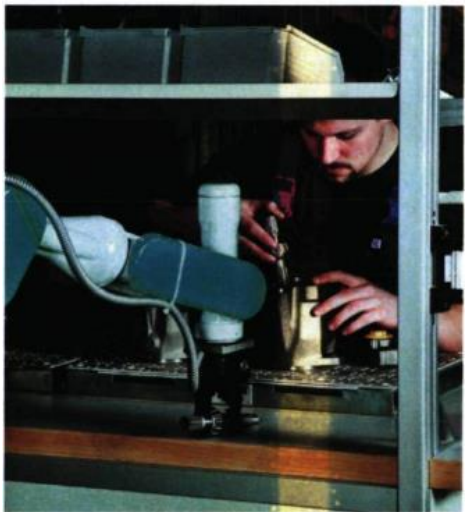
Figure 12 : Unimate premier robot industriel [Duchaine (2010)]

Faire en sorte que les robots puissent partager le même espace que les humains et travailler avec eux représente une évolution naturelle vers une robotique plus avancée (ou cobotique). Cette cohabitation potentielle a le pouvoir d'avoir un impact important dans les lieux de travail, dans le domaine de la santé et dans d'autres aspects de la vie quotidienne.

(a) Kuka LWR. Robot d'ipHR originalement conçu par le DLR et maintenant commercialisé par KUKA.

(b) Omnirob. Un prototype présenté par Kuka ajoutant une dimension de mobilité au LWR.

(c) Rob@work en situation d'interaction.



(d) Rob@work assistant de manière autonome.

Figure 13 : Exemples de robots conçus spécifiquement pour l'interaction physique avec l'humain [Duchaine (2010]

3.2 Contrôle en compliance et l'asservissement

Le contrôle en compliance et l'asservissement des robots manipulateurs en série sont des aspects cruciaux dans le domaine de la robotique moderne. La compliance, dans ce contexte, fait référence à la capacité d'un robot à s'adapter de manière souple aux forces et aux contraintes externes, reproduisant ainsi le comportement élastique et réactif des tissus biologiques. Cette caractéristique permet aux robots de travailler en collaboration avec les êtres humains de manière plus sûre et plus fluide, que ce soit dans des environnements industriels ou médicaux. L'asservissement, d'autre part, implique le contrôle précis des mouvements d'un robot pour qu'il suive des trajectoires souhaitées malgré les perturbations externes. Cela implique souvent l'utilisation de capteurs intégrés pour mesurer en temps réel les forces et les positions, permettant au robot de s'ajuster automatiquement pour maintenir la précision et la stabilité de ses mouvements. La combinaison de la compliance et de l'asservissement offre une synergie puissante, permettant aux robots manipulateurs en série de s'engager dans des tâches complexes avec une adaptabilité et une précision supérieure, tout en minimisant les risques de collision et en maximisant leur interaction avec l'environnement.

3.3 Les méthodes de contrôle en compliance

Divers auteurs se sont penchés sur le concept de compliance, dont une formalisation mathématique est établie dans des travaux tels que [Rouget (1983)] et [Badano (1993)] . Les forces appliquées F en un point du robot induisent un couple en un autre point, ce qui est décrit par l'équation :

$$F = K(D)D + B(D)\frac{dD}{dt} + I(D)\frac{d^2D}{dt^2} \qquad (1)$$

Cette équation révèle l'influence des matrices d'inertie I(D), de frottement B(D) et de raideur K(D), qui dépendent du déplacement du robot D.

L'obtention d'un comportement compliant pour un robot repose sur l'introduction de flexibilité et de souplesse dans son fonctionnement, ce qui peut être réalisé de deux manières distinctes, à savoir la compliance active et la compliance passive. Dans le cas de la compliance active, la flexibilité est mise en œuvre par le biais de stratégies logicielles complexes. Ce processus exige la détection précise des forces résultant du contact avec l'environnement, et ces forces sont ensuite gérées et contrôlées par des actionneurs asservis. Ainsi, le robot peut réagir de manière adaptative aux changements dans son environnement et ajuster son comportement en conséquence.

En revanche, la compliance passive se manifeste par la souplesse inhérente des structures mécaniques du robot. Cette souplesse est conçue de manière à permettre au robot de s'adapter naturellement aux variations d'efforts et de position résultant des interactions avec l'environnement. Les structures flexibles du robot sont conçues de telle manière qu'elles absorbent et compensent les erreurs de positionnement qui surviennent lors des contacts avec des surfaces ou des objets. Cette déformation naturelle des structures permet au robot de maintenir un comportement conforme tout en minimisant les forces potentiellement dommageables qui pourraient être générées lors du contact avec l'environnement.

Ainsi, les deux approches, active et passive, visent à conférer au robot la capacité de répondre de manière adaptative aux contraintes de son environnement. Chacune de ces approches a ses avantages et ses inconvénients, et le choix entre les deux dépend souvent des exigences spécifiques de la tâche, de l'application et de l'environnement dans lequel le robot opère.

	AVANTAGES	INCONVENIENTS
COMPLIANCE PASSIVE	• peu chère • fiable • simple à mettre en œuvre • réponses rapides	• dédiée à une seule application • ne contrôle pas l'effort de Contact.
COMPLIANCE ACTIVE	• adaptable suivant la tâche • commande fine des efforts de contact	• nécessite une loi de commande • nécessite des actionneurs et capteurs supplémentaires • complexe • onéreuse • fiabilité moyenne en milieu industriel

Tableau 1 : comparaison des différentes méthodes [Prelle (1997)]

Dans le contexte de mon sujet de stage, nous avons privilégié la stratégie de compliance active. Cette orientation découle de la nature de notre application de sollicitation mécanique. En effet, notre choix se justifie par le fait que l'environnement dans lequel nous opérons était mal connu et que la raideur de cet environnement était variable. Il s'agissait par ailleurs in fine (en terme causal) de contrôler les efforts subis par le segment anatomique, ces derniers agissant par conséquence sur le robot.

Les approches de contrôle en compliance active peuvent être classées en deux catégories, en fonction de la nature de la tâche à accomplir. Dans la première catégorie, on trouve les tâches qui nécessitent un contrôle en position tout en limitant les efforts de contact. Dans ce cas, les consignes sont définies en termes de position ou de vitesse. D'autre part, la deuxième catégorie englobe les tâches où les forces de contact sont régulées selon une trajectoire prédéfinie. Dans cette situation, les consignes sont exprimées en forces.

La littérature présente plusieurs approches de commande qui sont comparées et discutées, notamment dans des ouvrages tels que ceux de [Duchaine (2010], [Prelle (1997)], [Safeea (2020)].

3.3.1 La commande hybride position /effort

Ce type de commande a été proposé par [Raibert (1981)]. Elle a ensuite été reprise par de nombreux auteurs [Qin (2013)]. Le principe de cette méthode est qu'un degré de liberté (DDL) ne peut être contrôlé en position et en force en même temps. L'idée est donc de diviser l'espace des DDL en deux sous-espaces, avec le premier commandé en position et le deuxième piloté en force. A noter qu'un même actionneur peut intervenir sur plusieurs DDL, et donc des consignes peuvent être envoyées simultanément en position et en effort. On note S la matrice de sélection qui nous permet de choisir le type de commande pour chaque degré de liberté et elle est diagonale. L'élément s_i est est égale à 1 pour une commande en position et 0 pour une commande en effort.

La structure fonctionnelle de ce type de contrôle est représentée par le schéma bloc suivant :

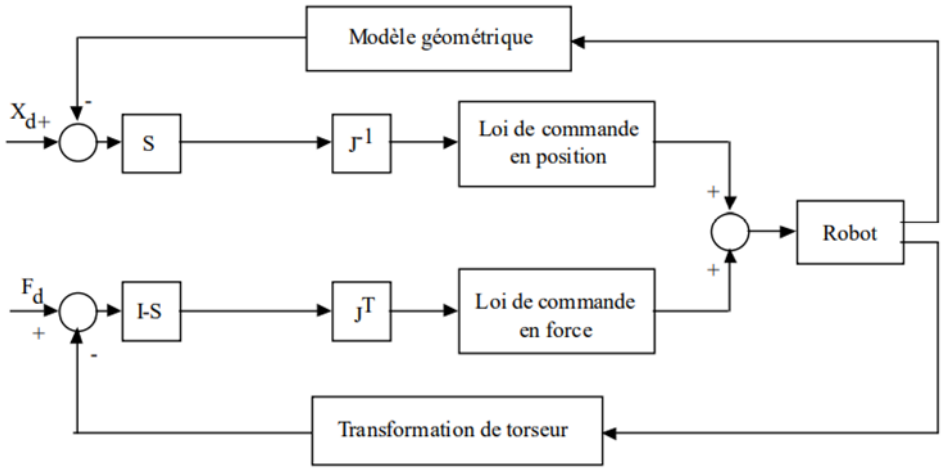


Figure 14 : Schéma bloc commande hybride position/force.[Raibert (1981)] [Qin (2013)]

Cette technique convient aux applications nécessitant des commandes différentes dans des directions différentes. Son inconvénient principal est qu'elle exige l'inversion de la matrice jacobienne, ce qui peut induire à une instabilité cinématique : comme évoqué par [Pujas (1995)] dans sa thèse la condition suffisante de stabilité dépend de la matrice jacobienne (donc de la configuration du robot) et nécessite une modélisation précise de l'environnement.

3.3.2 Commande en amortissement

Cette technique a été proposée par [Whitney (1969)]. Pour ce type de commande la consigne est en vitesse. Un retour en vitesse en boucle fermée est effectué à partir des forces extérieures mesurées sur l'effecteur avec l'introduction d'un coefficient d'amortissement k_f . La variable pilotée est donc la vitesse dans l'espace effecteur. La commande est représentée par le schéma fonctionnel suivant :

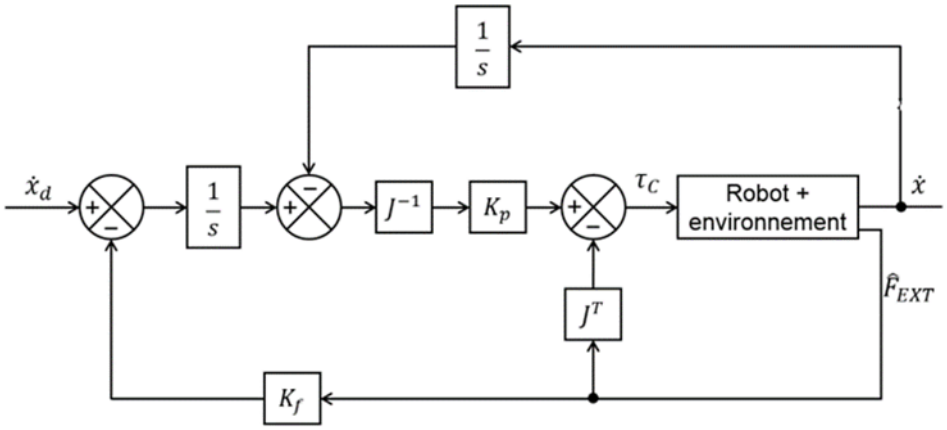


Figure 15 : Schéma bloc commande en amortissement [Sandoval Arevalo (2017)]

3.3.3 Commande par raideur active

En 1980, [Salisbury (1980)] propose de contrôler activement la raideur apparente d'un manipulateur. Le principe de cette méthode est de donner au robot le comportement d'un ressort à raideur programmable où la raideur est changée de façon logicielle pour pouvoir s'adapter aux évolutions de la tâche [Liégeois (2000)].

Cette méthode définit, en statique, une fonction linéaire entre les forces de contact et la position finale de l'effecteur via une matrice de raideur cartésienne :

$$F_c = K_c \, \Delta x_c$$

Avec :

F_c : effort de contact
K_c : matrice de raideur souhaitée
Δx_c: déplacement différentiel

Le couple différentiel des articulations est donné par :

$$\Delta \Gamma = J^T \, F_c$$

D'où :

$$\Delta x_c = J \, \Delta q$$

On obtient alors, la commande dans l'espace articulaire :

$$\Delta \Gamma = K_q \Delta q$$

Avec :
$\mathbb{J}$: la matrice jacobienne du manipulateur.
Δq : différentiel du vecteur articulaire.
K_q : la matrice de raideur dans l'espace articulaire.

$$K_q = \mathbb{J}^T K_c \mathbb{J}$$

Cette commande présente une régulation implicite de l'effort à travers l'introduction d'une matrice K_F considérée l'inverse de la raideur désirée.

Cette loi est représentée par schéma fonctionnel suivant :

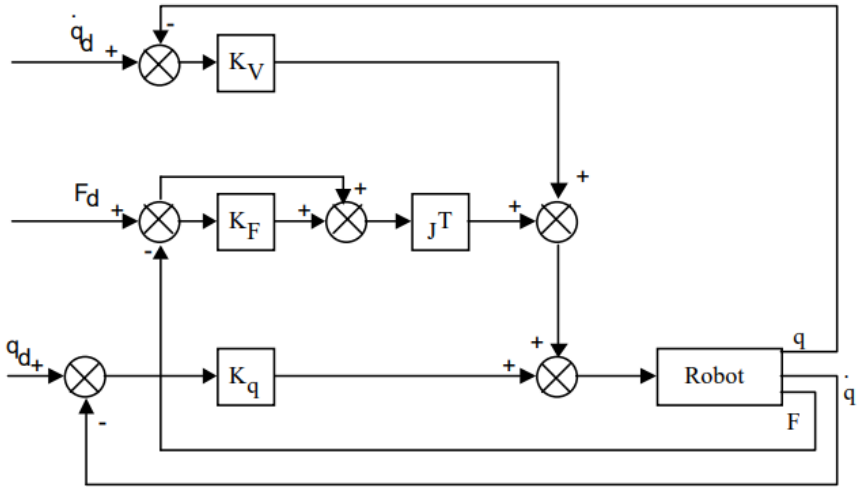


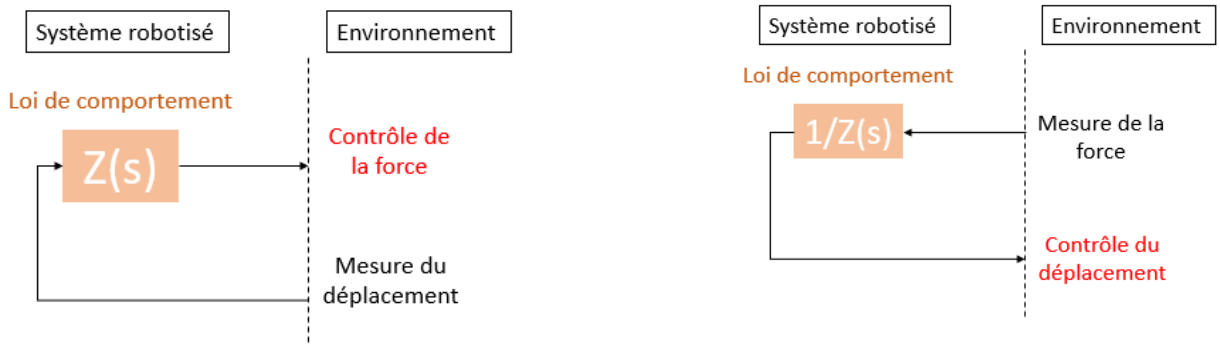
Figure 16 : schéma bloc commande par raideur active *[Prelle (1997)]*

La loi de commande s'écrit donc sous la forme suivante :

$$\Gamma = K_q(q_d - q) + K_v(\dot{q_d} - \dot{q}) + J^T (F_d + K_F(F_d - F))$$

3.4 Problématique de la commande en impédance et en admittance

La commande en impédance ou en admittance est une approche hybride qui cherche à établir une relation entre la force exercée et le déplacement du robot. Par exemple, on peut souhaiter que le robot se déplace proportionnellement à une force appliquée, ce qui serait semblable au comportement d'un ressort. Les approches en impédance et en admittance permettent toutes deux de définir cette relation entre force et déplacement. D'un point de vue théorique, ces approches sont presque équivalentes, comme on peut le voir dans la Figure 17. La principale différence réside dans la causalité : un contrôleur en impédance mesure le déplacement et ajuste la force en conséquence, tandis qu'un contrôleur en admittance mesure la force pour réguler le déplacement. Dans un monde idéal avec des mesures et un contrôle parfait des forces et des déplacements, les deux approches donneraient des résultats identiques pour la même loi de comportement. Cependant, en pratique, il existe d'importantes différences en termes d'implémentation et de performances entre ces approches. Il est à noter que, suite à échanges entre mon tuteur de stage et l'équipe de robotique des Hôpitaux Universitaire de Genève (avec qui une collaboration était entamée), nous avons plus particulièrement orienté notre réflexion sur l'implémentation en admittance.



(a) Commande en impédance (b) Commande en admittance

Figure 17 : Deux façon d'imposer une loi de comportement qui relie la force au déplacement

Les caractéristiques mécaniques d'un système sont fréquemment exprimées à l'aide de fonctions de transfert dans le domaine de Laplace pour une notation compacte. La fonction Z(s) est employée pour définir l'impédance, en représentant le rapport du signal de force f(s) au signal de déplacement x(s) dans le domaine de Laplace. D'un autre côté, l'admittance Y(s) est le réciproque de la fonction d'impédance.

Impédance : $Z(s) = \frac{f(s)}{x(s)}$ Admittance : $Y(s) = \frac{1}{Z(s)} = \frac{x(s)}{f(s)}$

Un système mécanique avec une inertie m, un amortissement linéaire b et une rigidité k a une impédance égale à :

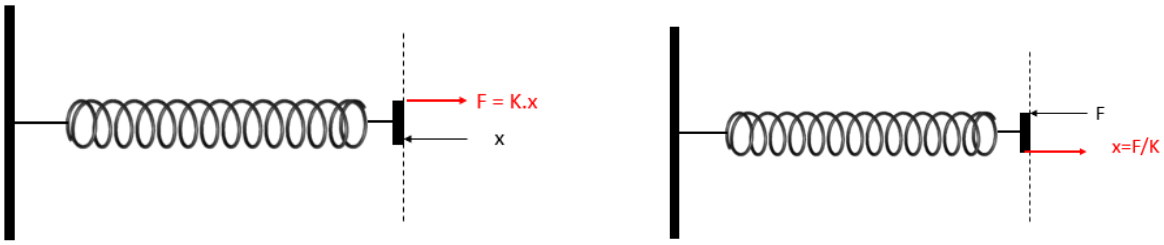
$$Z(s) = ms^2 + bs + k$$

L'opération de multiplié le signal de déplacement par l'impédance Z(s) est équivalent dans le domaine temporel à :

$$f(s) = Z(s)x(s) = (ms^2 + bs + k)x(s) \Leftrightarrow f(t) = m\frac{d^2}{dt^2}x(t) + b\frac{d}{dt}x(t) + kx(t)$$

Exemple :

Une loi de comportement mécanique qu'il est souvent désirable d'implémenter prend la forme générale de la loi linéaire de Hooke, i.e. représentant le comportement d'un ressort. Comme illustré à la Figure 18, l'approche en impédance se résume à mesurer un signal de position x et à le multiplier par une constante de rigidité k pour obtenir la force à appliquer. L'approche en admittance se résume à mesurer un signal de force f et le diviser par la constante de rigidité k pour obtenir le déplacement x à imposer. On utilise parfois la notion de compliance C = 1/k qui est l'inverse de la rigidité.



(b) Commande en impédance (c) Commande en admittance

Figure 18 : Imposition d'une loi de comportement f = K.x (ressort)

Dans la réalité, le choix entre les différentes approches est principalement influencé par le type d'actionneur utilisé au sein du système robotique. L'approche en impédance est plus appropriée pour les systèmes dotés d'actionneurs agissant comme des sources de force (par exemple, des moteurs électriques sans réducteur ou des cylindres pneumatiques). En revanche, l'approche en admittance est mieux adaptée aux systèmes où les actionneurs agissent comme des sources de déplacement, offrant un meilleur contrôle en position ou en vitesse, et est généralement employée dans les robots collaboratifs industriels.

3.5 Commande en admittance

L'idée fondamentale derrière la commande par admittance repose sur le fait que, quel que soit le contrôleur utilisé, le robot agit toujours comme un système physique spécifique à son effecteur. Bien que le contrôleur modifie la dynamique apparente de l'effecteur par rapport à la dynamique réelle du robot, basée sur ses composants mécaniques, le comportement final possède toujours une équivalence physique. En se basant sur cette idée, l'approche consiste à orienter cette dynamique apparente (aussi appelée dynamique virtuelle) de manière à ce que le comportement à l'endroit d'interaction suive celui d'un système physique désiré. Le rôle du contrôleur est ainsi de soit ajouter soit soustraire une certaine dynamique à celle du robot pour obtenir le comportement souhaité.

L'équation générale qui représente la dynamique virtuelle désirée est la suivante :

$$f = M\ddot{p} + C\dot{p} + K\Delta p$$

Avec :

P : vecteur position	M : matrice Masse
ṗ : vecteur vitesse	C : matrice d'amortissement
p̈ : vecteur d'accélération	K : matrice raideur

Ainsi, la commande en admittance se positionne naturellement comme une méthode d'intermédiaire entre les domaines de la position et de la force, ce qui la rend particulièrement pertinente pour la commande des robots destinés à collaborer avec les êtres humains. Pour assurer une collaboration efficace entre un humain et un robot, ce dernier doit jouer le rôle de suiveur avec succès. En se basant sur le sens du toucher comme unique moyen d'interaction, le robot détecte les forces au point d'interaction engendrées par les mouvements de l'humain et génère des mouvements appropriés en réponse. Dans cette perspective, la commande par admittance émerge comme une solution naturelle qui établit aisément le lien entre les forces perçues et les mouvements souhaités. En ce qui concerne les applications du LBMC : bien que le besoin décrit dans ce rapport ne soit pas haptique, l'approche en admittance semblait adaptée, à la fois pour y répondre et dans la perspective de développement d'autres applications mettant en œuvre une manipulation par un opérateur ou un sujet volontaire.

3.5.1 Commande en admittance dans l'espace articulaire

Lorsque nous utilisons le contrôle en admittance dans l'espace des articulations, des capteurs de force (en l'occurrence, de couples) mesurent la force actuelle à chaque articulation et envoient ensuite une commande de déplacement à l'actionneur correspondant. Les lois en admittance peuvent également être exprimées sous forme de matrices pour décrire le comportement de toutes les articulations en une seule équation

$$M\underline{\ddot{q}_d} + B\underline{\dot{q}_d} + K\underline{q_d} = \underline{\tau}$$

Avec : $\underline{q_d}$, $\dot{q}_d$, $\underline{\ddot{q}_d}$ vecteur de position, vitesse, accélération articulaire respectivement, et les Matrice définie positives M, B, K, matrice masse, amortissement, raideur respectivement, et $\underline{\tau}$ vecteur couple moteur mesuré pour chaque articulation du robot.

La loi de comportement représentée dans la Figure 19 correspond en réalité à la simulation d'un système composé d'une masse, d'un ressort et d'un amortisseur. Cette simulation prend en compte une force réelle captée par un capteur en tant qu'entrée. Ensuite, le résultat de cette simulation est transmis à un contrôleur de déplacement. En conséquence, le robot imite ainsi le système virtuel qui a été simulé.

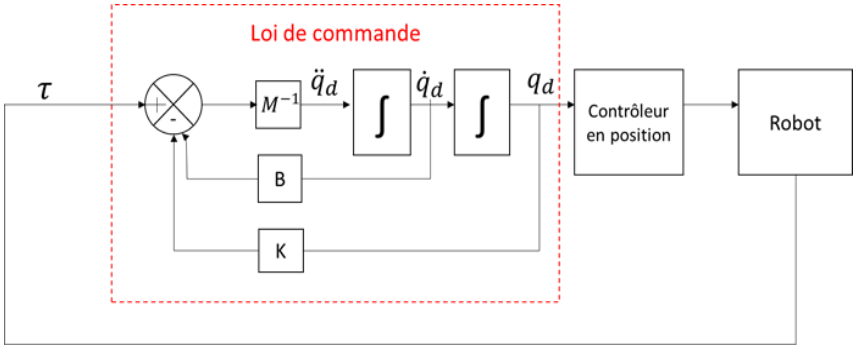


Figure 19 : Commande de l'admittance aux joints d'un robot : schéma bloc [Girard (2020)]

3.5.2 Commande en admittance dans l'espace opérationnel

L'approche en admittance dans l'espace de l'effecteur repose sur la mesure de la force appliquée à l'effecteur, en utilisant un capteur externe monté à l'extrémité du robot, ainsi que sur le contrôle du déplacement de l'effecteur à un niveau inférieur. Une loi de commande courante pour un comportement similaire à celui d'une masse-ressort-amortisseur à l'effecteur est la suivante :

$$M\,\ddot{X}_d + B\,\dot{X}_d + K\,X_d = f_e$$

$X_d, \dot{X}_d, \ddot{X}_d \in \mathbb{R}^{6\times1}$: vecteur déplacement, vitesse et accélération cartésiens.
$\underline{f_e} \in \mathbb{R}^{6x1}$: le vecteur de force erreur (la différence entre la force mesurée et désirée).
M $\in \mathbb{R}^{6x6}$: la matrice d'inertie virtuelle désirée.
B $\in \mathbb{R}^{6x6}$: la matrice d'amortissement virtuelle désirée.
K $\in \mathbb{R}^{6x6}$: la matrice de raideur virtuelle désiré.

