TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

CliCours.com

Clicours.COM

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CNC	Computer Numerical Control
CNN	Convolutional Neural Network
F/T	Force-Torque
GT	Ground-thruth
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
LSTM	Long Short-Term Memory
NN	Neural Network
PAP	Pick-and-place
PCA	Principal Component Analysis
R-CNN	Region based Convolutional Neural Network
RGB	Red-Green-Blue
VGG	Visual Geometry Group

LISTE DES SYMBOLES ET UNITÉS DE MESURE

cm	Centimètre
Hz	Hertz
N	Newton
m	Mètre
mm	Millimètre
m/s^2	Mètre par seconde carrée
Nm	Newton-mètre
Ns/m	Newton-seconde par mètre
rad/s^2	Radiant par seconde carrée
V	Vélocité

INTRODUCTION

Depuis maintenant quelques années, la venue de la robotique collaborative a changé considérablement la perception de la robotique dans le monde manufacturier (Zinn *et al.* (2004)). Avec cette nouvelle technologie, l'accès à l'automatisation d'une tâche pour un processus impliquant des interactions humains-robots a été grandement facilité, expliquant la popularité grandissante de ces types de robot sur des chaînes de montage dans de moyennes et petites entreprises. Ces appareils sont souvent utilisés pour faire des tâches répétitives aux côtés d'ouvriers pour assister à la production.

Figure 0.1 Utilisation typique d'un robot collaboratif, le service d'une CNC. Tirée de https ://www.universal-robots.com/case-stories/plc-industries/

Le service d'une CNC est un exemple d'usage très populaire de la robotique collaborative. À l'aide d'un robot, une entreprise peut automatiser le remplissage, le vidage et la palettisation des pièces sortant d'une CNC et ainsi utiliser l'opérateur pour des tâches plus critiques dans la production.

La popularisation de l'utilisation de robot collaboratif tente de régler le problème grandissant du manque de main-d'oeuvre dans le domaine manufacturier, mais génère aussi plusieurs autres

difficultés. Le nombre grandissant de robot dans une entreprise exécutant un nombre grandissant de tâches différentes affecte alors grandement la charge de travail de programmation. Pour que le robot fonctionne comme prévu, il faut généralement un programmeur hautement qualifié pour programmer manuellement les actions du robot en fonction des contraintes imposées par son environnement ; une ressource rare qui est généralement inaccessible au PME et qui augmente grandement le coût d'automatisation. Si vous ajoutez plusieurs robots à l'équation, la programmation devient une tâche de longue haleine nécessitant une connaissance précise des robots et des outils qu'ils utiliseront.

Face à ce problème, l'intérêt pour la "programmation par démonstration", un domaine de recherche en robotique, a gagné une audience croissante (Argall *et al.* (2009)). Les travaux impliquant la programmation par démonstration consistent généralement à faire une démonstration (physique ou simulée) d'une tâche à un robot pour que celui-ci répète la démonstration, enlevant du même coup le besoin de programmer manuellement la tâche à effectuer. Un exemple simple de programmation par démonstration pourrait être de guider physiquement le robot, à l'aide d'un contrôle par impédance, à faire une tâche pendant que celui-ci enregistre la trajectoire parcourue pour ensuite répéter exactement la démonstration effectuée. Un tel programme de contrôle est présent dans la suite *Force Copilot* de *Robotiq* (Robotiq (2018)), où l'on peut contrôler un robot de la compagnie *Universal Robot* à l'aide d'un capteur d'effort. Cette méthode nous permet non seulement d'enlever le besoin de programmer certains segments de la routine du robot, mais permet aussi à des personnes non expertes en robotique de programmer une tâche facilement et rapidement.

Bien que cette méthode de programmation facilite le travail pour certaines tâches faciles, la démonstration d'autres tâches nécessitant une plus grande précision peut devenir très ardue, voire impossible. Dans notre exemple, puisque le robot n'enregistre que les données spatio-temporelles de la démonstration, c'est-à-dire la trajectoire empruntée, des tâches qui requièrent

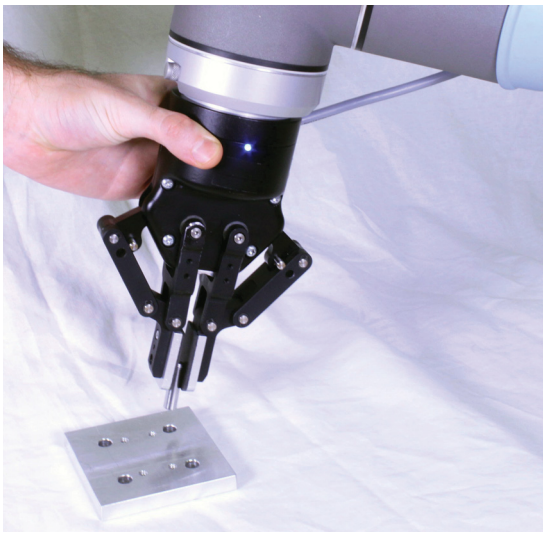


Figure 0.2 Démonstration d'une tâche d'insertion à l'aide d'un contrôle par impédance.

d'appliquer des forces spécifiques pourraient être difficiles à exécuter à partir des données enregistrées lors de la démonstration.

Dans le cadre de ce projet, nous étudierons la démonstration de tâche d'insertion et comment un système pourrait répéter ce type de tâche à l'aide d'une démonstration simple. Il n'est pas rare de voir des robots faire des insertions dans une tâche répétitive. Par exemple, un robot collaboratif pourrait insérer des vis dans des cavités pour qu'un ouvrier puisse ensuite continuer l'assemblage, ou encore, un robot collaboratif pourrait être utilisé pour insérer des composants électroniques sur un circuit imprimé pour que ceux-ci soient soudés. Ces tâches sont difficilement répétables lorsque démontrées physiquement au robot puisque plusieurs incertitudes sont impliquées dans l'exécution de la tâche : la pièce pourrait avoir bougé pendant la préhension entre la démonstration et l'exécution future, ou encore le trou de l'assemblage pourrait avoir bougé légèrement entre la démonstration et l'exécution résultant ainsi en une erreur d'exécution.

Dans le cadre de ce projet, nous proposons, entre autres, de détecter l'exécution d'une tâche d'insertion pendant la démonstration effectuée par l'opérateur pour ainsi être capable de trouver

Clicours.COM

une solution locale à cette tâche lors de la ré-exécution de la démonstration par le robot. Nous croyons qu'en détectant une tâche d'insertion et en appliquant une solution locale que nous pourrions augmenter le nombre de tâches enseignables à l'aide de technique d'apprentissage par démonstration.

Dans ce document, nous aborderons le développement d'une solution permettant de faire non seulement la démonstration d'une tâche d'insertion, mais de répéter celle-ci d'une manière intelligente et optimisée. Nous aborderons d'abord les travaux récents liés au sujet de la programmation par démonstration et des solutions pour le problème de l'insertion en robotique. Nous discuterons du développement d'un contrôle humain-robot destiné à faire la démonstration de tâches contact avec l'espace de travail du robot. Nous analyserons ensuite le développement d'un prototype fonctionnel de détection de tâche d'insertion à l'aide de signaux de force générés pendant la démonstration de la tâche. Nous continuerons cette analyse par le développement d'une solution robuste pour remplacer une tâche d'insertion détectée. Finalement, une solution complète d'autogénération de routines pour la réalisation d'une tâche d'insertion à partir d'une démonstration est expliquée et examinée dans le but de mettre de l'avant les développements futurs de la programmation par démonstration dans l'industrie manufacturière.

CHAPITRE 1

REVUE DE LITTERATURE

1.1 Enseignement d'une tâche d'insertion

Dans la recherche de moyens d'éviter de programmer manuellement un robot chaque fois qu'il doit apprendre une nouvelle tâche, plusieurs études ont tourné autour du concept de « programmation par démonstration », terme développé dans les recherches de Billard *et al.* (2008). Ce terme est utilisé dans plusieurs champs de recherches en robotique : pour les interactions humains-robots, le contrôle et le comportement des robots ou encore pour la vision par ordinateur visant le domaine de la robotique collaborative. Ces travaux visent principalement à construire des solutions à partir d'exemples, bon ou mauvais, de l'exécution d'une tâche. Cette méthode d'apprentissage par imitation vise aussi à réduire la tâche fastidieuse de programmer des robots manuellement en minimisant la quantité de code à générer manuellement par l'utilisateur pour enseigner une tâche.

Par exemple, Peternel *et al.* (2015) ont développé une façon de programmer une tâche d'assemblage par démonstration en contrôlant un robot à distance avec une interface haptique. Lors de l'enseignement de la tâche, les raideurs utilisées par l'opérateur ainsi que les trajectoires empruntées pour réaliser la tâche sont encodées sous la forme de DMP (Dynamic mouvement primitive : Schaal (2006)). Cette manière permet d'encoder la trajectoire à effectuer en épousant un nombre déterminé de gaussienne pour atteindre un point final. Cette procédure est particulièrement pertinente dans les situations où la présence directe d'un travailleur n'est pas possible. Le mouvement de l'enseignant peut être couplé à distance au contrôle du robot et enregistré pour une répétition future. Bien que la solution proposée ici soit prometteuse, nous considérons que l'ajout d'un appareil haptique pour faire l'enseignement d'une tâche n'est pas une solution souhaitée puisque celle-ci ajoute de l'équipement qui n'apporte pas de valeur directe à l'utilisateur.

Une solution qui pourrait respecter un peu plus les conditions vécues par une entreprise utilisant un robot collaboratif pourrait être de faire l'enseignement de la trajectoire d'une tâche à l'aide d'un système de démonstration kinesthésique. Cette méthode consiste à démontrer une tâche à l'aide d'un contact physique entre l'utilisateur et le robot. Dans les travaux de Hersch *et al.* (2008), une solution utilisant des démonstrations kinesthésiques est utilisée pour enregistrer la tâche à effectuer. L'utilisateur montre physiquement l'emplacement de la pièce à prendre au robot ainsi que la trajectoire à prendre pour ensuite générer une solution basée sur ces démonstrations. Nous croyons qu'une telle méthode est davantage pertinente lorsque utilisée avec un robot collaboratif puisque le contact direct avec le robot est possible.

Dans le but de faire des démonstrations kinesthésiques, un système de contrôle humain-robot doit être développé. Quelques solutions préexistantes peuvent être utilisées dans ce cas-ci ; par exemple, le système de "free-motion" fourni avec les robots Universal Robot (le robot utilisé dans le cadre de notre projet) utilise la lecture des courants envoyés au joint du robot pour détecter si un déplacement est demandé par l'utilisateur. Cette méthode simple permet de bouger le robot physiquement dans son espace de travail. Par contre, il est difficile de faire des mouvements fins, telle qu'une tâche d'insertion, avec cette méthode. Une solution plus adéquate serait d'utiliser un contrôle par impédance variable, tel que proposé par Duchaine & Gosselin (2007). Cette méthode permet de contrôler un robot à l'aide des forces mesurées par capteur d'effort. En plus d'avoir une mesure plus fine de la force appliquée par l'utilisateur, Duchaine & Gosselin (2007) proposent d'utiliser l'intention mesurée de l'utilisateur pour changer les paramètres du contrôle et ainsi faciliter le déplacement du manipulateur.

1.2 Solution pour l'exécution d'une tâche d'insertion

Puisqu'une tâche d'insertion requiert une position précise ainsi qu'un profil de force pour être capable de pénétrer l'assemblage sans tout casser, il nous est important de trouver une méthode d'insertion adéquate pour ne pas rejouer naïvement la tâche démontrée puisque plusieurs facteurs peuvent avoir changé entre la démonstration et l'exécution par le robot pouvant causer ainsi des erreurs.

De nombreux travaux ont tenté de surmonter le défi de l'insertion serrée. Les méthodes proposées incluent l'utilisation du retour d'effort, tel que fait par Broenink & Tiernego (1996), pour résoudre les incertitudes liées au positionnement lors de l'assemblage. Dans cet article, la méthode proposée pour régler ce problème utilise un système de vision ainsi qu'un contrôle par impédance durant la tâche d'insertion pour réduire la force appliquée par le robot pour faire pénétrer l'objet dans l'insertion. Dans un même ordre d'idées, les travaux de Hamner *et al.* (2010) proposent d'utiliser un asservissement visuel et en force ainsi que d'incorporer un contrôle de tâches réactives sophistiqué pour surmonter efficacement les incertitudes et les exceptions d'un système autonome d'insertion de câblage électrique pour le domaine manufacturier d'automobile. Bien que ces solutions semblent adéquates pour le problème que nous voulons résoudre, ceux-ci utilisent une caméra au coeur de l'exécution de la tâche, un outil que nous voulons éviter dans le cadre de ce projet. Nous souhaitons utiliser le minimum d'outils externes au robot pour l'exécution de la tâche.

Jasim *et al.* (2014) proposent une solution qui comporte une méthode très populaire dans le domaine de la robotique. Cet article se centre sur la méthode de recherche spirale pour localiser la position d'une insertion quand celle-ci est incertaine. Cette méthode, puisqu'elle n'utilise qu'un capteur d'effort (que nous utiliserons déjà pour le contrôle du robot), semble très adéquate dans le cadre de notre projet. Par contre, bien que cette méthode soit appropriée pour effectuer la tâche d'insertion, nous devons être capable de savoir quand l'utiliser. Ce qu'il nous faut désormais, c'est un moyen de déterminer automatiquement qu'une insertion a été réalisée pendant la phase d'enseignement kinesthésique, avant que les techniques d'insertion mentionnées puissent être utilisées.

1.3 Détection d'une tâche d'insertion à l'aide des signaux de force

En utilisant un modèle analytique, il nous est possible de caractériser une tâche d'insertion. Tel qu'expliqué par Gosselin (2006), une insertion est souvent caractérisée par une approche de la surface en contact, d'un glissement sur la surface et finalement de l'insertion. Il est ensuite possible de construire un modèle analytique en sachant que les mouvements libres du robot

peuvent être caractérisés par la trajectoire à parcourir et que la rigidité liée à l'environnement perçu par le robot (tel que dans une tâche d'insertion) peut être caractérisé par les forces et les moments mesurés. Un exemple d'insertion basée sur le modèle de force/moment est présenté dans les travaux de Kim *et al.* (1999). Nous considérons que pour rejouer une démonstration, il nous est primordial d'utiliser une démonstration basée sur la force plutôt que d'utiliser une démonstration basée uniquement sur la position du robot.

Par contre, dans le cadre de ce projet, puisque la démonstration est faite à partir d'un contrôle par admittance opéré par un utilisateur, il nous est difficilement possible de faire un modèle d'une insertion. Effectivement, lors de la démonstration d'une tâche d'insertion, les forces ressenties par la rigidité de l'environnement et les forces appliquées par l'utilisateur pendant l'enseignement sont complètement superposées. L'humain ajoute alors une nouvelle dimension au modèle d'insertion et est difficilement caractérisable. Il nous est toujours possible d'utiliser des techniques d'insertion utilisant un modèle de force, tel que vu à la section 1.2, mais il nous est difficile de déterminer la présence d'une tâche d'insertion l'intérieur d'une démonstration par admittance à l'aide d'un modèle. Puisque nous voulons savoir automatiquement où utiliser une technique d'insertion, il nous est primordiale de trouver une méthode pour détecter une tâche d'insertion dans un signal de force. Pour ce faire, nous avons penché sur l'option d'utiliser des algorithmes d'intelligences artificielles pour nous aider.

Depuis maintenant plusieurs années, l'intelligence artificielle est utilisée dans toute sorte de domaine différent. Par exemple, les travaux récents faits par Li *et al.* (2010) suggèrent l'utilisation de réseaux neuronaux convolutifs pour faire la classification de pièces musicales par genre. Ce genre de travaux est très pertinent pour nous, car cela suggère que les CNNs sont une méthode très efficace de classifier des signaux temporels, comme les signaux qui pourraient être générés par un capteur d'effort pendant l'enseignement d'une tâche. De plus, plusieurs travaux similaires faits sur des signaux audio, telle que la reconnaissance vocale effectuée par Abdel-Hamid *et al.* (2014) ou encore la classification des syllabes d'une phrase par Lee *et al.* (2009), nous prouvent que les CNN ont été testé et confirmé pour plusieurs autres tâches impliquant des signaux temporels.

Dans un domaine plus près du nôtre, Rad *et al.* (2015) ont prouvé qu'il est possible de prédire la venue d'une crise chez des patients atteints du trouble du spectre autistique à l'aide d'un CNN. Effectivement, l'article présente les résultats positifs obtenus lorsque des accéléromètres attachés à des patients mesurent les mouvements de ceux-ci. Les mouvements enregistrés sont ensuite utilisés pour entraîner un CNN à être capable de détecter des mouvements moteurs stéréotypés qui sont typiquement la prédiction d'une crise. Nous croyons que ce résultat sur des signaux spatio-temporels, tels que ceux que nous générerons, nous permettent de mettre de l'avant l'utilisation des CNNs pour la détection de tâche d'insertion dans des signaux de force d'une démonstration kinesthésique.

L'utilisation de CNNs appliqués à des tâches d'insertion ou de préhension sont souvent employés pour faire la reconnaissance de formes. Par exemple, De Gregorio *et al.* (2018) ont utilisé un CNN entraîné sur des données de capteurs tactiles afin de déterminer l'orientation d'un fil électrique dans un préhenseur pour ensuite faire l'insertion de celui-ci. Dans le même ordre d'idée, Kwiatkowski *et al.* (2017) ont utilisé un CNN pour reconnaître la stabilité/qualité d'une prise d'objets divers en étudiant des images tactiles lors de la fermeture du préhenseur. Bien que ces CNNs soit utilisés pour réaliser des tâches de manipulation, ceux-ci ne font qu'étudier des données statiques, c'est-à-dire à un moment donné, plutôt que d'étudier des données évoluant dans le temps. Dans le cadre de notre projet, nous étudierons des données provenant de forces générées pendant une tâche d'insertion, ce qui aura pour effet que ces données auront une dimension temporelle ajoutée.

CHAPITRE 2

CONTRÔLE PAR ADMITTANCE D'UN ROBOT COLLABORATIF POUR DES APPLICATIONS À RIGIDITÉ VARIABLE

Comme mentionné auparavant, le but de ce projet est d'être capable de faire la démonstration d'une tâche d'insertion à un robot avec une méthode "teach-by-showing" et que celui-ci soit capable de répéter cette tâche d'une manière automatique ou avec le moins d'entrée d'information possible par l'utilisateur. Avant de pouvoir développer un système capable de détecter et de gérer de telles tâches, il est primordial que nous nous penchions sur la manière dont nous démontrerons les tâches au robot.

Puisque nous utilisons un robot collaboratif, une manière bien évidente de démontrer des trajectoires et des tâches au système est par l'apprentissage kinesthésique, tel que fait dans de multiples travaux de recherche et même certains produits destinés à la production, tel que le programme ActiveDrive de Robotiq (Robotiq (2018)). Le principe de l'apprentissage kinesthésique repose sur le fait que l'instructeur démontre physiquement (avec un contact physique) au robot la tâche à apprendre. Pour ce faire, nous avons choisi d'utiliser un principe de contrôle par impédance en utilisant un capteur d'effort au poignet de notre robot.

Tel que développé par Hogan (1985), le contrôle par impédance classique d'un manipulateur robotique peut être résumé avec la formule suivante :

$$\mathbf{F} = K[\mathbf{X}_0 - \mathbf{X}] + B[\dot{\mathbf{X}}_0 - \dot{\mathbf{X}}] - M[\ddot{\mathbf{X}}_0 - \ddot{\mathbf{X}}] \tag{2.1}$$

où $\mathbf{X}_0$ est la position commandée, $\mathbf{X}$ est la position actuelle, M est la masse virtuelle, K est le coefficient de rigidité, B est le coefficient de viscosité et $\mathbf{F}$ est la force résultante. Pour l'optimisation du contrôle par impédance d'un robot utilisé en coopération avec un humain, nous devrions alors gérer les paramètres virtuels de masse, de viscosité et de rigidité. Par contre, tel que démontré dans Ikeura *et al.* (1994), l'effet de la rigidité devrait être négligé lors du contrôle par

cliCours.com

impédance. Effectivement, l'utilisation d'un terme de raideur virtuel à l'intérieur d'une commande par impédance donne un point d'équilibre attractif qui nuit au mouvement collaboratif libre. Ce terme est plutôt utilisé pour des trajectoires programmées utilisant une composante de force à l'aide d'une commande par impédance. Alors, tel que Duchaine & Gosselin (2007) le proposent, nous pouvons résumer le contrôle par impédance d'un robot collaboratif ainsi :

$$\mathbf{F} = M[\ddot{\mathbf{X}}_0 - \ddot{\mathbf{X}}] + B[\dot{\mathbf{X}}_0 - \dot{\mathbf{X}}] \tag{2.2}$$

En utilisant cette méthode, nous n'avons que deux paramètres à optimiser pour obtenir un contrôle adéquat. Par contre, comme on peut le constater rapidement, la sortie générée par cette formule nous donne la force résultante d'une vitesse et d'une accélération données. Puisque nous voulons développer un système de contrôle kinesthésique d'un robot collaboratif, l'entrée du système devrait alors être la force appliquée sur le robot. Nous devons alors modifier cette formule pour obtenir une méthode qui accepterait un vecteur de force en entrée pour obtenir une sortie en déplacement. Pour ce faire, nous nous somme inspiré de la proposition faite dans les travaux de Duchaine & Gosselin (2007) pour un contrôle en vélocité plutôt qu'en force. Nous proposons alors de modifier la fonction 2.2 ainsi :

$$[\dot{\mathbf{X}}_0 - \dot{\mathbf{X}}] = (\mathbf{F} - M[\ddot{\mathbf{X}}_0 - \ddot{\mathbf{X}}])B^{-1} \tag{2.3}$$

pour ainsi obtenir un contrôle en vitesse. Cette méthode devient alors un contrôle en admittance puisque celle-ci contrôle la vitesse du préhenseur plutôt que la force. Nous utiliserons cette formule et cette annotation pour le restant des développements fais dans ce document.

2.1 Contrôle robuste pour l'exécution de tâches à raideur variable

Nous avons discuté dans la dernière section du contrôle par admittance de base pour le déplacement d'un bras manipulateur. Pour le déplacement de base du robot, ces solutions sont plus

qu'adéquates. Par exemple, démontrer une nouvelle position à atteindre dans l'espace du robot lors de l'exécution d'un programme peut être démontré en déplaçant physiquement le robot à l'endroit voulu en utilisant un contrôle par admittance simple. Nous appellerons tout mouvement qui n'implique pas l'environnement du robot comme étant un mouvement en *free-motion*.

Par contre, ces solutions de contrôle peuvent devenir rapidement instables lors de démonstrations de tâches qui interagissent avec des objets rigides. Par exemple, les forces appliquées sur le préhenseur lors de la démonstration d'une tâche d'insertion sont souvent beaucoup plus élevées que celles appliquées par le démonstrateur.

2.1.1 Stabilité du contrôle par admittance dans un environnement à raideur variable

Pour démontrer le phénomène d'instabilité, nous avons fait une expérience rapide où le robot bouge dans une direction à une vitesse fixe en free-motion et est soudainement arrêté. Dans un cas, l'opérateur arrêtera le robot en cessant de contrôler celui-ci et dans un deuxième cas, le robot sera arrêté par une surface rigide.

La figure 2.1 présente les résultats des forces et la position du robot lorsque celui-ci est arrêté par l'opérateur en free-motion et lorsque celui-ci est arrêté par une surface rigide. On peut observer rapidement que la collision du robot avec la surface rigide génère une grande force instantanée dans le sens opposé de la direction précédente du préhenseur. Dans le cas ci-haut, la différence de force mesurée est près de 140 N, ce qui est près de 7 fois plus grand que la force utilisée pour guider le robot, soit environs 20 N. On peut observer que ce phénomène est complètement absent de la séquence d'arrêt en free-motion.

Évidemment, si l'on regarde la fonction de contrôle 2.3, un changement de force soudain dans une direction opposée provoquera un effet de rebond. Le résultat du changement soudain de force de la figure 2.1 peut être observé sur la courbe bleue du graphique. On remarque rapidement que la position finale du préhenseur diverge d'environ 2 mm de la surface de travail dû au rebond produit pour l'impact.

Clicours.COM

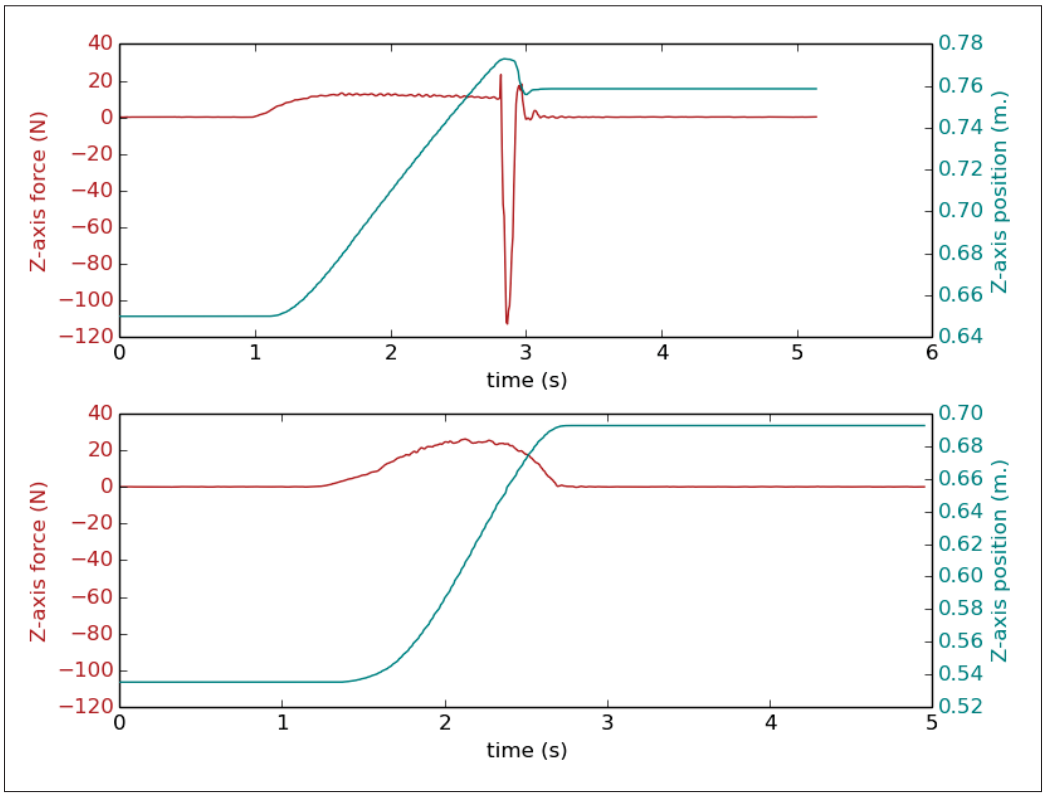


Figure 2.1 Résultats obtenus lors de l'expérience 1 (d = 1.5 Ns/m). **Graphique haut** : réponse en force et en position lorsque le préhenseur est arrêté par une surface rigide. **Graphique bas** : réponse en force et en position lorsque le préhenseur est arrêté par l'opérateur en free-motion.

Une solution simple pour éviter ce phénomène serait d'augmenter la valeur du terme d'amortissement d pour ainsi diminuer la vitesse résultante d'une force en entrée. Pour démontrer l'effet du changement l'amortissement dans la stabilité du contrôle, nous avons reproduit l'impact précédent avec un amortissement plus élevé, soit $d = 15Ns/m$.

En comparant les réponses obtenues lors d'un impact avec des amortissements différents sur la figure 2.2, on peut rapidement voir que le problème d'instabilité obtenu précédemment est atténué. En effet, lorsqu'on utilise un amortissement dix fois plus élevé, l'effet de rebond est pratiquement inexistant.

Par contre, bien que le contrôle soit stable, un nouveau problème survient ; les forces nécessaires pour faire bouger le robot sont maintenant beaucoup plus élevées. On peut voir sur la

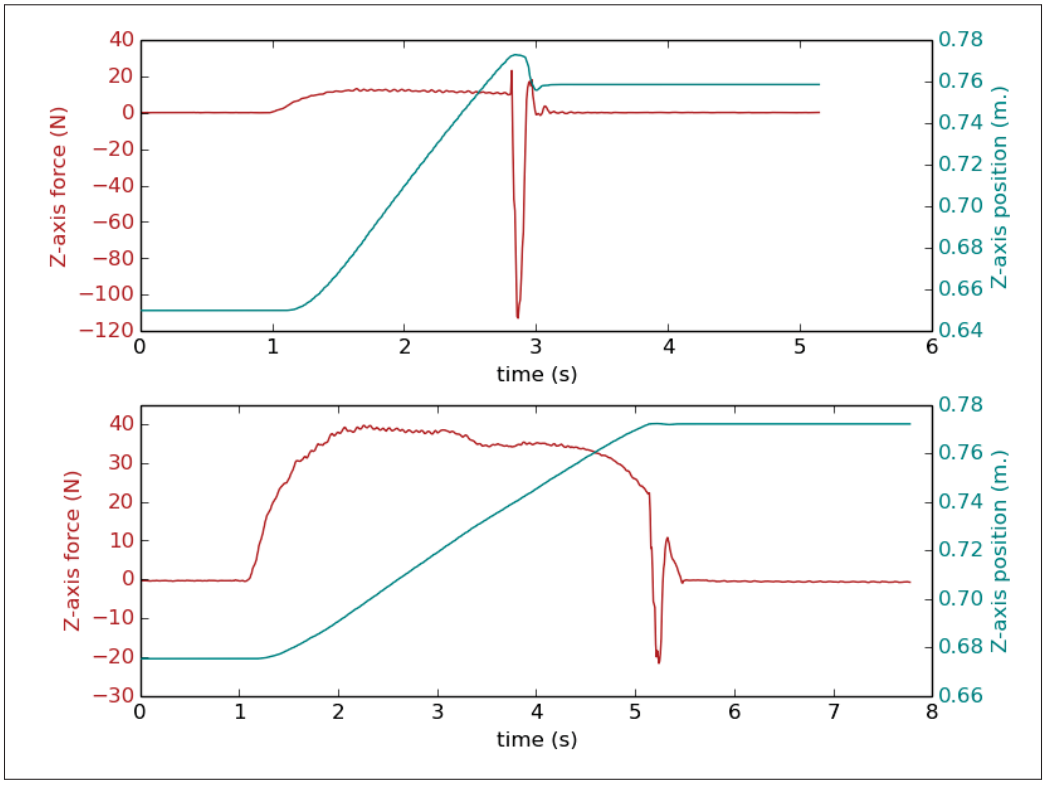


Figure 2.2 **Graphique haut** : réponse en force et en position lorsque le préhenseur est arrêté par une surface rigide avec d=1.5 Ns/m. **Graphique bas** : réponse en force et en position lorsque le préhenseur est arrêté par une surface rigide avec d=15 Ns/m.

figure 2.2 que la force utilisée pour contrôler le robot est maintenant d'environ 35 N, comparativement à environ 20 N pour un $d = 1.5Ns/m$. Avec un amortissement plus élevé, le contrôle du robot devient moins naturel qu'avec un amortissement plus souple puisque les forces requises sont beaucoup plus élevées.

2.1.2 Contrôle par admittance avec raideur variable

Comme nous l'avons constaté à l'expérience précédente, le contrôle par admittance de base nous pose un dilemme de stabilité. Avoir un taux d'amortissement bas nous permet d'avoir un contrôle fluide en free-motion, mais génère un effet de rebond lorsque le robot entre en contact avec une surface rigide. Un taux d'amortissement plus haut nous permet d'être stable lorsque nous obtenons une pointe soudaine de force, mais le contrôle en free-motion requiert alors une

plus grande force et devient désagréable à contrôler. Nous aurions alors besoin d'avoir un taux d'amortissement haut seulement lorsque le robot entre en contact avec une surface rigide.

Plusieurs travaux ont abordé l'idée d'avoir un taux d'amortissement variable dans un contrôle par admittance comme le nôtre. Tsumugiwa *et al.* (2002) ont proposé d'utiliser un taux d'amortissement qui serait directement relié à la rigidité sentie lors du contrôle par un utilisateur. D'autres travaux, réalisés par Duchaine & Gosselin (2007), proposent de construire un contrôle en vitesse du robot avec un taux d'amortissement variable relié à l'intention mesurée de déplacement de l'utilisateur. Par contre, les solutions proposées ici ont été développés dans le but de faire des tâches qui ne requièrent que de faire des déplacements en free-motion. Dans un cas comme le nôtre, où nous devons parfois déplacer le robot à l'intérieur d'une insertion rigide et étroite, ces solutions deviennent vite instables ou alors difficiles à contrôler.

Bien que ces travaux ne sont pas adéquats pour le système que nous voulons développer, nous croyons que nous pouvons grandement nous inspirer de ceux-ci. En effet, nous proposons de développer une solution basée sur un contrôle ayant un taux d'amortissement variable. Par contre, nous proposons ici une nouvelle méthode analytique pour déterminer lorsque le taux d'amortissement doit être augmenté pour que le système reste stable tout en ayant un contrôle agréable pour l'utilisateur.

Lorsque nous avons développé notre solution, nous avons déterminé, à l'aide d'essai et erreur, trois phénomènes qui peuvent causer un moment d'instabilité : le changement de force appliqué au contrôle est trop grand, le prochain mouvement calculé du robot est trop différent des derniers et lorsque les forces mesurées sur le robot oscillent trop rapidement.

2.1.2.1 Changement soudain de la force mesurée pour le contrôle du robot

Le cas classique d'instabilité dans une commande par admittance est lorsque le contrôle entre en contact avec une surface rigide, tel que présenté à lors de l'expérience de la section 2.1.1. Lorsque le préhenseur entre en contact avec la surface de travail, la force mesurée par le capteur de force est extrêmement différente de celle mesurée lors du contrôle en free-motion et crée

ainsi un effet de rebond. Nous devons alors être capable de déterminer lorsque ce moment arrive pour permettre au système d'augmenter le taux d'amortissement pour estomper cet effet indésirable.

Pour ce faire, nous calculons la différence entre la force mesurée au temps présent et la moyenne des 3 dernières forces mesurées. Si la différence est plus grande que 15 N, le système augmentera le taux d'amortissement pour permettre au contrôle de rester stable. Cette méthode est utilisée pour chacun des axes du capteur de force (X,Y,Z). De plus, nous utilisons cette méthode pour chacun des moments mesurés du capteur de force (rX,rY,rZ), mais dans leur cas, une différence de 1.2 Nm est utilisée pour déclencher une augmentation du taux d'amortissement.

Avec le temps, si aucun autre changement drastique de force mesurée n'est enregistré, le taux d'amortissement revient alors à son état initial.

2.1.2.2 Changement soudain de la commande en vitesse du robot

Lors de nos essais, nous avons observé que, dans certains cas, la force mesurée par le capteur de force change radicalement, mais pas assez soudainement pour déclencher une hausse du taux d'amortissement par le calcul proposé à la section précédente. Dans le cas où le changement de force n'est pas assez soudain mais quand même extrêmement élevé, nous obtenons toujours un effet de rebond du système. Nous proposons alors de mesurer le taux de changement de la commande envoyée au robot par le système de contrôle et si cette commande est trop différente des dernières envoyées, nous augmenterons alors le taux d'amortissement du système pour réduire l'instabilité.

Pour ce faire, après chaque calcul de direction à envoyer au robot à l'aide de la formule d'admittance, nous mesurons la différence entre la commande calculée au moment présent et la moyenne des commandes envoyées lors des 5 dernières itérations de la boucle de contrôle. Si l'accélération mesurée est de plus de 0.6 m/s^2 ou encore de 2 rad/s^2 pour les rotations, nous augmentons alors le taux d'amortissement du système.

Avec le temps, si aucun autre changement drastique de commande du système n'est enregistré, le taux d'amortissement revient alors à son état initial.

2.1.2.3 Oscillation du signal de force pour le contrôle

Toujours lors du développement de notre solution, nous avons observé un phénomène particulier lorsque nous effectuons une tâche d'insertion rigide. Lors de la pénétration de l'objet à insérer, les deux dernières solutions augmentant le taux d'amortissement préviennent le système de devenir instable. Tel que planifié, lorsque l'objet entre en contact avec la surface d'insertion, le système augmente le taux d'amortissement, car le contact initial de l'insertion génère habituellement un changement soudain de la force mesurée et l'algorithme proposé à la section 2.1.2.1 empêche le système de devenir instable. L'insertion se fait alors facilement avec un taux d'amortissement élevé et un contrôle stable.

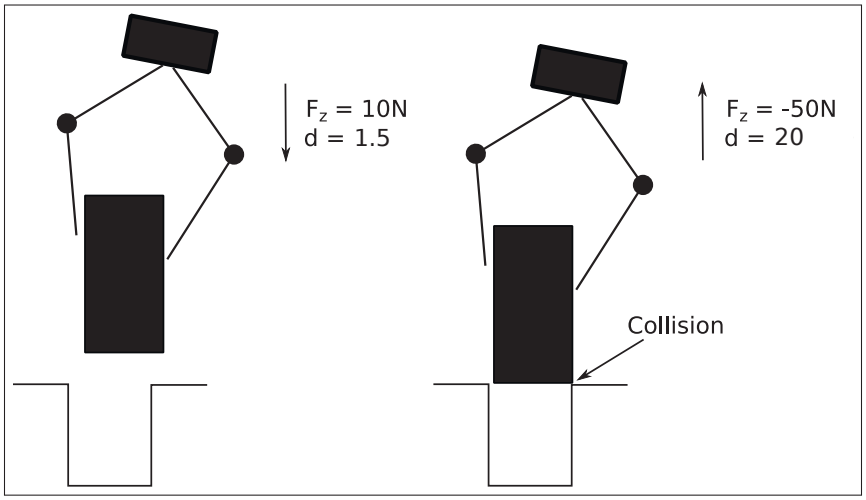


Figure 2.3 Forces ressenties par le capteur d'effort lors du contact avec l'entrée d'une insertion étroite.

Par contre, si l'objet inséré reste un certain temps dans le préhenseur du robot sans recevoir de commande de mouvement par l'utilisateur, le robot bougera légèrement à cause des faibles forces générées par le contact de la pièce dans l'insertion et ceci causera une petite oscillation dans le mouvement du robot.

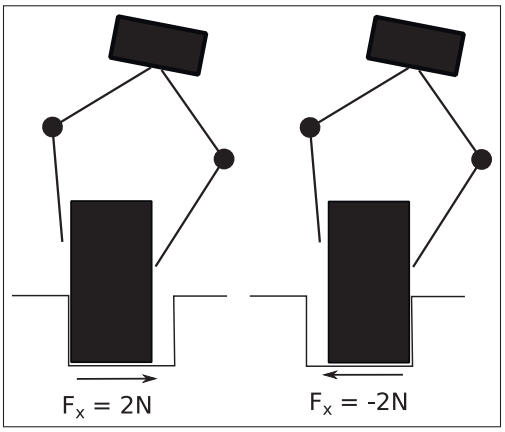


Figure 2.4 Forces oscillatoires ressenties par le capteur d'effort lorsque contraint dans le plan XY à l'intérieur d'une insertion.

Ensuite, puisque le système ne détecte plus de changements radicaux dans les forces mesurées (section 2.1.2.1) et dans les commandes envoyées (section 2.1.2.2), le taux d'amortissement retournera à son état initial pour le contrôle en free-motion. Finalement, puisque l'amortissement est bas et que le robot est toujours contraint dans l'insertion, les oscillations générées par les parois de l'insertion s'amplifieront de plus en plus dû à l'effet du rebond et causera le robot à faire des mouvements brusques jusqu'à ce que le système détecte des forces trop grandes. Pour éviter ce phénomène d'amplification de l'oscillation, nous forçons alors le système à garder un taux d'amortissement grand lorsqu'une petite oscillation est présente dans le signal de force.

Pour mesurer si une oscillation est présente dans le signal, nous enregistrons toutes les fois que le signal de force passe par zéro (zero-crossing). Nous avons déterminé que si le signal de force croise zéro plus de 3 fois par seconde, le système se trouve dans un état d'oscillation et le taux d'amortissement reste élevé. De plus, pour ne pas considérer un signal bruité centré à zéro comme étant un signal qui oscille, nous ajoutons une condition où le signal doit avoir une amplitude minimum de 2 N pour être considéré dans un état oscillatoire.

Par exemple, dans la figure 2.5, nous pouvons observer un effet oscillatoire dans le signal de force. Dans ce cas-ci, puisque nous avons 3 passages par zéro en moins d'une seconde, le système garderait un taux d'amortissement élevé.

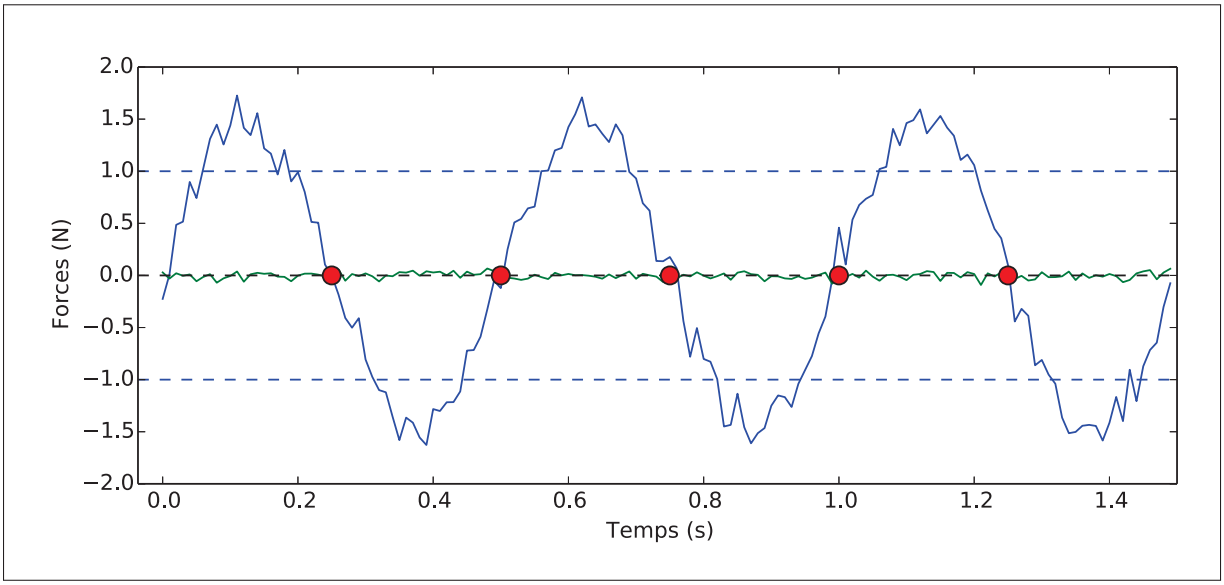


Figure 2.5 Exemples de signaux de forces oscillants. **En bleu :** un signal de force oscillant dépassant les limites d'amplitude et obtenant 3 passages par zéro en moins d'une seconde. **En vert :** un signal oscillant avec une amplitude trop petite pour être significative.

2.1.2.4 Boucle de contrôle

La boucle de contrôle développé dans le cadre de ce projet fonctionne à un taux de 125 hz et est séparée en deux parties. La première partie est une boucle d'acquisition des données du capteur de force ainsi que de calculs de déplacement. La deuxième est une boucle de contrôle transformant dans l'espace des joints du robot les déplacements cartésiens calculés par la première boucle et exécutant les mouvements demandés.

La boucle de calcul et d'acquisition a été programmée en Python 2.7 et fonctionne directement sur un système d'exploitation Ubuntu 14.04 externe au robot. Celle-ci acquiert les données de force provenant d'un capteur d'effort Robotiq FT 300 (6 axes) en plus de faire le calcul du prochain déplacement du robot à l'aide d'un contrôle par admittance avec un taux d'amortissement variable, comme expliqué précédemment dans ce chapitre. La boucle de contrôle peut être résumée par l'algorithme 2.1.