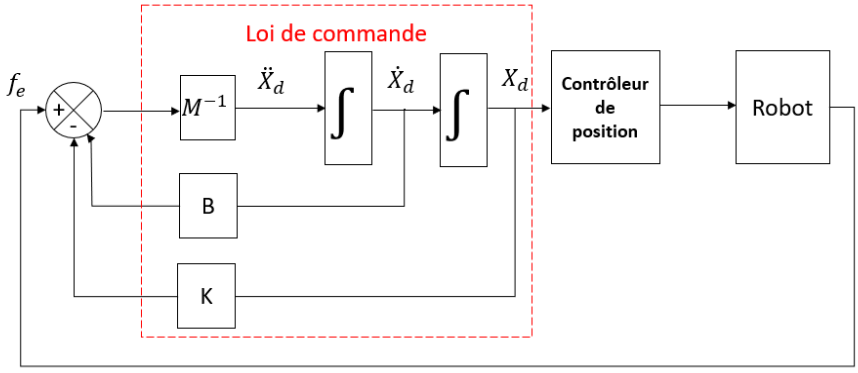


Figure 20 : Commande de l'admittance à l'effecteur d'un robot avec une loi de comportement de type masse-ressort-amortisseur : schéma bloc [Girard (2020)]

Grâce à cette loi de commande, si un humain pousse sur l'effecteur du robot, le robot se déplacera dans la direction de la poussée avec un déplacement proportionnel à l'intensité de la poussée. En l'absence de poussée, le robot restera immobile.

Une différence importante dans cette approche par rapport à l'approche précédente est que seules les forces appliquées à l'effecteur auront un effet sur la position du robot. Si une force est exercée sur les parties du robot situées en amont de la zone de détection, le contrôleur ne pourra pas tenir compte de cette force externe dans le calcul du déplacement du robot selon la loi de comportement souhaitée.

3.6 Présentation du robot industriel Kuka iiwa lbr 14

En entamant mon stage, j'ai débuté (Tâche 1) par une étude bibliographique approfondie sur le robot KUKA LBR iiwa 14. Cette étude s'est basée sur les documents techniques fournis par KUKA *[KUKARoboticArmMaterial (2015)]*, me permettant d'explorer ses différentes fonctionnalités. Par la suite, j'ai bénéficié de l'encadrement de mon tuteur pour apprendre à utiliser le robot et comprendre les différents paramètres clés (repères, modes de contrôle, etc…) J'ai ainsi réalisé des mouvements simples de ses axes et des mouvements par rapport à divers repères, définissables à l'aide du "Smart PAD" du robot. J'ai également pris le temps de me familiariser avec l'environnement logiciel "KUKA Sunrise Workbench", qui offre la possibilité de communiquer avec le robot et de paramétrer des éléments tels que la vitesse des mouvements, le choix de l'effecteur final (TCP) et la définition des aspects de sécurité du robot dans son espace de travail.

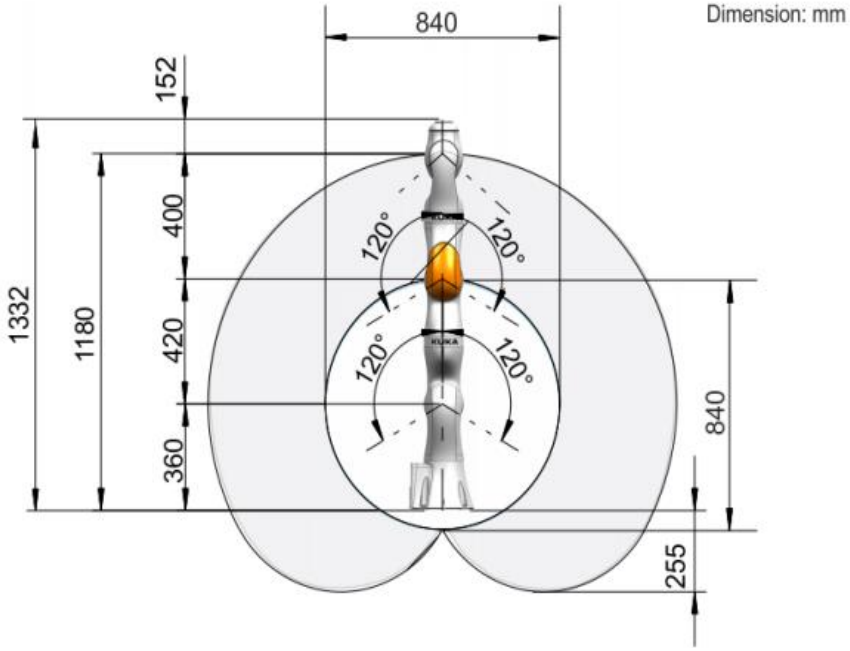


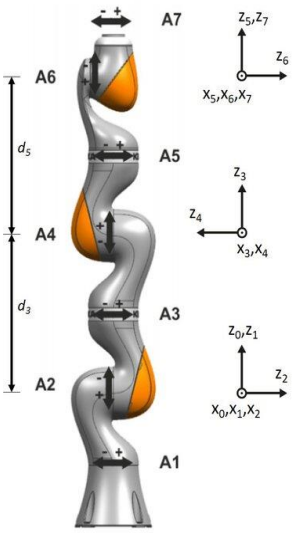
Figure 21 : Espace de travail, LBR iiwa 14 R820, vue de coté

Notre robot industriel est composé des éléments suivants :

- ⇨ Le manipulateur
- ⇨ Le contrôleur robotique Kuka Sunrise Cabinet
- ⇨ Le boîtier de commande portatif Kuka Smart PAD
- ⇨ Les câbles de liaison
- ⇨ Le logiciel
- ⇨ Les options et accessoires

- **Manipulateur :**

Le robot LBR iiwa est spécialement conçu pour les tâches de montage sensibles Il s'agit par ailleurs d'un des premiers robots 'sensibles' et capable de collaborer avec les humains, fabriqué en série. Le terme "LBR" signifie "Leichtbauroboter" (robot léger), tandis que "iiwa" signifie "intelligent industrial work assistant". Nous disposions de la version ayant une capacité de manipulation de charge utile de 14 kilogrammes à 1 m/s.



Link i	θ_i	d_i	a_i	α_i
1	q_1	0	0	0
2	q_2	0	0	$-\pi/2$
3	q_3	$0.42m$	0	$\pi/2$
4	q_4	0	0	$\pi/2$
5	q_5	$0.40m$	0	$-\pi/2$
6	q_6	0	0	$-\pi/2$
7	q_7	0	0	$\pi/2$

Figure 22 : paramètre DH robot kuka lbr iiwa 14 r820

Il offre une portée de 800 à 820 mm et se distingue par sa réactivité. Il dispose d'un mode de contrôle en compliance, via des méthodes pré-implémentées en Java, qui lui permet par exemple de manipuler des pièces fragiles en minimisant les risques de pincement et de cisaillement grâce à la régulation de la position et de la compliance.

Ses autres caractéristiques principales sont : la possibilité d'utilisation de 3 modes, du moins au plus collaboratifs et, dans ce dernier, la possibilité de le manipuler à la main via un bouton en tête d'effecteur. Lors des pauses, il peut ainsi être interrompu et guidé simplement en le manipulant.

Type	**Articulé, collaboratif.**
Nombre d'axes	**7 axes.**
Fonction	**De manutention, de palettisation, d'usinage, à émission optique à étincelles.**
Montage	**Au sol, pour montage au plafond.**
Charge maximale	**14 kg (30,865 lb).**
Rayon d'action	**820 mm (32,28 in).**
Répétabilité	**0,1 mm (0,0039 in).**
Précision	**Non fournie par Kuka (dépendante de la tâche. ex. charge utile/vitesse…)**

Tableau 2 : caractéristique robot lbr iiwa 14 r820

- **Kuka Sunrise Cabinet**

C’est le contrôleur du robot. Il est formé des éléments suivants :
• Un PC de commande
• Un boitier de commande portatif Smart PAD
• Un panneau de raccordement
Le contrôleur du robot joue divers rôles, notamment la gestion de la sécurité, le contrôle du robot lui-même, la supervision logique et la régulation des processus, tout en incorporant un traitement grâce à des capteurs intégrés. Toutes ces fonctionnalités sont regroupées au sein d'un unique contrôleur pour l'intégralité de l'installation. Le contrôleur est donc l’interface physique pour laquelle il était nécessaire de trouver un nouveau protocole de communication en raison du choix du passage à l'utilisation de Matlab.

Figure 23 : Contrôleur Kuka Sunrise Cabinet

- **Boitier de commande Kuka Smart PAD**

C'est un boîtier de commande portable qui rassemble toutes les fonctionnalités de contrôle et d'affichage nécessaires pour commander le bras robot. Il est doté d'un écran tactile appelé interface Smart PAD, qui peut être utilisé avec un stylet ou simplement avec un doigt.

Figure 24 : Smart PAD boitier de commande

- **Kuka Sunrise Workbench**

C’est l’outil pour la mise en service d’une station et pour le développement d’applications de robot. Il est utilisé via un environnement de développement et de programmation sous java, basé sur l’IDE Eclipse. Il dispose des fonctionnalités suivantes pour la mise en service et développement d’applications :

➢ Mise service :
• Installation du logiciel système
• Configuration de cellule robotisée (station)
• Edition de la configuration de sécurité
• Création de la configuration E/S
• Déploiement d’un projet sur le contrôleur de robot

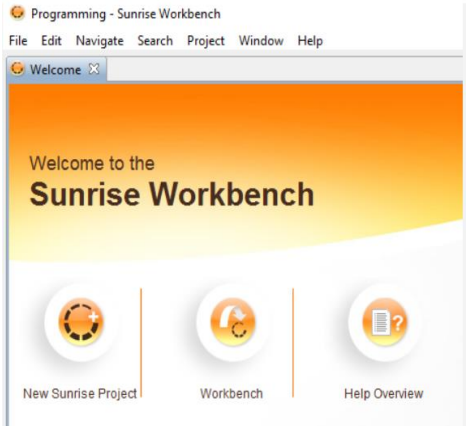


Figure 25 : kuka sunrise workbench

➢ Développement d’applications :
• Programmation les applications robot en utilisant le langage Java
• Gestion les projets et programmes
• Edition et gestion les données de durée d’exécution
• Synchronisation du projet

3.7 Approche et script existant

Je réfère le lecteur à la présentation du travail de mode de contrôle préexistant à mon stage en Annexe A, Cette présentation offre une compréhension approfondie des objectifs ainsi que des principes de correction visés. Ce mode de contrôle est développé en langage Java au sein de l'environnement Kuka Sunrise Workbench. Le protocole discuté dans la section précédente, intitulée "Contexte et Problématique", utilise ce mode de contrôle. Comme je n'avais jamais vraiment utilisé le langage auparavant, j'ai investi du temps initial pour me familiariser avec ses scripts de contrôle (Tâche 2). J'ai accompli cela en consultant des tutoriels et en examinant des exemples d'applications. Avec le soutien de mon tuteur de stage, j'ai réussi à décrypter le script de contrôle, un élément essentiel pour faire progresser le projet de mon stage. Dans le but de mieux appréhender son fonctionnement, j'ai traduit le code Java en un algorithme détaillé en annexe.

4. Matériels et méthodes

Cette section détaillée sur le matériel et les logiciels utilisés reflète la progression de mon apprentissage durant le stage. J'ai commencé par me familiariser avec le robot lui-même et à comprendre l'approche de contrôle existante codée en Java, implémentée par l'équipe de recherche du laboratoire en 2019-2020. Ensuite, je me suis plongé dans l'exploitation du capteur externe (logiciel constructeur et possibilités de communication en vue d'une mesure et d'une exploitation des données en temps réel via Matlab) pour des mesures précises, j'ai exploré la simulation virtuelle avec "CoppeliaSim" et enfin, j'ai exploité des Toolbox existantes (Toolbox Matlab RST et Toolbox KST, développée par [Safeea (2017)][Safeea (2017)] et modifiée par D. Gonzalez (équipe robotique HUG) pour communiquer et interagir avec le robot en temps réel. Ces éléments ont été essentiels pour parvenir à des résultats positifs à la fin de mon stage.

4.1 Capteur de force 6 axes

4.1.1 Description du matériel et logiciel utilisé

L'approche de contrôle actuelle repose sur l'utilisation des capteurs de couple intégrés dans les articulations du robot, avec une précision estimée à environ $\pm 2N$ pour les forces et $\pm 0.2\ N.m$ pour les moments. Le capteur externe déjà mentionné en introduction a été monté à l'extrémité du robot comme le montre la figure suivante :

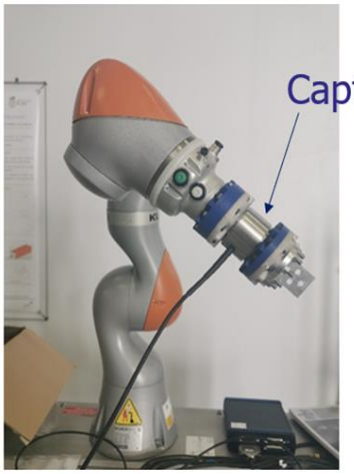


Figure 26 : capteur de force 6 axes (K6D40 500N/20Nm)

Les caractéristiques (matrices de calibration [Annexe B]) ainsi que les erreurs de mesure (affichées par le constructeur) associées aux 6 composantes sont présentées dans les tableaux suivants :

Référence	Fabriquant	Etendue F_x/F_y	Etendue F_z	Etendue M_x/M_y	Etendue M_z	Com
K6D40 500N/20Nm	Me-messytem	500 N	2000N	20Nm	40Nm	USB/ETHERCAT

Tableau 3 : Caractéristiques du capteur d'effort 6 axes.

Effort	F_x(N)	F_y(N)	F_z(N)	M_x(Nm)	M_y(Nm)	M_z(Nm)
Effort nominal (N/Nm)	500	500	2000	20	20	40
Erreur (N/Nm)	± 0,21	± 0,29	± 0,47	± 0,0602	± 0,0113	± 0,02

Tableau 4 : Précision (données constructeur) du capteur d'effort 6 axes.

Figure 27 : Système d'acquisition - 8 voies analogiques - (GSV-8DS)

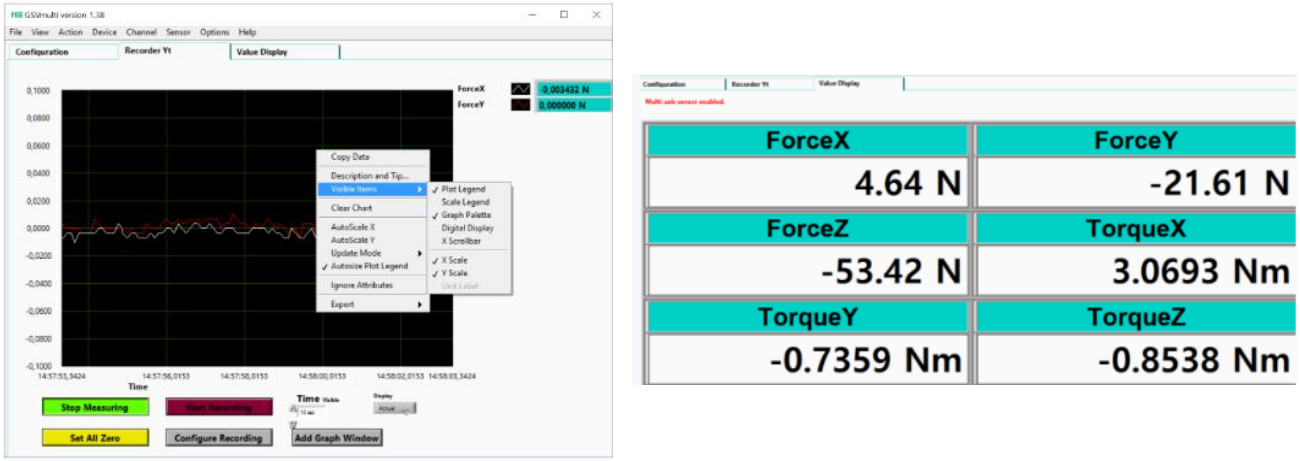
Le capteur est accompagné d'un amplificateur de mesures/système d'acquisition (Figure 27) qui a pour fonction de convertir le signal analogique provenant du capteur d'effort en données numériques exploitables dans une boucle de contrôle. Le système à notre disposition est équipé d'un amplificateur à 8 canaux, caractérisé par une haute résolution dans la plage de fréquences allant de 1 Hz à 48000 Hz. Il permet l'enregistrement simultané des données provenant des 8 canaux sans nécessiter de multiplexeur. De plus, il offre plusieurs options d'interfaces de communication, notamment un port USB, EtherCAT et CANbus, ainsi qu'une interface UART pour le contrôle des mesures à travers une carte électronique Raspberry.

Pour le traitement et la manipulation des données acquises, l'amplificateur est associé à un logiciel nommé "GSVmultichannel". Cette application propose une interface graphique qui permet de régler divers paramètres du capteur, tels que sa calibration, le filtrage des données, l'acquisition et la sauvegarde. Dans le contexte de notre étude visant à mettre en place un contrôle dans Matlab, j'ai recherché sur le site du fabricant "me-système" des méthodes pour traiter les données dans Matlab. Cette recherche a conduit à la découverte de bibliothèques ".dll" qui peuvent être intégrées à Matlab. Ces bibliothèques fournissent diverses fonctions, comme l'activation des canaux, l'acquisition de données, la configuration des fréquences d'acquisition, la calibration, etc. De plus, en utilisant des ressources disponibles sur GitHub *[me-systeme]*, j'ai pu m'appuyer sur des exemples fournis pour établir la communication nécessaire.

4.1.2 Travail réalisé et résultat

Après avoir accompli les tâches (1) et (2), je me suis entièrement concentré sur la tâche (3) qui consistait à me familiariser avec le capteur externe et à créer des scripts sous Matlab. Cette étape était cruciale car elle impliquait l'acquisition en temps réel de mesures, ce qui était l'objectif principal de mon stage. Cependant, avant de débuter la mise en œuvre des scripts, j'ai d'abord pris le temps de me familiariser avec le logiciel "GSVmultichannel". Ce logiciel est conçu pour la visualisation et l'acquisition de données à partir des conditionneurs GSV. Il offre la possibilité de paramétrer diverses options telles que la fréquence d'acquisition, la calibration, la sauvegarde des données et le tarage, etc. entre autres. Cette étape préliminaire m'a permis d'acquérir les connaissances nécessaires pour la création des scripts d'acquisition de données sous Matlab.

La figure ci-dessous illustre la visualisation en temps réel des données, tant sous forme graphique que numérique, à travers le logiciel GSVmultichannel :



(a) Visualisation graphique en temps réel des données (b) Visualisation numérique en temps réel des données

Figure 28 : Visualisation en temps réel des données "GSVmultichannel"

Le fabricant du capteur "me-systeme" propose des exemples d'intégration des bibliothèques "MEGSV86x64.h" et "MEGSV86x64.dll" sur GitHub *[me-systeme]*. Ces bibliothèques sont écrites en langage C/C++. J'ai utilisé la commande "*Calllib*" dans Matlab pour appeler les fonctions contenues dans le fichier ".dll". Cette approche m'a permis de lire les données du capteur et de les configurer selon les besoins.

L'organigramme ci-dessous illustre la structure de mon script (Annexe C) de lecture en temps réel et stockage des données sous Matlab :

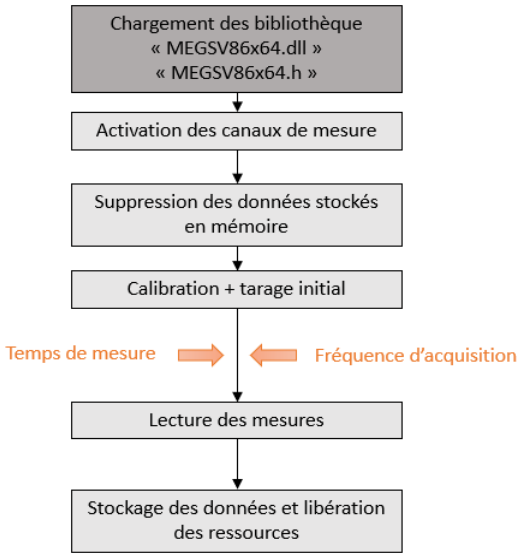


Figure 29 : Organigramme script acquisition de mesures

Après avoir effectué la lecture des mesures, j'ai procédé à leur enregistrement dans un fichier Excel, facilitant ainsi leur analyse et leur traitement ultérieur. Ci-dessous, vous trouverez un exemple représentatif de la visualisation de ces données au cours du temps :

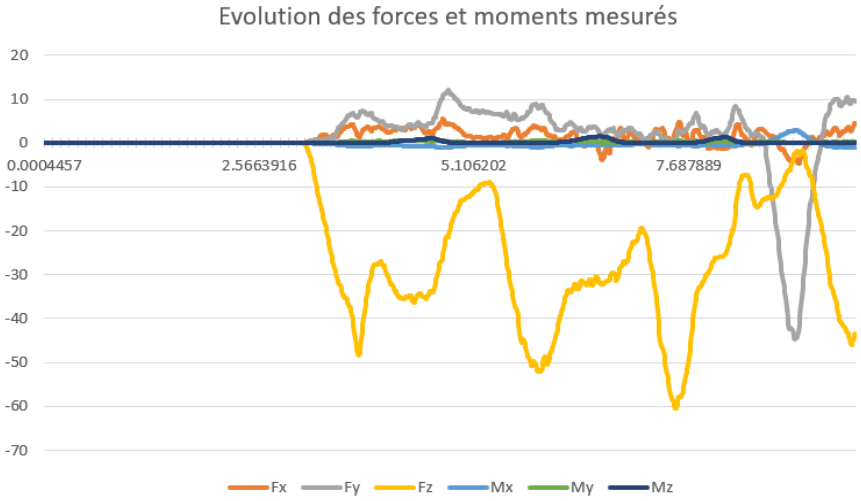


Figure 30 : exemple évolution des forces et moments mesurés sur les 6 axes du capteur

Toutefois, lors du montage du capteur sur le bras robot, une composante supplémentaire a été ajoutée sous la forme d'une pièce permettant de la fixer à la structure anatomique (comme illustré dans la figure 31). L'impact du poids de cette pièce, ajouté au capteur, doit être compensé dans les mesures enregistrées pendant le mouvement du robot. Cette correction est nécessaire pour obtenir les mesures d'efforts réellement appliqués à la bride du robot.

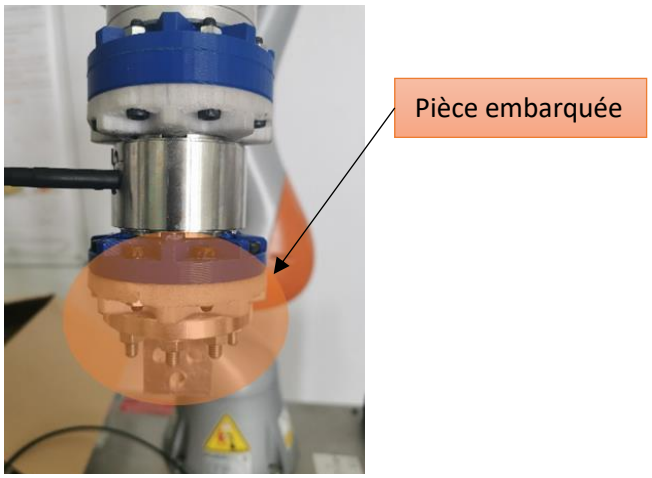


Figure 31 : Montage du capteur sur le bras robot

- **Soustraction de l'effet de la pièce embarquée**

Afin de compenser l'effet du poids de la pièce embarquée dans les mesures du capteur, j'ai créé une fonction (Annexe D) qui calcul l'expression des forces et moments de la pièce dans le repère tourné du capteur, cette fonction prend comme entré les trois angles de rotation successive X, Y, Z du repère flange (i.e. capteur) par rapport au repère base du robot ainsi un vecteur de position de ce repère du flange par rapport au repère base du robot. Par la suite, j'ai effectué la soustraction de ces valeurs calculées à partir des mesures du capteur, permettant ainsi d'obtenir des mesures corrigées et dépourvues de l'effet du poids de la pièce embarquée.

Le script principale « *main()* » interagit à la fois avec le capteur et le robot en même temps, et utilise également des fonctions de la Toolbox KST qui m'a permis d'obtenir les angles de rotation et les positions à chaque instant pendant le mouvement du robot.

Voici un exemple de test que j'ai élaboré pour valider le script. Dans cet exemple, j'applique une rotation de -60 degrés autour de l'axe X au robot. Cette rotation démarre à partir de la position indiquée dans la figure 32. Pendant ce mouvement, j'enregistre l'évolution des forces et les moments. J'effectue ces enregistrements à la fois sans soustraction de l'effet de la pièce et avec soustraction de cet effet.