Algorithme 2.1 Algorithme d'ajustement de l'amortissement d et calcul du prochain déplacement par rapport aux forces mesurées.

```
1  Sortie : Le prochain vecteur de déplacement V
2   Calcul du taux d'amortissement d
3  if d_timer > 0 then
4  |   d_timer-=1
5  end
6  d = 1.5 + (d_timer × 100)
7  Calcul de V avec contrôle en vitesse
8  V(t) = (ft(t) − m × ΔV/dt)/d
9  Ajustement du taux d'amortissement d_timer
10 A–Est-ce que la force mesurée change trop rapidement ?
11 if Δft > 15 then
12 |   V(t) = 0
13 |   d_timer+=50
14 end
15 B–Est-ce que le prochain mouvement calculé est trop rapide ?
16 if ΔV > 0.005 then
17 |   V(t) = 0
18 |   d_timer+=50
19 end
20 C–Est-ce qu'une oscillation est détectée ?
21 if d_timer>0 then
22 |   θ = get_oscillation()
23 |   if θ ⩾ 3 then
24 |   |   d_timer+=50
25 |   end
26 |
```

On peut observer que le taux d'amortissement variable dans la boucle de contrôle est fait à partir de la variable d_{timer}. Lorsque le système détermine que d doit être augmenté pour que le contrôle reste stable, la variable d_{timer} est augmentée de 50 et est ensuite ajoutée à d dans la prochaine itération de la boucle. Cette variable d_{timer} est ensuite décrémentée à chaque itération de la boucle pour permettre un retour vers le taux d'amortissement initial en douceur. Il est important de noter aussi que les variables ΔV et Δft sont calculées à chaque itération de la boucle à partir des dernières valeurs de V et ft.

Clicours.com

La valeur V calculée est ensuite envoyée, par réseau, à un robot UR5 cb2 roulant une boucle de contrôle pour effectuer les mouvements demandés par la boucle précédente. Celle-ci effectue les opérations suivantes en continu.

1. Reçoit le vecteur de mouvement V du programme de contrôle.
2. Transforme le vecteur cartésien $V_{capteur}$ qui est dans le plan du capteur de force pour obtenir le vecteur V_{robot} dans le plan de la base du robot.
3. Transforme V_{robot} dans l'espace joint du robot en effectuant la cinématique inverse du robot à partir de V_{robot}.
4. Effectue le mouvement des joints calculé pour 0.008 secondes (le temps d'une boucle).

Lors du démarrage de cette boucle, celle-ci s'attend à recevoir une nouvelle commande de déplacement à tous les 0.008 secondes. Si après un délai de 2 secondes pendant l'exécution de la boucle aucune commande n'est reçue, le contrôle du robot s'arrête.

Pour le restant des travaux réalisés dans ce document, le contrôle développé ici est utilisé. Celui-ci sera utilisé pour faire des démonstrations de tâches et pour enregistrer les trajectoires faites par l'utilisateur. Nous étudierons les signaux de force enregistrés pendant le contrôle pour connaître si nous sommes capable de comprendre la nature des tâches effectuées et de découvrir si nous sommes capables de modifier les trajectoires effectuées par l'utilisateur dans le but d'optimiser les tâches enseignées.

CHAPITRE 3

PREUVE DE CONCEPT : LES RÉSEAUX DE NEURONES CONVOLUTIFS POUR LA DÉTECTION DE TÂCHES D'INSERTION PENDANT L'APPRENTISSAGE KINESTHÉSIQUE D'UN ROBOT COLLABORATIF

Dans ce chapitre, nous aborderons un système de détection automatique de tâche d'insertion basé sur une architecture de réseau neuronal convolutif (CNN) qui pourrait être intégré dans la plupart des systèmes robotiques à apprentissage kinesthésique. Notre méthode analyse les signaux de force et torque (F/T) appliqués par l'opérateur à l'effecteur du robot pendant la phase de démonstration pour en extraire les caractéristiques essentielles du signal associé à la tâche d'insertion. Après la détection d'une tâche d'insertion, le robot pourra automatiquement remplacer la portion de la routine apprise où l'insertion a lieu pour la remplacer par une solution plus robuste et adéquate que de répéter les mouvements exacts du démonstrateur.

Les CNNs ont prouvé au fil des années qu'ils sont un algorithme d'apprentissage machine donnant des résultats impressionnants, et ce, dans plusieurs domaines. Des exemples d'utilisation des CNNs pour des problèmes de traitement de signaux ne sont pas rares, que ce soit pour la reconnaissance de la parole (Lee *et al.* (2009)), la reconnaissance de style musical (Li *et al.* (2010)) ou des signaux plus spatio-temporels (Rad *et al.* (2015)). L'utilisation de cette technique d'extraction de caractéristiques est justifiée par la nature du signal de nos expériences, car le même capteur F/T est utilisé pour un contrôle par admittance et l'acquisition des données pendant les démonstrations. L'utilisation du même capteur F/T signifie que la dynamique du contrôle par l'opérateur et de l'interaction externe de l'effecteur s'y retrouve superposée, un problème difficile à découpler en utilisant de simples modèles déterministes.

Sur la figure 3.1, nous pouvons voir les forces et les couples résultant générés lors d'une tâche d'insertion et lors d'une tâche de pick-and-place. Uniquement en regardant la figure, il est assez difficile de déterminer les caractéristiques qui caractériseraient chaque tâche. Un CNN est une approche adéquate pour découvrir et apprendre automatiquement les caractéristiques nécessaires à la distinction des deux tâches. À notre connaissance, notre travail représente la

Clicours.COM

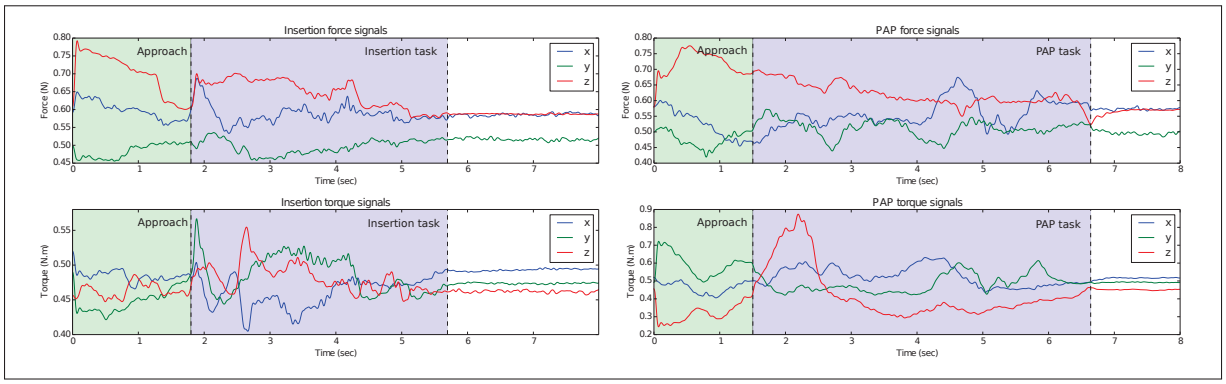


Figure 3.1 Exemples de signaux de force lors d'une tâche d'insertion (gauche) et lors d'une tâche de PAP (droite).

première fois qu'un système de détection d'insertion automatique a été appliqué aux méthodes d'apprentissage kinesthésique d'un robot collaboratif.

Pour évaluer notre approche, nous avons utilisé huit objets différents pour créer une base de données. Celle-ci contient des signaux F/T d'insertion, de tâches de pick-and-place et des mouvements aléatoires effectués avec les huit objets différents. Nous donnons un aperçu des détails théoriques des techniques appliquées dans la section 3.2 de ce chapitre. La section 3.3 décrit le matériel, les logiciels ainsi que les méthodes d'acquisition de données utilisées lors des expériences. La section 3.4 présente les résultats obtenus. Enfin, la section 3.6 termine par une discussion de l'impact de ce travail et des projets futurs potentiels.

3.1 Définition d'une tâche d'insertion

Tout d'abord, puisque nous parlerons de tâches d'insertion tout au long de ce document, il est primordial de définir proprement l'action. Selon le dictionnaire Larousse, une insertion est : "Mettre, glisser, introduire quelque chose sous ou dans quelque chose", ou bien "Introduire quelque chose dans un ensemble". La deuxième définition s'applique particulièrement bien dans le domaine de la robotique, où nous définissons une insertion comme étant l'action d'introduire un objet dans un ensemble pour créer un assemblage.

Dans le cours de ce projet, nous rétrécirons davantage la définition d'insertion. Pour qu'une action d'assemblage soit définie comme une insertion, l'objet inséré doit être contraint spatialement dans au moins 4 axes. Par exemple, dans le cas typique "peg-in-hole", l'assemblage est contraint et n'a que deux degrés de liberté, soit l'axe Z en translation et en rotation.

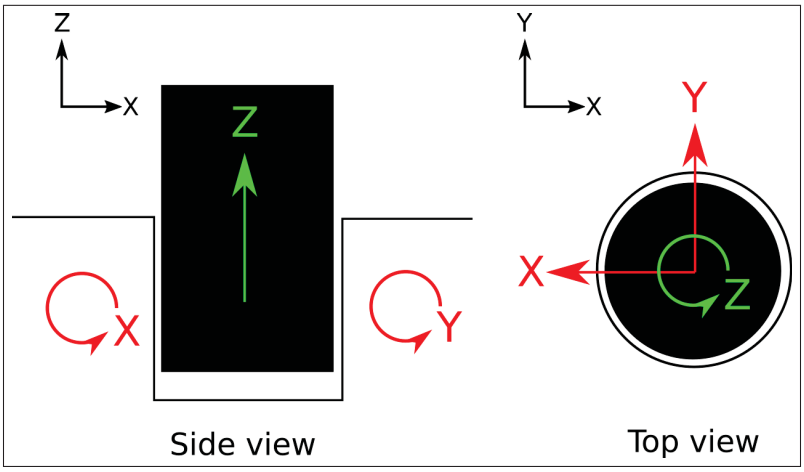


Figure 3.2 Démonstration des axes contraints lors d'un insertion "peg-in-hole". **En vert** : les axes non-contraints (Z en translation et Z en rotation). **En rouge** : les axes contraints pendant l'action.

En comparaison, l'action de dépôt d'un objet, comme dans une tâche de pick-and-place, n'est contraint que dans un axe, soit l'axe des Z lorsque l'objet entre en contact avec la surface de travail.

De plus, pour le cadre de ce projet, les insertions effectuées seront des assemblages rigides et serrés. Effectivement, nous excluons toutes insertions qui pourraient être faites sur des corps mous, comme l'insertion d'une aiguille dans un patient dans le cadre d'une tâche médicale, et nous excluons aussi toutes insertions où l'objet à insérer et le socle de l'insertion comprennent un jeu de plus de 2 mm.

3.2 Méthodologie

L'objectif principal de ce travail est de déterminer automatiquement la présence d'une tâche d'insertion lors d'un enseignement kinesthésique. Une fois qu'un système peut déterminer au-

tomatiquement le moment où une insertion s'est produite dans la démonstration basée sur la position, il devient possible pour le robot de remplacer la partie de la démonstration contenant l'insertion par une solution mieux adaptée à la tâche. Nous pensons que cela pourrait être possible en étudiant les signaux du capteur de F/T de la phase d'enseignement pour distinguer les tâches d'insertion des autres tâches.

Dans nos expériences, un classificateur CNN est utilisé pour distinguer les tâches d'insertion enseignées avec le contrôle par admittance. Ce type de classificateur est connu pour être un moyen puissant d'extraire des caractéristiques dans des signaux spatio-temporels. Certains travaux montrent que l'extraction de caractéristiques via les CNNs est plus attrayante que les méthodes d'extraction faites à la main, c'est-à-dire à l'aide d'un modèle fait par un expert sur le sujet, car cette dernière méthode nécessite une connaissance spécifique de la nature de chaque signal (Rad *et al.* (2015)). De plus, le classificateur CNN est intéressant à utiliser dans notre cas particulier, car nous voulons détecter des modèles sur des signaux de F/T spatio-temporels. Ces signaux peuvent être principalement représentés par les caractéristiques qui auront été déterminées par le réseau.

La méthode proposée dans cet article consiste à acquérir les forces exercées par l'opérateur sur le robot pendant l'enseignement, comme si elles représentaient la sortie du modèle de contrôle moteur de l'opérateur lui-même. Plusieurs travaux soutiennent l'idée que le système nerveux central humain stocke les capacités de contrôle moteur en tant que modèles (Doya (2000), Imamizu *et al.* (2000), Hikosaka *et al.* (2002)). En supposant que ces modèles puissent être capturés par la force exercée sur le robot, il pourrait être possible, avec l'aide de réseaux de neurones, de différencier ces modèles les uns des autres et de détecter la tâche d'insertion prévue par le démonstrateur.

Nous savons que toute tâche est une combinaison de plusieurs petits mouvements. Par exemple, une tâche d'insertion serrée comprend les étapes suivantes : approche, alignement angulaire et translationnel de l'assemblage, l'insertion et l'arrêt du mouvement à la fin de l'insertion. Nous allons capturer le profil des signaux de F/T de la trajectoire démontré par l'opérateur pour

chaque étape impliquée dans une tâche. Ensuite, nous testerons notre hypothèse selon laquelle les CNN sont efficaces pour détecter les caractéristiques importantes du signal F/T liées à ces étapes.

Dans la section suivante, nous exposerons brièvement le concept des CNNs, puis nous expliquerons comment le nôtre a été conçu et appliqué à notre ensemble de données.

3.2.1 Apprentissage des caractéristiques importantes à l'aide des CNNs

Les CNNs consistent généralement en une ou plusieurs couches convolutives à propagation avant combinées à des couches de sous-échantillonnage, qui sont souvent suivies de couches entièrement connectées. Ce qui rend un CNN unique (comparé aux autres réseaux de neurones) est que dans chaque couche de convolution, chaque neurone est uniquement connecté localement à un sous-ensemble de la couche précédente et partage son poids avec tous les autres neurones de la même "feature map". La feature map de sortie est la combinaison de tous les neurones superposés partageant les mêmes poids. En d'autres termes, la feature map en sortie est la convolution d'un filtre sur la totalité des données de la couche précédente et donne la réponse obtenue lors de l'application du filtre sur les données d'entrée. L'utilisation de plusieurs filtres crée alors différentes feature maps réagissant à différents modèles,et résume ainsi l'entrée en termes de combinaisons de réponses obtenues avec plusieurs filtres sur des features maps qui indique la position de ces modèles dans le signal d'entrée.

Les features maps créées après une couche de convolution sont généralement introduites dans une couche de sous-échantillonnage par moyenne ou de sous-échantillonnage par valeur maximale afin de réduire la taille des données et de produire une invariance au décalage du signal (Scherer *et al.* (2010)). Par exemple, laissons $X^{n\times a\times b}$ - où n est le nombre de feature maps de taille $a \times b$ - être l'entrée d'une couche de sous-échantillonnage sans chevauchement $k \times j$. Le résultat en sortie sera une compression des données dans un format $X^{n\times \frac{a}{k}\times \frac{b}{j}}$.

La dernière couche de pooling du CNN est généralement aplatie et introduite dans une couche de réseau de neurones (NN) entièrement connectée pour obtenir la classe prédite finale de

l'entrée en utilisant une fonction d'activation softmax. Les couches entièrement connectées sont utilisées pour combiner les informations de chaque feature map de sortie des couches précédentes.

3.2.2 Utilisation d'un CNN pour la classification des tâches d'insertion à l'aide des signaux d'un capteur F/T

Afin d'utiliser un CNN pour la détection d'insertion, nous avons entraîné le système avec des signaux de force et torque appliqués sur le préhenseur du robot pendant l'ensemble de la tâche d'insertion ; en commençant par l'approche et en terminant par l'arrêt complet du robot à la fin de l'insertion. Nous avons ensuite généré un réseau qui classifie les signaux d'entrée d'une manière binaire.

Considérons deux groupes de trois signaux : $F_x, F_y, F_z \in \mathbb{R}^n$ et $T_x, T_y, T_z \in \mathbb{R}^n$, où F représente la catégorie des forces en translation, T représente la catégorie des moments de forces (Torque) et n est la longueur du signal. Assumons une matrice $X \in \mathbb{R}^{n\times 6}$ étant le résultat de la concaténation des 6 signaux précèdent ($F_{x,y,z}$ et $T_{x,y,z}$). X sera alors l'entrée du réseau qui considère chaque signal F/T comme une dimension de l'entrée. Finalement, considérons $y \in 0,1$ comme étant l'étiquette associée à un X donné, où pour une entré X_i, $y_i = 1$ est la détection d'une tâche d'insertion et $y_i = 0$ est l'absence d'une tâche d'insertion.

Les applications typiques utilisant les CNNs, telles que la vision par ordinateur, nécessitent l'utilisation d'un filtre carré. Cependant, dans notre cas, nous avons des signaux unidimensionnels, c'est-à-dire que la hauteur des filtres est de un. Ce détail simplifie le réglage des paramètres des filtres, car seule la largeur de ceux-ci doit être définie. La même simplification s'applique aux couches de sous-échantillonnage.

Une technique que nous avons empruntée au domaine de la vision par ordinateur concerne la manière dont un CNN gère généralement les images en couleur. Lors des couches de convolution, les filtres générés ont des connexions avec toutes les dimensions de l'image dans leur propre voisinage ; les dimensions étant les canaux rouge, vert et bleu d'une image RGB. Dans

notre cas, la concaténation de chaque signal de force et de moment dans une matrice à 6 dimensions signifie que les filtres peuvent accéder à tous les signaux F/T dans un certain voisinage, ou en d'autres mots, dans un certain laps de temps du signal.

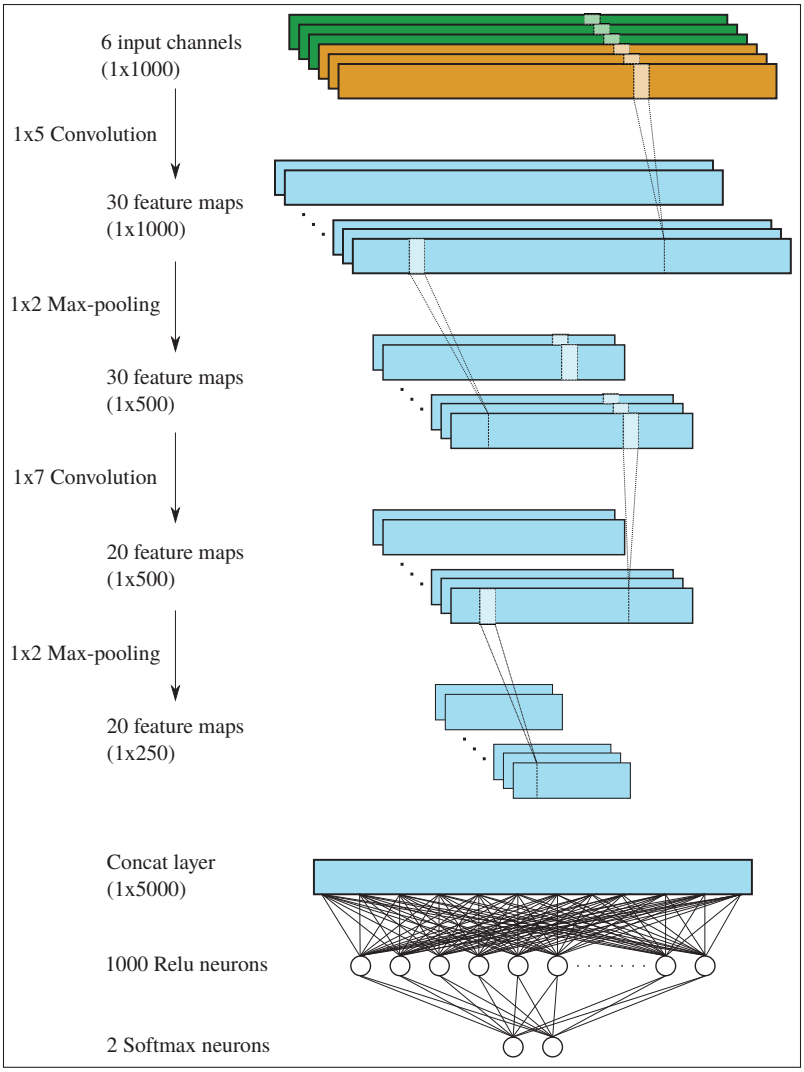


Figure 3.3 Architecture CNN utilisée dans ce travail. Les étiquettes à côté des flèches à gauche indiquent les opérations ; les étiquettes à côté des blocs à droite indiquent les couches.

3.2.3 Approche proposée

L'objectif principal de ce travail est de détecter les tâches d'insertion par le biais de la séquence d'enseignement par démonstration kinesthésique du robot. À cette fin, l'opérateur montre d'abord au robot la tâche à accomplir en utilisant le capteur F/T pour déplacer le préhenseur du robot. Pendant que le démonstrateur montre la tâche au robot, l'ensemble complet des signaux du capteur est enregistré. Ces signaux de forces sont ensuite utilisés pour déterminer si une tâche d'insertion a été effectuée pendant l'apprentissage de la tâche par le robot. Pour ce faire, les données sont d'abord divisées en échantillons de temps de 8 secondes avec un pas de 80 millisecondes. Cela génère des fenêtres de signaux qui sont ensuite envoyées au CNN, qui aura été entraîné antérieurement, pour déterminer si l'une des fenêtres du signal de démonstration est liée à une tâche d'insertion. Finalement, si une fenêtre de signal est identifiée comme étant une insertion, le robot pourrait alors remplacer cette partie de la démonstration de l'opérateur par une méthode d'insertion plus avancée et plus robuste aux incertitudes, par exemple, une insertion de type spirale (Jasim *et al.* (2014)). La figure 3.4 présente un diagramme de l'approche proposée.

Bien sûr, un CNN doit d'abord être entraîné pour appliquer les bons filtres ; dans ce cas, pour filtrer les fonctionnalités qui caractérisent les insertions.

3.3 Montage expérimental

3.3.1 Acquisition des données

Pour acquérir les données d'entraînement et de test, nous avons installé un manipulateur UR5 d'Universal Robots, un capteur FT 300 et une pince à 2 doigts de Robotiq. Le même capteur F/T a été utilisé pour le contrôle par admittance et pour acquérir des données, faisant en sorte que toutes les dynamiques du contrôle de l'opérateur ont été mélangées avec la dynamique de l'action effectuée et de l'interaction avec l'environnement du robot. Par exemple, si l'opérateur forçait le déplacement du préhenseur sur une surface dure telle qu'une table, l'interaction de

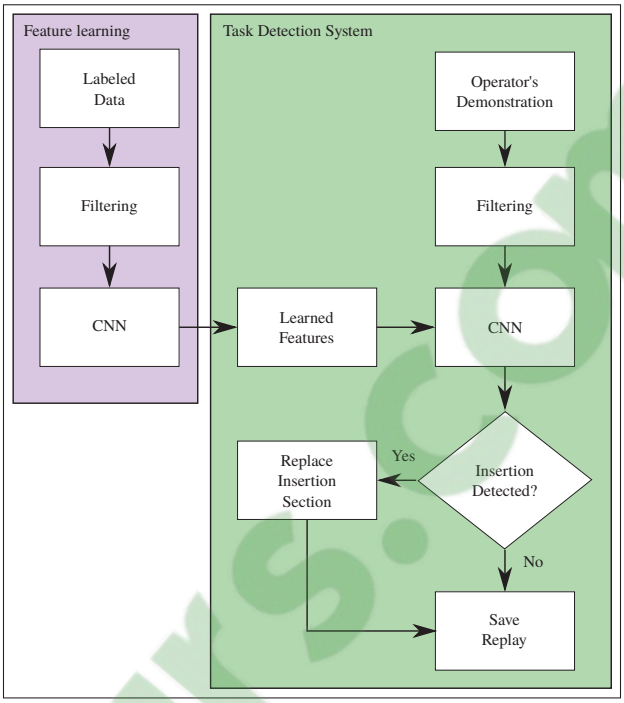


Figure 3.4 Étapes de l'approche proposée.

l'opérateur avec le robot était inséparable de l'interaction du robot avec la table, donc les forces résultantes sont alors capturées d'une manière superposée.