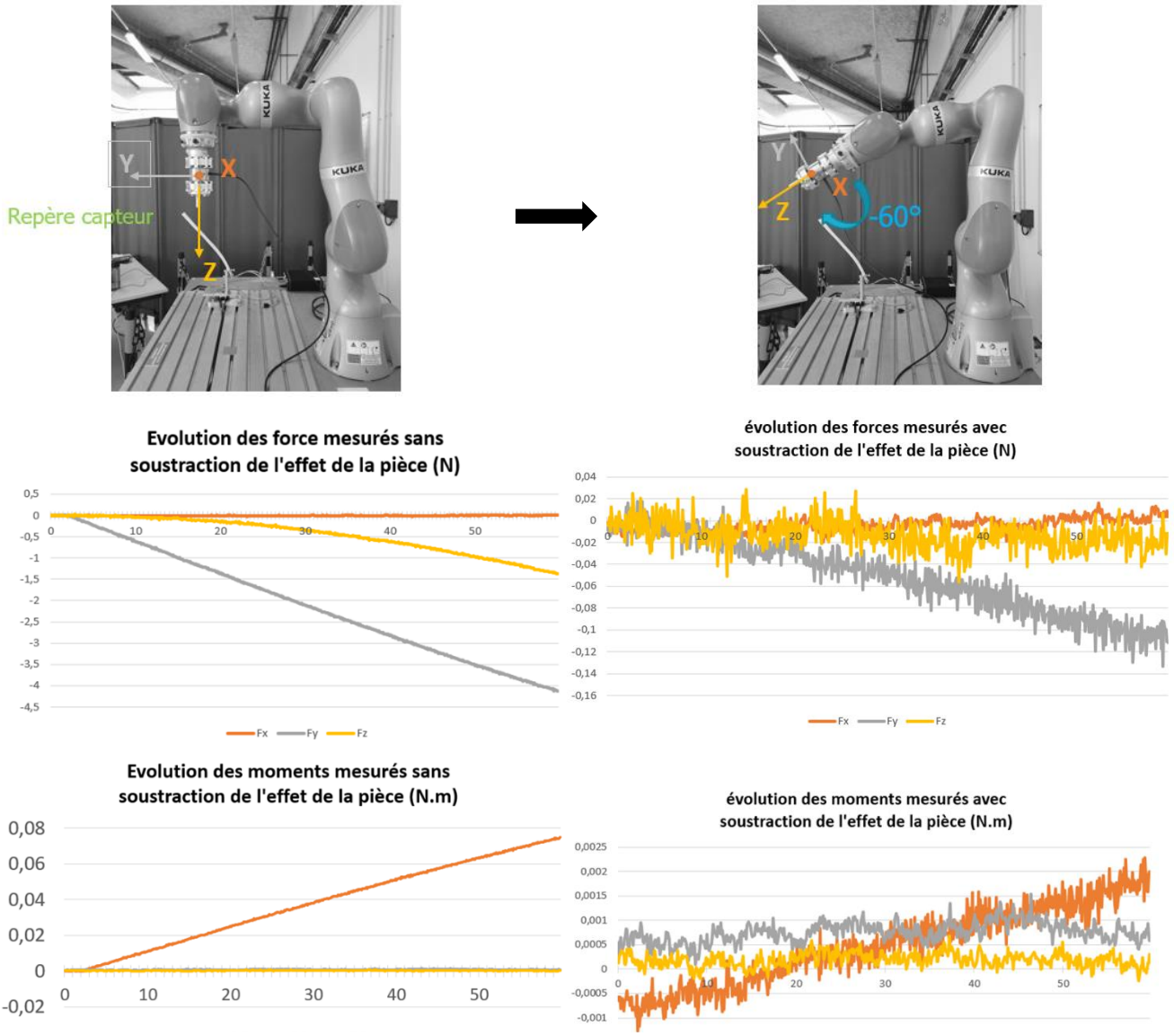


Figure 32 : exemple test de validation du script soustraction de la pièce embarqué

Résultat observé : Lorsque nous comparons les mesures effectuées, il est clair qu'il existe une différence significative entre les valeurs de forces et de moments enregistrées, entre les deux cas de mesures : avec et sans la soustraction de l'effet de la pièce embarquée. Les mesures de force et de moment diminuent considérablement après l'application de l'algorithme de soustraction. Ces valeurs oscillent près de zéro.

4.1.3 Discussion

L'acquisition en temps réel des mesures provenant du capteur constituait l'une des étapes préliminaires essentielles de mon stage. L'implémentation du script d'acquisition a exigé une maîtrise approfondie d'un ensemble considérable de fonctions fournies par la bibliothèque (MEGSV86x64.dll et MEGSV86x64.h), ainsi qu'une compréhension du format des données invoquées dans Matlab et de leur exploitation. Cette phase a également nécessité une comparaison minutieuse entre les mesures obtenues à l'aide de mes scripts et celles fournies par le logiciel d'acquisition "GSVmultichannel" lors de différents tests. Pour valider mes scripts, j'ai effectué plusieurs essais de mouvements et, en ce qui concerne la soustraction de l'effet de la pièce embarquée, j'ai mené des essais répétés. Malgré des oscillations des mesures autour de zéro qui ne sont pas strictement nulles, je suis globalement satisfait de la précision obtenue à cette étape. Cependant, j'ai identifié certaines sources potentielles d'imprécision, notamment une estimation approximative de la force due au poids de la pièce dans mon script et des bruits de mesure. J'ai également découvert des problèmes de dérive lors de nombreux tests d'enregistrement des mesures dans une position statique du robot [Annexe O]. Dans ces enregistrements, j'ai observé une amplification des mesures au fil du temps, sans possibilité de correction immédiate. En tout cas, ces résultats positifs constituent une base sur laquelle je pouvais m'appuyer pour progresser dans les étapes suivantes. Notamment pour entamer l'implémentation de la partie simulation et commande du robot conformément à mes objectifs.

4.2 Logiciel de simulation « CoppeliaSim »

4.2.1 Description du matériel et logiciel utilisé

"CoppeliaSim" est un logiciel de simulation largement utilisé dans le domaine de la robotique et de l'automatisation. Il offre un environnement virtuel puissant pour modéliser et simuler divers scénarios impliquant des robots et d'autres systèmes automatisés. Avec "CoppeliaSim", il est possible de créer des environnements virtuels réalistes en ajoutant des robots, des objets et en définissant des comportements interactifs. La simulation permet de réaliser des tests et des validations sans mettre en danger le matériel réel. L'une des caractéristiques clés de ce logiciel est sa capacité à visualiser en temps réel les résultats de la simulation, ce qui permet d'analyser les trajectoires, les forces et les interactions pour obtenir des informations précieuses sur le comportement des robots. "CoppeliaSim" peut également être intégré avec d'autres logiciels, tels que Matlab/Simulink, pour le développement et la validation d'algorithmes de contrôle. Dans le cadre de notre projet, "CoppeliaSim" a été utilisé pour simuler le comportement du robot KUKA LBR iiwa 14 et pour tester nos approches de contrôle en admittance, ce qui a grandement contribué à la réussite de notre étude.

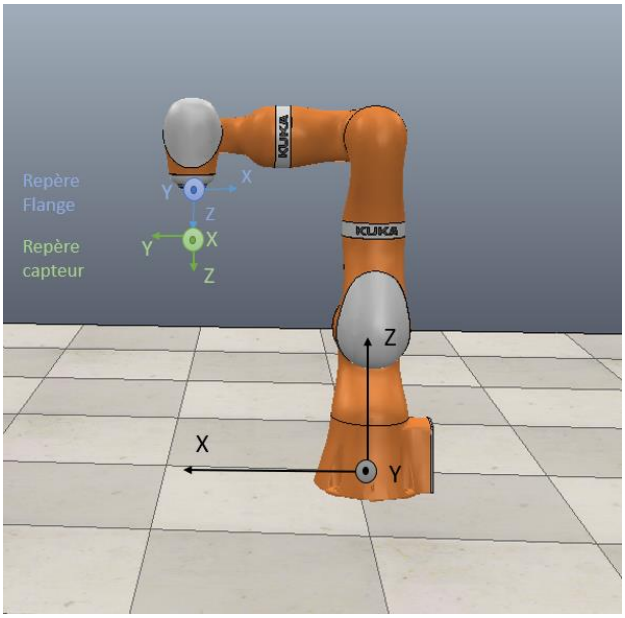


Figure 33 : illustration de modèle du robot kuka lbr iiwa 14 sous CoppeliaSim

4.2.2 Travail réalisé et résultat

Avant d'entrer dans la mise en œuvre des scripts de commande du robot, j'ai d'abord pris le temps de me familiariser avec l'utilisation du logiciel de simulation "CoppeliaSim" (Tâche 4). Ce logiciel offre la possibilité de créer des scènes contenant un ou plusieurs objets, et de les contrôler de manière individuelle en utilisant des scripts intégrés, ou à distance à l'aide de scripts dans d'autres langages tels que C/C++, Matlab, Octave, Python, Java et Lua. Étant donné mon intérêt à mettre en place une boucle de commande via Matlab/Simulink, le fonctionnement est comparable à celui d'une API distante conventionnelle. En d'autres termes, Matlab envoie une requête au serveur CoppeliaSim, décrivant l'action à effectuer sur la simulation ou le robot. Cette requête peut contenir des informations sur les mouvements, les positions, ou d'autres paramètres de contrôle. Ensuite, le serveur CoppeliaSim traite la requête et renvoie une réponse à Matlab, informant ainsi le client de l'état de la simulation ou du robot après l'action entreprise (Figure 34)

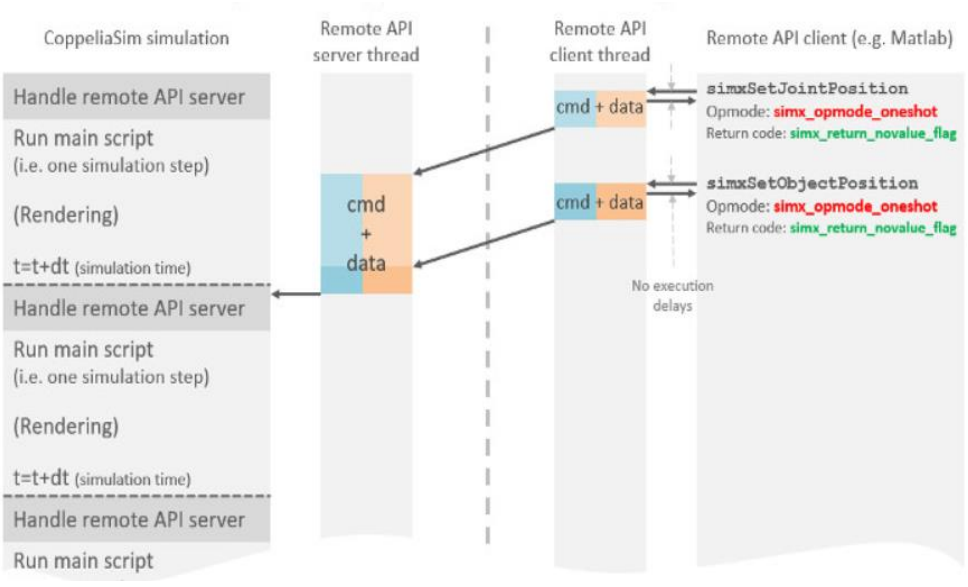


Figure 34 : Structure de l'API distante de CoppeliaSim

Cette interaction fluide et bidirectionnelle permet à Matlab de contrôler et de surveiller en temps réel les éléments de la simulation dans CoppeliaSim, ou même d'intégrer ces interactions dans des boucles de commande plus complexes au sein de l'environnement Matlab/Simulink.

Une fois que j'ai saisi le fonctionnement en m'appuyant sur des tutoriels et des explications approfondies disponibles sur le site du logiciel *: (https://www.coppeliarobotics.com/),* j'ai entamé la phase de mise en œuvre des scripts visant à accomplir à la fois la cinématique directe et la cinématique indirecte.

- **Script de cinématique directe**

Le principe fondamental de mon script de cinématique directe du robot [Annexe E] se résume à la séquence d'actions suivante : en premier lieu, j'établis la connexion avec « CoppeliaSim » et j'extrais l'identification de chaque articulation en m'appuyant sur le modèle géométrique du robot dans CoppeliaSim. Par la suite, je confère à l'utilisateur la capacité de sélectionner une vitesse et une position articulaire souhaitée, exprimée sous la forme d'un vecteur de dimension (7,1). Chaque composante de ce vecteur correspond à une position articulaire en radians pour chaque axe du robot. Une fois cette sélection effectuée, j'utilise une fonction spécifique d'envoi de position, où la position choisie est transmise en tant que paramètre. Pour simplifier, le processus de mon script peut être schématisé comme suit :

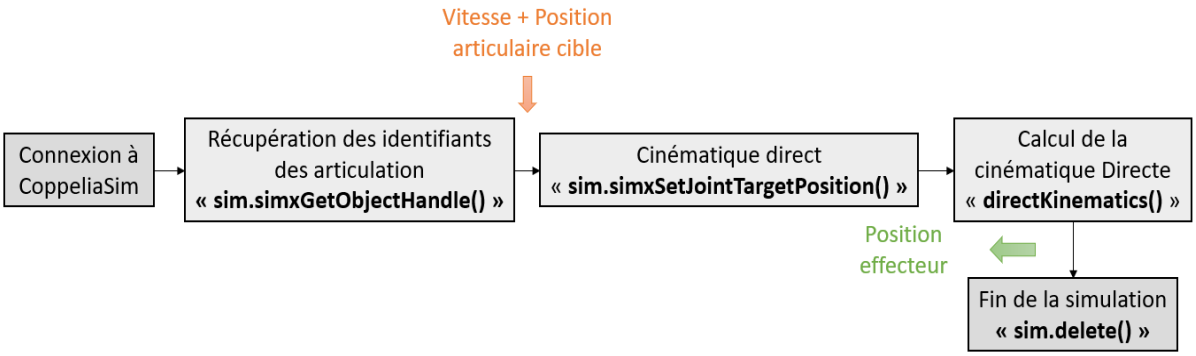
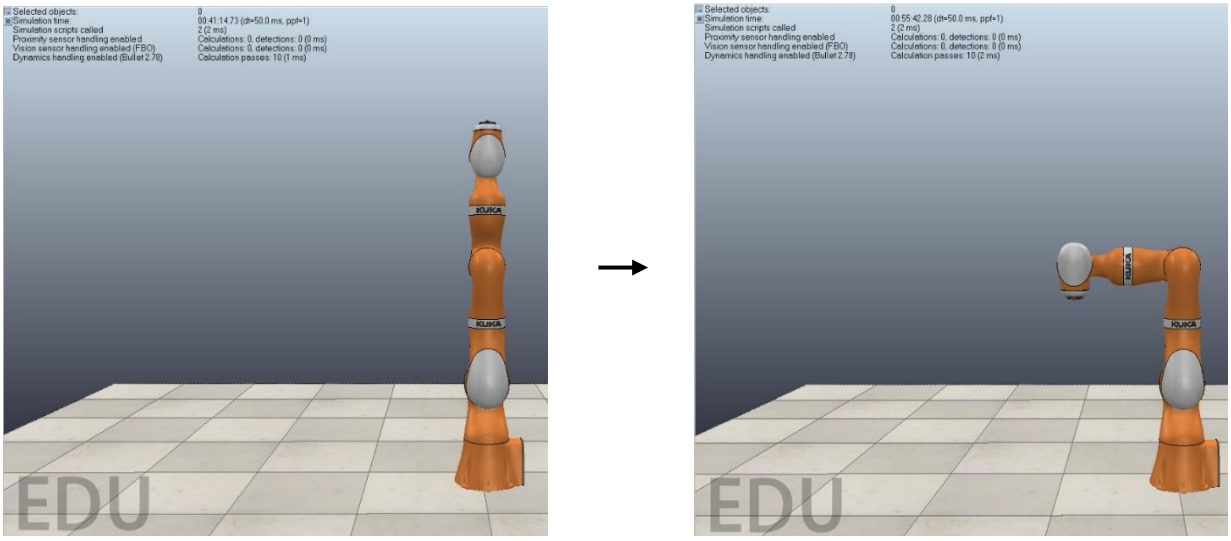


Figure 35 : Algorithme de cinématique directe sous CoppeliaSim

Voici un exemple qui illustre un test d'une cinématique directe :



`jPos= [0.0; 0.0; 0.0; 0.0; 0.0; 0.0; 0.0]`

`jPos= [0.0; 0.0; 0.0; -pi/2; 0.0; pi/2; 0.0]`

Figure 36 : exemple Sous CoppeliaSim d'une cinématique directe

- **Script de cinématique inverse**

Le principe fondamental de mon script de cinématique inverse du robot [Annexe F] peut être décrit par la séquence d'actions qui suit : tout d'abord, j'établis une connexion avec « CoppeliaSim » et j'obtiens l'identification de chaque articulation. Par la suite, je confère à l'utilisateur la capacité de choisir une vitesse articulaire ainsi qu'une position cartésienne souhaitée de l'effecteur dans l'espace de travail du robot. Cette position est définie sous forme d'un vecteur tridimensionnel (1,3), et une orientation est représentée par un autre vecteur (1,3) en relation avec le repère de base du robot. Ensuite, j'utilise la fonction « inverseKinematics() » de la Toolbox KST. Cette fonction sert à trouver les bonnes positions des articulations. Elle utilise deux méthodes d'optimisation itératives basées sur le gradient : l'algorithme de projection BFGS et l'algorithme de Levenberg-Marquardt. Ces deux méthodes partent d'une estimation initiale de la solution et cherchent à minimiser une certaine fonction de coût. Si l'une de ces méthodes converge vers une configuration où le coût est presque nul dans une tolérance spécifiée, alors elle a trouvé une solution au problème de cinématique inverse. Pour simplifier, le schéma suivant illustre les différentes étapes du processus de mon code :

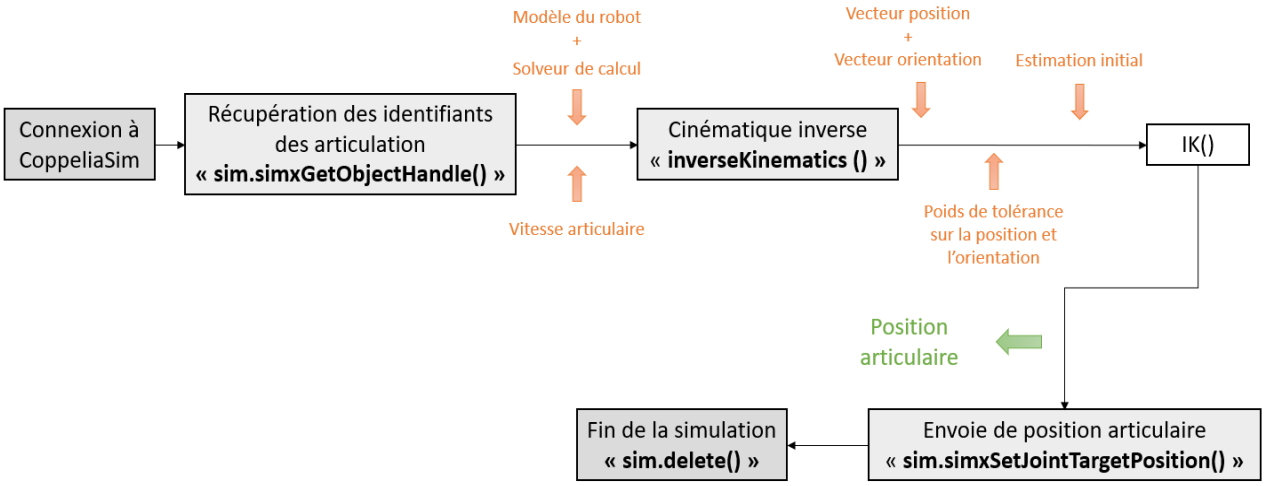
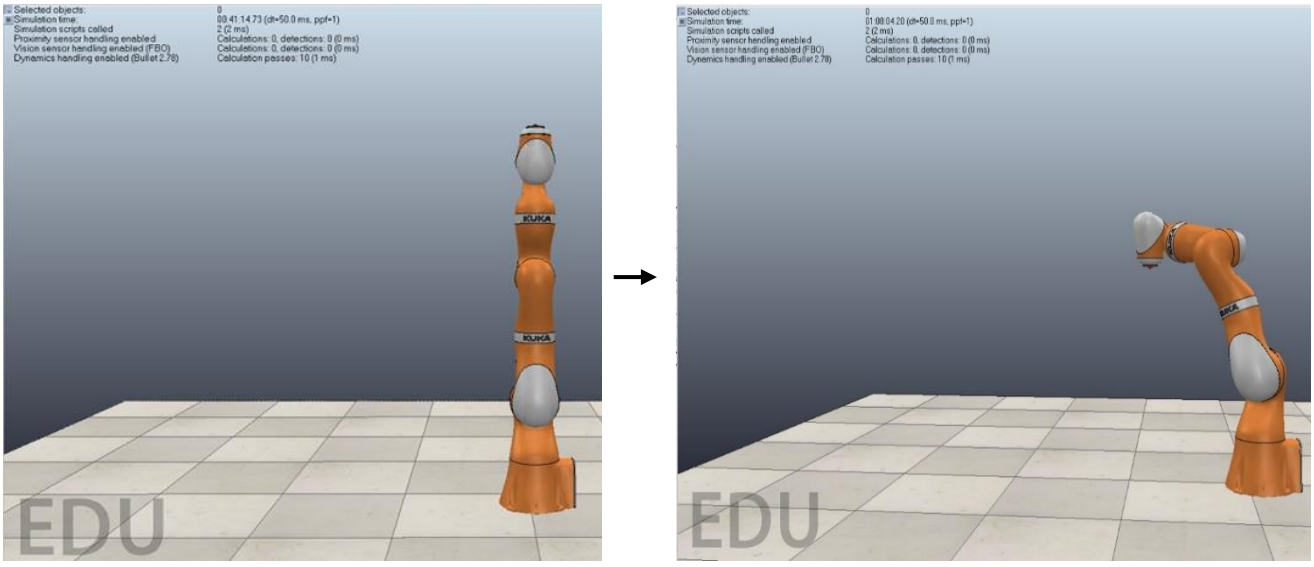


Figure 37 : Algorithme de cinématique inverse sous CoppeliaSim

Voici un exemple qui illustre un test d'une cinématique inverse :




```
Position_effecteur = [0.0, 0.0, 0.0]
Orientation_effecteur = [0.0, 0.0, 0.0]
```

```
Position_effecteur = [0.4 -0.36 0.62]
Orientation_effecteur = [-pi, 0.0, pi]
```

Figure 38 : exemple sous CoppeliaSim d'une cinématique inverse

- **Script suivi de trajectoire**

Afin d'établir le suivi d'une trajectoire pour le robot simulé, le script [Annexe G] présente des variations au niveau des entrées et sorties. Dans cette version, j'accorde à l'utilisateur la possibilité de déterminer la vitesse articulaire souhaitée ainsi que les positions cartésiennes de l'effecteur. Les positions cartésiennes sont représentées sous forme d'une matrice de dimensions (n,6), où chaque ligne correspond à un point dans l'espace. Les trois premières colonnes décrivent la position de l'effecteur, tandis que les trois colonnes suivantes indiquent son orientation. En sortie, le script génère une matrice (n,7) qui contient les positions articulaires obtenues grâce au solveur de cinématique inverse. Le schéma suivant décrit ce processus :

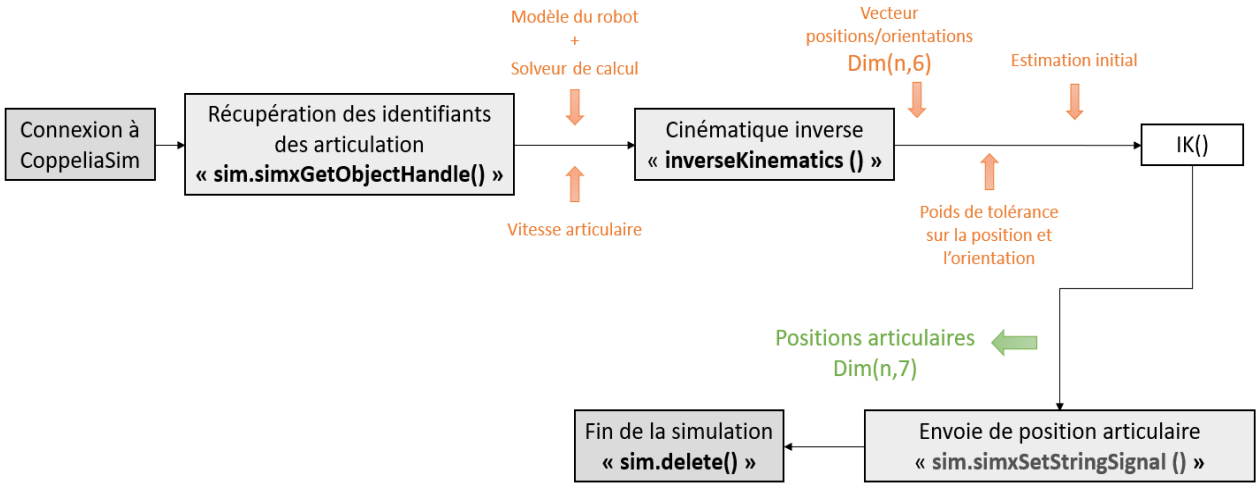


Figure 39 : Algorithme suivi de trajectoire sous CoppeliaSim

L'utilisation de la fonction **simxSetStringSignal()** me permet d'envoyer les positions articulaires calculées par le solveur de cinématique inverse à CoppeliaSim. Ces positions articulaires sont encapsulées au sein d'une chaîne de caractères que je transmets comme un signal au simulateur. CoppeliaSim interprète ensuite ce signal et applique les positions articulaires correspondantes aux articulations du modèle robotique.