Nous avons utilisé huit objets différents pour effectuer les tâches d'insertion et les PAP (Pick-And-Place). Nous avons effectué un total de 886 expériences, soit : 302 insertions, 453 PAP et 131 mouvements aléatoires. L'objectif de ce travail étant de détecter les tâches d'insertion, nous avons divisé les expériences en deux classes : les insertions (classe 1) et toutes autres mouvements (classe 2).

Pour créer la base de données de la classe 1, nous avons effectué des insertions serrées et rigides. Pour ce faire, nous avons attaché un étau mécanique à une table et fixé l'assemblage (la pièce qui reçoit l'insertion) dans l'étau.

L'objet à insérer était fermement saisi par la pince à deux doigts et guidé lors de l'insertion par l'opérateur à l'aide du contrôle par admittance. Les huit dernières secondes de données

(signaux du capteur) de chaque tâche d'insertion ont été enregistrées afin d'entraîner le réseau de neurones.

La partie PAP des données (classe 2) a été collectée de manière similaire, sauf que les objets ont été placés sur un endroit précis sur une surface plane au lieu d'être insérés dans un étau.

Finalement, les mouvements aléatoires (classe 2) ont été généré en faisant bouger le manipulateur UR5 de plusieurs manières différentes, soit : de longs mouvements continus, des mouvements saccadés courts, des séquences sans mouvement, ou encore, en le secouant dans plusieurs directions.

Les données ont été collectées en utilisant un programme Python fonctionnant sous Ubuntu 14.04 à un taux d'acquisition de 125 Hz. Chaque échantillon compilé contient 8 secondes de données provenant de toutes les directions du capteur F/T, résultant en des matrices $\mathbb{R}^6$ de longueur 1000 ($8s \times 125Hz = 1000$). Ces échantillons sont le résultat d'une capture des signaux F/T à la fin de chacune des démonstrations, c'est-à-dire que nous avons enregistré seulement les approches et insertions/PAP de chacune des démonstrations. Le CNN, qui a été utilisé pour analyser ces données, a été généré à l'aide de la bibliothèque d'apprentissage automatique open source Tensorflow.

3.3.2 Prétraitement des données

Pour réduire le bruit induit par le capteur F/T dans les signaux de force utilisés dans la détection de tâches, nous avons utilisé un filtre passe-bas de type Butterworth d'ordre quatre comme étape de prétraitement de la base de donnée. Cela a été fait auparavant par Myers *et al.* (2001) dans leurs travaux sur les techniques d'enseignement par démonstration, car comme ils le démontrent dans leur papier, le mouvement des membres humains lors de l'exécution de trajectoires volontaires est d'environ 5 Hz alors que les mouvements involontaires (réflexes) se produisent généralement autour de 10 Hz. Comme nous n'avons besoin que d'informations à basse fréquence pour caractériser les mouvements de l'opérateur pendant l'enseignement, nous pouvons supprimer toute information inutile dans les données qui perturberaient la séquence

d'apprentissage du réseau de neurones. Nous avons défini une fréquence de coupure de 12 Hz avec un filtre numérique Butterworth d'ordre quatre.

3.4 Expériences et résultats

3.4.1 Méthodologies d'entraînement et de test

Pour valider notre approche, nous avons utilisé la méthode d'entraînement/validation "Leave-One-Subject-Out" comme cela a été fait dans les travaux de Rad *et al.* (2015). Cette technique consiste à créer un ensemble de données de test à partir de toutes les données obtenues pour un objet donné et à utiliser le restant des démonstrations pour la base de données d'entraînement. Cela garantit que le réseau généré n'a jamais vu aucun exemple de signal généré à partir d'un objet de test. La base de données de test est ensuite utilisée pour évaluer que le système est capable de généraliser ce qu'il a appris sur de nouveaux objets que celui-ci n'aurait jamais vu. Dans nos expériences, seuls quatre des huit objets (goujon métallique, tube de PVC, variateur de lumière et l'insertion en X) ont été utilisés à tour de rôle comme ensemble de test, car c'étaient les seuls objets possédant à la fois des données de tâches d'insertion et des données PAP.

Les bases de données d'entraînement ont été augmentées en faisant glisser chaque échantillon de signal vers la droite et la gauche sur une période de temps d'un maximum d'une seconde. Cette action a triplé la quantité de données pour l'entraînement.

Pour tenter d'obtenir un résultat à comparer avec nos méthodes d'extraction de caractéristiques (CNN), la première expérience menée consistait à classer les signaux d'entrée de test sans extraction de caractéristiques. Pour ce faire, nous avons directement montré les échantillons de données du capteur F/T à un réseau de neurones complètement connecté similaire à celui utilisé dans la sortie des couches de convolution de notre CNN. Ce réseau consiste en une couche cachée de 1000 neurones qui est suivie par une autre couche de deux neurones softmax, chacune associé à une classe.

Clicours.COM

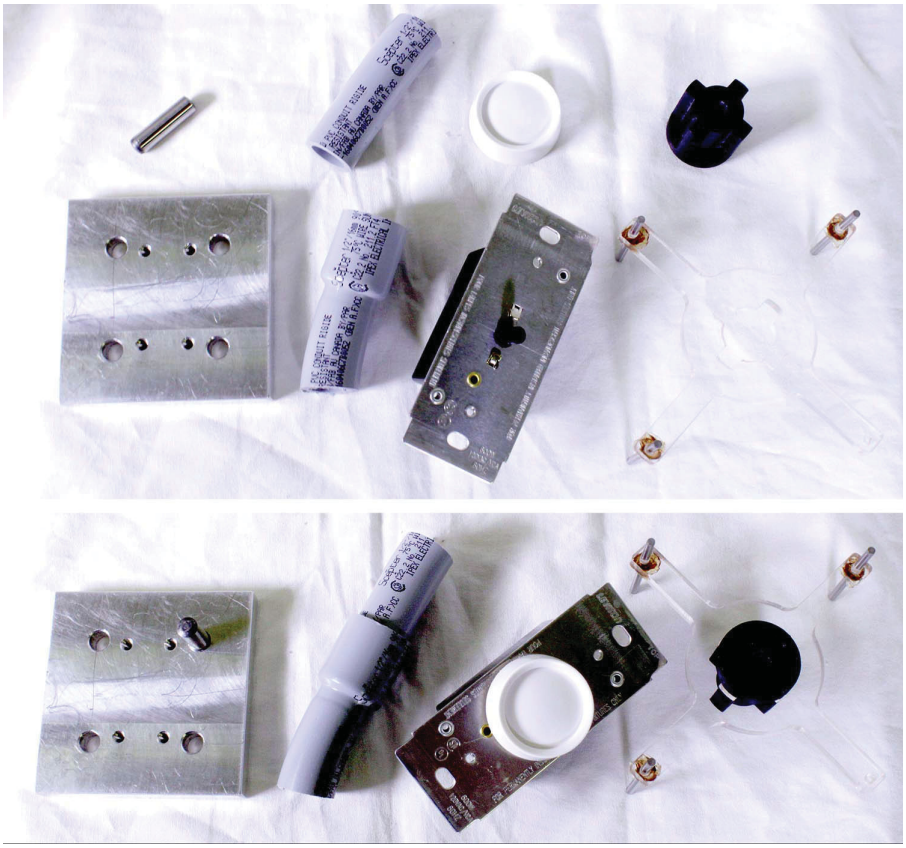


Figure 3.5 Les objets utilisés dans les bases de données de test. De gauche à droite : goujon métallique, tube de PVC, variateur de lumière et l'insertion en X.

Pour que les signaux F/T correspondent à l'entrée de ce réseau, nous avons concaténé toutes les forces et torques des données des capteurs F/T pour obtenir un tableau de 6000 points de données de longueur. Nous nous référerons à cette première expérience comme étant une référence de performance pour notre CNN.

La deuxième expérience effectuée consistait à classer le même ensemble de données de tests que dans la première expérience, mais cette fois-ci en utilisant notre technique d'extraction de caractéristiques proposée à la section 3.2.2 (CNN). En comparant ce travail avec la première expérience, il était possible d'évaluer si l'extraction de caractéristiques donne de meilleurs résultats en termes de détection des tâches d'insertion et ainsi prouver que cette méthode est un bon candidat pour faire partie d'un système d'apprentissage par démonstration.

Dans les deux expériences, nous avons optimisé nos réseaux en utilisant un algorithme d'Adam (Kingma & Ba (2014)) pour calculer les gradients de nos réseaux. Afin de maximiser la généralisabilité du réseau entraîné et d'empêcher le réseau de se surentraîner, nous avons utilisé un taux d'abandon de 0,3 (Srivastava *et al.* (2014)) dans chaque couche de nos réseaux de neurones.

Finalement, puisque les deux réseaux contiennent plusieurs modules à caractère aléatoire, comme la distribution normale de l'initialisation des poids des neurones ou l'algorithme stochastique d'optimisation basé sur les gradients de premier ordre, les réseaux ne donneront pas toujours le même résultat pour une base de données de test spécifique à chaque entraînement. Ainsi, nous avons généré dix réseaux différents en utilisant la base de données d'entraînement pour chaque base de données de test et avons évalué la précision de chacun de ces réseaux sur les données de test associés. Nous présentons la moyenne et l'écart-type des résultats dans le tableau 3.1 qui, à notre avis, représente fidèlement les résultats attendus si quelqu'un essayait de recréer l'expérience.

3.4.2 Résultats expérimentaux

Comme on peut le voir à l'aide du tableau 3.1, la précision générale de notre CNN de 82%, dépasse le résultat obtenu avec l'expérience de référence, soit 78%, avec une marge de 4%. Le fait que le résultat moyen de notre CNN soit plus élevé montre clairement l'avantage de l'extraction de caractéristiques. Nous pouvons également considérer le résultat général comme un indicateur des bonnes performances du CNN.

Tableau 3.1 Résultats obtenus lors des quatre expériences de test.

Résultats	**Expériences**	**Goujon**	**Tube**	**Variateur**	**Insertion X**	**Moyennes**
Accuracy	Référence	0.7±0.03	0.73±0.2	0.79±0.04	0.89±0.05	0.78
	CNN	**0.74±0.02**	**0.79±0.04**	**0.85±0.04**	**0.91±0.03**	**0.82**
Precision	Référence	0.65±0.02	0.9±0.04	**0.95±0.03**	1.00	**0.88**
	CNN	**0.68±0.02**	**0.91±0.0**4	0.89±0.03	1.00	0.87
Recall	Référence	0.89±0.06	0.41±0.07	0.53±0.11	0.77±0.12	0.65
	CNN	**0.96±0.02**	**0.57±0.09**	**0.73±0.09**	**0.8±0.07**	**0.77**

Si nous examinons de plus près les résultats obtenus avec le premier objet (goujon métallique), nous pouvons voir que la précision du réseau est plutôt faible (68%) et que le taux de rappel (recall rate) est relativement élevé (96%), ce qui signifie que la majorité des erreurs commises par le système étaient des faux positifs. Le tout est probablement dû au fait que les mouvements générés lors de l'étape d'approche des démonstrations de PAP sont similaires aux mouvements générés lors de l'étape d'approche de la tâche d'insertion. Étant donné que l'objet est petit et que les démonstrations de PAP exigeaient que l'objet soit placé dans un endroit très précis, le placement de l'objet nécessitait de nombreux ajustements précis au cours de l'étape d'approche et des ajustements similaires nécessaires pendant l'approche d'une insertion.

Le phénomène inverse peut être observé avec l'objet 2 (tuyaux en PVC). Ce test a permis d'obtenir une excellente précision (91%) et un faible taux de rappel (57%), ce qui signifie que certaines des insertions n'ont pas été détectées (faux-négatif). Cela peut s'expliquer par le fait que l'exécution d'insertion avec le tuyau de PVC était beaucoup plus facile qu'avec les autres objets. Étant donné que peu d'ajustements avaient à être effectués lors de l'insertion du tuyau dans l'autre, nous croyons que le réseau considère que les ajustements pré-insertions comme les principales caractéristiques pour faire la discrimination entre une tâche d'insertion et une démonstration de PAP. L'absence d'ajustement par l'opérateur avant l'insertion a rendu cette tâche difficile à détecter.

3.5 Exemple d'utilisation du CNN sur des démonstrations complètes

L'expérience précédente a montré comment un CNN entraîné pour faire la détection de tâche d'insertion peut être utilisé pour détecter ces tâches avec une bonne précision. Cependant, le réseau développé n'est capable d'analyser que 8 secondes (1 000 points de données) du signal F/T à la fois, ce qui est considérablement inférieur à ce qui serait nécessaire pour analyser les données d'une démonstration complète. Dans cette section, nous explorerons comment notre réseau peut identifier une tâche d'insertion avec un signal d'entrée d'une longueur indéterminé.

3.5.1 Utilisation du CNN pour des entrées plus longues

Étant donné que le CNN entraîné prend une entrée de taille fixe de 8 secondes en raison de ses dernières couches entièrement connectées, un signal de démonstration standard est habituellement plus long (en termes de temps) et doit être segmenté ou redimensionné dans un format compatible avec le réseau.

Un moyen populaire consiste à utiliser un CNN sur des régions (R-CNN, Girshick *et al.* (2014)). Cette technique, généralement utilisée dans le traitement des images pour la localisation d'objets, utilise des algorithmes de proposition de région pour sélectionner les régions positives potentielles de l'image. Ces régions peuvent être trouvées en utilisant plusieurs techniques, notamment la technique de "objectness" (Alexe *et al.* (2012)) ou la recherche sélective (Uijlings *et al.* (2013)). Cependant, nos données n'étant pas aussi substantielles que celles utilisées dans le traitement classique des images, il n'est pas nécessaire d'utiliser la méthode de proposition de région. En effet, les techniques proposées sont utilisées pour sauver du temps lors du traitement des images puisque le nombre de régions qui peuvent être étudiées est très grand. Dans notre cas, puisque notre signal ne contient qu'une dimension, faire l'étude de l'entièreté des régions possibles du signal ne cause pas de problème au niveau du temps de calcul.

Nous avons plutôt simplement segmenté le signal entier en fenêtres de 8 secondes (ici l'équivalent de 1 000 points de données, la grandeur de l'entrée du CNN). La segmentation a été effectuée sur l'ensemble du signal en passant la fenêtre d'entrée du CNN sur l'ensemble du signal avec un pas de 10 (0,08 seconde).

Une fois la segmentation terminée, nous sommes en mesure d'analyser la totalité de la longueur du signal. Au lieu d'utiliser une sortie binaire déclarant si le signal est une tâche d'insertion ou non, nous utiliserons la sortie softmax de notre CNN pour obtenir un niveau de confiance pour chaque entrée que nous montrons au réseau. Nous accumulons alors ces niveaux de confiance dans un vecteur de la même longueur que la démonstration complète en entrée. Pour chaque 8 secondes de signal que nous analysons, nous ajoutons la valeur de sortie softmax à l'endroit correspondant aux 8 secondes dans le vecteur de sortie.

En général, en supposant que la durée de la démonstration est n, les étapes de la tâche de détection peuvent être énumérées comme suit :

1. Organiser les signaux de force de la démonstration dans une matrice $F^{6\times n}$.
2. Construire un vecteur de sortie y^n pour accumuler les niveaux de confiance du CNN sur toute la durée de la démonstration.
3. Faire $n/10$ nouvelles matrices d'entrées $N^{6\times 1000}$ à partir de la démonstration complète, en faisant glisser une fenêtre de longueur 1000 sur la matrice F avec un pas de 10.
4. Introduire les nouvelles matrices N dans le CNN précédemment entraînées et sauvegarder la valeur de sortie softmax, r, pour la détection d'une tâche d'insertion (cette valeur correspond à un niveau de confiance compris entre [0, 1]).
5. Ajouter la valeur r aux valeurs correspondantes du vecteur de sortie y. Par exemple, la sortie de la matrice $N(0:1000)$ est ajoutée à $y(0:1000)$ et la sortie de la matrice N(t :t+1000) est ajoutée à $y(t:t+1000)$.

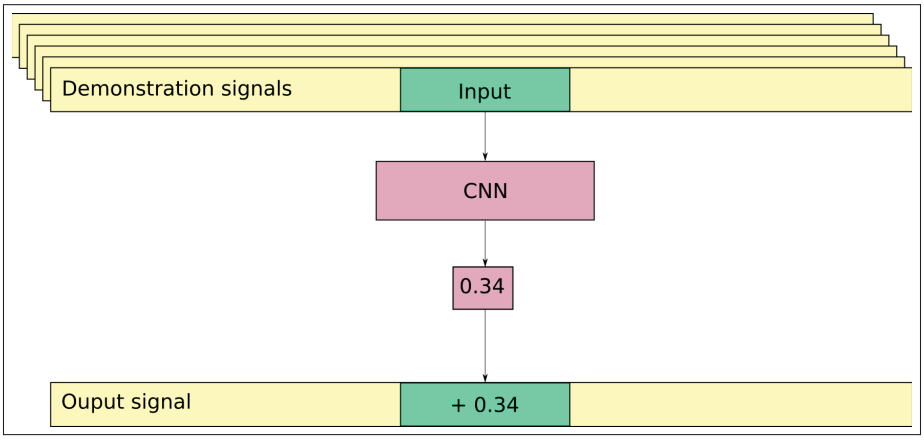


Figure 3.6 Comment le vecteur de sortie est produit à partir d'une démonstration complète.

Suivre les étapes ci-dessus formera un vecteur de sortie de la même longueur que l'entrée et indiquera à quel endroit, sur le signal d'origine, une tâche d'insertion est la plus probable s'il y a lieu.

3.5.2 Exemple d'utilisation du CNN pour une démonstration complète

Dans cette sous-section, nous décrivons comment nous avons utilisé la méthode mentionnée dans la section 3.2.2 pour analyser les signaux F/T d'une démonstraion complète. Pour cet exemple, nous avons pris le signal d'une démonstration où une tâche d'insertion et une tâche PAP ont été effectuées. Les objets utilisés dans cette démonstration sont illustrés à la figure 3.7. La tâche démontrée consistait à prendre la prise de courant du chargeur de batterie et à l'insérer sur le chargeur, puis à ramasser la batterie et à la placer à un endroit précis de la surface de travail. Comme pour les expériences précédentes, cette démonstration a été réalisée avec des objets non utilisés pour entraîner le réseau.

Figure 3.7 Objets utilisés pour l'expérience de démonstration complète. En haut à gauche : prise de courant ; à droite : chargeur de batterie ; en bas à gauche : batterie.

La figure 3.8 montre le vecteur de sortie de la démonstration complète ainsi que les signaux F/T associés à la démonstration. Dans cet exemple, une valeur cumulée de 30 dans le vecteur de sortie a été choisie arbitrairement comme étant la valeur qui entraîne la détection d'une tâche d'insertion.

Nous pouvons observer dans le vecteur de sortie que le temps prévu pour une tâche d'insertion couvre presque complètement le temps annoté pour la tâche d'insertion. Avec une valeur de

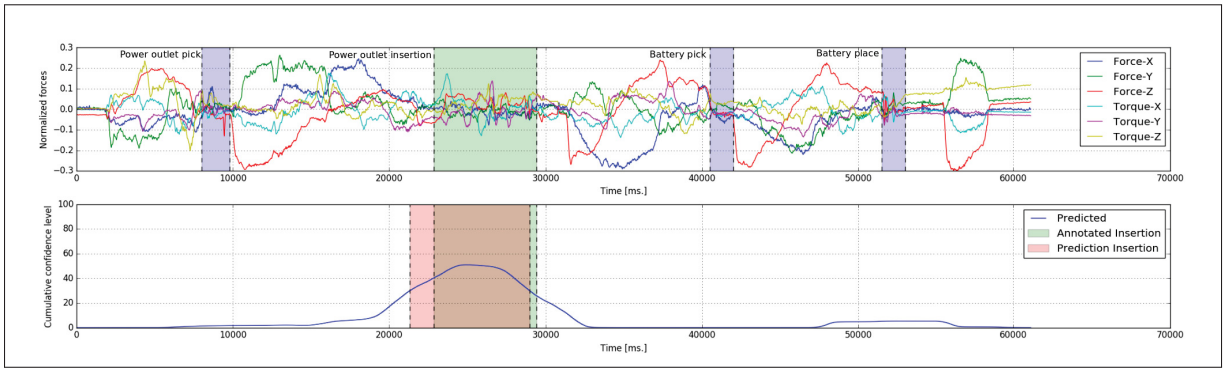


Figure 3.8 En haut : signaux F/T complets pour une démonstration d'enseignement kinesthésique, avec les tâches annotées. En bas : vecteur de sortie contenant les valeurs cumulées de softmax pour la détection de tâches d'insertion.

seuil de 30, 93,8% de l'insertion annotée est considérée comme une insertion par le CNN et seulement 19,5% de la région d'insertion prédite n'est pas annotée en tant qu'insertion.

Bien que la tâche d'insertion n'ait pas été parfaitement détectée en connaissant la position générale dans le temps de la tâche, il est possible de trouver le moment dans la démonstration où le robot s'est arrêté et de trouver ainsi le point d'arrêt de l'insertion. Ce faisant, on pourrait éventuellement remplacer cette partie de la démonstration par une technique d'insertion plus appropriée.

La deuxième chose que nous pouvons observer est que le vecteur de sortie semble avoir une petite activation à l'endroit de la tâche PAP. Dans ce cas, la valeur de sortie de cette région ne suffit manifestement pas pour être considérée comme une tâche d'insertion et est donc correctement classée.