Voici un exemple qui illustre un test de suivi de trajectoire (translation sur l'axe X) :

```
cartesian_trajectory =
[0.4 -0.4 0.6 pi  0.0  -pi
 0.4 -0.2 0.6 pi  0.0  -pi
 0.4  0.0 0.6 pi  0.0  -pi
 0.4  0.2 0.6 pi  0.0  -pi
 0.4  0.4 0.6 pi  0.0  -pi]
```

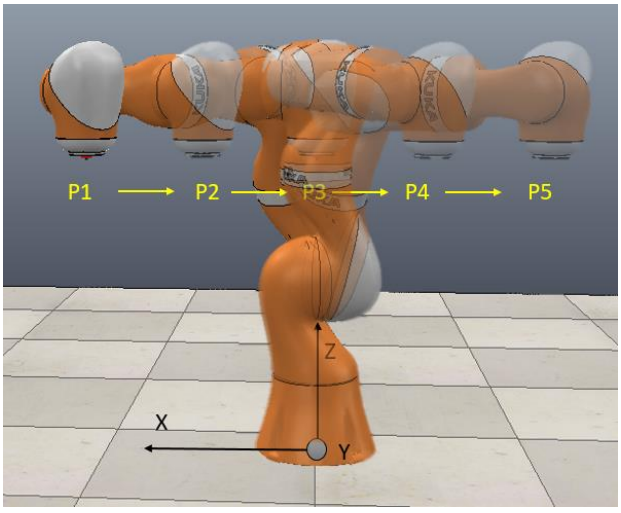


Figure 40 : exemple de suivi de trajectoire (en translation)

4.2.3 Discussion

L'utilisation du logiciel CoppeliaSim a marqué une étape essentielle dans mon avancement, puisqu'elle m'a permis d'établir une base solide et une architecture initiale pour mes scripts. Cela m'a facilité la mise en œuvre de la cinématique directe, indirecte et du suivi de trajectoire sur un modèle de robot KUKA LBR iiwa 14. J'ai ainsi pu exercer un contrôle précis sur la vitesse des mouvements. Cependant, il est important de noter que mon utilisation de CoppeliaSim se limitait principalement à la visualisation des résultats obtenus à partir de mes calculs de cinématique. En effet, les calculs étaient basés sur les modèles de dynamique du robot fournis dans Matlab. Ce que j'envoyais à CoppeliaSim était donc essentiellement le résultat de mes calculs.

Après avoir acquis une compréhension approfondie du fonctionnement de l'outil de cinématique virtuelle sur le robot simulé sous CoppeliaSim, ainsi que des entrées et sorties nécessaires pour établir des cinématiques directes ou inverses, j'ai pu utiliser ces connaissances pour progresser dans l'implémentation de scripts de contrôle de cinématique directe et inverse sur le robot réel. La familiarité avec les fonctions de calcul et les solveurs de la Toolbox KST a joué un rôle crucial dans cette transition, en me permettant de transférer les principes et les méthodes de simulation vers des applications pratiques sur le robot réel.

4.3 Commande du robot réel

4.3.1 Description du matériel et logiciel utilisé

La KST : Kuka Sunrise Toolbox représente un ensemble d'outils destinés à faciliter le contrôle des robots KUKA iiwa 7/14 R 800/820 à partir d'un ordinateur externe en utilisant Matlab. Cette boîte à outils est accessible via un dépôt GitHub, mis à disposition sous une licence open source par son créateur [Safeea (2017)]. Le dépôt inclut divers éléments tels que les fonctions Matlab dédiées à la manipulation des robots Kuka, un ensemble de codes Java à importer dans Sunrise Workbench pour établir la connexion avec le manipulateur, ainsi que des tutoriels et un guide d'utilisation. Toutefois, lors de ma tentative d'utilisation de la KST, j'ai rencontré des problèmes de communication avec le robot. Après avoir discuté de la situation avec l'équipe de Genève, avec laquelle nous collaborons, il est apparu qu'ils avaient développé une version modifiée de la boîte à outils KST afin de résoudre ces problèmes et permettre une communication fluide avec le robot, ainsi que l'envoi de commandes efficaces. Par conséquent, nous avons utilisé leur boîte à outils pour mener des essais de cinématique directe et inverse sur le robot. Nous avons exploité les diverses fonctions fournies par cette boîte pour effectuer ces tests et évaluer la cinématique du robot.

Figure 41 illustre l'architecture et le schéma de communication de la KST. La KST s'exécute sur un ordinateur externe/à distance et communique avec KUKA Sunrise via TCP/IP à travers un réseau Ethernet en utilisant le connecteur X66 du robot. La KST met en œuvre un client TCP/IP qui communique avec le serveur Java (KST Server et KST Main) exécuté dans le Sunrise.OS. Le KST Server et le KST Main sont fournis avec la boîte à outils KST.

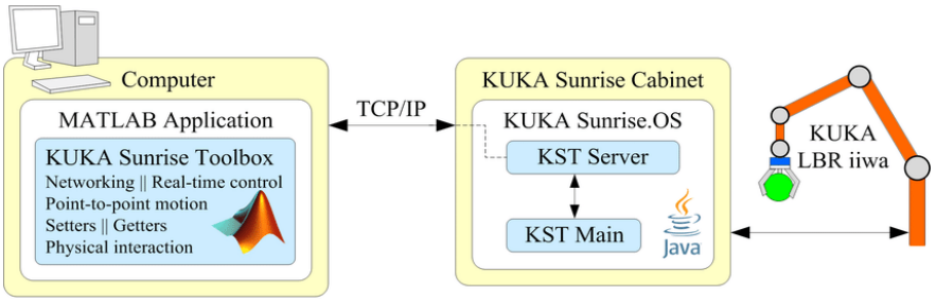


Figure 41 : Architecture et schéma de communication de la KST.

4.3.2 Travail réalisé et résultat

Avant d'entreprendre l'élaboration des scripts de cinématique directe et inverse sur le robot, il était impératif de mener une recherche approfondie au sein de diverses ressources, notamment le *[KSTUsersGuide]* fourni par KUKA. Mon premier objectif était d'établir une connexion solide entre Matlab et le robot (Tâche 5). À cet effet, j'ai procédé à une exploration minutieuse des informations bibliographiques disponibles.

La mise en place d'une connexion réseau entre le robot et le PC implique plusieurs étapes, que j'ai suivies de la manière suivante :

- Utilisation d'un câble Ethernet pour connecter le port X66 du robot au port Ethernet du PC.
- Ensuite, sur le PC externe, je change l'adresse IP du PC en une adresse statique dans la plage d'adresses IP du robot.
- Puis Synchronisation du KST server (MatlabToolboxServer) dans le contrôleur du robot qui est une application codée en Java dans l'environnement Kuka Sunrise Workbench.

Par la suite, pour établir la connexion entre le client Matlab du PC et le serveur, il est essentiel de mettre en œuvre les lignes comme démontré dans la figure (42), et de lancer ce script une fois que le serveur (MatlabToolboxServer) a déjà été démarré :

```
%% Créer un objet robot
ip='172.31.1.147'; % IP du controlleur
arg1=KST.LBR14R820; % choix du modèle iiwa14R820
arg2=KST.Medien_Flansch_Touch_pneumatisch; % choix du type du flange
Tef_flange=eye(4); % matrice de transformation de l'eefecteur au flange
iiwa=KST(ip,arg1,arg2,Tef_flange); % creation de l'objet


%% Connexion avec le serveur
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);
```

Figure 42 : code pour établir connexion IP avec le robot kuka r820

La boite à outils est fournie avec des exemples de tutoriels (démo) rédigés en langage de script Matlab, les exemples varient en termes de difficulté : certains sont simple et présentent une implémentation directe des fonctions de mouvement, tandis que d'autres sont plus complexes.

La mise en place d'une connexion entre Matlab et le robot ne s'est pas avérée aussi rapide et simple que démontré précédemment. En effet, j'ai rencontré plusieurs problèmes et bugs informatiques lors de mes tentatives de lancement des exemples d'application de la boîte à outils KST. Cependant, grâce

à l'assistance de mon tuteur et de l'équipe de recherche de Genève, nous avons pu identifier et résoudre les diverses erreurs, ce qui m'a finalement permis d'établir une connexion efficace. En fin de compte, j'ai pu progresser dans la mise en œuvre des scripts de cinématique directe et inverse sur le véritable robot.

- **Script de la cinématique directe**

Le script de cinématique directe appliqué au robot réel [Annexe H] suit une approche similaire à celle utilisée pour le robot simulé. Les différences se situent principalement au niveau des fonctions employées pour établir la connexion et envoyer les commandes. Dans ce contexte, j'ai recours aux fonctions fournies par la Toolbox KST pour interagir avec le robot. L'organigramme suivant expose de manière détaillée les étapes du script de cinématique directe :

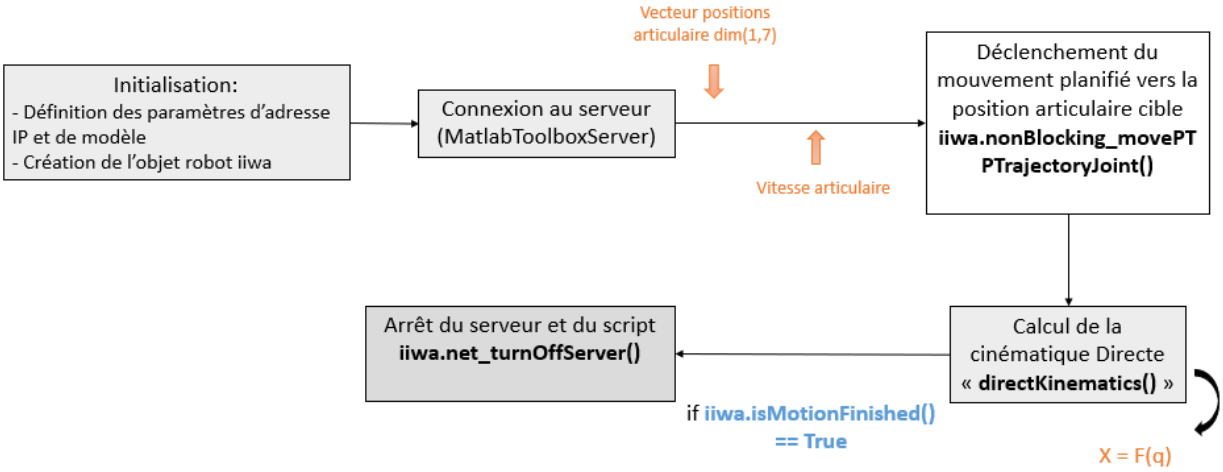


Figure 43 : Algorithme de cinématique directe du robot

- **Script de la cinématique inverse :**

De manière similaire à l'approche décrite pour la simulation du robot, le principe de la cinématique inverse sur le robot réel [Annexe I] suit les étapes énoncées dans l'organigramme suivant :

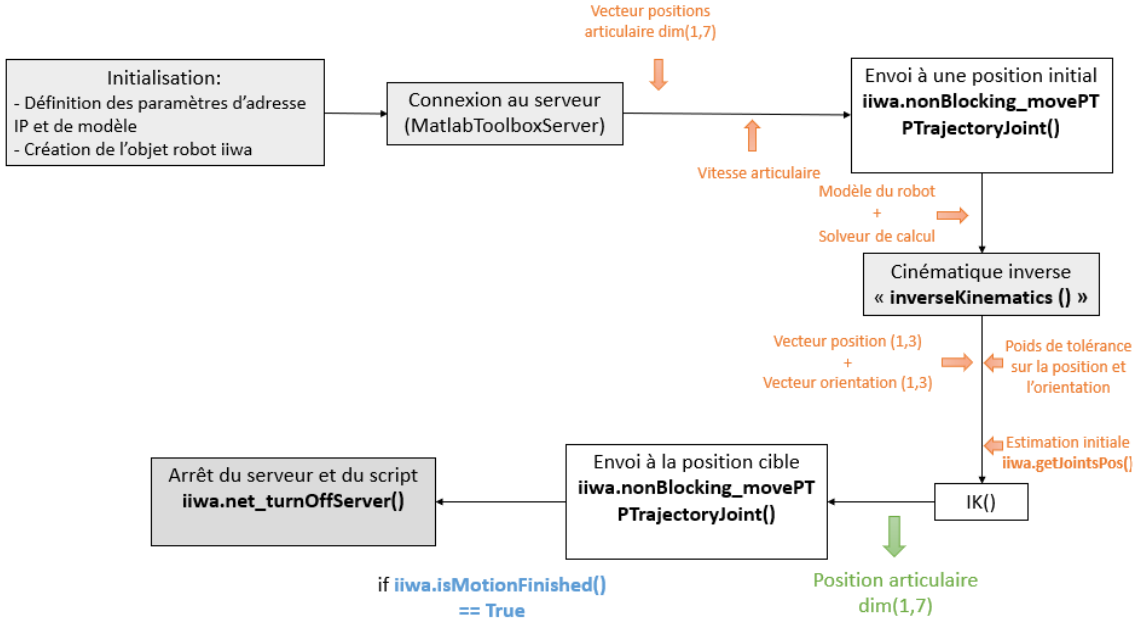


Figure 44 : Algorithme de cinématique inverse du robot

Afin d'évaluer l'efficacité de mon script implémenté et le calcul de la cinématique inverse à l'aide des fonctions de la Toolbox KST, ainsi que les modifications apportées au modèle dynamique du robot fourni par Matlab, j'ai mis en œuvre une méthode pour vérifier si le robot atteint correctement la position cible souhaitée, définie par des vecteurs de position et d'orientation établis par l'utilisateur.

Pour ce faire, j'ai suivi les étapes suivantes :

1 - J'ai programmé le robot pour se déplacer vers une position cible spécifiée en utilisant le script de cinématique inverse.

2 - J'ai utilisé les fonctionnalités du smart-pad pour vérifier si le robot se positionne correctement dans la position et l'orientation désirées.

Cette approche m'a permis de vérifier visuellement et manuellement si le robot atteint la position cible correcte en fonction des vecteurs de position et d'orientation que j'ai fourni.

Voici un exemple de cinématique inverse où le flange de mon robot est utilisé comme effecteur (TCP). J'ai inclus les étapes suivantes dans mon script :

```
Translation = [0.4 0.0 0.6]
Rotation = [-pi, 0.0, -pi]
```

1 - J'ai défini une position et une orientation :

2 - J'ai exécuté mon script.

3 - J'ai vérifié l'affichage du positionnement et de l'orientation du flange sur le smart-pad du robot.

Finalement, j'ai constaté que la position et l'orientation affichées sur le smart-pad du robot correspondent exactement à celles que j'avais définies dans le script. Cela se présente de la manière suivante dans l'illustration :

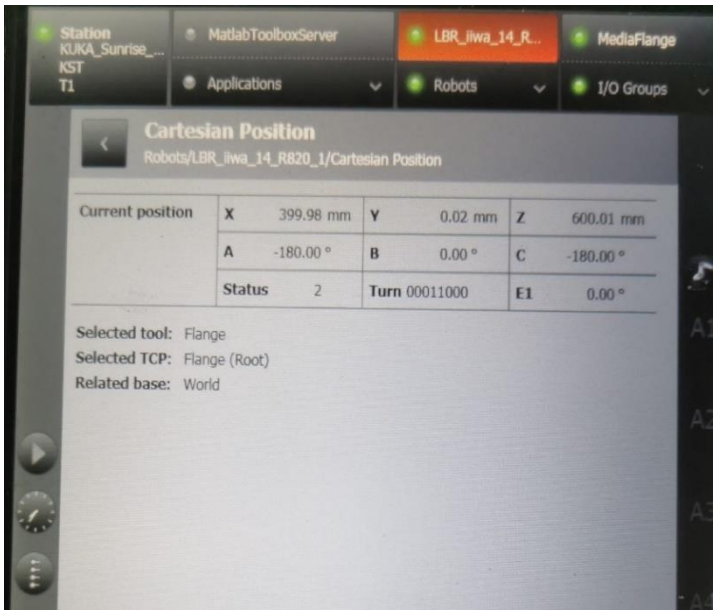


Figure 45 : Affichage de position cartésienne sur Smart-Pad

De plus, dans mon code, j'ai inclus une fonctionnalité permettant à l'utilisateur de sélectionner un autre point d'effecteur. Cette modification a pour effet de calculer la cinématique du robot au niveau du point d'outils (TCP). Pour mettre cela en œuvre, j'ai ajusté la matrice de transformation homogène "Tef_flange" au début du script. Cette matrice représente la transformation du flange vers le point d'effecteur choisi par l'utilisateur. Cette flexibilité offre à l'utilisateur la possibilité de calculer la cinématique du robot en fonction de différents points d'outils, offrant ainsi une adaptabilité aux besoins spécifiques de la tâche ou de l'application.

- **Suivi de trajectoire**

Après avoir défini la trajectoire sous forme d'une matrice de dimension (n,6), avec les trois premières colonnes décrit les positions désirées et 3 dernières colonnes les orientations souhaitées, le processus de suivi de trajectoire entre en action. Chaque position cartésienne répertoriée dans la matrice est traitée séquentiellement. Pour chaque position, le robot est programmé de manière à effectuer des mouvements fluides et précis, en utilisant la cinématique inverse pour déterminer les positions articulaires requises. La gestion de la vitesse de déplacement entre les différentes positions est également mise en place [Annexe J].

Pendant l'exécution du suivi de trajectoire, j'envoie des commandes au robot réel pour qu'il suive les positions spécifiées dans la matrice. À chaque étape, le robot s'ajuste et s'oriente en conséquence pour suivre la trajectoire souhaitée. Les informations de position et d'orientation affichées sur le smart-pad du robot permettent une vérification visuelle en temps réel de l'alignement avec la trajectoire cible.

En résumé, après avoir évalué les fonctionnalités de la cinématique du robot dans un environnement virtuel, et pouvoir contrôler le robot réel en utilisant la Toolbox KST, cela m'a permis de franchir une étape importante vers l'implémentation de la boucle de commande en admittance.

- **Boucle de contrôle en admittance**

Dans cette section, je vais expliquer en détail le concept de la boucle de commande en admittance que j'ai mis en œuvre (tâche 6). Pour cette implémentation, j'ai opté pour la commande en admittance dans l'espace opérationnel, comme décrit dans la section 3.5.2. Cette décision a été prise afin de pouvoir contrôler les forces appliquées à l'effecteur dans l'espace des tâches, plutôt qu'au niveau des articulations du robot.

Le schéma illustré ci-dessous présente la structure de la boucle de commande mise en place dans l'environnement Simulink :

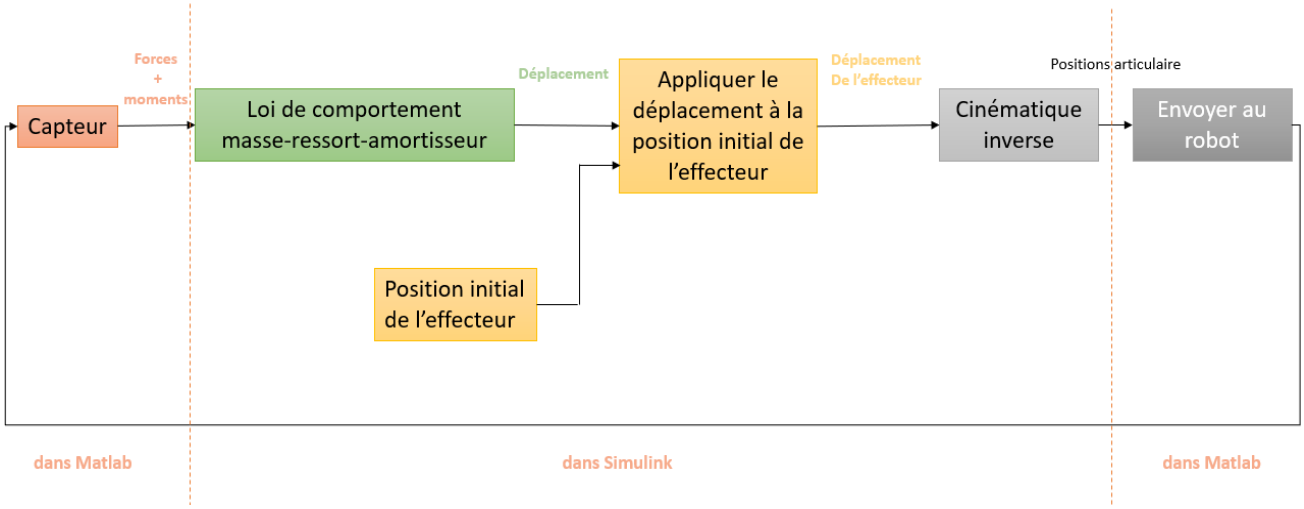


Figure 46 : Boucle de contrôle en admittance implémentée

1- Bloc Capteur

Dans ce bloc, je collecte les valeurs mesurées des forces et des moments à partir du script Matlab, en utilisant la même séquence d'acquisition présentée dans la section 4.3.1. Après avoir déterminé la fréquence et la durée de l'acquisition des mesures (qui doivent correspondre à celles de la simulation), je transfère finalement les valeurs de mesure dans Matlab vers le bloc capteur dans Simulink en utilisant la fonction assignin(). La sortie de ce bloc est un vecteur de dimension (6,1), ou chaque ligne corresponds à la force mesurée selon l'axe X, Y, et Z du capteur, et le moment selon l'axe X, Y, Z respectivement.

2- Bloc loi de comportement masse-ressort-amortisseur

La représentation visuelle de ce bloc est illustrée par le schéma ci-dessous :

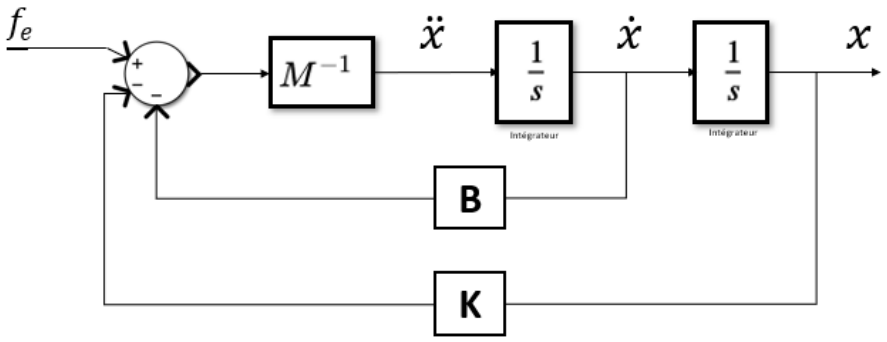


Figure 47 : illustration de loi de comportement masse-ressort-amortisseur sous Simulink

Le schéma en Figure 47 représente un schéma bloc détaillant une loi de commande de nature plus générique. Cette loi de comportement est formulée comme une équation de masse-ressort-amortisseur appliquée à l'effecteur du robot dans la suite de la boucle de contrôle. Dans cette expression, "M" représente la matrice masse, "B" est la matrice coefficients d'amortissements et "K" représente la matrice de raideur, ces trois matrices sont de dimension (6,6). L'équation s'écrit comme suit :

$$M\,\ddot{x} + B\,\dot{x} + K\,x = f_e$$

Ici, $\ddot{x}$ correspond à la seconde dérivée temporelle du vecteur position désiré, $\dot{x}$ est la première dérivée temporelle de la position désirée, et x représente la position désirée, ces vecteurs sont de dimension (6,1). Le côté droit de l'équation est associé à la force f_e appliquée sous la forme d'un vecteur (6,1). Cette représentation schématique illustre comment les éléments de masse, d'amortissement et de ressort sont combinés pour générer une réponse adaptée en fonction des forces appliquées à l'effecteur. En ajustant les paramètres M, B et K dans cette configuration, il est possible d'influencer la manière dont le système réagit aux entrées de force (i.e., position, vitesse accélération).

3- Bloc calcul de déplacement appliqué sur l'effecteur

Au sein de ce bloc, j'ai mis en place une fonction qui extrait un vecteur contenant la position actuelle ainsi que l'orientation de l'effecteur. Ce vecteur, de dimension (6,1), est ensuite soumis au processus de déplacement calculé dans le bloc précédent, qui est la loi de comportement désirée. À la sortie de ce processus, j'obtiens une nouvelle position et une nouvelle orientation résultant de ce déplacement appliqué. En somme, cette fonction agit comme un maillon essentiel entre la position et l'orientation actuelles de l'effecteur et le processus de déplacement basé sur la loi de comportement.

4- Bloc de cinématique inverse

Après avoir déterminé le déplacement de l'effecteur grâce au bloc précédent, j'entame une étape de cinématique inverse en utilisant le modèle dynamique fourni par Matlab. Cette démarche me permet d'obtenir les positions articulaires correspondantes aux déplacements calculés. Une fois ces positions articulaires obtenues, je les envoie de nouveau vers Matlab. Dans ce cadre, j'utilise les fonctions dédiées pour transmettre les commandes de position articulaire vers le robot, que ce soit le modèle simulé ou le robot réel. Cette séquence d'opérations assure ainsi que les déplacements souhaités de l'effecteur se traduisent en positions articulaires adéquates, permettant ainsi de piloter le robot dans l'espace de travail souhaité.

En résumé, Toutes les étapes précédemment détaillées s'inscrivent dans une boucle de contrôle fermée sophistiquée, visant à assurer une interaction harmonieuse entre le robot et son environnement. À partir du bloc capteur, les mesures de forces et de moments sont collectées en temps réel. Ensuite, ces données sont acheminées vers le bloc loi de comportement masse-ressort-amortisseur, où elles sont traitées selon une équation qui combine les paramètres de masse, d'amortissement et de ressort pour générer des réponses adaptées aux forces appliquées à l'effecteur. Le résultat de cette étape est ensuite utilisé par le bloc calcul de déplacement appliqué à l'effecteur pour définir une nouvelle position et orientation en tenant compte du déplacement calculé. Cette information est ensuite utilisée pour effectuer une cinématique inverse à l'aide du bloc dédié, obtenant ainsi les positions articulaires correspondantes. Enfin, ces positions articulaires sont transmises au robot, qu'il soit simulé ou réel, grâce à une série de commandes. Cette boucle de contrôle en admittance permet au robot de réagir de manière adaptée aux forces et aux mouvements environnants, créant ainsi un système réactif et précis capable de collaborer efficacement avec son environnement.

- **Test et résultat de la boucle d'admittance**

Initialement, j'ai élaboré une boucle d'admittance pour le modèle simulé du robot dans CoppeliaSim, avant de la mettre à l'épreuve sur le robot réel. Concernant les mesures de force, j'ai mis en place une procédure initiale [Annexe K], où j'ai introduit des signaux sinusoïdaux en entrée, définissant leurs paramètres tels que l'amplitude, la fréquence et la phase. J'ai ensuite configuré les matrices de masse, de ressort et d'amortissement en fonction des déplacements souhaités. Dans un second temps, j'ai établi la connexion entre les mesures réelles du capteur et le modèle du robot simulé.