3.6 Discussion

Dans ce chapitre, nous avons montré qu'il est possible de détecter automatiquement les tâches d'insertion en enseignant un robot à l'aide d'un contrôle par admittance. Notre méthode utilisant des réseaux de neurones pour analyser les forces et les couples appliqués par l'opérateur pendant l'enseignement a montré des résultats prometteurs. Nos résultats indiquent également

que l'extraction des caractéristiques (par les CNN) peut légèrement améliorer le taux de détection global par rapport à celui d'un réseau plus simple.

Bien que la méthode utilisée dans ce chapitre semble prometteuse pour la détection des tâches d'insertion dans les démonstrations d'enseignement kinesthésique, il est important de noter que d'autres types de réseaux de neurones pourraient également fonctionner. Notamment, les réseaux de neurones récurrents, comme les LSTM, ont donné d'excellents résultats dans des tâches temporelles telles que la reconnaissance de la parole (Graves *et al.* (2013)). Ce type de réseau pourrait être comparé à la technique utilisée ici dans de futurs travaux.

Dans d'autres travaux éventuels, nous pourrions intégrer différents types de capteurs dans la boucle en plus du capteur de force afin d'améliorer la caractérisation de la tâche par le système. Par exemple, Roberge *et al.* (2016) ont utilisé un autre type de capteur, un capteur tactile dans ce cas-ci, et ont prouvé qu'il est possible de détecter le glissement d'un objet qui est saisi par un préhenseur sur une surface externe. Cette méthode de détection par rétroaction tactile pourrait être utilisée pour renforcer le taux de prédiction de notre travail.

clicours.com

CHAPITRE 4

DÉVELOPPEMENT D'UN SYSTÈME DE REMPLACEMENT DE TÂCHE D'INSERTION

Comme nous avons déjà abordé auparavant, l'utilisation d'un système de suivi de trajectoire pour faire la programmation d'un robot collaboratif accélère grandement le temps de programmation pour faire une tâche et permet aussi de programmer des mouvements plus humains. Ces systèmes utilisent un contrôle kinesthésique et enregistrent les mouvements effectués pendant la phase de démonstration pour créer une "replay" qui répète point par point la démonstration exécutée.

Certains systèmes de suivi de trajectoires, tel que le système de suivi de trajectoire "Active Drive" de la compagnie Robotiq, permettent l'intégration facile d'une replay d'une démonstration à l'intérieur d'un programme à l'aide d'une simple ligne de code. Par contre, ces suivis de trajectoire assument que l'opérateur est capable de démontrer une trajectoire valide en tout temps puisque les replays générés respecteront exactement ce que la démonstration était. En connaissance de cause, nous savons déjà qu'il est difficile d'enseigner certaines tâches à l'aide d'une replay basé sur l'aspect spatio-temporel d'une démonstration. Plus précisément, nous savons que l'enseignement d'une tâche d'insertion est difficile à faire puisque celle-ci requière que les pièces de l'assemblage soient à la position exacte où elles étaient lors de la démonstration pour que la tâche soit répétable. S'il était possible d'évaluer qu'un tel type de tâche a été effectué pendant la démonstration, il serait donc possible de remplacer cette partie de la démonstration par une séquence plus robuste. Nous pourrions donc finalement démontrer la totalité de la tâche à effectuer à l'aide d'une démonstration et il ne serait plus nécessaire de programmer textuellement la séquence à effectuer du robot.

Nous croyons avoir prouvé lors du dernier chapitre qu'il est possible de détecter l'exécution de tâche d'insertion lors d'une démonstration au robot faite à l'aide d'un contrôle par admittance. Par contre, la méthode utilisée ne permet qu'étudier que 8 secondes de signal à la fois et ne permet que de détecter si une insertion est présente sans la localiser. Nous avons ensuite regardé

Clicours.com

s'il était possible de localiser l'insertion à l'intérieur d'une démonstration à l'aide d'un algorithme simple qui applique la démonstration itérativement au réseau de neurones pour détecter les sections où une insertion est présente avec un certain succès.

Dans ce chapitre, nous discuterons du développement d'un réseau capable de faire la détection et la localisation d'une tâche d'insertion. Nous comparerons cette nouvelle architecture avec celle présentée précédemment dans la section 3.2.2 à l'aide d'expérimentation similaire. Dans un deuxième temps, nous proposons une façon de segmenter une démonstration pour isoler une tâche d'insertion et l'éliminer pour la remplacer par une technique plus robuste aux incertitudes.

4.1 Amélioration du CNN : localisation d'une tâche dans un signal

Nous avons développé un réseau de neurones convolutifs à la section 3.2.2 de ce document capable de déterminer avec un taux de succès de 82% si une insertion est présente. Bien que ce taux de reconnaissance représente un pas dans la bonne direction, celui-ci est encore loin de convenir pour un usage réel dans un milieu industriel. Nous considérons par contre que ce réseau prouve que la détection est possible et que le développement d'un système plus robuste serait possible après quelques itérations de prototype.

Bien que les systèmes de vision par ordinateur sont désormais d'une efficacité étonnante, cela n'a pas toujours été le cas. La croissance de la performance des systèmes de vision peut s'expliquer par plusieurs raisons, mais une des raisons primaires est la création de bases de données communes pour la recherche mondiale, tel que ImageNet (Deng *et al.* (2009)). En ayant cette base de données disponible pour tous, plusieurs équipes ont développé des réseaux de neurones qui sont maintenant des références dans le domaine. Tel qu'expliqué dans les travaux de Nguyen *et al.* (2018) et affiché dans la figure 4.1, les réseaux gagnants du ILSVRC (ImageNet Large Scale Visual Recognition Challenge) sont devenus des standards dans l'industrie : AlexNet en 2012 (le premier réseau profond à avoir gagné), les VGG16-19, le GoogLeNet et plus récemment les ResNets.

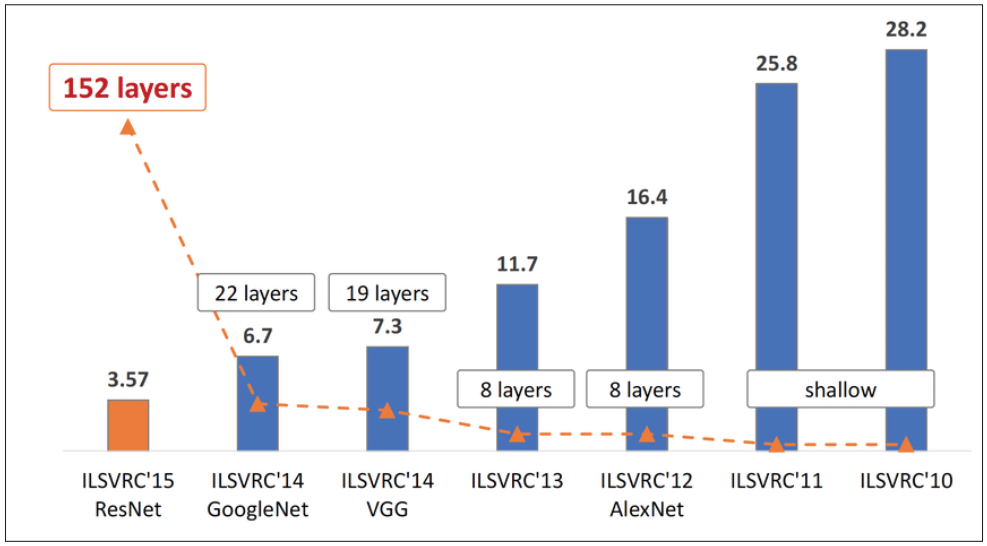


Figure 4.1 Évolution des résultats obtenus lors de la compétition au fil des ans. Le nombre de couche de neurones augmente vs. le pourcentage d'erreur obtenu. Tirée de https ://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

De nos jours, ces réseaux sont utilisés partout et il est même possible de télécharger des versions de ces réseaux pré-entraînés pour faciliter l'intégration de ceux-ci à une application.

Nous avons donc décidé d'utiliser une architecture standard en sachant que ceux-ci ont été étudié et éprouvé à maintes reprises dans le domaine de la vision par ordinateur. Dans notre cas, nous avons choisi d'implémenter une architecture VGG (Simonyan & Zisserman (2014)). Ce réseau a prouvé qu'il est possible d'avoir de bonnes performances en utilisant que des filtres 3x3 pour l'ensemble du réseau. Les avantages de ce réseau résident dans sa simplicité d'implémentation et sa bonne généralisation sur un ensemble d'application.

Bien qu'il existe des réseaux de style VGG pré-entraînés disponibles par téléchargement, ceux-ci sont souvent entraînés grâce à la base de données ImageNet. Nous ne pouvons donc pas utiliser un de ces réseaux pré-entraînés pour notre projet puisque la nature de nos signaux d'entrée est complètement différente de ceux utilisés pour entraîner ces réseaux. Nous construirons alors notre propre base de données ainsi que nos méthodologies d'entraînement.

Dans cette section, nous discuterons d'une nouvelle architecture CNN plus profonde pour faire la détection et la localisation de tâches d'insertion, les méthodes utilisées pour construire notre base de données et nous comparerons les performances de ce nouveau réseau avec ceux obtenus précédemment.

4.1.1 Approche proposée

Pour cette itération du CNN que nous utiliserons, nous avons opté pour une architecture du style VGG, comme présenté dans les travaux de Simonyan & Zisserman (2014).

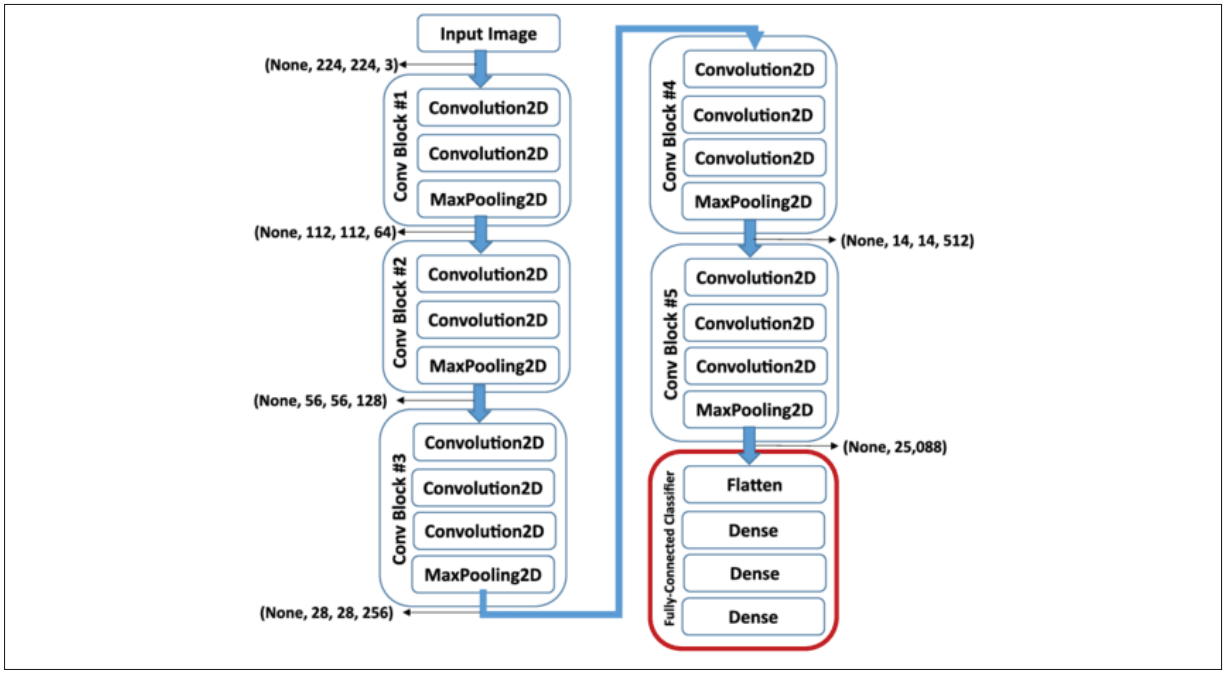


Figure 4.2 Architecture typique d'un réseau VGG16. Tirée de https ://www.researchgate.net/figure/A-schematic-of-the-VGG-16-Deep-Convolutional-Neural-Network-DCNN-architecture-trained_fig2_319952138

Comme on peut le voir à la figure 4.2, l'architecture d'un réseau VGG (dans ce cas-ci, VGG-16 puisqu'il contient 16 couches de profondeur) est sous-sectionné par bloc de convolution. Chacun de ces blocs contient 2 ou 3 couches de convolution 3x3 suivi d'une opération de sous-échantillonnage pour diminuer la grandeur des entrées de la prochaine couche. Finalement,

après plusieurs blocs de convolution, la sortie obtenue est envoyée à un réseau de neurones complètement connecté pour faire la classification de l'entrée du réseau.

Dans notre cas, nous utiliserons une architecture telle qu'illustrée dans la figure 4.3.

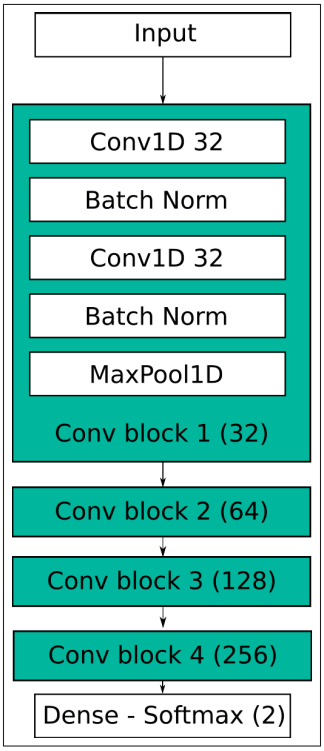


Figure 4.3 L'architecture utilisée pour la classification de segment de démonstration, inspirée de l'architecture VGG16.

Notre architecture est composée de 4 blocs de convolution. Chacun de ces blocs est composé de plusieurs couches, notamment :

- *Deux couches de convolution* : ces couches ont chacune le même nombre de filtres pour le même bloc (par exemple, 32 filtres pour le bloc 1). Les filtres utilisés pour l'ensemble du réseau, c'est-à-dire pour tous les blocs de convolution, sont d'une taille 3x1, puisque nous avons un signal temporel.
- *Deux couches de normalisation par lot (batch normalization)* : ces couches permettent dans un premier temps d'accélérer le temps d'entraînement du réseau de neurones en normalisant

la sortie de chacune des couches de convolution et ainsi en réduisant la valeur de l'entrée de la couche de convolution suivante. En ayant des valeurs d'entrées normalisées, les neurones ne risquent pas de prendre des valeurs extrêmes pour compenser une entrée ayant des valeurs trop grandes. Dans un deuxième temps, ces couches de normalisation permettent au réseau de mieux généraliser puisque les données normalisées contiennent peu de bruit. La couche suivante s'entraîne donc sur des données contenant du bruit et devient plus robuste (Ioffe & Szegedy (2015)).

La normalisation par lot se fait en soustrayant la moyenne du lot et en divisant le résultat par l'écart-type du lot. Ces couches ajoutent donc deux nouveaux paramètres à entraîner, soit la moyenne des lots et l'écart-type pour chacune des sorties de convolution.

- *Une couche de sous-échantillonage par maximum* : cette couche est toujours la dernière de chacun des blocs de convolution et permet de réduire la taille de l'entrée du bloc suivant et ainsi réduit le nombre de paramètre à optimiser pour les couches suivantes. Cette couche ajoute aussi une invariance au déplacement dans le temps et à la mise à l'échelle (scaling) des caractéristiques importantes pour la reconnaissance d'insertion.

Les blocs de convolution utilisés dans notre projet contiennent respectivement 32, 64, 128 et 256 filtres par couches de convolution interne. Finalement, la sortie du dernier bloc de convolution est envoyée dans une couche de neurones complètement connectés avec des activations du type softmax pour obtenir une probabilité que le signal contienne une tâche d'insertion.

4.1.1.1 Intégration d'un système de localisation de tâches d'insertion en sortie du CNN

Contrairement au réseau que nous avons développé au chapitre 3, nous voulons maintenant aussi être capable de situer dans le temps une tâche d'insertion à l'intérieur d'un signal d'entrée. Pour ce faire, nous pouvons utiliser un réseau, tel que celui proposé à la figure 4.3, dont la sortie aurait une activation linéaire plutôt qu'une activation de type Softmax. En utilisant une sortie avec une activation linéaire, nous proposons d'entraîner le réseau à être capable de trouver le moment précis du début de l'insertion ainsi que la fin de la séquence.

Puisque faire la classification du signal (discrimination) et trouver le temps de la tâche d'insertion (régression) nécessitent de découvrir les caractéristiques importantes dans le signal, nous croyons que plusieurs blocs de convolution peuvent être utilisés conjointement pour les deux tâches à exécuter. Nous croyons que les premières couches du CNN s'entraîneront à être capable de filtrer des caractéristiques de base du signal d'entrée qui sont utilisées pour la classification et la localisation d'une tâche d'insertion. C'est pourquoi nous proposons une architecture CNN démontrée à la figure 4.4.

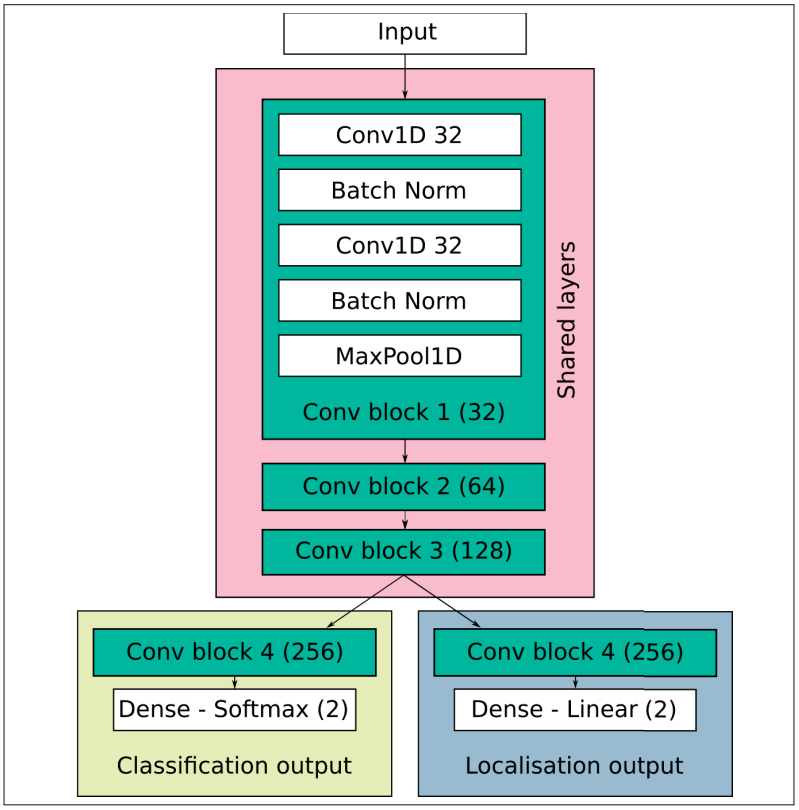


Figure 4.4 Architecture complète du réseau utilisée dans ce projet. Après le troisième bloc de convolution, deux blocs de convolutions spécialisés sont entraînés pour classifier l'entrée et localiser une insertion.

Dans cette architecture, un bloc de convolution spécialisé est formé pour apprendre les fonctions importantes pour classer les signaux d'insertion (section jaune de la figure 4.4) après le troisième bloc de convolution. La sortie de ce bloc est introduite dans deux neurones ayant une

sortie Softmax d'une manière entièrement connectée pour obtenir une classification binaire du signal d'entrée.

En plus d'être capable de détecter si un segment de signal contient une tâche d'insertion, nous devons également être en mesure de localiser le moment où l'insertion a été effectuée pendant la démonstration. Ceci est fait par un bloc de convolution spécialisé formé pour apprendre les fonctionnalités importantes pour localiser les tâches d'insertion (section bleue de la figure 4.4). Comme la partie localisation du système n'est pas une sortie binaire, nous avons utilisé deux neurones entièrement connectés avec une activation linéaire pour représenter le temps de début et le temps de fin de la tâche d'insertion dans le signal d'entrée.

4.1.2 Acquisition et traitement de la base de données

4.1.2.1 Acquisition de démonstrations complètes pour la construction d'une base de données

Dans le but d'être capable de faire la détection de tâches d'insertion dans une démonstration complète, nous n'avons pas utilisé la base de données fabriquée dans le cadre des expérimentations du chapitre 3 puisque les échantillons de signaux utilisés dans cette partie ne provenaient pas de démonstrations complètes, mais bien de la séquence finale de l'insertion.

Nous avons donc construit une nouvelle base de données à l'aide de démonstrations complètes de tâches à l'aide du contrôle par admittance développé au chapitre 2. Les démonstrations effectuées comportent plusieurs actions : la préhension d'objets, le dépôt d'objets sur une surface, l'insertion d'objets rigides, des suivies de trajectoire précise et le serrage de pièces dans un espace contrainte.

Pour chacune des démonstrations enregistrées, nous avons annoté le temps que les tâches présentées ont été effectué pour ensuite entraîner notre réseau à les localiser. Pour ce faire, l'opérateur devait appuyer sur une touche et tenir celle-ci enfoncée tout au long de la tâche effectuée pour annoter le temps d'exécution. Par exemple, pendant la démonstration, lorsque l'opérateur

s'apprêtait à insérer une clé USB, celui-ci a appuyé sur une touche du clavier pour indiquer au programme d'acquisition que la tâche a débuté. La touche est ensuite tenu enfoncée tout au long de l'insertion, puis relâchée pour indiquer au programme d'acquisition que la tâche est terminée. Après avoir terminé une démonstration complète, le démonstrateur devait ensuite indiquer le type de tâche effectué pour chacune des annotations enregistrées. En reprenant l'exemple précédent, suite à la démonstration complète, l'opérateur a dû indiquer au programme d'acquisition de données que la tâche annotée était une insertion.

Puisque l'annotation du temps de la tâche était faite manuellement, le temps de début d'une tâche d'insertion pouvait grandement varier entre les démonstrations. Effectivement, ce qui représente le début de la tâche d'insertion n'est pas clair pour chacun , tel qu'illustré à la figure 4.5. Nous prendrons note de cette erreur lors des résultats obtenus après l'entraînement du réseau à la section 4.1.4.1.