Concrètement, j'ai appliqué des forces ou des moments directs sur le capteur en utilisant ma main, tout en surveillant les réactions correspondantes sur le modèle du robot simulé dans l'environnement CoppeliaSim. Cette phase m'a permis de valider la boucle d'admittance en confirmant la cohérence entre les mesures réelles et les réponses simulées du robot [Annexe L].

Voici un exemple de test illustrant le fonctionnement de la boucle en admittance appliquée au robot simulé :

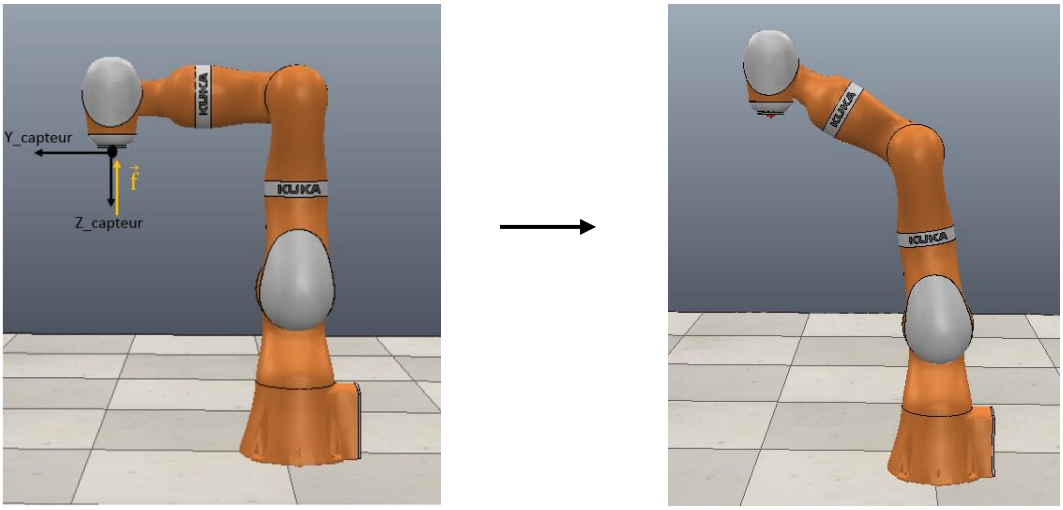


Figure 48 : exemple d'application de la boucle d'admittance sur le robot simulé

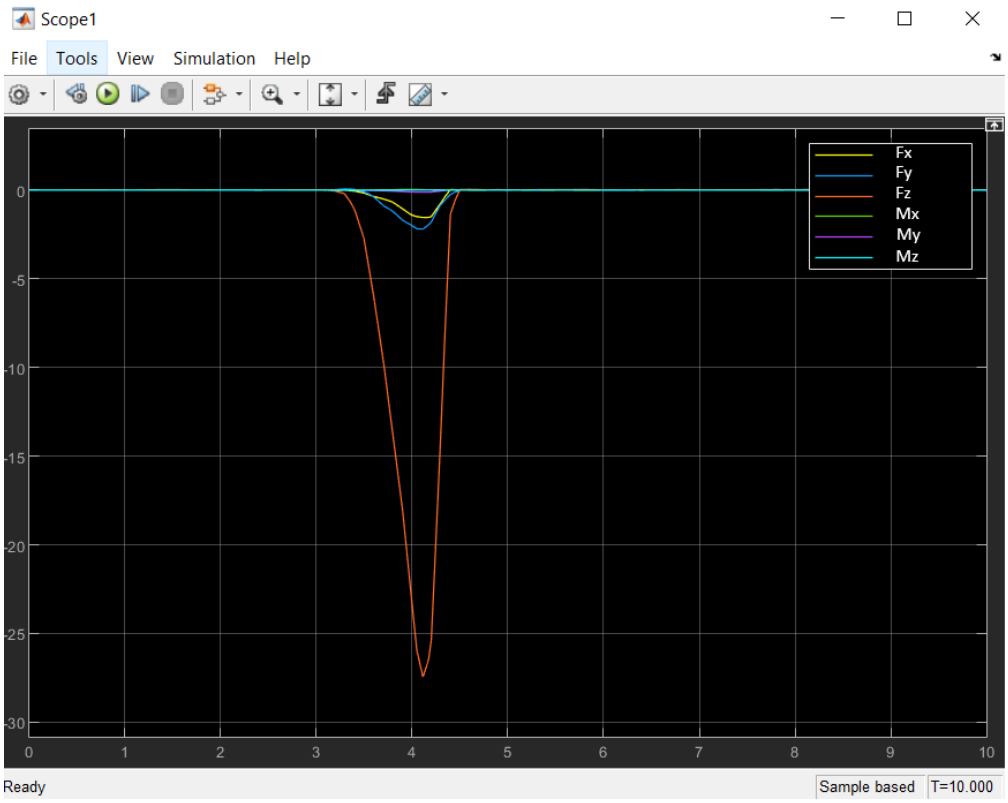


Figure 49 : évolution des forces (N) et moments (N.m) au cours du temps (10 sec d'acquisition)

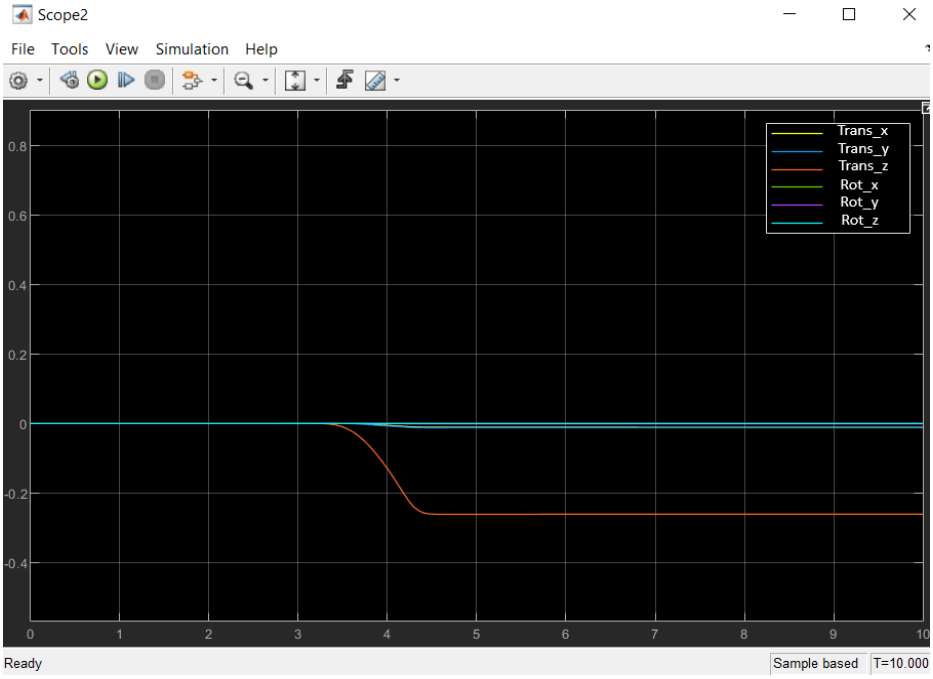


Figure 50 : déplacement de l'effecteur (m) après simulation de la boucle en admittance

Comme illustré par cet exemple, j'ai exercé une force manuelle sur la bride du robot réel dans la direction de l'axe (-Z) du capteur (Figure 49 et Figure 50). En réponse, l'effecteur du robot s'est déplacé vers le haut, suivant la direction de la poussée. Il est important de noter que mon application n'était pas strictement verticale, ce qui explique pourquoi le capteur a enregistré des efforts dans d'autres directions. Cependant, ces efforts sont relativement faibles par rapport à la direction de la force appliquée. Les valeurs de déplacement et de vitesse de l'effecteur, en réponse aux forces appliquées, dépendent directement des paramètres que nous définissons pour les matrices de masse, raideur et amortissement dans le bloc de la loi de comportement de type masse-ressort-amortisseur. Ces paramètres jouent un rôle fondamental dans la manière dont l'effecteur réagit aux forces extérieures et influence la dynamique de son mouvement.

Les résultats encourageants obtenus lors de l'application de la boucle d'admittance sur le robot simulé m'ont permis de passer à la phase d'implémentation des scripts nécessaires pour appliquer cette boucle au robot réel.

La première étape de la démarche que j'ai suivie pour valider mon script était la suivante : tout d'abord, j'ai lancé le script permettant une communication simultanée avec le capteur et le robot. Après avoir défini une fréquence d'acquisition et une durée de mesure (par exemple 10 secondes), j'ai exercé une pression manuelle sur la bride du robot dans une direction choisie. À la fin de cette période d'acquisition des mesures du capteur, l'effecteur du robot a commencé à se déplacer dans la même direction que celle de la pression manuelle exercée. Pour valider les scripts mis en place, j'ai répété cette procédure de test dans différentes directions, vérifiant ainsi si l'effecteur du robot suivait avec précision la direction de la pression manuelle [Annexe M].

La deuxième étape consiste à choisir une courte période d'acquisition des mesures des capteurs (1 seconde) et j'ai créé un autre script [Annexe N] qui permet de lancer la procédure dans une boucle. Cela signifie que dès que la communication avec le capteur et le robot est établie, je commence à effectuer des mesures de force et de moment chaque seconde. Ensuite, le robot se déplace en fonction des mesures enregistrées pendant cette période. Un exemple de validation de ce test est illustré dans la figure ci-dessous :

(a) Position initiale (b) Position finale

Figure 51 : exemple test de la boucle d'admittance sur le robot réel.

Ici, je réalise un test sur une poutre en téflon attachée à la bride, qui constitue un substitut intéressant du rachis (la poutre présente des raideurs différentes dans trois plans de flexion, induisant une forme de couplage mécanique). En effectuant une manipulation manuelle du robot, j'ai positionné le robot dans cette configuration initiale Figure 51 (a). Dans cette position, la poutre applique des forces et des moments dans différentes directions. Une fois que le script est lancé, la procédure démarre : le capteur mesure les forces et les moments appliqués chaque seconde, et le robot se déplace à chaque incrément pour les annuler jusqu'à atteindre une position neutre Figure 51 (b) où aucune force ni moment n'est appliqué sur la bride du robot.

La boucle d'admittance fonctionne bien dans cette approche incrémentale, mais elle comporte un problème pratique, à savoir le bruit des moteurs du robot à chaque petit déplacement (étape 2). Ce problème est discuté dans la partie suivante.

4.3.3 Discussion

L'accomplissement de l'implémentation des scripts pour la cinématique directe et inverse sur le robot réel a été une étape essentielle pour saisir l'approche de commande en admittance du robot au travers des fonctionnalités de la Toolbox KST. Cette étape m'a permis de comprendre comment récupérer les informations du robot, envoyer des commandes et le contrôler de manière précise. Cependant, l'utilisation du modèle dynamique pour les calculs de cinématique directe et inverse dans Matlab a nécessité quelques ajustements pour correspondre au robot réel. Ces ajustements comprenaient des aspects tels que la taille du bride et l'adaptation de la position et de l'orientation de l'effecteur dans le modèle dynamique pour qu'ils correspondent au repère dans lequel les mesures étaient effectuées. De plus, j'ai intégré des fonctions qui permettaient à l'utilisateur de personnaliser la taille de l'outil utilisé, ce qui a nécessité des modifications sur le modèle du robot fourni par Matlab afin de l'adapter au robot réel.

Mon stage a atteint ce point spécifique où j'ai pu voir les résultats de la boucle d'admittance mise en œuvre. Pour parvenir à ce résultat, il était nécessaire d'établir une communication avec le robot et le capteur en même temps. La procédure impliquait de choisir un intervalle d'acquisition des mesures, de lancer le script, et ensuite d'exercer une force sur la bride du robot. Lorsque l'acquisition des mesures était complète, l'effecteur du robot commençait à se déplacer dans la direction de la poussée exercée. Ces résultats ont été positifs et ont confirmé le succès du contrôle en admittance sur le robot réel.

Cependant, l'objectif principal consistait à parvenir à une synchronisation précise entre l'acquisition des mesures et le mouvement de l'effecteur. Idéalement, cela nécessiterait que l'acquisition des mesures et l'envoi de commandes au robot se fassent en continu et de manière synchronisée. Cette démarche devait se dérouler entièrement au sein de Simulink, éliminant ainsi les phases disjointes où les données étaient transmises entre Matlab et Simulink pour la simulation et le contrôle. Cette transition était complexe et chronophage, nécessitant des ajustements et des modifications au script d'acquisition des mesures dans Matlab, ainsi que l'exploration de bibliothèques pour la communication en temps réel avec le capteur.

A ce stade, j'ai reconnu la nécessité de réaliser les mesures de force en temps réel directement dans Simulink en utilisant les bibliothèques fournies pour la communication avec le capteur. Cela a nécessité des ajustements par rapport au script d'acquisition des mesures dans Matlab, mais j'ai investi du temps pour réussir à mettre en place cette approche.

En ce qui concerne l'envoi des positions articulaires à la fin de la boucle vers le robot réel, il s'est avéré nécessaire d'utiliser des fonctions spécifiques pour cette tâche. J'ai exploré l'idée de remplacer la fonction "*nonBlocking_movePTPTrajectoryJoint()*" par "*realTime_startDirectServoJoints()*". Cette dernière fonction aurait permis d'envoyer en temps réel les commandes de position articulaire au robot. Pour ce faire, j'ai consacré du temps à me familiariser avec l'utilisation de cette fonction.

Cependant, au moment de l'implémentation sur le robot réel, j'ai rencontré certaines difficultés dans l'utilisation de cette fonction. De plus, il est important de noter que cette fonction n'a pas encore été modifiée ou développée par l'équipe robotique HUG, avec laquelle le laboratoire collabore. En fin de compte, la fonction que j'ai utilisée pour envoyer les positions articulaires au robot était une version modifiée mise en place par D. Gonzalez de l'équipe robotique HUG. Cette fonction a été opérationnelle et a permis d'assurer l'envoi des commandes nécessaires au contrôle des positions articulaires sur le robot réel. Cependant, lorsque j'ai intégré cette fonction dans une boucle où je prenais des mesures du capteur à intervalles réguliers (par exemple, toutes les 1 seconde) et envoyais les résultats des positions articulaires au robot, j'ai remarqué que le robot émettait des bruits, un peu comme s'il activait des freins entre chaque itération de déplacement (ce n'était néanmoins pas le cas), et résultant en des discontinuités soudaines (petits déplacements à fortes accélérations). Ce problème existait uniquement à partir de la deuxième itération et de manière apparemment erratique (parfois en début, parfois en fin du mouvement, parfois de manière à peine perceptible, parfois avec des discontinuités de déplacement de l'ordre du centimètre, ce qui évidemment inacceptable dans le cadre du besoin de notre application). Par conséquent, je suis d'avis que cette fonction ne convient pas à cette procédure spécifique. C'est pourquoi je pense que l'idée de la remplacer par la fonction "*realTime_startDirectServoJoints()*" résoudra le problème des petits bruits que nous entendons entre chaque itération et permettra un control en temps continu.

Bien que le stage n'ait ainsi pas abouti à la mise en œuvre complète et optimisée de cette synchronisation en temps réel, les résultats obtenus ainsi que les démarches entreprises ont fourni des perspectives importantes pour la poursuite du projet et la résolution de ces défis techniques.

5. Conclusion et perspectives

En résumé, mon stage a été une expérience enrichissante qui m'a permis de plonger au cœur de la robotique et de la commande. J'ai réussi à mettre en place des scripts de cinématique directe, indirecte et de suivi de trajectoire sur un modèle de robot KUKA LBR iiwa 14, en utilisant des outils tels que CoppeliaSim et la Toolbox KST. De plus, j'ai pu mettre en œuvre une boucle de commande en admittance et exploiter les mesures de forces et de moments provenant d'un capteur externe intégré. Ces tâches ont exigé une combinaison de compétences en automatique, robotique et mécanique. En conséquence ce projet m'a permis de consolider et d'étendre les compétences acquises au cours de ma formation, tout en me permettant d'explorer de manière pratique le domaine des robots industriels et de contribuer à son avancement.

Cependant, en raison du niveau limité d'expertise en robotique au sein du laboratoire, je n'ai pas pu achever la tâche de synchronisation en temps réel entre les mesures du capteur et les mouvements du robot de manière continue, ce qui aurait été essentiel pour réaliser des protocoles d'essai. La complexité de cette tâche nécessite un niveau élevé d'expertise pour affiner et optimiser les fonctions de la Toolbox KST en vue d'envoyer des commandes au robot réel de manière plus optimale. Malheureusement, cette étape n'a pas encore été développée par l'équipe de robotique HUG à Genève. En conséquence, j'ai dû me limiter à l'utilisation des fonctions disponibles.

Bien que je ressente un certain regret en ce qui concerne cette limitation, je préfère mettre en avant ma satisfaction concernant ma contribution à ce projet et les progrès que j'ai pu réaliser. Je reste convaincu que les connaissances acquises durant ce stage me seront précieuses dans mes projets futurs et ma poursuite de carrière dans le domaine de la robotique et de l'automatisation.

En ce qui concerne les perspectives du projet, voici les aspects à améliorer :

1 - Continuer dans la même approche de contrôle, c'est-à-dire la commande en admittance, et améliorer l'implémentation actuelle en utilisant un mode de contrôle continu dans Simulink en utilisant les fonctionnalités DirectServo. L'objectif est d'initier un contrôle en temps réel souple dans l'espace des articulations du robot.

2 - Effectuer une estimation et une analyse précises des matrices du modèle d'admittance (masse, ressort, amortissement) en intégrant le modèle dynamique du manipulateur. Et identifier l'influence de chacun de ces paramètres sur la dynamique du robot et de choisir une régulation adaptée à l'application.

3 - Intégrer un soustracteur dans la boucle d'admittance entre le moment mesuré autour d'un axe spécifique, par exemple, pour une flexion/extension, inclinaison latérale ou rotation axiale et le moment de consigne.

4 - Ajouter la mesure en temps réel de la position et de la vitesse de l'effecteur dans la boucle de contrôle. Cela permettra d'effectuer des corrections de trajectoire en même temps que le contrôle des efforts.

Ces améliorations devraient contribuer à améliorer la robustesse et la précision du mode de contrôle en admittance pour répondre aux besoins spécifiques de l'application.

Bibliographie

[Horgan (2001)] D. Robots in laparoscopic surgery. In Journal of Laparoendoscopic & Advanced Surgical Techniques (2001), vol. 11, Mary Ann Liebert, Inc., pp. 415-419 – URL : https://www.researchgate.net/publication/11545677_Robots_in_Laparoscopic_Surgery [cité en page 415-419]

[Rouget (1983)] Contribution à l'étude structurelle et fonctionnelle d'organes compliants passifs pour robots d'assemblage. Thèse de docteur-ingénieur : Ecole Nationale Supérieure de Mécanique et des Microtechniques, Besançon, 1983,URL : https://www.sudoc.fr/004920589 [cité en page 203]

[Badano (1993)] Contribution au développement d'une méthodologie d'assemblage automatisé par approche stochastique et compliance. Application à l'insertion de pièces cylindriques sans chanfrein. Thèse de doctorat : Institut National des Sciences Appliquées, Ly.URL : http://www.theses.fr/1993ISAL0005

[Duchaine (2010] Commande des robots destinés à intéragir physiquement avec l'humain. (2010) - URL : https://robot.gmc.ulaval.ca/fileadmin/documents/Theses/vincent_duchaine.pdf [Cité en pages 10 et 16].

[Prelle (1997)] Contribution au contrôle de la com pliance d'un bras de robot à actionnement électropneumatique, Lyon, INSA, Disser tation, 1997 - URL : https://www.researchgate.net/profile/Christine_Prelle/publication/37813263_Contribution_au_controle_de_la_compliance_d'un_bras_de_robot_a_actionnement_electropneumatique/links/00b49529610402cd10000000/Contribution-au-controle-de-la-compliance-dun-bras-de-robot-a-actionnement-electropneumatique.pdf [Cité en pages ix, 10, 12 et 13]

[Safeea (2020)] Safe Collaborative Robotic Mani pulators, 00500 : : Universidade de Coimbra, Dissertation, 2020 URL : https://pastel.hal.science/tel-03070309/file/safeea.pdf [Cité en page 10].

[Raibert (1981)] ,Craig John : Hybrid position/force control of manipulators. (1981) URL : http://fab.cba.mit.edu/classes/865.15/classes/measurement/hybrid-position-force.pdf [Cité en page 10].

[Qin (2013)] , Jinna : Commande hybride position/force robuste d'un robot manipulateur utilisé en usinageet/ou en soudage, Ecole nationale supérieure d'arts et métiers-ENSAM, Dissertation, 2013 - URL : https://pastel.hal.science/file/index/docid/940035/filename/QIN.pdf [Cité en page 10].

[Pujas (1995)] , Berthe A. : Etude de la robustesse de schémas de commande position/force pour robots à deux bras, Dissertation, 1995 [Cité en page 11].

[Whitney (1969)] , Daniel E. : Resolved motion rate control of manipulators and human prostheses. In : IEEE Transactions on man-machine systems 10 (1969), Nr. 2, S. 47–53 - URL : https://ieeexplore.ieee.org/abstract/document/4081862

[Sandoval Arevalo (2017)] , Juan S :Contribution à la commande en couple de robots redondants avec contrainte de RCM dans un contexte d'interaction physique humain-robot, Orléans, Dissertation, 2017 - URL : https://hal.science/tel-02001865/

[Salisbury (1980)] Active stiffness control of a manipulator in cartesian coordinates, IEEE Conference on decision & control, Albuquerque (New Mexico), 1980, URL : https://ieeexplore.ieee.org/abstract/document/4046624 [cité en page 95-100].

[Liégeois (2000)] , Alain : Modélisation et commande des robots manipulateurs. In : Techniques de l'Ingénieur, S 7 (2000), S. 730 URL : https://www.techniques-ingenieur.fr/base-documentaire/archives-th12/archives-automatique-et-ingenierie-systemes-tias0/archive-1/modelisation-et-commande-des-robots-manipulateurs-r7730/ [Cité en page 11].

[Girard (2020)] Modélisation, analyse et commande des systèmes robotisés, URL : https://alexandregirardca.files.wordpress.com/2020/04/modelisationanalysecommanderobots-versionh20-alexandregirard-2020-04-21.pdf

[KUKARoboticArmMaterial (2015)] Récupéré sur https://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Spez_LBR_iiwa_en.pdf

[Safeea (2017)] *Kuka Sunrise Toolbox - URL : https://github.com/Modi1987/KST-Kuka-Sunrise-Toolbox*

[me-systeme] Récupéré sur : https://github.com/me-systeme/GSV-8_Matlab/tree/master

[KSTUsersGuide] Récupéré sur : https://usermanual.wiki/Document/KSTUsersGuide.772454249.pdf

Annexe A : Approche et script existant (contrôle en force)

Objectif de contrôle

L'objectif principal de ce mode de contrôle réside dans l'application d'une rotation précise autour d'un axe spécifique, tout en annulant les forces et moments externes indésirables agissant dans les autres directions. Ceci permet au robot de maintenir une trajectoire précise en dépit des perturbations provenant de forces extérieures.

Processus d'algorithme

- Mesure initiale et tarage :

Initialement, le robot effectue une mesure des forces et moments externes au point d'interaction entre l'outil et la pièce anatomique. Il utilise les capteurs de couple intégrés à chaque articulation du robot, dont les données sont projetées vers ce point d'interaction. Ensuite, ces mesures sont "tarées" lorsque le robot se trouve dans une position neutre, permettant d'obtenir des références de départ.

-Boucle de contrôle en force :

La boucle de contrôle fonctionne par itérations, guidées par un angle défini par l'utilisateur, en rotation autour de l'axe d'intérêt, passant par le Centre Instantané de Rotation (CIR) de la pièce anatomique. Au début de chaque itération, la boucle traite les moments résiduels autour des axes autres que l'axe ciblé pour la rotation. Si ces moments dépassent les valeurs spécifiées par l'utilisateur ([-limTorque, limTorque]), une correction est appliquée pour les réduire. Ensuite, la boucle se tourne vers les forces résiduelles dans les directions non souhaitées. Si les forces externes dans ces directions dépassent les valeurs spécifiées ([-limForce, limForce]), une correction est effectuée pour les diminuer. Ces corrections sont effectuées en boucle (les forces interagissant avec les moments exprimés en un point par le biais des bras de leviers) et par ajustements incrémentiels dans les directions des forces ou moments résiduels détectés par le robot. Les valeurs de ces incréments sont également définies par l'utilisateur.

-Itération :

La boucle de contrôle continue à itérer entre la correction des moments et la correction des forces jusqu'à ce que la rotation autour de l'axe d'intérêt soit achevée ou que l'angle maximal d'itération, défini par l'utilisateur, soit atteint. Ce processus itératif permet d'affiner progressivement les ajustements pour annuler les perturbations externes et garantir un mouvement précis du robot.