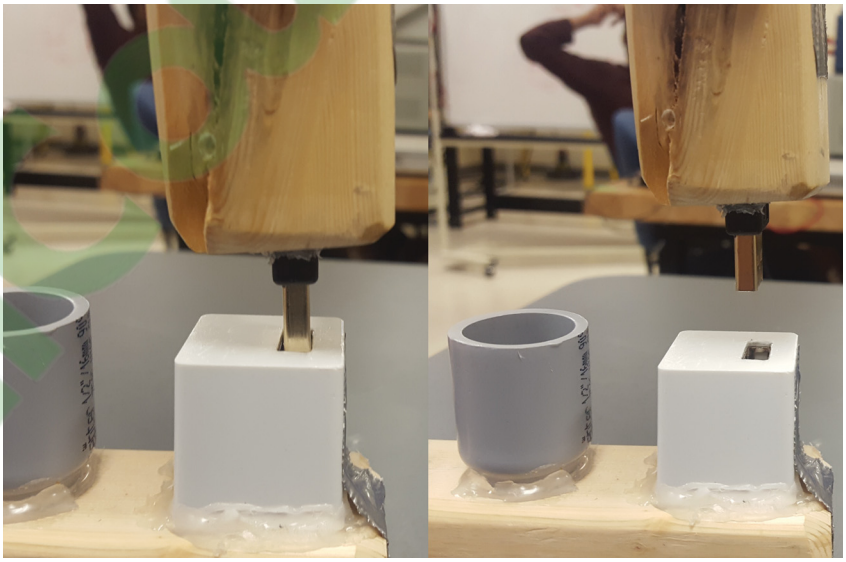


Figure 4.5 Il est difficile de dire à quelle distance du trou la tâche d'insertion débute. À quelques cm ou au contact ?

Cliquez.com

4.1.2.2 Construction de la base de données d'entraînement et de test

Parmi les 1183 démonstrations de tâches enregistrées, 100 démonstrations ont été sélectionnées aléatoirement pour composer une base de données de test et le restant des démonstrations ont été utilisées pour composer une base de données d'entraînement et de validation pour notre CNN. Les démonstrations de notre base de données d'entraînement contiennent alors : 976 tâches d'insertion, 757 tâches de PAP, 201 tâches de serrage et 197 tâches de suivi de trajectoire sur une surface rigide. Les 100 démonstrations retenues pour notre base de données de tests contiennent : 91 tâches d'insertion et 70 tâches de PAP.

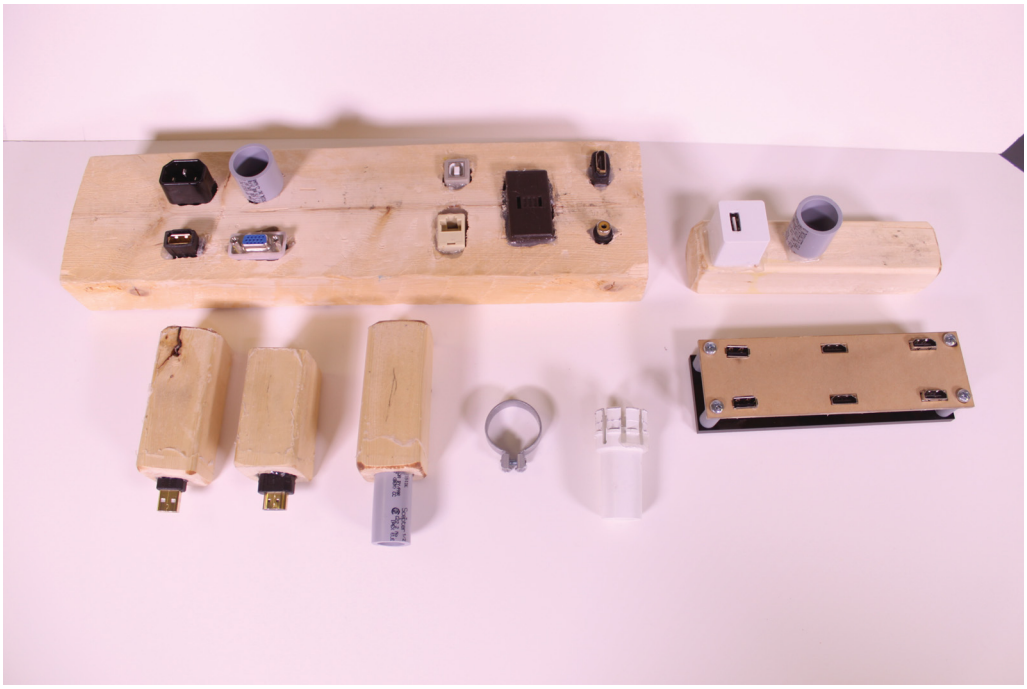


Figure 4.6 Les objets utilisés pour les tâches de la base de données. Une variété de connecteurs standards, tels que des connecteurs USB et HDMI.

En plus d'utiliser les objets de la base de données faite au chapitre 3, nous avons utilisé les objets de la figure 4.6. Ceux-ci consistent principalement de connecteurs standard utilisés par plusieurs dans la vie de tous les jours. Par contre, ces connecteurs ont souvent une tolérance mécanique très petite lors de l'insertion ; en particulier le connecteur HDMI.

Les signaux de démonstration collectés sont en matrice $X^{n\times 9}$, où n est la longueur dans le temps de la démonstration. X peut être considéré comme la concaténation des forces appliquées sur l'effecteur $F_{x,y,z}$, les couples appliqués sur l'effecteur $T_{x,y,z}$ et les vitesses de l'effecteur dans l'espace cartésien $V_{x,y,z}$. Pour rendre cette matrice de longueur n admissible pour l'entrée de notre réseau qui est d'une longueur de 1000 (8 secondes à 125 hz), il nous a été nécessaire de générer des fenêtres $X^{1000\times 9}$ à partir de notre démonstration complète $X^{n\times 9}$. Pour obtenir les fenêtres de signal de longueur fixe d'une démonstration complète, nous avons procédé de la manière suivante :

1. Créer $(n-1000)/100$ fenêtres de longueur 1000 en glissant une fenêtre d'acquisition de longueur 1000 avec un pas de 100 sur la matrice de démonstration complète.
2. Donner une étiquette pour la présence de tâches d'insertion à l'intérieur des fenêtres créées. (Voir section 4.1.3.1 pour les détails de l'étiquetage.)
3. Pour toutes les fenêtres étiquetées positivement pour la présence d'insertions, associer le temps de départ et de fin de la tâche d'insertion par rapport à la position de la fenêtre. (Voir section 4.1.3.2 pour les détails sur l'annotation du temps de la tâche par rapport à la fenêtre de signal.)
4. Puisque le nombre de fenêtres étiquetées négatives est beaucoup plus grand que celles étiquetées positives, réduire le nombre de fenêtres négatives pour obtenir un ratio de $1,2:1$ en enlevant aléatoirement des fenêtres négatives du lot de fenêtres créées.
5. Ajouter les fenêtres (positives et négatives) restantes à la base de données d'entraînement.
6. Répéter les étapes 1 à 5 pour toutes les démonstrations dédiées à la construction de la base de données d'entraînement.

4.1.3 Méthodologie d'entraînement du CNN à multiples sorties

Comme mentionné dans la section précédente 4.1.2.1, chaque inférence d'une entrée s'est vue attribuée une classe binaire (pas d'événement ou tâche d'insertion) et une position de départ et de fin pour l'occurrence de l'insertion. Ce réseau multi-tâches implique que nous devons

Clicours.COM

minimiser deux fonctions de coût pendant l'entraînement, où la partie classification serait une optimisation de l'entropie croisée (cross-entropy loss)

$$L_{class} = -(y_{gt} log(y_p) + (1 - y_{gt}) log(1 - y_p)) \tag{4.1}$$

et la régression serait une optimisation de l'erreur quadratique moyenne.

$$L_{reg} = \frac{1}{n_{batch}} \sum (y_p - y_{gt})^2 \tag{4.2}$$

Ces deux fonctions d'optimisation pourraient être réunies en une seule fonction en les additionnant à l'aide d'une valeur pondérée, mais dans notre cas, la partie de régression du réseau ne devrait pas être entraînée sur un exemple négatif (pas d'insertion), car il n'y a pas de position à apprendre lorsque le signal ne contient pas d'événement. Ainsi, nous avons entraîné ce réseau en deux parties.

4.1.3.1 Entraînement du réseau pour la classification de tâches d'insertion

La classification des segments a été effectuée en prenant des segments de signaux de 8 secondes (la taille de l'entrée du réseau) et en étiquetant positivement ou négativement chacun d'entre eux pour la présence d'une tâche d'insertion. Pour être un segment positif, la fenêtre de 8 secondes doit chevaucher au minimum 70% du segment d'insertion annoté et pour être un segment négatif, la fenêtre ne doit pas chevaucher plus de 30% du segment d'insertion annoté. Les fenêtres d'entrées chevauchant une tâche d'insertion avec un rapport entre ces deux seuils précédents n'ont pas été utilisées pour l'entraînement, car celles-ci portent à confusion. Nous les annotons donc comme indéfinies.

$$y_{gt} = \begin{cases} 0, & r_i \leq 0.3 \\ 1, & r_i \geq 0.7 \\ undefined, & 0.3 < r_i < 0.7 \end{cases} \tag{4.3}$$

Ici, y_{gt} est l'annotation finale (gt : ground-thruth) d'un segment de démonstration et r_i est le rapport de la région d'insertion annotée dans la démonstration d'origine couverte par la fenêtre de segment. La figure 4.7 montre des exemples de fenêtres considérées positives et négatives pour la présence d'une tâche d'insertion, ainsi qu'un exemple de fenêtre indéfinie.

Maintenant que nous avons déterminé l'étiquette (label) assortie pour chacune des fenêtres de la base d'entraînement, nous pouvons entraîner le réseau à classifier chacune de ces fenêtres. Pour ce faire, nous divisons la base de données d'entraînement en deux parties ; nous utilisons 90% des fenêtres de signal pour l'entraînement et le 10% restant pour la validation du réseau. Bien entendu, les exemples présents dans la base de données sont mélangés aléatoirement avant de faire la séparation pour s'assurer que nous avons un échantillon représentatif de l'ensemble dans la base de validation.

Nous entraînons le réseau avec l'algorithme d'optimisation stochastique Adam (Kingma & Ba (2014)) sur l'erreur d'entropie croisée. Nous utilisons des lots de 100 données ainsi qu'un algorithme d'arrêt précoce d'entraînement (early stopping) et un algorithme de baisse du taux d'apprentissage (learning rate decay).

L'algorithme d'arrêt précoce d'entraînement permet d'arrêter l'entraînement du réseau lorsque l'erreur en classification de la base de validation arrête de s'améliorer. Dans notre cas, nous avons paramétré le système pour que celui-ci arrête l'entraînement après 10 époques d'entraînement où le résultat sur la base de validation ne change pas pour le mieux pour éviter d'avoir un réseau surentraîné (overfitted) sur la base d'entraînement.

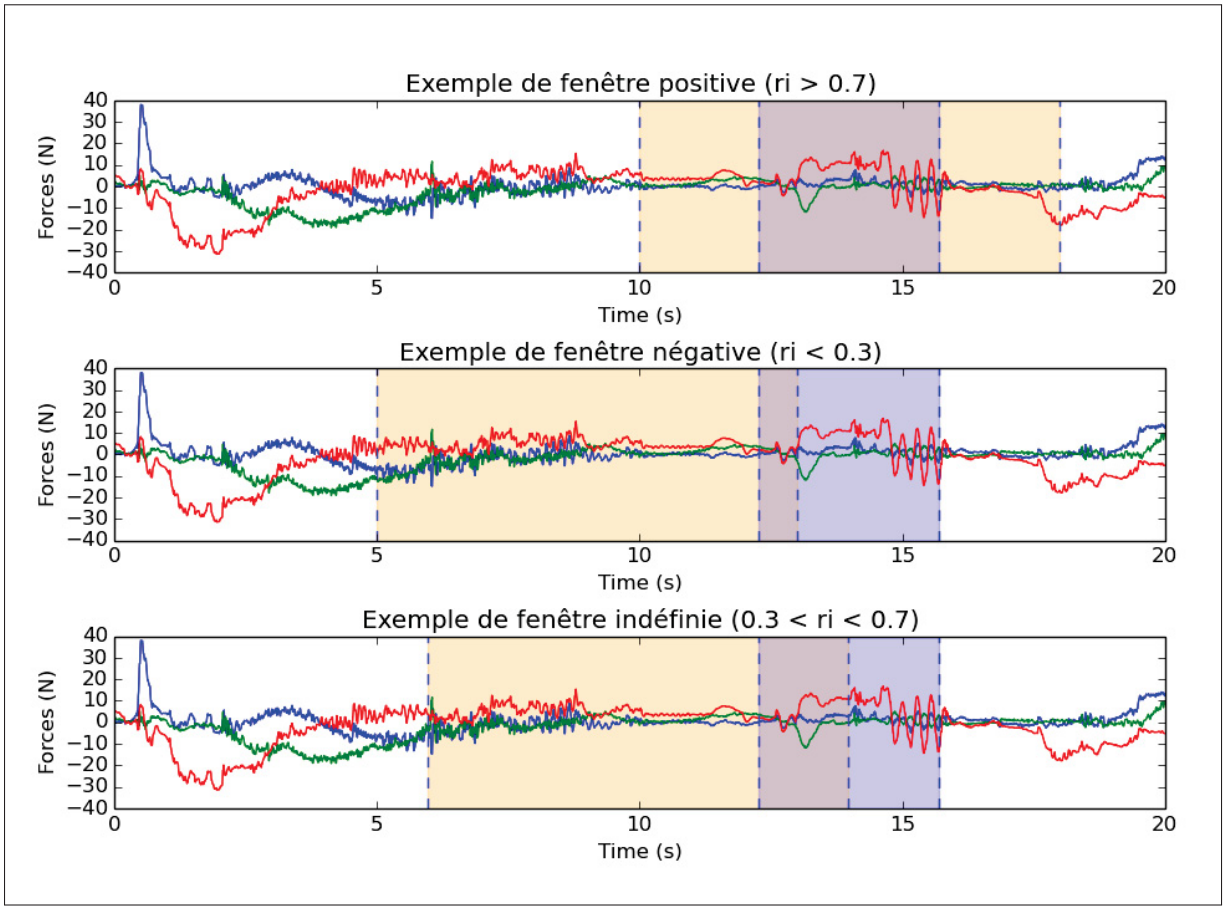


Figure 4.7 Exemples de segment positif, négatif et invalide pour l'entraînement du réseau. En bleu : le temps annoté de la tâche à l'intérieur de la démonstration. En beige : une fenêtre de signaux $X^{1000\times 9}$ obtenue en sous-échantillonnant la démonstration.

L'algorithme de baisse du taux d'apprentissage permet d'ajuster le taux d'entraînement (learning rate) lorsque le résultat en validation cesse de s'améliorer. Réduire le taux d'apprentissage permet parfois de débloquer l'entraînement lorsque celui-ci tombe dans un état où il oscille possiblement autour de la solution optimale parce que l'ajustement des poids du réseau est trop grand. Dans notre cas, nous réduisons le taux d'apprentissage par un facteur de 10 lorsque l'apprentissage ne s'améliore pas après 4 époques.

4.1.3.2 Entraînement du réseau pour la localisation de tâches d'insertion

Une fois que la partie classification du réseau est entraînée sur tous les exemples valides, les poids des 3 premiers blocs de convolution du réseau sont gelés et la partie de régression du réseau est entraînée. Seuls les exemples positifs de données d'insertion sont utilisés pour former cette partie. La figure 4.8 illustre la différence entre l'entraînement des deux derniers blocs.

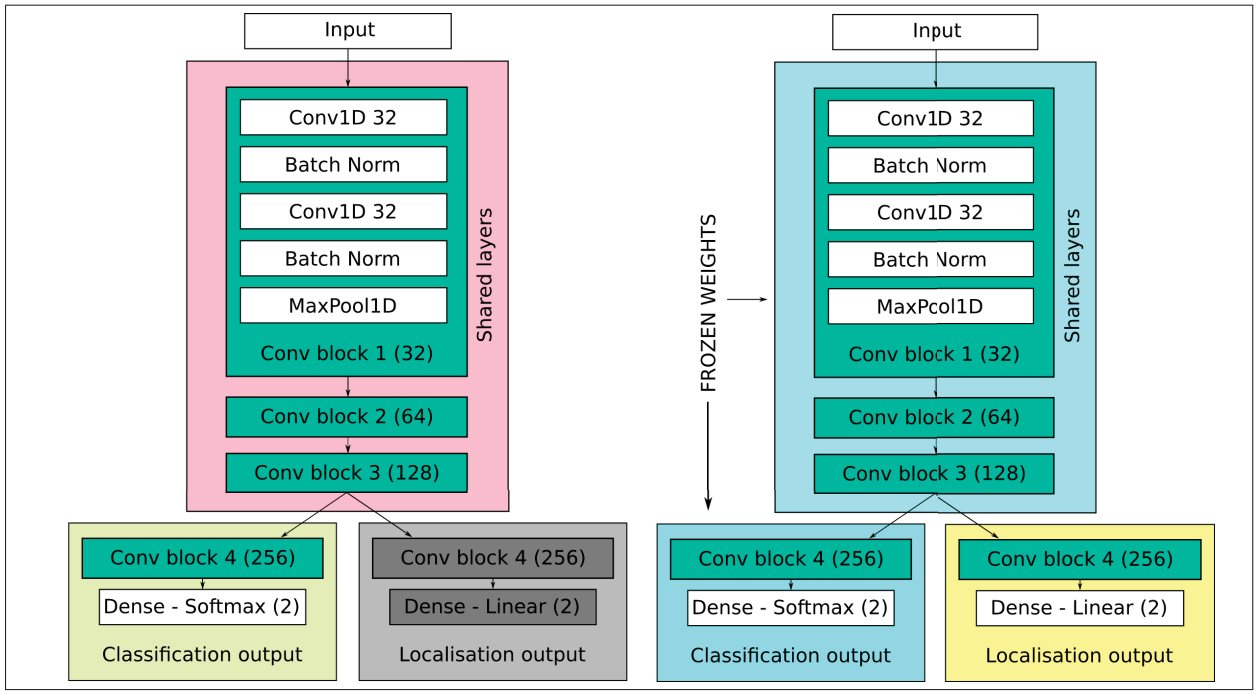


Figure 4.8 Entraînement par étapes. Étape 1 : Entraînement de la sortie en classification du système. Étape 2 : Geler les poids des premières couches de convolution et entraîner la sortie en régression.

Afin d'entraîner la partie régression de notre réseau, nous devons avoir les étiquettes (labels) nécessaires à chaque fenêtre de signal de notre base de données d'entraînement et de validation. Pour ce faire, nous avons pris chacune des fenêtres de signal que nous avions préalablement étiqueté comme étant positives à la section 4.1.3.1 et nous avons donné un index de début et de fin de l'insertion selon la position de cette fenêtre dans le signal de démonstration complète. Pour obtenir les valeurs voulues, nous appliquons :

$$\{start, end\}_{rel} = \{start, end\}_{abs} - i \tag{4.4}$$

où i est l'index de début de la fenêtre envoyé au réseau par rapport à la démonstration complète.

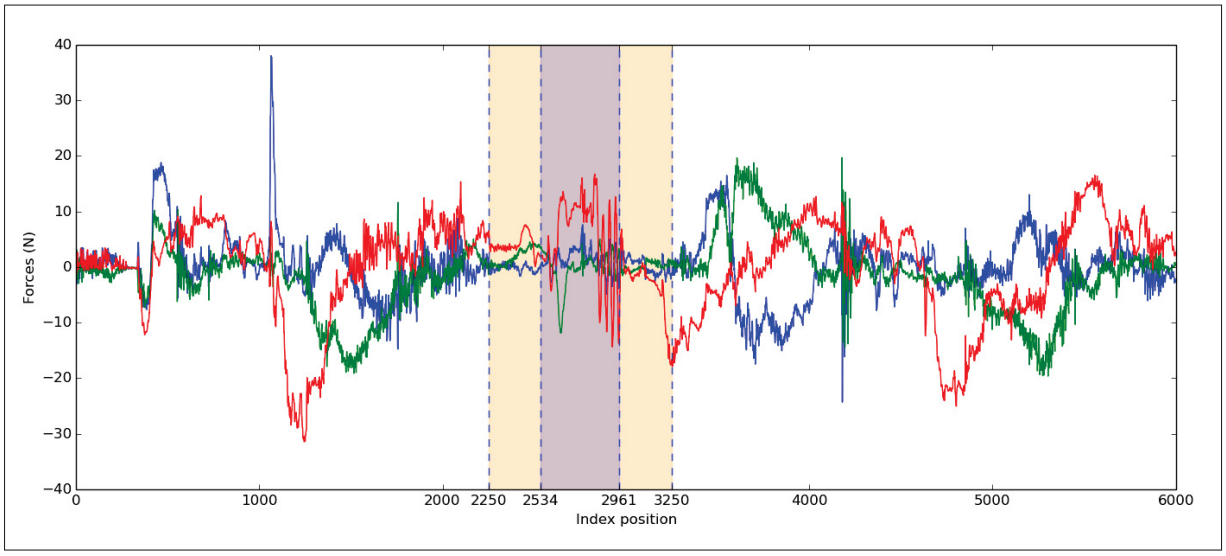


Figure 4.9 Exemple des forces enregistrées lors d'une démonstration d'une tâche d'insertion. En bleu : la région annotée comme étant la séquence d'insertion par le démonstrateur. En beige : une fenêtre de signaux $X^{1000\times 9}$ obtenue en sous-échantillonnant la démonstration.

Par exemple, on peut voir à la figure 4.9 que la fenêtre couvre complètement la tâche d'insertion à l'intérieur de la démonstration complète ; cette fenêtre sera donc classifié e comme positive. Bien que les indexs absolus de début et de fin de la tâche d'insertion soient respectivement 2534 et 2961, nous devons ajuster ceux-ci pour qu'ils soient relatifs à la fenêtre montrée au réseau. Nous obtenons alors les étiquettes suivantes :

$$\{start, end\}_{rel} = \{2534, 2961\}_{abs} - 2250 = \{284, 711\} \tag{4.5}$$

Maintenant que nous avons déterminé les étiquettes assorties pour chacune des fenêtres de la base d'entraînement, nous pouvons entraîner le réseau à localiser les tâches d'insertion à l'intérieur des fenêtres de signal. Pour ce faire, nous utilisons la même division de de la base de données que celle faite à la section 4.1.3.1. Puisqu'une grande partie du réseau est déjà pré-entraînée sur 90% de la base de données, nous devons utiliser les mêmes données pour l'entraînement, question d'être certain qu'il n'y aurait pas de biais à cause d'un surentraînement.