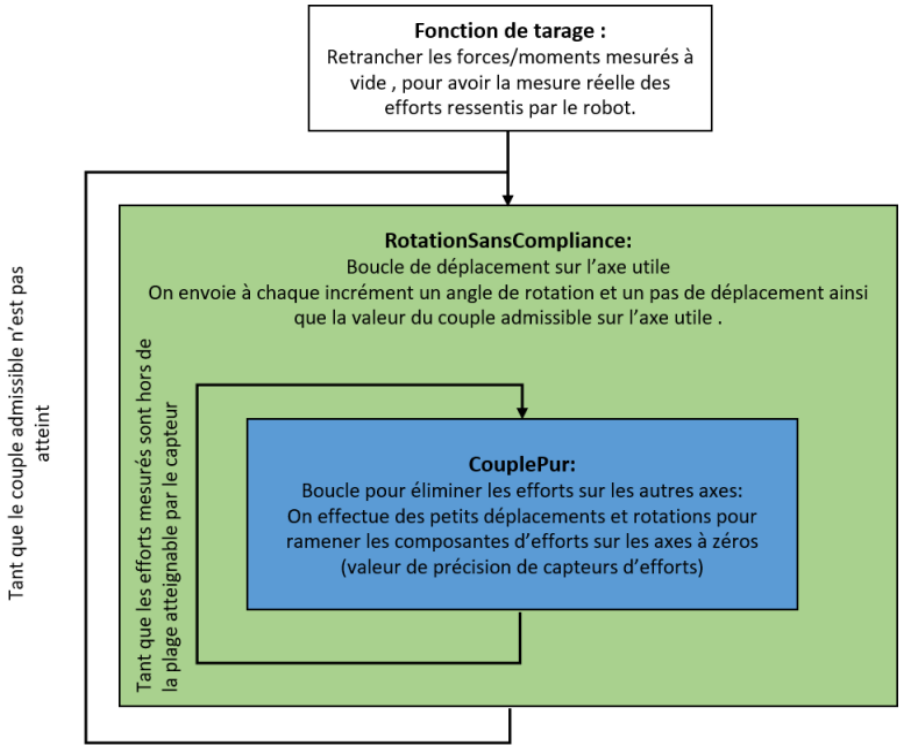
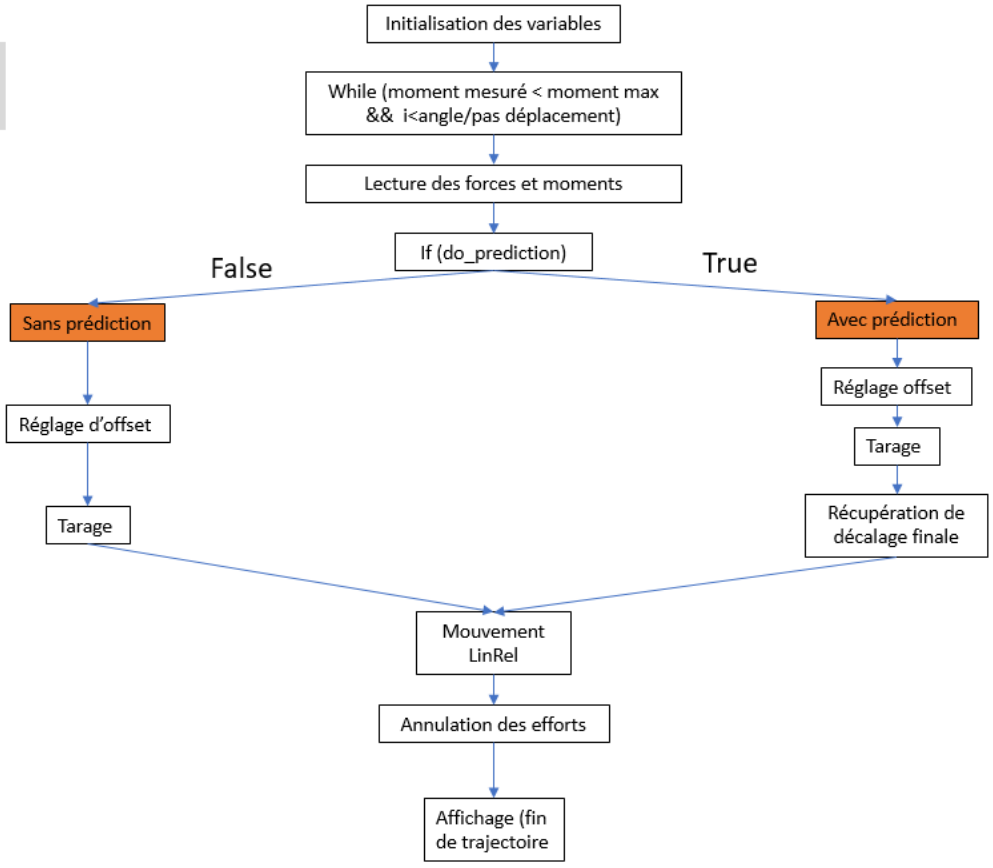
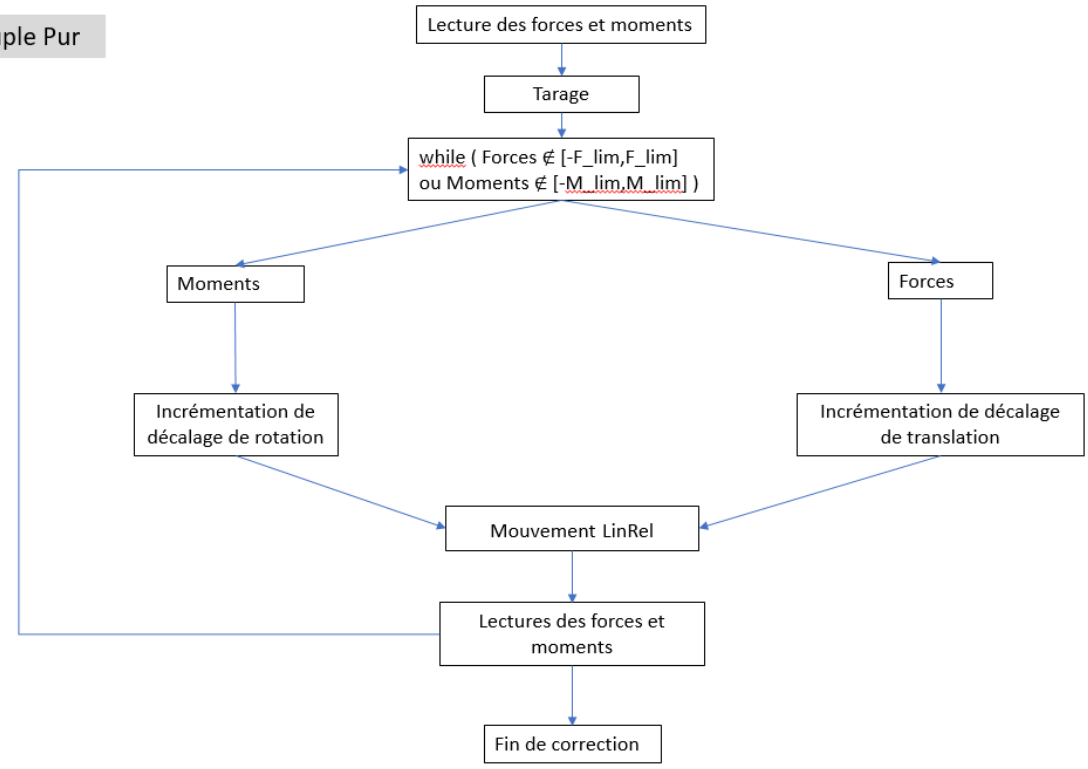


Figure 52 : algorithme de l'approche en force

Rotation sans compliance



Couple Pur



Limitations de ce mode de contrôle

- Vitesse insuffisante en raison du grand nombre d'itérations nécessaires pour neutraliser les forces.
- L'ajustement des efforts est basé sur une consigne de déplacement.
- Sensibilité aux perturbations inattendues : il peut être perturbé par des événements inattendus, comme des changements rapides dans la charge ou des interactions avec l'environnement, ces perturbations pourraient faire bouger le robot de manière instable ou imprévisible.

Annexe B : Matrice de calibration du capteur de force 6 axes (K6D40 500N/20Nm)

Tab.3: calibration matrix Ap

Reference \ Channel	1	2	3	4	5	6
Fx in N/mV/V	10.572	-474.296	458.342	4.915	-451.012	492.159
Fy in N/mV/V	-552.217	261.621	258.676	-547.396	263.304	259.794
Fz in N/mV/V	-807.24	-808.907	-757.018	-768.662	-784.692	-811.236
Mx in Nm/mV/V	-9.107	-11.226	-11.286	-9.478	19.386	19.649
My in Nm/mV/V	18.356	17.494	-16.776	-18.238	-1.029	1.445
Mz in Nm/mV/V	-11.962	11.449	-11.588	12.29	-11.264	12.064

Tab.4: calibration matrix Bp

Reference \ Channel	1	2	3	4	5	6
Fx in N/(mV/V)²	5.558	7.53	0.733	-8.269	-6.725	1.384
Fy in N/(mV/V)²	-4.162	14.754	1.471	-14.944	-5.82	9.769
Fz in N/(mV/V)²	17.132	14.094	18.421	-19.4	-14.305	-15.126
Mx in Nm/(mV/V)²	0.181	0.746	0.066	-0.679	-0.593	0.269
My in Nm/(mV/V)²	0.185	-0.61	-0.223	0.559	-0.331	0.311
Mz in Nm/(mV/V)²	-1.158	0.435	-0.04	-0.399	1.193	0.006

Annexe C : script d'acquisition et stockage des mesures du capteur

```
%% ce script permet de lire plusieurs données du capteur à la fois dans un
% même vecteur en faisant appel à des fonction#c à l'aide de callib
% il faut avoir dans le même répertoire les fichier MEGSV86x64.dll ,MEGSV86x64.h
% et Errorcodes.h + fichier(.dat) et les deux fichiers de la matrice de
% calibration
%pour mieux comprendre l'appel des fonctions regardez(gsv86dll_referencemanual)
%MEGSV86x64.h  contient les détails des fonctions avec les entrées /sorties
% - les fichiers .dll /.h pour l'acquisition de données sont requis pour
% lancer ces script
% - en cas de message d'erreur (extendet=-1): lancer les deux commandes
% pour vider le buffer et décharger la dll
% calllib('MEGSV86x64','GSV86release',com)
% unloadlibrary('MEGSV86x64')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% variable à modifier si necessaire avant de commencer le programme :
% com, freq, time_stop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear
clc

%% load  Library   MEGSV86x64.dll & MEGSV86x64.h (Reading data)
com =5; % definition du port USB
if ~libisloaded('MEGSV86x64')  % chargement des libreries
    loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
end

%% Activation des canaux
[extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
calllib('MEGSV86x64', 'GSV86startTX', com);

calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la mémoire
de l'appareil.

%% %%%%%%%%%%%%%%%%%%%%         Calibration           %%%%%%%%%%%%%%%%%%%%%%%%%
dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
DatFilePath = libpointer('cstring',dat);
[er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% initialisation de fréquence d'acquisition
freq = 100; % Hz
calllib('MEGSV86x64', 'GSV86setFrequency', 5, freq);

%% Tarage initial
calllib('MEGSV86x64', 'GSV86setZero', 5, 0);

%%%%%%%%%%%%%%%%%
%% Lecture des données
tic %startTime = datetime('now');
stop = false;

% creation des tableaux vides pour stockage des données
T = []; % stockage vecteur du temps
A = []; % stockage vecteur des données dimension(1,8)
%%%%%%%%%%%%%%%%
time_stop = 10; % definition du temps d'acquisition en secondes
pause('off')
vec_zero = [0 0 0 0 0 0 0 0]; %vecteurs zeros (1,8) ,qui est transmis entre chaque intervalle
```

```
                                % de mesure transmis par le capteur
s = 0; % définition d'une variable pour calculer le nombre des mesures acquis
%%%%%%%%%%%%%%%
while ~stop
    t = toc; %seconds(datetime('now') - startTime)

    % Appel à la fonction pour lire les données
    [error, data]=calllib('MEGSV86x64','GSV86readMultiple',5, 0, libpointer('doublePtr',zeros(1,8)),...
        8, libpointer('int32Ptr',int32(0)), libpointer('int32Ptr',int32(0)) );

    % condition pour supprimer les vecteurs zeros (1,8) ,intervalle entre
    % chaque mesure transmis par le capteur
    if ~isequal(data, vec_zero)
        s = s + 1;
        A=vertcat(A,data(1:6)) % matrice des mesures (s,8)
        T(end+1)=t; % vecteur du temps
    end

    % condition pour arreter l'acquisition
    if t > time_stop
        stop = true;
    end
end
disp(['nombre de mesures : ', num2str(s)]); % affichage de nombre de mesures lus

%% export des données vers des fichiers csv et les sauvegarder :
headers = {'Time (s)', 'Fx','Fy','Fz','Mx','My','Mz'}; % Entêtes des colonnes
xlswrite('Mesures1.xlsx', headers, 'data', 'A1'); % Écriture des entêtes
xlswrite('Mesures1.xlsx',A,'data','B2');          % Ecriture des mesures
xlswrite('Mesures1.xlsx', T(:), 'data', 'A2');    % Ectriture du temps en seconde

%% Libération des ressources
calllib('MEGSV86x64', 'GSV86release', com);
unloadlibrary('MEGSV86x64');
```

Annexe D : script acquisition des mesures capteur et soustraction de l'effet de la pièce embarqué

```
%% ce script permet de communiquer avec le robot et le capteur externe en meme
%% temps et soustraire l'effet de la piece embarqué des mesures capteur au cours
%% de mouvement du robot
% @author : Tir abd elhafid

%% avant de lancer le script Veuillez préciser le temps d'execution du code
%% dans la variable (time_stop) en secondes

%% GSV86setZero = toujours en position ou le Z_capteur vers le bas (Parallele à Z_world)

%% remarque: Parfois faut relancer Matlab après l'utilisation de GSV-8 sinon les mesures seront pas
prises
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear
clc

%% %%%%%%%%%%%% Connexion avec le robot %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
warning('off')
%% Create the robot object
ip='172.31.1.147'; % The IP of the controller
arg1=KST.LBR14R820; % choose the robot iiwa7R800 or iiwa14R820
arg2=KST.Medien_Flansch_Touch_pneumatisch; % choose the type of flange
Tef_flange=eye(4);
iiwa=KST(ip,arg1,arg2,Tef_flange); % create the object
%% Start a connection with the server
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);


%% %%%%%%%%%%%Connexion avec le capteur externe %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% load  Library   MEGSV86x64.dll & MEGSV86x64.h (Reading data)
com =5;
if ~libisloaded('MEGSV86x64')
    loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
end
%% Activate channels
[extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
calllib('MEGSV86x64', 'GSV86startTX', com);

calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la mémoire
de l'appareil.

%% Calibration
dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
DatFilePath = libpointer('cstring',dat);
[er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );

%% initialisation of the frequency
freq = 10; % Hz (force sensor acquisition frequency)
% faut choisir une fréquence d'acquisition du capteur faible
% environ 10-15 HZ car la fréquence d'envoie Controleur-PC est bkp plus faible, environ
% 10-15 HZ et cela pour avoir une cohérence dans la résultat de
% soustraction de l'effet de la piece.

calllib('MEGSV86x64', 'GSV86setFrequency', com, freq);
```

```
calllib('MEGSV86x64', 'GSV86setZero', com, 0);
% Parfois faut essayer de lancer script 2 fois de suite
%  (Z_capteur // Z_world)) pour 2 second, pour faire le tarage correctement

% calllib('MEGSV86x64','GSV86setModeNoiseCut',com,0)
% possibilité de réduction de bruit via GSV-multi aussi

%% Lecture des données
tic %startTime = datetime('now');
stop = false;
i = 0; % incrément nombre des mesures
T = []; % vecteur stockage du temps
A = []; % vecteur stockage des mesures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
time_stop = 10;
pause('off')
vec_zero = [0 0 0 0 0 0 0 0];
%vecteurs zeros (1,8) ,qui est transmis entre chaque intervalle
% de mesure transmis par le capteur
TT = []; %vecteur stockager du résultat de soustraction
FF=[];% stockage vecteur expression des forces/moment
%% %%%%%%%%%%%%%%%%%% Boucle de lecture   %%%%%%%%%%%%%%%%%%%%%%%%%
while ~stop

    t = toc; %seconds(datetime('now') - startTime)
    [error, data]=calllib('MEGSV86x64','GSV86readMultiple',5, 0, libpointer('doublePtr',zeros(1,8)),...
        8, libpointer('int32Ptr',int32(0)), libpointer('int32Ptr',int32(0)) );

    if ~isequal(data, vec_zero)% supprimer les vecteurs zeros entre chaque mesures selon freq choisis
        A=vertcat(A,data); % stockage vecteur des mesures
        T(end+1)=t; % stockage vecteur de mesures

        forceZ_estime = 6.64; % l'effet du poids de la masse après le capteur
        % récuperer la position du Flange dans world
        cpos = getEEFCartesianPosition( iiwa.t_Kuka ); % du flange
        O1_O2 = [cell2mat(cpos(1)), cell2mat(cpos(2)), cell2mat(cpos(3))];
        % récuperer l'orientation du flange dans world
        orie = getEEFCartesianOrientation( iiwa.t_Kuka ); % (du flange) il y'a une différence entre la
séquence utilisée dans
        % cette fonction et celle utilisée pour envoyer une orientation au robot  (ici selon X puis Y
puis Z)
        alpha = cell2mat(orie(1));                  % envoi au robot (orien selon Z puis Y puis X)
        beta = cell2mat(orie(2));
        psi = cell2mat(orie(3));
        % calcule l'expression des forces et moments de la pièce embarqué
        % dans le repère tourné
        F = calcul_forces_moments(alpha, beta, psi, O1_O2);
        FF = [FF;F]; % stockage vecteur expression des forces/moments
        i = i+1; % incrément nombre des mesures


        % soustraction de l'effet de la pièce du mesures capteur:
        T1 = [ A(i,1)-FF(i,1), A(i,2)-FF(i,2), forceZ_estime+A(i,3)-FF(i,3), A(i,4)-FF(i,4),  A(i,5)-
FF(i,5),  A(i,6)-FF(i,6) ] %résultat soustraction
        TT = [TT;T1];

        if t > time_stop
            stop = true;
            iiwa.net_turnOffServer();
            break;
        end

    end
end

% disp (TT); % affichage des mesures avec soustraction
disp(['nombre de mesures : ', num2str(i)]); % nombres des mesures
%% export des données vers des fichiers csv et les sauvegarder :
```

```
headers = {'Time (s)', 'Fx','Fy','Fz','Mx','My','Mz'}; % Entêtes des colonnes
xlswrite('Mesures_sans_soustraction_piece.xlsx', headers, 'data', 'A1'); % Écriture des entêtes
xlswrite('Mesures_sans_soustraction_piece.xlsx', T(:), 'data', 'A2');
xlswrite('Mesures_sans_soustraction_piece.xlsx',A(:,1:6),'data','B2');

xlswrite('Mesures_avec_soustraction_piece.xlsx', headers, 'data', 'A1'); % Écriture des entêtes
xlswrite('Mesures_avec_soustraction_piece.xlsx', T(:), 'data', 'A2');
xlswrite('Mesures_avec_soustraction_piece.xlsx',TT,'data','B2');


%% Libération des ressources
calllib('MEGSV86x64', 'GSV86release', 5);
unloadlibrary('MEGSV86x64');
```

```
%% cette fonction calcule l'expression de la force et moment de la masse embarqué mesurés par le capteur
externe
%% dans le repère tourné à chaque incrément de rotation


% alpha = angle de rotation selon axe X
% beta = angle de rotation selon axe Y
% psi = angle de rotation selon axe Z


function T = calcul_forces_moments(alpha, beta,psi,O1_O2)

    H12 = transform_matrix( alpha, beta, psi , O1_O2); % transform matrix entre repère world et flange.

    O2_O3= [0.0; 0.0; 0.050]; % distance entre Flange et centre capteur (+ 50 mm) selon Z du Flange.
    H23 = transform_matrix( 0, 0, pi/2 , O2_O3); % transform matrix entre repère flange et capteur
    %%%%%
    H13 = H12*H23; % transform matrix entre world et repère capteur
    % orientation capteur = orientation flange
    % théoriquement : Force_dans_repère_capteur = H12 * Force_dans_repère_tourné_(ce qu'on cherche)
    % je connais Force_dans_repère_capteur ( à partir des mesures GSV avant tarage et je connais H12,
donc :
    % O1_F = H12 * O2_F
    % donc : O2_F = inverse(H12) * O1_F
    forceZ_estime = 6.64; % force mesuré selon axe Z descendant du capteur de toute la pièce embarqué.
    O1_F = [0; 0; -forceZ_estime; 0];
    O2_F = inverse(H13) * O1_F;  %% vecteur force dans le repère tournant
    AB = [0; 0.0001228; 0.017967]; % (m)  ==> centre de masse trouvé en exploitant ce que le capteur
mesure (corrigé manuellement)
    Moment = cross(AB,O2_F(1:3,:)); % (N.m) ==> vecteur moment dans le repère tournant

    T = [O2_F(1:3,:)', Moment'];

end
```

```
% cette fonction permet de construire la matrice homogène de transformation
% entre deux repère, ou prends en entré les angles en radian, et (t) est le
% vecteur translation (1,3)

function H = transform_matrix(alpha, beta, psi , t)
% alpha = angle autour de l'axe X
% beta = angle autour de l'axe Y
% psi = angle autour de l'axe Z
% t = vecteur de translation

H = transl(t)*trotz(psi)*troty(beta)*trotx(alpha);
end
```

```
% cette fonction permet de calculer matrice homogène de transformation
% inverse, prends en entré la matrice homogène à inverser (doc. Veron)
function T = inverse(H)
   T = eye(4);
   T(1:3,1:3) = transpose(H(1:3,1:3));
   T(1:3,4) = - transpose(H(1:3,1:3))*H(1:3,4);
end
```

Annexe E : Script simulation de cinématique directe sous CoppeliaSim

```
% ce script permet de communiquer avec CoppeliaSim et réaliser une simulation
% de cinématique direct sur le modèle robot kuka iiwa lbr 14 R820
% @author : Tir abd elhafid

clear
close all
clc
%% %%%%%%%%%% Connexion avec Coppeliasim %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sim=remApi('remoteApi');   % utiliser bibliothèque 'remApi'
sim.simxFinish(-1);        % fermer la connexion existante
clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5); % ouvre une nouvelle connexion
%% vérification de la connexion:
if(clientID>-1)
   disp('connected');
   jHandles=zeros (7,1);
   % récupération des identifiants des articulations
   for i=1:7
       s=['LBR_iiwa_14_R820_joint',num2str(i)];
       [res,daHandle]=sim.simxGetObjectHandle (clientID, s, sim.simx_opmode_blocking);
       jHandles(i)=daHandle;
   end
end

%% %%%%%%%% Direct kenimatic %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% définition de la position articulaire souhaité en Radian
jPos= [0.0; 0.0; 0.0; -pi/2; 0.0; pi/2; 0.0];

%% %%%%%%%%%%%%% Réglage de la Vitesse des Articulations %%%%%%%%%%%%%%%%%
relative_joint_speed = 0.05;
vrep_joint_original_speeds = {1.9199, 1.9199, 2.2340, 2.2340, 3.5605, 3.2114, 3.2114};
% taken from joint "Upper velocity limit" field [rad/s]
sim.simxPauseCommunication(clientID, 1);
for i = 1:7
    errCode = sim.simxSetObjectFloatParameter( ...
        clientID, jHandles(i),  sim.sim_jointfloatparam_upper_limit, ...
        vrep_joint_original_speeds{i} * relative_joint_speed, sim.simx_opmode_oneshot );
    if errCode ~= 0 && errCode ~= 1
        sim.simxPauseSimulation(clientID, sim.simx_opmode_oneshot_wait);
        error("setting simulated joint speed failed, error code " + num2str(errCode))
    end
end
sim.simxPauseCommunication(clientID, 0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% envoyer la configuration target initial pour le robot simulé sous CoppeliaSim
for i=1:7
   [errorCode]=sim;
   sim.simxSetJointTargetPosition(clientID,jHandles(i),jPos(i),sim.simx_opmode_oneshot);
end
Tef_flange = eye(4); % matrice de transforme entre flange et point TCP choisis sur l'outil
[X,J] = directKinematics(jPos,Tef_flange); % calcul de la cinématique direct
disp(X);

%% fin de simulation
sim.delete();
```

Annexe F : script simulation de cinématique indirect sous CoppeliaSim

```
% ce script permet de communiquer avec CoppeliaSim et réaliser une simulation
% de cinématique inverse sur la base du modèle robot kuka iiwa lbr 14 R820
% @author : Tir abd elhafid

clear
close all
clc

%% %%%%%%%%%% Connexion avec Coppeliasim %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sim=remApi('remoteApi');  % utiliser bibliothèque 'remApi'
sim.simxFinish(-1);       % fermer la connexion existante
clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5); % ouvre une nouvelle connexion
% vérification de la connexion:
if(clientID>-1)
   disp('connected');
   jHandles=zeros (7,1);
   % récupération des identifiants des articulations
   for i=1:7
       s=['LBR_iiwa_14_R820_joint',num2str(i)];
       [res,daHandle]=sim.simxGetObjectHandle (clientID, s, sim.simx_opmode_blocking);
       jHandles(i)=daHandle;
   end
end
% envoie à une position initial au début de simulation
jPos= [0.0; 0.0; 0.0; 0.0; 0.0; 0.0; 0.0];

%% %%%%%%%%%%%%%%% Réglage de la Vitesse des Articulations %%%%%%%%%%%%%%%
relative_joint_speed = 0.05;
vrep_joint_original_speeds = {1.9199, 1.9199, 2.2340, 2.2340, 3.5605, 3.2114, 3.2114};
% taken from joint "Upper velocity limit" field [rad/s]
sim.simxPauseCommunication(clientID, 1);
for i = 1:7
    errCode = sim.simxSetObjectFloatParameter( ...
        clientID, jHandles(i),  sim.sim_jointfloatparam_upper_limit, ...
        vrep_joint_original_speeds{i} * relative_joint_speed, sim.simx_opmode_oneshot );
    if errCode ~= 0 && errCode ~= 1
        sim.simxPauseSimulation(clientID, sim.simx_opmode_oneshot_wait);
        error("setting simulated joint speed failed, error code " + num2str(errCode))
    end
end
sim.simxPauseCommunication(clientID, 0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% envoyer la configuration target initial pour le robot simulé sous CoppeliaSim
for i=1:7
   [errorCode]=sim;
   sim.simxSetJointTargetPosition(clientID,jHandles(i),jPos(i),sim.simx_opmode_oneshot);
end

%% %%%%%%%%%%%%%%%%%%%%%%  inverse kenimatic   %%%%%%%%%%%%%%%%%%%%%%%%%%
% j'apporte une correction sur position de 'iiwa_link_ee_kuka' par rapport
% à la dernière articulation :
% sur le modèle robot_rbt par défaut la position de 'iiwa_link_ee_kuka' est
% à (0.045mm) par rapport à la dernère articulation selon Z_flange, avec la fonction
% correctComponentLength() j'ajoute 0.026mm pour que le point TCP
% 'iiwa_link_ee_kuka' correspond à la taille du notre flange "Touch
% Pneumatic flange" : 0.045mm ==> 0.071mm

robot_rbt = loadrobot("kukaIiwa14"); % charger modèle dynmq du robot
robot_rbt.DataFormat = "row"; % change the format struct => row
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8)+joint_length_additions(1, 9));
% la dernière valeur du vecteur 'joint_length_additions' peut etre modifiée pour qu'elle corresponds à
```

```
% la taille de l'outils càd après Flange. (ici j'effectue une
% inverse-cinématique par rapport au Flange)

solver_parameters = struct("EnforceJointLimits", 1); % Configuration des paramètres du solveur
ik = inverseKinematics('RigidBodyTree', robot_rbt, "SolverAlgorithm", "BFGSGradientProjection",
"SolverParameters", solver_parameters);

%% choix de la position désiré :
% --------
translation = [0.4 -0.36 0.62];
rot = [-pi, 0.0, pi];
% --------

transform_matrix = makehgtform('translate', translation, 'zrotate', rot(3), 'yrotate', rot(2), 'xrotate',
rot(1) );
weights = [1 1 1 1 1 1]; % trois premiers poids de tolérance pour ROT et 3 derniers sur Trans
initial_guess = [0.0 0.0 0.0 -pi/2 0.0 pi/2 0.0];

[joint_position, solution_info] = ik('iiwa_link_ee_kuka', transform_matrix, weights, initial_guess);
disp(joint_position);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% envoie de la solution de configuration trouvée par le solveur pour le robot simulé sous CoppeliaSim
for i=1:7
   [errorCode]=sim;
   sim.simxSetJointTargetPosition(clientID,jHandles(i),joint_position(i),sim.simx_opmode_oneshot);
end

%% fin de simulation
sim.delete();



function correctComponentLength(rbt, componentIndex, lengthAddition)
    component = rbt.Bodies(componentIndex);
    tform = component{1}.Joint.JointToParentTransform;

    % find the length line position in the matrix (it differs following the component)
    col = 4;
    line = 3;
    if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
        line = 2;
    end

    old_length = tform(line, col);
    new_length = old_length + lengthAddition;

    disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

    tform(line, col) = new_length;
    setFixedTransform(component{1}.Joint, tform);
    replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end
```

Annexe G : script simulation de suivi de trajectoire sous CoppeliaSim

```
% ce script permet de communiquer avec CoppeliaSim et réaliser une simulation
% de cinématique inverse "suivie de trajectoire" sur la base du modèle robot
% kuka iiwa lbr 14 R820
% @author : Tir abd elhafid

clear
close all
clc

%% %%%%%%%%%% Connexion avec Coppeliasim %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sim=remApi('remoteApi');  % utiliser bibliothèque 'remApi'
sim.simxFinish(-1);       % fermer la connexion existante
clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);
%% vérification de la connexion:
if(clientID>-1)
   disp('connected');
   jHandles=zeros (7,1);
   % récupération des identifiants des articulations
   for i=1:7
       s=['LBR_iiwa_14_R820_joint',num2str(i)];
       [res,daHandle]=sim.simxGetObjectHandle (clientID, s, sim.simx_opmode_blocking);
       jHandles(i)=daHandle;
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% envoie à une position initial au début de simulation
jPos= [0.0; 0.0; 0.0; -pi/2; 0.0; pi/2; 0.0];

%% %%%%%%%%%%%%%%% Réglage de la Vitesse des Articulations %%%%%%%%%%%%%%%
relative_joint_speed = 0.03;
vrep_joint_original_speeds = {1.9199, 1.9199, 2.2340, 2.2340, 3.5605, 3.2114, 3.2114};
% taken from joint "Upper velocity limit" field [rad/s]
sim.simxPauseCommunication(clientID, 1);
for i = 1:7
    errCode = sim.simxSetObjectFloatParameter( ...
        clientID, jHandles(i),  sim.sim_jointfloatparam_upper_limit, ...
        vrep_joint_original_speeds{i} * relative_joint_speed, sim.simx_opmode_oneshot );
    if errCode ~= 0 && errCode ~= 1
        sim.simxPauseSimulation(clientID, sim.simx_opmode_oneshot_wait);
        error("setting simulated joint speed failed, error code " + num2str(errCode))
    end
end
sim.simxPauseCommunication(clientID, 0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% envoyer la configuration target initial pour le robot simulé sous CoppeliaSim
for i=1:7
   [errorCode]=sim;
   sim.simxSetJointTargetPosition(clientID,jHandles(i),jPos(i),sim.simx_opmode_oneshot);
end

%% %%%%%%%%%%%%%%%%%%%%%%  inverse kenimatic   %%%%%%%%%%%%%%%%%%%%%%%%%
% j'apporte une correction sur position de 'iiwa_link_ee_kuka' par rapport
% à la dernière articulation :
% sur le modèle robot_rbt par défaut la position de 'iiwa_link_ee_kuka' est
% à (0.045mm) par rapport à la dernère articulation selon Z_flange, avec la fonction
% correctComponentLength() j'ajoute 0.026mm pour que le point TCP
% 'iiwa_link_ee_kuka' correspond à la taille du notre flange "Touch
% Pneumatic flange" : 0.045mm ==> 0.071mm

robot_rbt = loadrobot("kukaIiwa14"); % charger modèle dynmq du robot
robot_rbt.DataFormat = "row"; % change the format struct => row
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
```

```
correctComponentLength(robot_rbt, 10, joint_length_additions(1,
8)+joint_length_additions(1, 9));
% la dernière valeur du vecteur 'joint_length_additions' peut etre modifiée pour qu'elle corresponds à
% la taille de l'outils càd après Flange. (ici j'effectue une
% inverse-cinématique par rapport au Flange)

solver_parameters = struct("EnforceJointLimits", 1); % Configuration des paramètres du solveur
ik = inverseKinematics('RigidBodyTree', robot_rbt, "SolverAlgorithm", "BFGSGradientProjection",
"SolverParameters", solver_parameters);


%% --- choix de position cartésian désiré :
cartesian_trajectory = [0.4 -0.4 0.6 pi  0.0  -pi;...
                        0.4 -0.2 0.6 pi  0.0  -pi;...
                        0.4  0.0 0.6 pi  0.0  -pi;...
                        0.4  0.2 0.6 pi  0.0  -pi;...
                        0.4  0.4 0.6 pi  0.0  -pi];
% ----------------------------------------------------
joint_positions = [];
for s = 1:size(cartesian_trajectory,1)
   translation = [cartesian_trajectory(s, 1) cartesian_trajectory(s, 2) cartesian_trajectory(s, 3)];
   rot = [cartesian_trajectory(s, 4), cartesian_trajectory(s, 5), cartesian_trajectory(s, 6)];
   transform_matrix = makehgtform('translate', translation, 'zrotate', rot(3), 'yrotate', rot(2),
'xrotate', rot(1) );
   weights = [1 1 1 1 1 1]; % trois premiers poids de tolérance pour ROT et 3 derniers sur Trans

   %% %%%%%%%% get joint angle %%%%%%
   joint_angle = zeros(1,7);
   for i = 1:7
       [res, joint_angle(i)] = sim.simxGetJointPosition(clientID, jHandles(i), sim.simx_opmode_blocking);
   end
   initial_guess = joint_angle;
   [joint_position, solution_info] = ik('iiwa_link_ee_kuka', transform_matrix, weights, initial_guess);
   joint_positions = [joint_positions;joint_position];
end
disp(joint_positions);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% envoie de la solution de configuration trouvée par le solveur pour le robot simulé sous CoppeliaSim
sim_target_trajectory_i = size(joint_positions, 1);
joint_trajectory_flatten = reshape(joint_positions', 1, []);
errCode = sim.simxSetStringSignal(clientID, 'trajectory', sim.simxPackFloats(joint_trajectory_flatten),
sim.simx_opmode_oneshot_wait);
if errCode ~= 0
    self.vrep.simxPauseSimulation(clientID, sim.simx_opmode_oneshot_wait);
    error("simulated trajectory upload failed, error code " + num2str(errCode))
end

%% fin de simulation
sim.delete();


function correctComponentLength(rbt, componentIndex, lengthAddition)
    component = rbt.Bodies(componentIndex);
    tform = component{1}.Joint.JointToParentTransform;

    % find the length line position in the matrix (it differs following the component)
    col = 4;
    line = 3;
    if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
        line = 2;
    end

    old_length = tform(line, col);
    new_length = old_length + lengthAddition;

    disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));
```

```
    tform(line, col) = new_length;
    setFixedTransform(component{1}.Joint, tform);
    replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end
```

Annexe H : script de cinématique directe sur le robot réel

```
% ce script permet de communiquer et réaliser une cinématique direct sur
% le modèle robot kuka iiwa lbr 14 R820
% @author : Tir abd elhafid
clear
close all
clc;
warning('off')

%% Créer un objet robot
ip='172.31.1.147'; % IP du controlleur
arg1=KST.LBR14R820; % choix du modèle iiwa14R820
arg2=KST.Medien_Flansch_Touch_pneumatisch; % choix du type du flange
Tef_flange=eye(4); % matrice de transformation de l'eefecteur au flange
iiwa=KST(ip,arg1,arg2,Tef_flange); % creation de l'objet


%% Connexion avec le serveur
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);

%% choisir une position cible
position_initial = [0.0       0.0  0.0  -pi/2   0.0  pi/2        0.0];
relVel=0.1; % vitesse entre 0.01 et 1 ('%' des vitesses max des couples moteurs)
iiwa.nonBlocking_movePTPTrajectoryJoint(position_initial, relVel);

% calcul de cinématique direct
[X,J] = directKinematics(position_initial,Tef_flange); % calcul de la cinématique direct
disp(X);
%% Arret du serveur et du script
stop = false;
while ~stop
    finished = iiwa.isMotionFinished();% demander au serveur si mouv est fini,
                                       % si aucun mouv en progression, return true
    if finished
        iiwa.net_turnOffServer();
        break;
    end
end
```

Annexe I : script de cinématique inverse sur robot réel

```
% ce script permet de communiquer et réaliser une cinématique inverse sur
% le modèle robot kuka iiwa lbr 14 R820
% @author : Tir abd elhafid

clear
close all
clc;
warning('off')

%% Créer un objet robot
ip='172.31.1.147'; % IP du controlleur
arg1=KST.LBR14R820; % choix du modèle iiwa14R820
arg2=KST.Medien_Flansch_Touch_pneumatisch; % choix du type du flange
Tef_flange=eye(4); % matrice de transformation de l'eefecteur au flange
% ici : on établit une cinématique inverse par rapport au Flange comme
% effecteur, si on souhaite faire par rapport à un autre point (par exp :
% l'extrémité de l'outil), on a qu'a modifié la matrice de transformation 'Tef_flange'
iiwa=KST(ip,arg1,arg2,Tef_flange); % creation de l'objet


%% Connexion avec le serveur
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);

%% envoie à une position initial
position_initial = [0.0      0.0   0.0    -pi/2   0.0    pi/2  0.0];
relVel=0.1;
iiwa.nonBlocking_movePTPTrajectoryJoint(position_initial, relVel);

%% %%%%%%%%%%%%%%%%%%%%%%  inverse kenimatic   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % j'apporte une correction sur position de 'iiwa_link_ee_kuka' par rapport
 % à la dernière articulation :
 % sur le modèle robot_rbt par défaut la position de 'iiwa_link_ee_kuka' est
 % à (0.045mm) par rapport à la dernère articulation selon Z_flange, avec la fonction
 % correctComponentLength() j'ajoute 0.026mm pour que le point TCP
 % 'iiwa_link_ee_kuka' correspond à la taille du notre flange "Touch
 % Pneumatic flange" : 0.045mm ==> 0.071mm
robot_rbt = loadrobot("kukaIiwa14"); % charger modèle dynmq du robot
robot_rbt.DataFormat = "row"; % change the format struct => row
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8));

solver_parameters = struct("EnforceJointLimits", 1); % Configuration des paramètres du solveur
ik = inverseKinematics('RigidBodyTree', robot_rbt, "SolverAlgorithm", "BFGSGradientProjection",
"SolverParameters", solver_parameters);

%% choix de la position désiré :
% --------
translation = [0.4 0.0 0.6];
rot = [-pi, 0.0, -pi];
% --------

transform_matrix = makehgtform('translate', translation, 'zrotate', rot(1), 'yrotate', rot(2), 'xrotate',
rot(3) );
weights = [1 1 1 1 1 1]; % trois premiers poids de tolérance pour ROT et 3 derniers sur Trans
initial_guess = cell2mat(iiwa.getJointsPos());

[joint_position, solution_info] = ik('iiwa_link_ee_kuka', Tef_flange*transform_matrix, weights,
initial_guess);
```

```
disp(joint_position);

[X,J]=directKinematics(joint_position,Tef_flange); % calcul de : x = F(q)
iiwa.nonBlocking_movePTPTrajectoryJoint(joint_position, relVel);

%% turn off server et arret du script
stop = false;
while ~stop
    finished = iiwa.isMotionFinished();
    if finished
        iiwa.net_turnOffServer();
        break;
    end
end


function correctComponentLength(rbt, componentIndex, lengthAddition)
    component = rbt.Bodies(componentIndex);
    tform = component{1}.Joint.JointToParentTransform;

    % find the length line position in the matrix (it differs following the component)
    col = 4;
    line = 3;
    if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
        line = 2;
    end

    old_length = tform(line, col);
    new_length = old_length + lengthAddition;

    disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

    tform(line, col) = new_length;
    setFixedTransform(component{1}.Joint, tform);
    replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end
```

Annexe J : script de suivi de trajectoire du robot réel

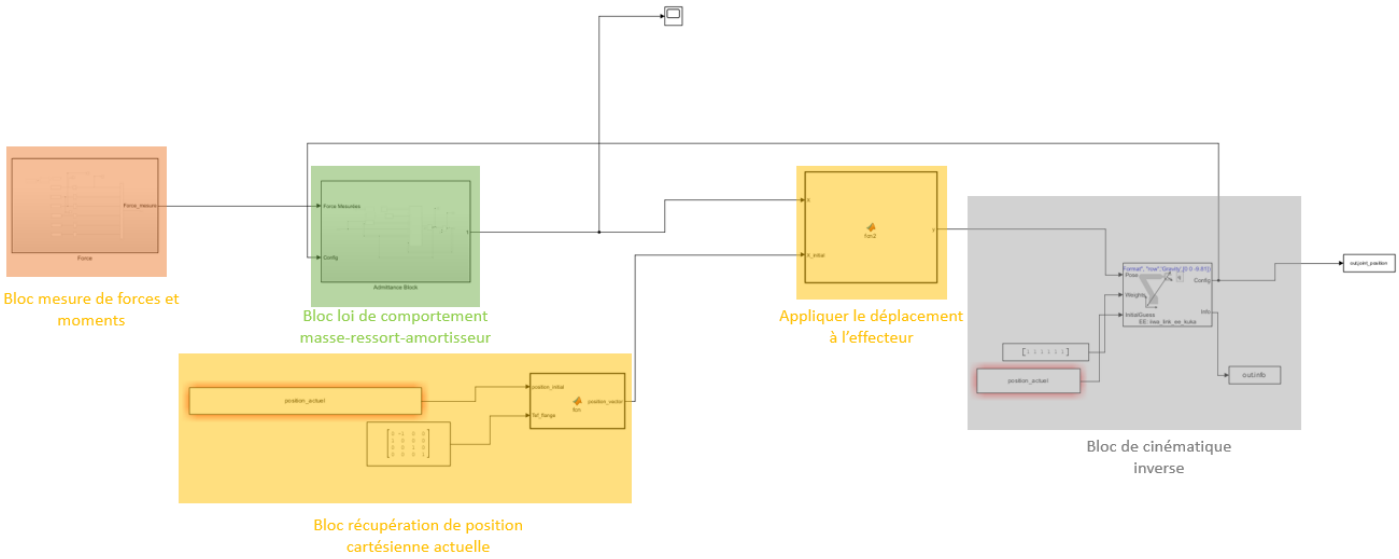
```
% ce script permet de communiquer et réaliser une cinématique inverse sur
% le modèle robot kuka iiwa lbr 14 R820
% @author : Tir abd elhafid

clear
close all
clc;
warning('off')

%% Créer un objet robot
ip='172.31.1.147'; % IP du controlleur
arg1=KST.LBR14R820; % choix du modèle iiwa14R820
arg2=KST.Medien_Flansch_Touch_pneumatisch; % choix du type du flange
Tef_flange=eye(4); % matrice de transformation de l'eefecteur au flange
% ici : on établit une cinématique inverse par rapport au Flange comme
% effecteur, si on souhaite faire par rapport à un autre point (par exp :
% l'extrémité de l'outil), on a qu'a modifié la matrice de transformation 'Tef_flange'
iiwa=KST(ip,arg1,arg2,Tef_flange); % creation de l'objet


%% Connexion avec le serveur
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);

%% envoie à une position initial
position_initial = [0.0      0.0   0.0    -pi/2   0.0    pi/2  0.0];
relVel=0.1;
iiwa.nonBlocking_movePTPTrajectoryJoint(position_initial, relVel);

%% %%%%%%%%%%%%%%%%%%%%%%  inverse kenimatic   %%%%%%%%%%%%%%%%%%%%%%%%%%%
 % j'apporte une correction sur position de 'iiwa_link_ee_kuka' par rapport
 % à la dernière articulation :
 % sur le modèle robot_rbt par défaut la position de 'iiwa_link_ee_kuka' est
 % à (0.045mm) par rapport à la dernère articulation selon Z_flange, avec la fonction
 % correctComponentLength() j'ajoute 0.026mm pour que le point TCP
 % 'iiwa_link_ee_kuka' correspond à la taille du notre flange "Touch
 % Pneumatic flange" : 0.045mm ==> 0.071mm
robot_rbt = loadrobot("kukaIiwa14"); % charger modèle dynmq du robot
robot_rbt.DataFormat = "row"; % change the format struct => row
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8));

solver_parameters = struct("EnforceJointLimits", 1); % Configuration des paramètres du solveur
ik = inverseKinematics('RigidBodyTree', robot_rbt, "SolverAlgorithm", "BFGSGradientProjection",
"SolverParameters", solver_parameters);

%% choix de la trajectoire désiré :
% --------
cartesian_trajectory = [0.4  -0.30  0.6  -pi  0.0  -pi;...
                        0.4   0.0   0.6  -pi  0.0  -pi;...
                        0.4   0.30  0.6  -pi  0.0  -pi];
% --------

joint_positions = [];
for s = 1:size(cartesian_trajectory,1)
    translation = [cartesian_trajectory(s, 1) cartesian_trajectory(s, 2) cartesian_trajectory(s, 3)];
    rot = [cartesian_trajectory(s, 4), cartesian_trajectory(s, 5), cartesian_trajectory(s, 6)];

    transform_matrix = makehgtform('translate', translation, 'zrotate', rot(1), 'yrotate', rot(2),
'xrotate', rot(3) );
```

```
    weights = [1 1 1 1 1 1]; % trois premiers poids de tolérance pour ROT et 3
derniers sur Trans
    initial_guess = cell2mat(iiwa.getJointsPos());
    [joint_position, solution_info] = ik('iiwa_link_ee_kuka', Tef_flange*transform_matrix, weights,
initial_guess);
    joint_positions = [joint_positions; joint_position];
    [X,J]=directKinematics(joint_positions,Tef_flange);
end

disp(joint_positions);
iiwa.nonBlocking_movePTPTrajectoryJoint(joint_positions, relVel);

%% turn off server et arret du script
stop = false;
while ~stop
    finished = iiwa.isMotionFinished();
    if finished
        iiwa.net_turnOffServer();
        break;
    end
end


function correctComponentLength(rbt, componentIndex, lengthAddition)
    component = rbt.Bodies(componentIndex);
    tform = component{1}.Joint.JointToParentTransform;

    % find the length line position in the matrix (it differs following the component)
    col = 4;
    line = 3;
    if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
        line = 2;
    end

    old_length = tform(line, col);
    new_length = old_length + lengthAddition;

    disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

    tform(line, col) = new_length;
    setFixedTransform(component{1}.Joint, tform);
    replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end
```

Annexe K : Script boucle de contrôle en admittance Virtuelle sous CoppeliaSim

```
%% ce script permet de simuler le robot sous CoppeliaSim et appliquer la boucle
%% de controle en admittance défini dans le fichier simulink.
% @author : Tir abd elhafid

%% utilisation :
%% 1 - lancer CoppeliaSim et charger le modèle robot kuka lbr iiwa 14 R820, puis lancer la simulation.
%% 2 - choisir le signal de force à appliquer dans simulink.
%% 3 - lancer ce script et relancer la simulation dans Coppeliasim

clear
close all
clc
%% %%%%%%%%%%%%%%%%% Connexion avec CoppeliaSim  %%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%% et envoie à une position initial %%%%%%%%%%%%%%%%%%
vrep=remApi('remoteApi');
vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
% vérification de la connexion:
if(clientID>-1)
   disp('connected');
   jHandles=zeros (7,1);
   % récupération des identifiants des articulations
   for i=1:7
       s=['LBR_iiwa_14_R820_joint',num2str(i)];
       [ress,daHandle]=vrep.simxGetObjectHandle (clientID, s, vrep.simx_opmode_blocking);
       jHandles(i)=daHandle;
   end
end
%% %%%%%%%%%%%%%% Set joint speed %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
relative_joint_speed = 0.05;
vrep_joint_original_speeds = {1.9199, 1.9199, 2.2340, 2.2340, 3.5605, 3.2114, 3.2114}; % taken from joint
"Upper velocity limit" field [rad/s]
for i = 1:7
    errCode = vrep.simxSetObjectFloatParameter( ...
        clientID, jHandles(i),  vrep.sim_jointfloatparam_upper_limit, ...
        vrep_joint_original_speeds{i} * relative_joint_speed, vrep.simx_opmode_oneshot );
    if errCode ~= 0 && errCode ~= 1
        vrep.simxPauseSimulation(clientID, vrep.simx_opmode_oneshot_wait);
        error("setting simulated joint speed failed, error code " + num2str(errCode))
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% définition de la position articulaire initial en Radian
position_initial= [0.0; 0.0; 0.0; -pi/2; 0.0; pi/2; 0.0];
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% envoyer la configuration target initial pour le robot simulé sous CoppeliaSim
joint_trajectory_flatten = reshape(position_initial', 1, []);
errCode = vrep.simxSetStringSignal(clientID, 'trajectory', vrep.simxPackFloats(joint_trajectory_flatten),
vrep.simx_opmode_oneshot_wait);

%% %% %%%%%%%%%%%%%%%%%%% get joint angle %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
joint_angle = zeros(1,7);
for ii = 1:7
   [ress, joint_angle(ii)] = vrep.simxGetJointPosition(clientID, jHandles(ii),
vrep.simx_opmode_blocking);
end
initial_guess = joint_angle;
%% %%%%%%%%%%%%%%%%% chargement du modèle robot %%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%% et def des matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

robot_rbt = loadrobot("kukaIiwa14");
robot_rbt.DataFormat = "row";
%% définir la matrice de transformation entre flange et point effecteur désiré:
```

```
% matrice de transformation entre flange et capteur, à modifier la valeur
% Tef_flange(3,4) selon la taille de l'outil.
Tef_flange = [0 -1 0 0;...
              1  0 0 0;...
              0  0 1 0;...
              0  0 0 1];
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026];
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8), Tef_flange);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% lancer la simulation sur simulink
open_system('Admittance_CoppeliaSim.slx');
out=sim('Admittance_CoppeliaSim.slx');

% %%%%%%% récupération des positions articulaires %%%%%%%%%%%%%%%%%%%%%%%%
config = out.joint_position;
joint_positions = config.Data;


% Réorganise les dimensions
joint_positions = permute(joint_positions, [3, 1, 2]);
joint_positions = reshape(joint_positions, [size(joint_positions, 1), size(joint_positions, 2)]);


% envoie de la solution de configuration trouvée par le solveur pour le robot simulé sous CoppeliaSim
joint_trajectory_flatten = reshape(joint_positions', 1, []);
errCode = vrep.simxSetStringSignal(clientID, 'trajectory', vrep.simxPackFloats(joint_trajectory_flatten),
vrep.simx_opmode_oneshot_wait);



%% fin de simulation
vrep.delete();



function correctComponentLength(rbt, componentIndex, lengthAddition, Tef_flange)

    component = rbt.Bodies(componentIndex);
    tform = component{1}.Joint.JointToParentTransform;

    tform = tform*Tef_flange;

    % find the length line position in the matrix (it differs following the component)
    col = 4;
    line = 3;
    if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
        line = 2;
    end

    old_length = tform(line, col);
    new_length = old_length + lengthAddition;

    disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

    tform(line, col) = new_length;
    setFixedTransform(component{1}.Joint, tform);
    replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end
```