Nous entraînons le réseau avec l'algorithme d'optimisation stochastique Adam (Kingma & Ba (2014)) sur l'erreur quadratique moyenne. Nous utilisons des lots de 100 données ainsi qu'un algorithme d'arrêt précoce d'entraînement (early stopping) et un algorithme de baisse du taux d'apprentissage (learning rate decay), de la même manière qu'à la section 4.1.3.1.

4.1.3.3 Segmentation d'une démonstration à partir des fenêtres de signal générées pour la détection de tâches d'insertion

L'utilisation du CNN pour faire la détection de tâche d'insertion à l'intérieur d'une démonstration complète requière quelques prétraitements et post-traitements de données pour obtenir un résultat utilisable dans un système. Pour obtenir les zones de temps contenant une tâche d'insertion à l'intérieur d'une démonstration, le système développé effectue les tâches suivantes :

1. Générer n fenêtres de signal d'une grandeur de 8 secondes chaque, de la même manière que pour générer la base de données d'entraînement (tel que démontré à la section 4.1.2.2).
2. Passer toutes les fenêtres générées à l'entré du CNN. Pour chaque fenêtre :
   - Si la fenêtre est prédite positive pour la présence d'une tâche d'insertion avec un taux de confiance $> 95\%$, enregistrer le temps de début et de fin prédit de l'insertion et convertir ceux-ci par rapport à la démonstration complète (tel que démontré à la section 4.1.3.2). Annoter cette zone de la démonstration comme étant une tâche d'insertion.
3. Accumuler toutes les zones positives générées par l'entrée des fenêtres de signal au réseau et les associer à la zone correspondante dans la démonstration complète.

4.1.4 Expérience et résultats

Pour tester l'efficacité de notre système à détecter des tâches d'insertion comprises dans des démonstrations complètes, nous avons construit une base de données d'entraînement contenant 1083 démonstrations complètes où des tâches d'insertion, des tâches de PAP et d'autres tâches aléatoires ont été effectuées. Plusieurs objets ont été utilisés pour effectuer ces tâches, comme démontré à la section 4.1.2.2. Après la phase d'entraînement, 100 autres démonstrations contenant des tâches d'insertion et des tâches PAP ont été enregistrées pour être utilisées comme un ensemble de tests. Les performances du CNN ont été mesurées sur cette base de test en mesurant les performances de détection, soit la précision et le rappel (recall rate) des détections, ainsi que la précision de localisation des tâches.

Puisque nous voulons mesurer les performances de détection sur un problème de localisation d'événement dans le temps, nous définissons une détection comme étant "bonne" si la zone de la tâche d'insertion prédite par le système couvre au minimum la moitié de la zone réelle annotée de la tâche d'insertion. On peut décrire cette relation ainsi :

$$r(p,gt) = \frac{min(p_{end}, gt_{end}) - max(p_{start}, gt_{start})}{gt_{end} - gt_{start}} \tag{4.6}$$

Ici, p est le temps prédit (début et fin) de la tâche d'insertion et gt est le temps réel annoté de la tâche d'insertion. Dans nos expériences effectuées, nous considérons qu'une tâche d'insertion est détectée lorsque le ratio $r(p,g)$ est plus grand que 0.5, donc que la tâche d'insertion soit couverte au minimum à moitié par la zone prédite.

4.1.4.1 Résultats de détection sur démonstrations complètes

En utilisant la base de données de tests contenant 100 démonstrations, nous avons obtenu les résultats de détection présentés dans le tableau 4.1.

Tableau 4.1 Résultats de détection de tâche dans la base de données de test d'insertion enseigné par une démonstration complète

	Predicted positive	**Predicted negative**
Ground-truth positive	86	5
Ground-truth negative	7	-
Recall rate	94.51%	
Precision rate	92.47%	
F1 score	**93.48%**	

Ici, nous pouvons observer 86 tâches bien classifiées (Ground-truth positive/Predicted positive). Cinq données n'ont pas été détectées par notre système et ont donc été classifiées négativement (Ground-truth positive/Predicted negative). Finalement, notre système a généré 7 faux positifs (Ground-truth negative/Predicted positive), c'est-à-dire que certaines sections de nos démonstrations ne contenant pas de tâches d'insertion ont été classifiées comme positive.

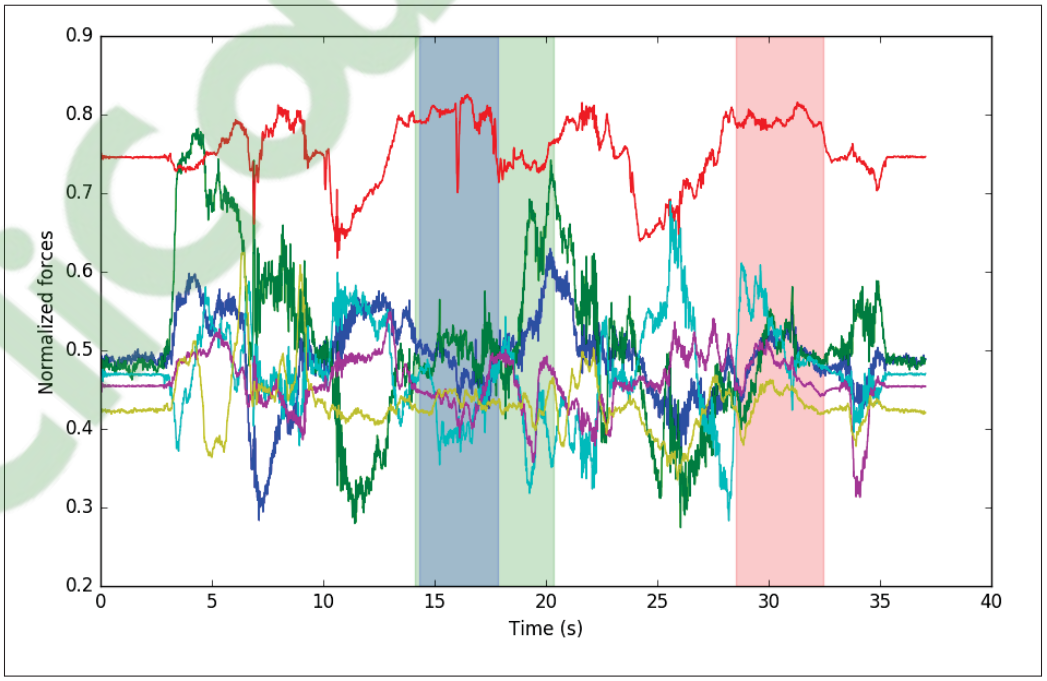


Figure 4.10 Exemple de bonne détection de deux tâches d'insertion à l'intérieur d'une démonstration. En bleu : les périodes où des insertions ont eu lieu annoté par le démonstrateur. En vert : les périodes prédites par le système où des insertions ont lieu. En rouge : les périodes où une tâche de PAP ont lieu.

Nous pouvons observer un taux de rappel de 94,51%, ce qui signifie que 86 des 91 tâches d'insertion présentées dans les démonstrations ont été correctement localisées.

Un taux de précision de 92,47% a été obtenu sur les démonstrations présentées, ce qui signifie que certaines régions de tâches d'insertion prédite par le système n'étaient pas de véritable insertion. Dans ces prédictions faussement positives, la plupart visaient une séquence dans la démonstration où l'opérateur posait un objet sur l'atelier (PAP). Un exemple de détection faussement positive peut être observé à la figure 4.11.

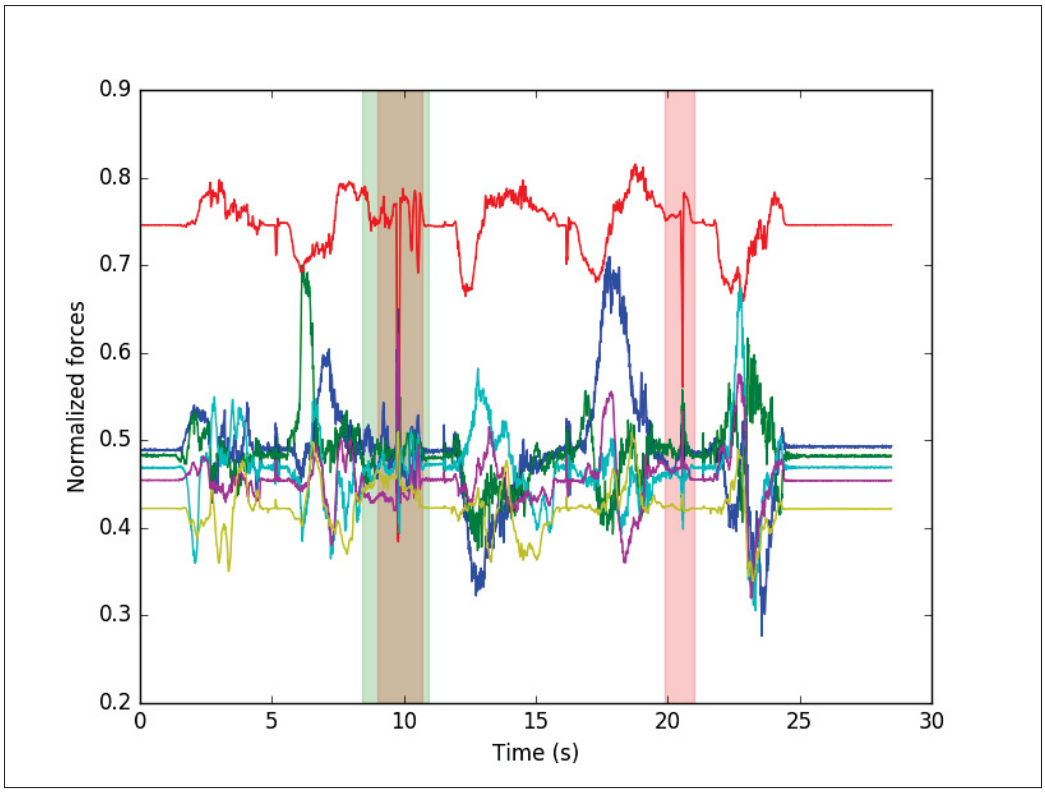


Figure 4.11 Exemple de détection faussement positive. En vert : les périodes prédites par le système où des insertions ont lieu. En rouge : les périodes où une tâche de PAP ont lieu. Cette démonstration ne contient aucune tâche d'insertion.

Dans cet exemple, le système a classifié une tâche de PAP comme étant une insertion. Bien que ce résultat ne soit pas un comportement recherché, ce type de tâche est très similaire à une tâche d'insertion (déposer un objet vs. insérer un objet) et les signaux générés lors de la démonstration expliquent la prédiction faussement positive obtenue ici.

On peut remarquer que le tableau 4.1 ne contient pas le nombre de vrais négatifs. Nous n'avons pas compilé ce résultat puisqu'il n'est pas important pour notre problème. Puisque nous ne voulons que détecter les tâches d'insertion à l'intérieur d'une démonstration, tout le reste des signaux de démonstration sont négatifs et il est donc difficile de mesurer à quel point nous avons bien classifié les signaux négatifs. Par contre, puisque nous n'avons pas compilé ce résultat, il nous est impossible de calculer le taux de succès (accuracy) de notre système puisque la formule du "accuracy" est :

$$accuracy = \frac{\sum True\ positive + \sum True\ negative}{population} \tag{4.7}$$

Dans notre cas, nous utilisons alors comme métrique principale pour mesurer la performance de notre système le F1-score. Cette métrique utilise la précision et le taux de rappel pour obtenir un résultat sans tenir compte des vrais négatifs et permet d'avoir un résultat plus juste sur un ensemble de données de test qui a une disproportion entre le nombre de données positives et négatives. Le F1-score peut être obtenu en calculant la moyenne harmonique entre la précision et le taux de rappel. Dans notre cas, nous avons obtenu un F1-score de 93.48%.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.8}$$

4.1.4.2 Résultats de localisation sur démonstrations complètes

En plus d'être capable de faire la détection d'une tâche d'insertion, le système doit être capable de situer la tâche dans le temps convenablement. Pour mesurer cette métrique, nous n'utiliserons pas le taux de chevauchement de la zone d'insertion (comme souvent utilisé précédemment pour définir si une détection est bonne ou non), mais nous utiliserons la notion d'IoU

Clicours.COM

(Intersection over Union). Cette métrique, souvent utilisée dans le domaine de la vision par ordinateur, nous permet de connaître si une localisation est bonne ou mauvaise en calculant le ratio de l'intersection (ou le chevauchement) de la localisation prédite et de la vraie localisation de l'objet sur l'union de ces deux régions.

Pour calculer l'IoU d'une localisation dans un espace une dimension (comme dans notre cas), nous procédons ainsi :

$$IoU = I/U \tag{4.9}$$

où I est le chevauchement de la zone prédite et de la vraie zone

$$I = min(p_{end}, gt_{end}) - max(p_{start}, gt_{start}) \tag{4.10}$$

et où U est l'union de ces deux même zone

$$U = max(p_{end}, gt_{end}) - min(p_{start}, gt_{start}) \tag{4.11}$$

Ici, p est le temps prédit (début et fin) de la tâche d'insertion et gt est le temps réel annoté de la tâche d'insertion. Comme le démontre la figure 4.12, plus le ratio entre le chevauchement et l'union des deux zones est près de 1.0, plus la localisation est bonne.

Les erreurs moyennes de la localisation des tâches d'insertion peuvent être observées dans le tableau 4.2. En plus d'avoir calculé l'IoU sur les résultats obtenus, nous avons calculé les erreurs moyennes sur les prédictions du temps de début du segment et de fin du segment. Les

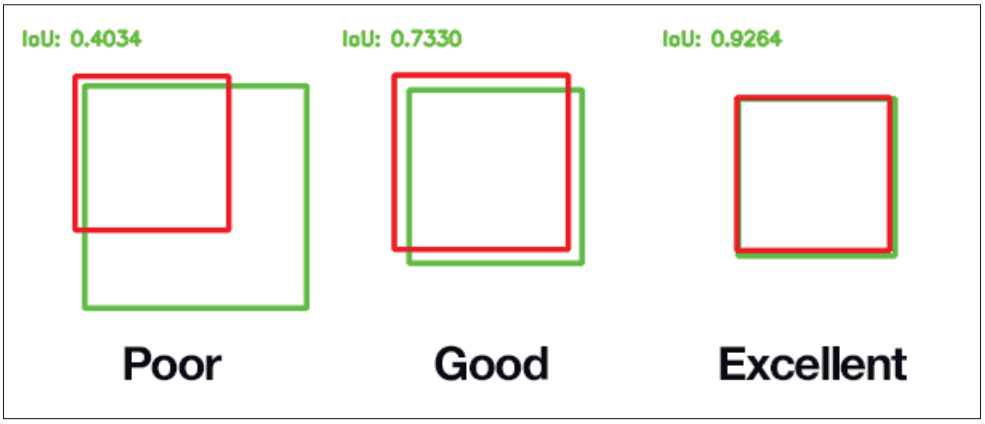


Figure 4.12 Exemple de IoUs. Tirée de https ://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Tableau 4.2 Résultats de la localisation de tâches d'insertion dans des démonstrations complètes.

Erreur moyenne de temps de départ (ms.)	113.77 ±80.19
Erreur moyenne de temps de fin (ms.)	46.86 ±50.2
IoU moyen	0.7304 ±0.1344

résultats obtenus lors des essais montrent une erreur moyenne de 113,77 millisecondes pour la prédiction du début d'une tâche d'insertion et une erreur moyenne de 46.86 millisecondes est observée pour la prédiction du point final du segment. On peut conclure facilement que la prédiction de la fin d'un segment de tâche d'insertion est beaucoup plus précise que la prédiction du début du segment.

Il est intéressant de constater que l'écart-type des erreurs moyennes est, pour les deux cas, très élevé. Prenons par exemple l'erreur moyenne sur le début du segment, l'écart-type ici est de 80.19 ms, ce qui nous indique qu'une grande majorité des résultats obtenus se situe entre [33.58,193.96] millisecondes. L'écart-type est donc très grand par rapport à la moyenne obtenue. Par contre, ce phénomène peut être expliqué facilement par la façon dont les données ont été récolté. En effet, comme expliqué à la section 4.1.2, étant donné que les démonstrations utilisées pour entraîner le CNN ont été annotées par le démonstrateur, le début et la fin de

l'insertion ont été annoté de manière arbitraire par le démonstrateur et peuvent être différents selon le point de vue de la tâche à faire par l'utilisateur. Il est donc évident que le CNN ne soit pas parfait pour une telle tâche puisque les données sont variables dans le temps.

Regardons maintenant l'IoU moyen obtenu, c'est-à-dire 73.04%. Ce résultat nous indique que le système localise les segments d'insertion avec un taux de succès acceptable. Nous utilisons cette métrique, plutôt que de simplement prendre le taux de chevauchement du segment annoté avec celui prédit, pour être certain que la localisation soit adéquate.

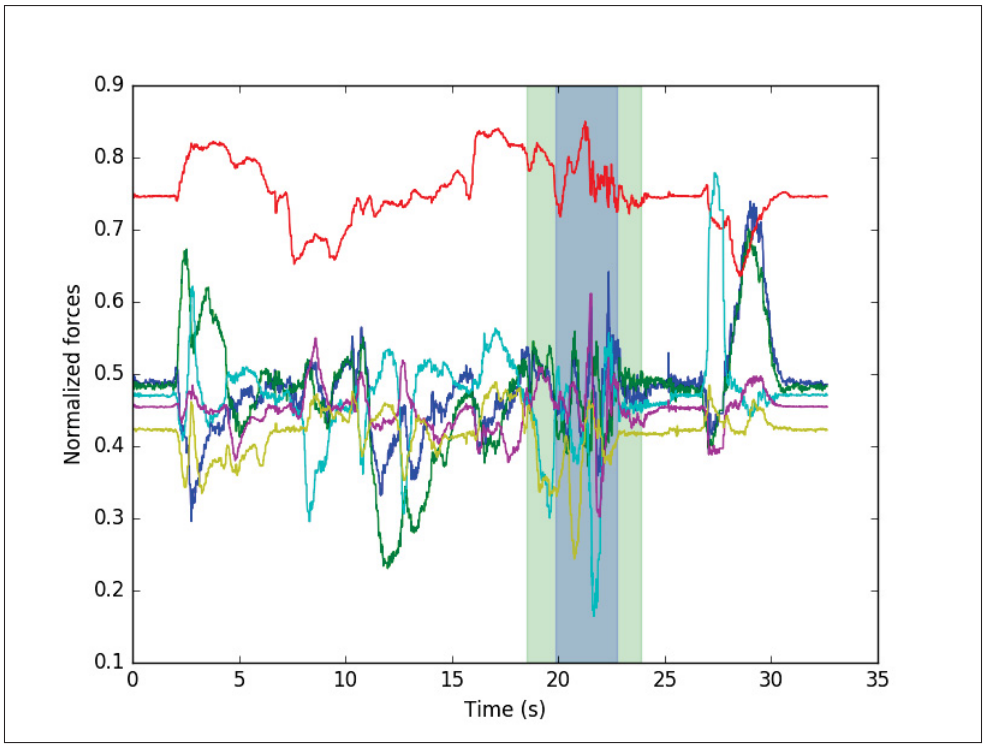


Figure 4.13 Exemple d'une bonne détection d'une tâche d'insertion avec un mauvais IOU. En bleu : les périodes où des insertions ont eu lieu annoté par le démonstrateur. En vert : les périodes prédites par le système où des insertions ont lieu.

Dans la figure 4.13, nous pouvons observer un exemple qui illustre bien l'utilité du IoU. Dans cet exemple, si nous utilisions la métrique démontrée à l'équation 4.6, nous obtiendrions un résultat de 100% puisque le segment réel est complètement couvert par la zone prédit par le système. Malheureusement, nous pouvons observer dans cet exemple une erreur de près de

1.5 secondes pour le début et la fin de segment prédit, ce qui résulte en une prédiction plutôt mauvaise. L'IoU de cet exemple est de 52.9%, ce qui représente beaucoup mieux le résultat actuel obtenu.

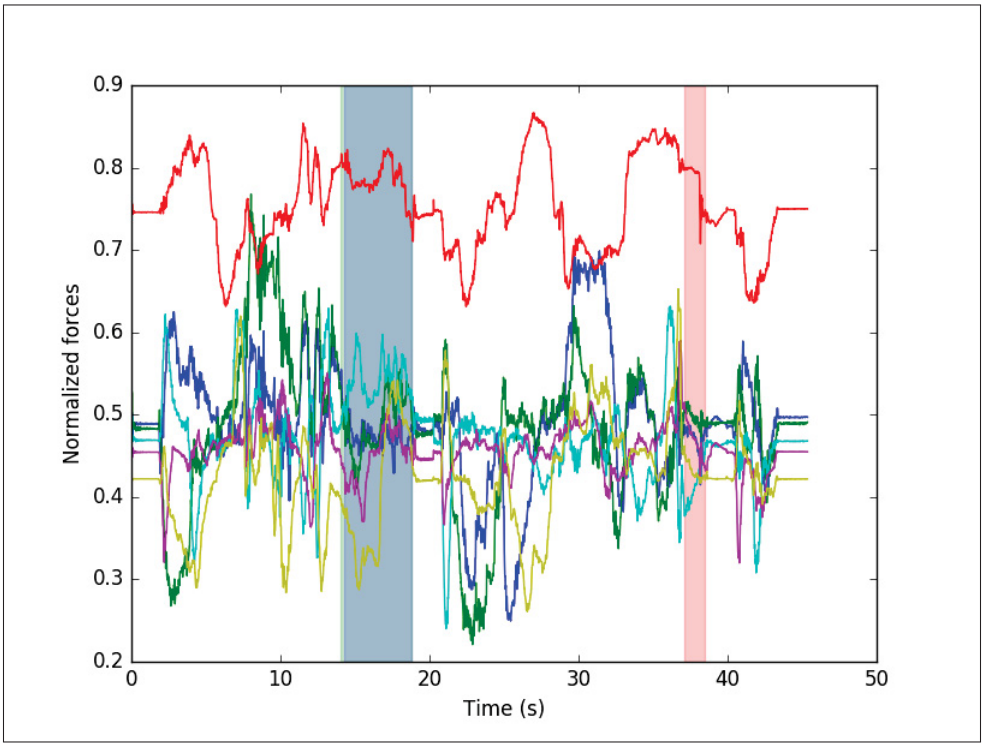


Figure 4.14 Exemple d'une bonne détection d'une tâche d'insertion avec un bon IOU. En bleu : les périodes où des insertions ont eu lieu annoté par le démonstrateur. En vert : les périodes prédites par le système où des insertions ont lieu. En rouge : les périodes où une tâche de PAP ont lieu.