Boucle de contrôle en admittance sur Simulink

Annexe L : Script boucle de contrôle en admittance Virtuelle sous CoppeliaSim avec mesures capteur

```
%% Ce script permet de simuler le robot sous CoppliaSim et communiquer
%% avec le capteur en meme temps pour controler le robot simulé en boucle
%% d'admittance.
% @author : Tir abd elhafid


%% utilisation :
% 1 - lancer la simulation du robot sur CoppeliaSim
% 2 - le temps et freq d'acquisition du capteur sont défini ici (attention
%     le temps de simulation sur Simulink doit etre le meme que le temps
%     d'acquisition du capteur)
% 3 - lancer ce script et et attendre 1 sec environ.
% 4 - pousser doucement avec la main sur l'effecteur selon une direction choisi
% 4 - une fois l'acquisition des mesures est terminée, relancer la
%     simulation sur CoppeliaSim ('start') et remarquer le déplacement du robot selon
%     la direction d'application du force ou moments.

%% Remarque :
% les paramètres M, K, C dans Simulink peuvent etre modifié selon la dynamique
% désiré (i.e proportionnalité entre force appliqué et déplacement désiré)

clear
close all
clc
%% %%%%%%%%%%%%%%%%%% Connexion avec CoppeliaSim  %%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%% et envoie à une position initial %%%%%%%%%%%%%%%%%%%%
vrep=remApi('remoteApi');
vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
% vérification de la connexion:
if(clientID>-1)
   disp('connected');
   jHandles=zeros (7,1);
   % récupération des identifiants des articulations
   for i=1:7
       s=['LBR_iiwa_14_R820_joint',num2str(i)];
       [ress,daHandle]=vrep.simxGetObjectHandle (clientID, s, vrep.simx_opmode_blocking);
       jHandles(i)=daHandle;
   end
end
%% %%%%%%%%%%%%%%%% Set joint speed %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
relative_joint_speed = 0.05;
vrep_joint_original_speeds = {1.9199, 1.9199, 2.2340, 2.2340, 3.5605, 3.2114, 3.2114}; % taken from joint
"Upper velocity limit" field [rad/s]
for i = 1:7
    errCode = vrep.simxSetObjectFloatParameter( ...
        clientID, jHandles(i),  vrep.sim_jointfloatparam_upper_limit, ...
        vrep_joint_original_speeds{i} * relative_joint_speed, vrep.simx_opmode_oneshot );
    if errCode ~= 0 && errCode ~= 1
        vrep.simxPauseSimulation(clientID, vrep.simx_opmode_oneshot_wait);
        error("setting simulated joint speed failed, error code " + num2str(errCode))
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% définition de la position articulaire initial en Radian
position_initial= [0.0; 0.0; 0.0; -pi/2; 0.0; pi/2; 0.0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% envoyer la configuration target initial pour le robot simulé sous CoppeliaSim
for i=1:7
   vrep.simxSetJointTargetPosition(clientID,jHandles(i),position_initial(i),vrep.simx_opmode_oneshot);

end
```

```
%% %%%%%%%%%%%%%%%%%%% get joint angle %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
joint_angle = zeros(1,7);
for ii = 1:7
   [ress, joint_angle(ii)] = vrep.simxGetJointPosition(clientID, jHandles(ii),
vrep.simx_opmode_blocking);
end

initial_guess = joint_angle;


%% %%%%%%%%%%%%%%%%% Connexion Capteur %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% load  Library   MEGSV86x64.dll & MEGSV86x64.h (Reading data)
com =5; % definition du port USB
if ~libisloaded('MEGSV86x64')  % chargement des libreries
    loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
end
%% Activation des canaux
[extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
calllib('MEGSV86x64', 'GSV86startTX', com);

calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la mémoire
de l'appareil.
%% Calibration
com = 5;
dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
DatFilePath = libpointer('cstring',dat);
[er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% initialisation de fréquence d'acquisition
freq = 10; % Hz !!!!!!!!!!!!!!!!!!!!!!
calllib('MEGSV86x64', 'GSV86setFrequency', 5, freq);

calllib('MEGSV86x64', 'GSV86setZero', 5, 0); % Tarage initial

%% Lecture des données
tic %startTime = datetime('now');
stop = false;
i = 0;
%%%%%%%%%%%%%%%%
time_stop = 10; % definition du temps d'acquisition

num_iterations = time_stop * freq;
A = zeros(num_iterations, 8);
T = zeros(num_iterations,1);
% D = zeros(num_iterations,2);
D1 = zeros(num_iterations,2);D4 = zeros(num_iterations,2);
D2 = zeros(num_iterations,2);D5 = zeros(num_iterations,2);
D3 = zeros(num_iterations,2);D6 = zeros(num_iterations,2);
pause('off')
vec_zero = [0 0 0 0 0 0 0 0];
s = 0;

%%%%%%%%%%%%%%%%
while ~stop

    t = toc; %seconds(datetime('now') - startTime)

    [error, data]=calllib('MEGSV86x64','GSV86readMultiple',5, 0, libpointer('doublePtr',zeros(1,8)),...
        8, libpointer('int32Ptr',int32(0)), libpointer('int32Ptr',int32(0)) );

    if ~isequal(data, vec_zero)
        s = s + 1;
        A(s, :)=data % matrice des mesures (s,8)
```

```
        T(s) = t; % vecteur du temps

        % D(:, 1) = T;
        % D(:, 2) = A(:, 3);
        D1(:, 1)=T;D2(:, 1)=T;D3(:, 1)=T;
        D4(:, 1)=T;D5(:, 1)=T;D6(:, 1)=T;

        D1(:, 2) = A(:, 1);D2(:, 2) = A(:, 2);D3(:, 2) = A(:, 3);
        D4(:, 2) = A(:, 4);D5(:, 2) = A(:, 5);D6(:, 2) = A(:, 6);
    end
    % condition pour arreter l'acquisition
    if t > time_stop
        stop = true;
    end
end
disp(s); % affichage de nombre de mesures lus
% data_timeseries = timeseries(D(:, 2), D(:, 1));
% assignin('base', 'data_timeseries', data_timeseries);
data_timeseries1 = timeseries(D1(:, 2), D1(:, 1));
assignin('base', 'data_timeseries1', data_timeseries1);
data_timeseries2 = timeseries(D2(:, 2), D2(:, 1));

assignin('base', 'data_timeseries2', data_timeseries2);
data_timeseries3 = timeseries(D3(:, 2), D3(:, 1));
assignin('base', 'data_timeseries3', data_timeseries3);
data_timeseries4 = timeseries(D4(:, 2), D4(:, 1));
assignin('base', 'data_timeseries4', data_timeseries4);
data_timeseries5 = timeseries(D5(:, 2), D5(:, 1));
assignin('base', 'data_timeseries5', data_timeseries5);
data_timeseries6 = timeseries(D6(:, 2), D6(:, 1));
assignin('base', 'data_timeseries6', data_timeseries6);

%% Libération des ressources
calllib('MEGSV86x64', 'GSV86release', com)
unloadlibrary('MEGSV86x64')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %% lancer la simulation sur simulink
open_system('Admittance_virtuelle.slx');
out=sim('Admittance_virtuelle.slx');

%% %%%%%%% récupération des positions articulaires %%%%%%%%%%%%%%%%%%%%%%%%%
config = out.joint_position;
joint_positions = config.Data;


% Réorganise les dimensions
joint_positions = permute(joint_positions, [3, 1, 2]);
joint_positions = reshape(joint_positions, [size(joint_positions, 1), size(joint_positions, 2)]);


%% envoie de la solution de configuration trouvée par le solveur pour le robot simulé sous CoppeliaSim
joint_trajectory_flatten = reshape(joint_positions', 1, []);
errCode = vrep.simxSetStringSignal(clientID, 'trajectory', vrep.simxPackFloats(joint_trajectory_flatten),
vrep.simx_opmode_oneshot_wait);


%% fin de simulation
vrep.delete();
```

Annexe M : Script boucle de contrôle en admittance sur le robot réel

```
clear
close all
clc

%% %%%%%%%%%%%%%%%%%%% Connexion robot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Create the robot object
ip='172.31.1.147';
arg1=KST.LBR14R820;
arg2=KST.Medien_Flansch_Touch_pneumatisch;
Tef_flange=eye(4);
iiwa=KST(ip,arg1,arg2,Tef_flange);

%% Start a connection with the server
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);

%% %%%%%%%%%%%%%%%%%%% envoie à la position initial %%%%%%%%%%%%%%%%%%%%%%
% joint_initial = [ 0.0       0.0   0.0    -pi/2   0.0    pi/2  0.0];
% relVel=0.2;
% iiwa.nonBlocking_movePTPTrajectoryJoint(joint_initial, relVel);

%% %%%%%%%%%%%%%%%%%% chargement du modèle robot %%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%% et def des matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
robot_rbt = loadrobot("kukaIiwa14");
robot_rbt.DataFormat = "row";
Tef_flange = [0 -1 0 0;...
              1  0 0 0;...
              0  0 1 0;...
              0  0 0 1];
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
% joint_length_additions(1, 9) =  "taille de l'outil souhaité" selon Z-flange
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8) + joint_length_additions(1, 9),
Tef_flange);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com =5; % definition du port USB
if ~libisloaded('MEGSV86x64')  % chargement des libreries
    loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
end
%% Activation des canaux
[extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
calllib('MEGSV86x64', 'GSV86startTX', com);

calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la mémoire
de l'appareil.

%% calibration
com = 5;
dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
DatFilePath = libpointer('cstring',dat);
[er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );

calllib('MEGSV86x64', 'GSV86setZero', 5, 0); % Tarage initial
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% cas 1 : mettre time_stop(capteur + simulink) = 10 sec

stop = false;
while ~stop
    % récuperer la position actuel des joints
```

```
    finished = iiwa.isMotionFinished();
    if finished
        cpos = iiwa.getJointsPos();
        position_actuel = [cell2mat(cpos(1)); cell2mat(cpos(2)); cell2mat(cpos(3));...
                    cell2mat(cpos(4)); cell2mat(cpos(5)); cell2mat(cpos(6)); cell2mat(cpos(7))];
        break;
    end
end
% -----------------
time_stop = 10; % temps d'acquisition des mesures

freq = 10; % frequence d'acquisition capteur
move_robot(iiwa, time_stop, freq);

% -----------------
%% turn off server
stop = false;
while ~stop
    finished = iiwa.isMotionFinished();
    if finished
        iiwa.net_turnOffServer();
        break;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% fonction pour faire les meures des forces et moments, envoyer
% à Simulink pour simuler la boucle d'admittance et récuperer les position
% articulaire à la fin de la boucle, puis envoyer la commande au robot.
function move_robot(iiwa, time_stop, freq)

        %% %%%%%%%%%%%%%%%%% Capteur %%%%%%%%%%%%%%%%%%%%%%%%%%
        %% %%%%%%%%%%%%%%%%%  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% load  Library   MEGSV86x64.dll & MEGSV86x64.h (Reading data)
        com =5; % definition du port USB
        if ~libisloaded('MEGSV86x64')  % chargement des libreries
            loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
        end
        %% Activation des canaux
        [extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
        calllib('MEGSV86x64', 'GSV86startTX', com);

        calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
        calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la
mémoire de l'appareil.

        com = 5;
        dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
        DatFilePath = libpointer('cstring',dat);
        [er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% initialisation de fréquence d'acquisition
        calllib('MEGSV86x64', 'GSV86setFrequency', 5, freq);

        %%%%%%%%%%%%%%%%%
        %% Lecture des données
        tic %startTime = datetime('now');
        stop = false;
        %%%%%%%%%%%%%%%%%
        num_iterations = time_stop * freq;
        A = zeros(num_iterations, 8); % vecteur stockage des mesures capteur
        T = zeros(num_iterations,1); % vecteur stockage du temps
        % initialisation des vecteurs de stockage de chaque mesures forces
        % et moments
        D1 = zeros(num_iterations,2);D4 = zeros(num_iterations,2);
        D2 = zeros(num_iterations,2);D5 = zeros(num_iterations,2);
        D3 = zeros(num_iterations,2);D6 = zeros(num_iterations,2);
```

```
        pause('off')
        vec_zero = [0 0 0 0 0 0 0 0];
        s = 0;

        %%%%%%%%%%%%%%
        %% lancer la simulation sur simulink
        open_system('Admittance_simulink.slx');

        disp(['Début mesures capteur pour un délai : ', num2str(time_stop) , ' secondes']);
        while ~stop  % or while s < num_iterations
            t = toc; %seconds(datetime('now') - startTime)

            [~, data]=calllib('MEGSV86x64','GSV86readMultiple',5, 0,
libpointer('doublePtr',zeros(1,8)),...
                8, libpointer('int32Ptr',int32(0)), libpointer('int32Ptr',int32(0)) );

            if ~isequal(data, vec_zero)
                s = s + 1;
                A(s, :) = data; % matrice des mesures (s,8)
                T(s) = t; % vecteur du temps


                D1(:, 1)=T;D2(:, 1)=T;D3(:, 1)=T;
                D4(:, 1)=T;D5(:, 1)=T;D6(:, 1)=T;

                D1(:, 2) = A(:, 1);D2(:, 2) = A(:, 2);D3(:, 2) = A(:, 3);
                D4(:, 2) = A(:, 4);D5(:, 2) = A(:, 5);D6(:, 2) = A(:, 6);

            end
            % assigner les mesures aux variables dans simulink
            data_timeseries1 = timeseries(D1(:, 2), D1(:, 1));
            assignin('base', 'data_timeseries1', data_timeseries1);
            data_timeseries2 = timeseries(D2(:, 2), D2(:, 1));
            assignin('base', 'data_timeseries2', data_timeseries2);
            data_timeseries3 = timeseries(D3(:, 2), D3(:, 1));
            assignin('base', 'data_timeseries3', data_timeseries3);
            data_timeseries4 = timeseries(D4(:, 2), D4(:, 1));
            assignin('base', 'data_timeseries4', data_timeseries4);
            data_timeseries5 = timeseries(D5(:, 2), D5(:, 1));
            assignin('base', 'data_timeseries5', data_timeseries5);
            data_timeseries6 = timeseries(D6(:, 2), D6(:, 1));
            assignin('base', 'data_timeseries6', data_timeseries6);
            % disp(data_timeseries.Data);

            % condition pour arreter l'acquisition
            if t > time_stop
                stop = true;
            end

        end
        disp('fin mesures capteur');

        out=sim('Admittance_simulink.slx'); % établir la simulation de la boucle dans simulink


        %%%%% récupération des positions articulaires %%%%%%%%%%%%%%%%%%%%%%%%%
        config = out.joint_position;
        joint_positions = config.Data;


        % Réorganise les dimensions
        joint_positions = permute(joint_positions, [3, 1, 2]);
        joint_positions = reshape(joint_positions, [size(joint_positions, 1), size(joint_positions, 2)]);
        disp(size(joint_positions));
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % envoyer les positions articulaires au robot
        relVel=0.1;
        disp('Appuyer pour effectuer le mouvement');
```

```
        % joint_positions = remove_repeated_lines(joint_positions, 1e-3);
        joint_positions = calculerMoyenneParLigne(joint_positions);
        iiwa.nonBlocking_movePTPTrajectoryJoint( joint_positions, relVel);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %% Libération des ressources
        calllib('MEGSV86x64', 'GSV86release', com);
        unloadlibrary('MEGSV86x64');

end



function correctComponentLength(rbt, componentIndex, lengthAddition, Tef_flange)

            component = rbt.Bodies(componentIndex);
            tform = component{1}.Joint.JointToParentTransform;

            tform = tform*Tef_flange;

            % find the length line position in the matrix (it differs following the component)
            col = 4;
            line = 3;
            if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
                line = 2;
            end

            old_length = tform(line, col);
            new_length = old_length + lengthAddition;


            disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

            tform(line, col) = new_length;
            setFixedTransform(component{1}.Joint, tform);
            replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end



function moyenne = calculerMoyenneParLigne(matrice)
    nbLignes = size(matrice, 1);
    moyenne = zeros(1, size(matrice, 2));

    for i = 1:nbLignes
        moyenne = moyenne + matrice(i, :);
    end

    moyenne = moyenne / nbLignes;
end
```

Annexe N : Script boucle de contrôle en admittance sur le robot réel (approche incrémentale)

```
clear
close all
clc

%% %%%%%%%%%%%%%%%%%%%% Connexion robot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Create the robot object
ip='172.31.1.147';
arg1=KST.LBR14R820;
arg2=KST.Medien_Flansch_Touch_pneumatisch;
Tef_flange=eye(4);
iiwa=KST(ip,arg1,arg2,Tef_flange);

%% Start a connection with the server
flag=iiwa.net_establishConnection();
if flag==0
  return;
end
pause(1);

%% %%%%%%%%%%%%%%%%%%%% envoie à la position initial %%%%%%%%%%%%%%%%%%%%%%
% joint_initial = [ 0.0       0.0    0.0     -pi/2    0.0     pi/2   0.0];
% relVel=0.2;
% iiwa.nonBlocking_movePTPTrajectoryJoint(joint_initial, relVel);

%% %%%%%%%%%%%%%%%%%%% chargement du modèle robot %%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%% et def des matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
robot_rbt = loadrobot("kukaIiwa14");
robot_rbt.DataFormat = "row";
Tef_flange = [0 -1 0 0;...
              1  0 0 0;...
              0  0 1 0;...
              0  0 0 1];
joint_length_additions = [0, 0, 0, 0, 0, 0, 0, 0.026, 0.0];
% joint_length_additions(1, 9) =  "taille de l'outil souhaité" selon Z-flange
correctComponentLength(robot_rbt, 10, joint_length_additions(1, 8) + joint_length_additions(1, 9),
Tef_flange);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% load  Library   MEGSV86x64.dll & MEGSV86x64.h (Reading data)
com =5; % definition du port USB
if ~libisloaded('MEGSV86x64')  % chargement des libreries
    loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
end
%% Activation des canaux
[extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
calllib('MEGSV86x64', 'GSV86startTX', com);

calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la mémoire
de l'appareil.

com = 5;
dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
DatFilePath = libpointer('cstring',dat);
[er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );

calllib('MEGSV86x64', 'GSV86setZero', 5, 0); % Tarage initial


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cas 2 : dans une boucle (mesures chaque 1 sec)
% stopp = false;
% while ~stopp
for i = 1:10
```

```
    stop = false;
    while ~stop
        finished = iiwa.isMotionFinished();
        if finished
            cpos = iiwa.getJointsPos();
            position_actuel = [cell2mat(cpos(1)); cell2mat(cpos(2)); cell2mat(cpos(3));...
                       cell2mat(cpos(4)); cell2mat(cpos(5)); cell2mat(cpos(6)); cell2mat(cpos(7))];
            break;

        end
    end
% -----------------
    time_stop = 1;
    freq = 10;
    joint_positions = move_robot(iiwa, time_stop, freq);

end

% -----------------
%% turn off server
stop = false;
while ~stop
    finished = iiwa.isMotionFinished();
    if finished
        iiwa.net_turnOffServer();
        break;
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = move_robot(iiwa, time_stop, freq)

        %% %%%%%%%%%%%%%%%%% Capteur %%%%%%%%%%%%%%%%%%%%%%%%%%
        %% %%%%%%%%%%%%%%%%%  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        com =5; % definition du port USB
        if ~libisloaded('MEGSV86x64')  % chargement des libreries
            loadlibrary('.\MEGSV86x64.DLL', '.\MEGSV86x64.h')
        end
        %% Activation des canaux
        [extendet] = calllib('MEGSV86x64', 'GSV86actExt', com);
        calllib('MEGSV86x64', 'GSV86startTX', com);

        calllib('MEGSV86x64','GSV86clearDeviceBuf',com);  %Clear and reset device measuring values buffer
        calllib('MEGSV86x64','GSV86clearDLLbuffer',com);  %supprime toutes les données stockées dans la
mémoire de l'appareil.

        com = 5;
        dat = "<drive>:\\<path\\<K6D-CalibrationMatrix_Plus_19405258>.dat";
        DatFilePath = libpointer('cstring',dat);
        [er_calib]=calllib('MEGSV86x64','GSV86writeFTsensorFromFile', com,1, DatFilePath );


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% initialisation de fréquence d'acquisition
        calllib('MEGSV86x64', 'GSV86setFrequency', 5, freq);
        %% Lecture des données
        tic %startTime = datetime('now');
        stop = false;

        %%%%%%%%%%%%%%%%
        num_iterations = time_stop * freq;
        A = zeros(num_iterations, 8);
        T = zeros(num_iterations,1);
        D1 = zeros(num_iterations,2);D4 = zeros(num_iterations,2);
        D2 = zeros(num_iterations,2);D5 = zeros(num_iterations,2);
        D3 = zeros(num_iterations,2);D6 = zeros(num_iterations,2);
```

```
pause('off')
vec_zero = [0 0 0 0 0 0 0 0];
s = 0;

%%%%%%%%%%%%%%
%% lancer la simulation sur simulink
open_system('Admittance_simulink.slx');


disp(['Début mesures capteur pour un délai : ', num2str(time_stop) , ' secondes']);
while ~stop % or while s < num_iterations
    t = toc; %seconds(datetime('now') - startTime)

    [~, data]=calllib('MEGSV86x64','GSV86readMultiple',5, 0,
libpointer('doublePtr',zeros(1,8)),...
        8, libpointer('int32Ptr',int32(0)), libpointer('int32Ptr',int32(0)) );

    if ~isequal(data, vec_zero)
        s = s + 1;

        A(s, :) = data; % matrice des mesures (s,8)
        T(s) = t; % vecteur du temps

        D1(:, 1)=T;D2(:, 1)=T;D3(:, 1)=T;
        D4(:, 1)=T;D5(:, 1)=T;D6(:, 1)=T;

        D1(:, 2) = A(:, 1);D2(:, 2) = A(:, 2);D3(:, 2) = A(:, 3);
        D4(:, 2) = A(:, 4);D5(:, 2) = A(:, 5);D6(:, 2) = A(:, 6);

    end
    data_timeseries1 = timeseries(D1(:, 2), D1(:, 1));
    assignin('base', 'data_timeseries1', data_timeseries1);
    data_timeseries2 = timeseries(D2(:, 2), D2(:, 1));
    assignin('base', 'data_timeseries2', data_timeseries2);
    data_timeseries3 = timeseries(D3(:, 2), D3(:, 1));
    assignin('base', 'data_timeseries3', data_timeseries3);
    data_timeseries4 = timeseries(D4(:, 2), D4(:, 1));
    assignin('base', 'data_timeseries4', data_timeseries4);
    data_timeseries5 = timeseries(D5(:, 2), D5(:, 1));
    assignin('base', 'data_timeseries5', data_timeseries5);
    data_timeseries6 = timeseries(D6(:, 2), D6(:, 1));
    assignin('base', 'data_timeseries6', data_timeseries6);
    % disp(data_timeseries.Data);

    % condition pour arreter l'acquisition
    if t > time_stop
        stop = true;
    end

end
disp('fin mesures capteur');

out=sim('Admittance_simulink.slx');
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%% récupération des positions articulaires %%%%%%%%%%%%%%%%%%%%%%%%%
config = out.joint_position;
joint_positions = config.Data;


% Réorganise les dimensions
joint_positions = permute(joint_positions, [3, 1, 2]);
y = reshape(joint_positions, [size(joint_positions, 1), size(joint_positions, 2)]);
% disp(joint_positions);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

relVel=0.1;
disp('Appuyer pour effectuer le mouvement');
```

```
        % y = remove_repeated_lines(joint_positions, 1e-3);

        y = calculerMoyenneParLigne(joint_positions);
        iiwa.nonBlocking_movePTPTrajectoryJoint( y, relVel);


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %% Libération des ressources
        calllib('MEGSV86x64', 'GSV86release', com);
        unloadlibrary('MEGSV86x64');


end



function correctComponentLength(rbt, componentIndex, lengthAddition, Tef_flange)

            component = rbt.Bodies(componentIndex);
            tform = component{1}.Joint.JointToParentTransform;

            tform = tform*Tef_flange;

            % find the length line position in the matrix (it differs following the component)
            col = 4;
            line = 3;
            if (mod(componentIndex, 2) == 1) || (tform(line, col) == 0)
                line = 2;

            end

            old_length = tform(line, col);
            new_length = old_length + lengthAddition;

            disp("Component " + num2str(componentIndex-1) + " length: " + num2str(old_length) + " -> " +
num2str(new_length));

            tform(line, col) = new_length;
            setFixedTransform(component{1}.Joint, tform);
            replaceJoint(rbt, component{1}.Name, component{1}.Joint);
end




function moyenne = calculerMoyenneParLigne(matrice)
    % Obtient le nombre de lignes dans la matrice
    nbLignes = size(matrice, 1);
    % Initialise un vecteur de moyenne avec des zéros, ayant la même longueur que le nombre de colonnes
dans la matrice
    moyenne = zeros(1, size(matrice, 2));
    % Parcourt chaque ligne de la matrice
    for i = 1:nbLignes
        % Ajoute les éléments de la ligne actuelle au vecteur de moyenne
        moyenne = moyenne + matrice(i, :);
    end
    % Divise le vecteur de moyenne par le nombre de lignes pour obtenir la moyenne réelle
    moyenne = moyenne / nbLignes;
end
```

Annexe O : Problème drift des mesures du capteur