Dans un autre ordre d'idée, la figure 4.14 nous montre un autre exemple où une métrique, telle que celle montrée à l'équation 4.6, donnerait un résultat de 100%. Par contre, contrairement à l'exemple précèdent, la prédiction est quasiment exact (la superposition entre la zone prédite et réelle est pratiquement parfaite) et nous obtenons alors un IoU de 94.15%.

4.2 Segmentation d'une démonstration pour l'optimisation de la séquence

Maintenant que nous sommes capables de déterminer si une tâche d'insertion a eu lieu pendant une démonstration et de la localiser dans le temps, il est temps de segmenter la démonstration

originale pour obtenir une tâche rejouable plus optimisée. Par segmenter, nous voulons dire étiqueter/modifier une certaine section de la démonstration originale dans le but d'utiliser une solution externe pour faire la tâche en question ; dans notre cas, une tâche d'insertion.

Jusqu'à maintenant, nous avons observé les forces appliquées sur le robot pendant la démonstration. Dans la figure 4.15, les forces ainsi que la position du robot pendant la démonstration sont affichées. Nous avons utilisé le système de détection de tâche d'insertion pour trouver une tâche d'insertion adéquatement (entre 30 sec. et 33 sec.).

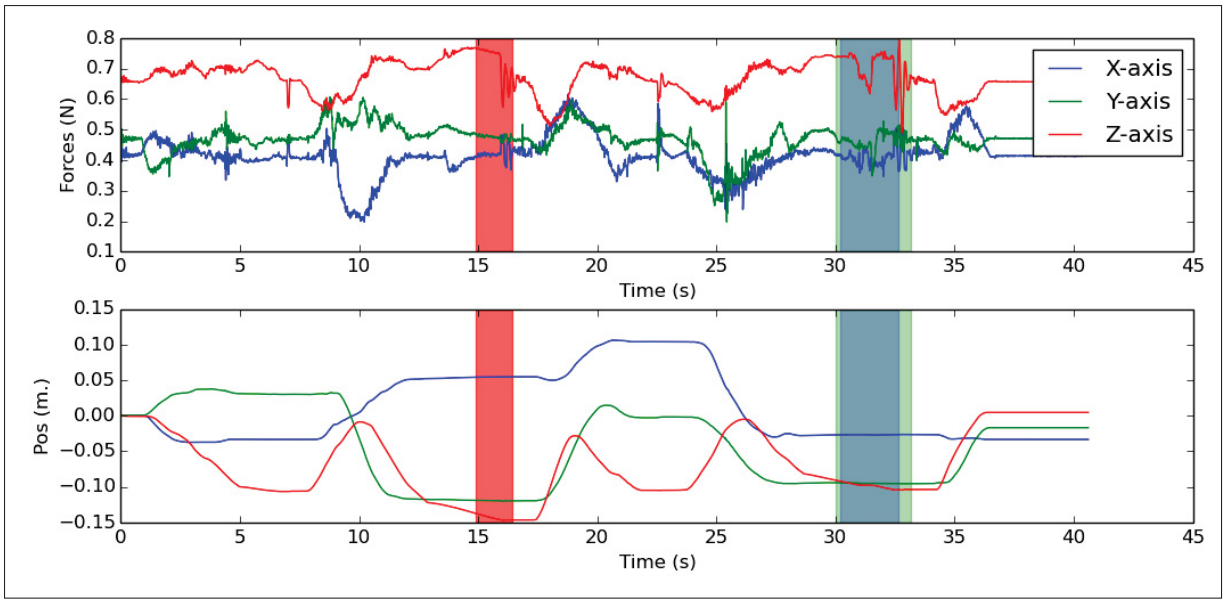


Figure 4.15 Exemple d'une démonstration. En rouge : segment de la démonstration où une tâche de placement d'objet sur une surface est effectuée. En bleu : segment de la démonstration où une tâche d'insertion est effectuée. En vert : segment de la démonstration classifié comme étant une tâche d'insertion par le système.

La figure 4.16 nous expose la trajectoire générée pendant la démonstration dans un espace cartésien. On y retrouve toutes les étapes de la démonstration, voir : une prise d'objet, le dépôt de celui-ci, une autre prise d'objet et finalement l'insertion du dernier.

Le segment de la trajectoire de la figure 4.16 surligné en rouge correspond au segment déterminé justement par notre système comme étant une tâche d'insertion. Bien que nous ayons le temps et l'endroit dans la trajectoire où l'insertion a lieu, cette section représente la section où

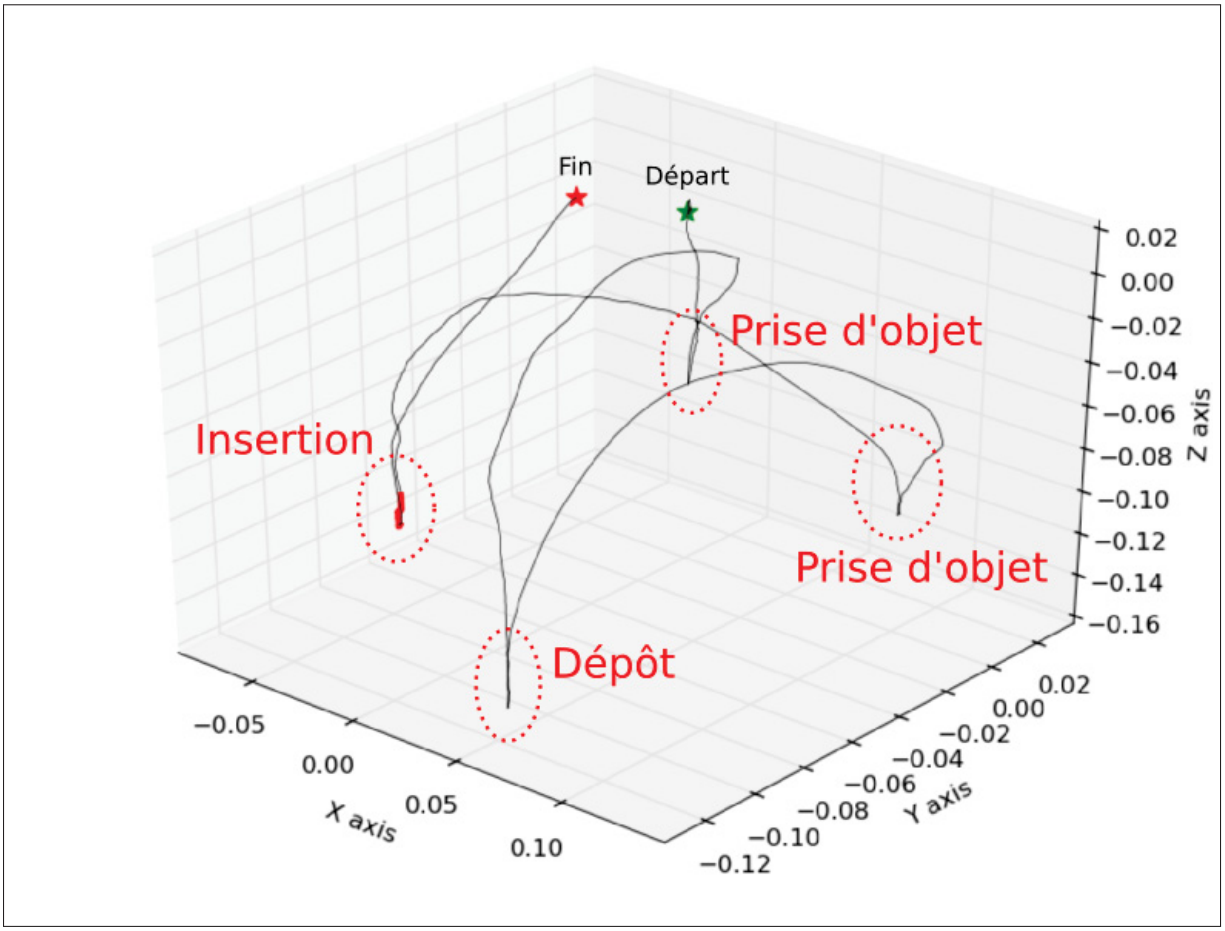


Figure 4.16 Représentation isométrique de la trajectoire démontrée par l'opérateur.

la tâche est effectuée. Puisque nous voulons remplacer cette section avec une technique d'insertion robuste, comme une insertion spirale présentée à la section 5.1, nous nous devons de trouver un point d'approche pour l'insertion. Dans cette section, nous discuterons des différentes méthodes possibles pour déterminer un point d'approche à l'aide de la trajectoire enregistrée pendant la démonstration de la tâche.

4.2.1 Évaluation du point d'approche pour l'insertion

Pour faire l'évaluation de la location du point d'approche de notre nouvelle insertion, nous avons quelques choix à faire. Tout d'abord, nous devons déterminer un point référence pour débuter l'évaluation du point d'approche. Ce point de référence sera utilisé pour déterminer

la position du point d'approche. Par exemple, si nous voulons un point d'approche à 5 cm au-dessus de l'assemblage, nous pourrions prendre le point de contact de la pièce avec l'assemblage (début de l'insertion) comme point de référence et faire une translation dans l'axe Z de 5 cm pour obtenir notre point d'approche.

Deux choix évidents s'offrent à nous quant au choix d'un point de référence, soit : la position de début de la tâche d'insertion ou la position de fin. Bien évidemment, puisque nous ne connaissons pas à l'avance la profondeur de l'insertion démontrée, il serait plus facile d'utiliser le début du segment d'insertion comme point d'approche. Par contre, comme établi avec les résultats de la section 4.1.4.2, la précision obtenue pour la prédiction du début du segment d'insertion par rapport au début réel de l'insertion pendant la démonstration est beaucoup moins précise que la prédiction de la fin de celui-ci. Nous pouvons donc utiliser ce point de référence avec un niveau de confiance plus élevé. En revanche, nous devrons indiquer au système la profondeur de l'insertion pour être certain d'avoir une approche convenable. L'interaction avec l'utilisateur rendra notre système semi-automatique puisque le jugement de l'opérateur sera utilisé pour générer la nouvelle tâche d'insertion.

Dans le reste de cette section, nous utiliserons la fin de la tâche d'insertion comme point de référence pour développer un algorithme d'évaluation d'un point d'approche. Il est à noter que malgré notre choix, il est toujours possible de changer le point de référence sans pour autant changer les algorithmes proposés.

4.2.1.1 Point d'approche basé sur la trajectoire de démonstration

Le premier algorithme que nous proposons ici est basé sur la trajectoire précédant le point final de l'insertion. Cette méthode consiste à choisir une distance minimum adéquate δ pour un point d'approche de l'insertion et traverser itérativement la trajectoire précédant la fin de l'insertion (le point de référence choisi) jusqu'à ce que nous ayons parcouru la distance δ déterminée. Avec cette méthode, notre point d'approche serait forcément sur la trajectoire générée pendant la démonstration.

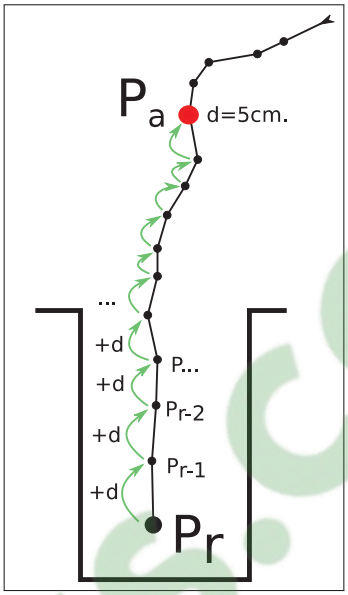


Figure 4.17 On remonte point par point la trajectoire depuis le point de référence pour atteindre notre point d'approche sur la trajectoire originale.

Pour trouver le point d'approche dans la trajectoire T, nous partons du point de référence de l'insertion $p_r \in T(p_1, ..., p_n)$ et calculons la distance euclidienne entre ce point et le point précédent sur la trajectoire. La distance euclidienne dans un espace 3 dimensions se calcule ainsi :

$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2} \tag{4.12}$$

où a et b sont des points dans un espace cartésien 3 dimensions. Nous répétons ce calcul pour chaque point précédent dans la trajectoire T jusqu'à ce que nous ayons parcouru la distance δ déterminée auparavant et nous inscrivons le point final comme étant le point d'approche. L'algorithme 4.1 résume le processus emprunté ici.

Cette méthode simple nous permet d'avoir un point qui est directement sur la trajectoire générée pendant la phase de démonstration et ceci nous fournit un point d'approche adéquat lorsque

clicours.com

Algorithme 4.1 Algorithme de suivi de trajectoire pour atteindre un point d'approche.

Entrées : *La trajectoire $T = p_1, ..., p_r$ précédant le point de référence et où p_r est le point de référence, le seuil de distance ε*
Sortie : *Le point d'approche p_a*
Initialisation de la distance calculé δ et de l'index de déplacement θ
$\delta = 0$
$\theta = 0$
while $\delta < \varepsilon$ **do**
    Calculer la distance entre les prochain points
    $d(p_\theta, p_{\theta-1}) = \sqrt{\sum_{i=1}^{3}(p_{\theta_i} - p_{\theta-1_i})^2}$
    Ajouté distance δ+=d
    Incrémenter θ+=1
end
Retourner le point d'approche
$p_a = p_\theta$

le démonstrateur génère une approche à l'insertion appropriée. Par exemple, si nous reprenons la démonstration analysée à la figure 4.16 et que nous utilisons une distance d'approche de 5 cm, nous obtenons un point d'approche tel qu'affiché à la figure 4.18.

Par contre, si la trajectoire générée pendant la démonstration ne contient pas une approche appropriée, nous pourrions obtenir des résultats plutôt inadaptés. Prenons la trajectoire démontrée à la figure 4.19 comme exemple d'approche incorrecte fait par l'opérateur. Dans ce cas, en utilisant toujours une distance de 5 cm, nous obtenons un point d'approche qui est nettement inadéquat.

En effet, puisque la trajectoire enregistrée ici ne comporte pas une approche verticale vers le point d'insertion, le point d'approche calculé par notre algorithme se retrouve à 5 cm, mais avec un décalage indésiré sur le plan XY. Dans ce cas-ci, il serait difficile pour une technique d'insertion d'être capable d'effectuer l'insertion avec succès puisque le point de départ de l'insertion serait décalé.

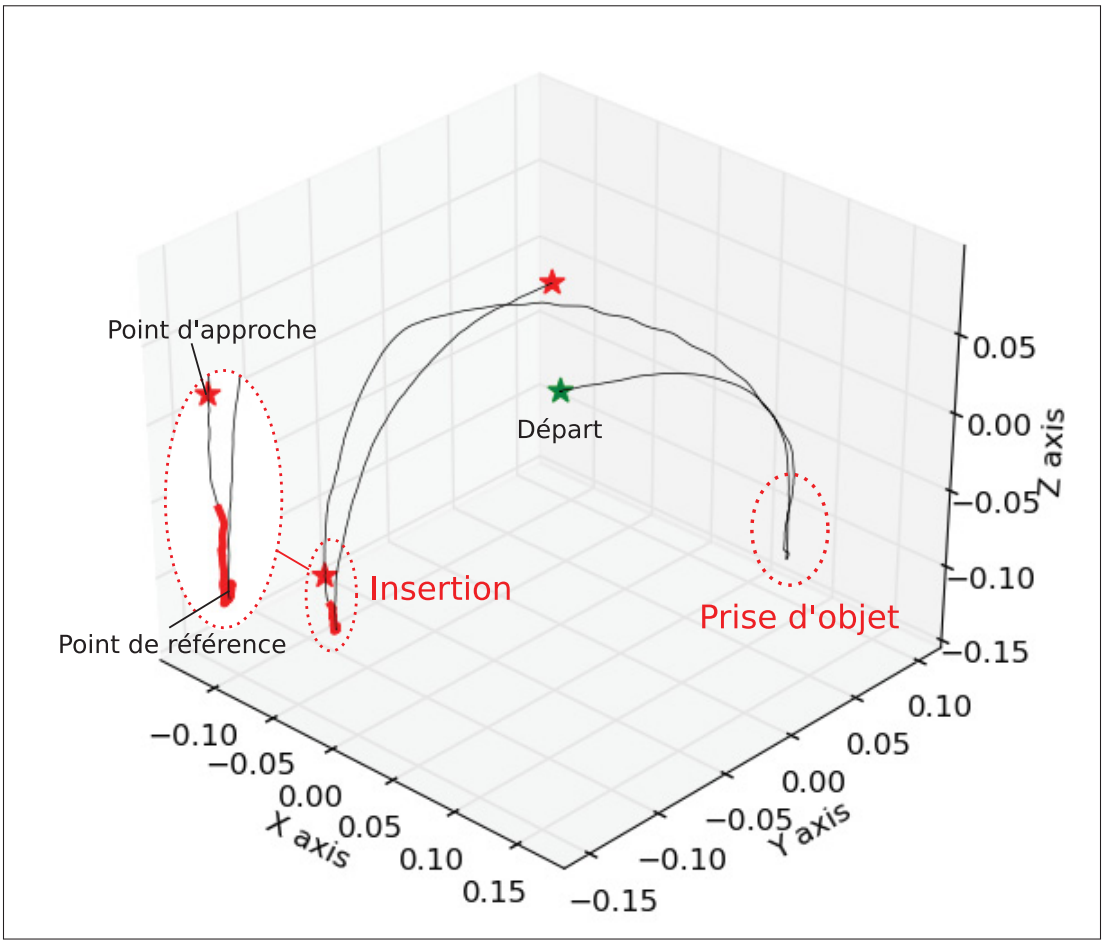


Figure 4.18 Point d'approche obtenu avec l'algorithme présenté.

4.2.1.2 Point d'approche basé sur l'analyse de la composante principale des points de l'insertion

L'approche discutée préalablement fonctionne bien si l'on assume que le démonstrateur effectue une trajectoire adéquate contenant une approche qui est relativement dans l'axe de l'insertion. Cependant, dans le but de faire une solution qui répète intelligemment des trajectoires enseignées, nous ne voulons pas assumer que l'opérateur ait des connaissances préalables en robotique et donc nous ne voulons pas supposer que celui-ci effectuera une approche convenable à chaque démonstration de tâches d'insertion.

Clicours.COM

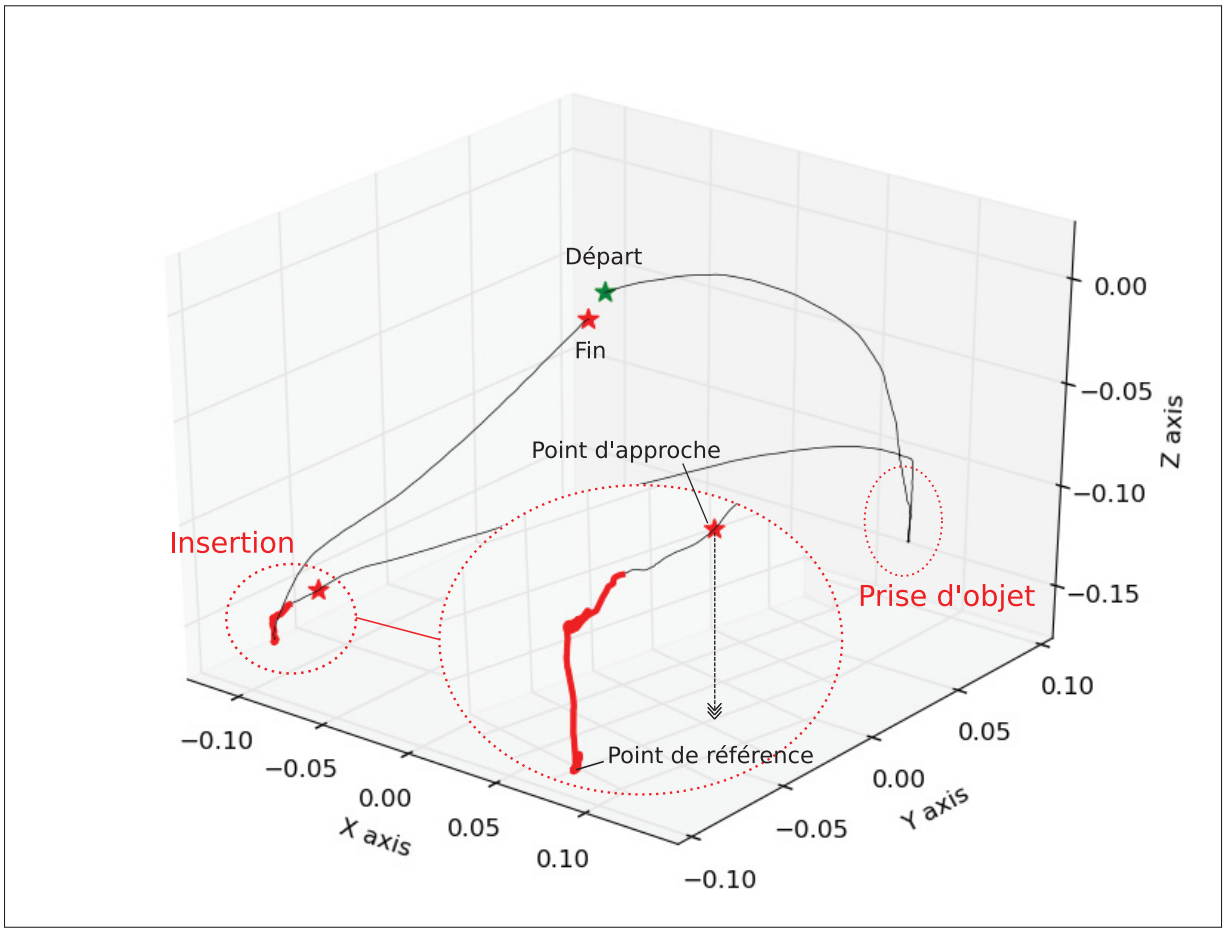


Figure 4.19 Point d'approche erroné obtenu avec l'algorithme présenté.

Dans la section 4.2.1.1, nous obtenions un point d'approche qui était à la distance voulue du point de référence, par contre, dans certain cas, le point d'approche peut être décalé dans les axes orthogonaux à l'axe d'insertion et ainsi créer un problème lors de l'utilisation d'une technique d'insertion. Ce que nous recherchons alors est un point d'insertion qui est directement sur l'axe d'insertion.

Pour calculer l'axe de l'insertion, nous utilisons le principe de composante principale (PCA). Le PCA nous permet d'enlever des dimensions de nos données tout en conservant le plus d'information possible. Nous pouvons remplacer plusieurs dimensions de nos données en créant un nouvel axe qui passera par le centre géométrique de nos données et qui maximisera la variance

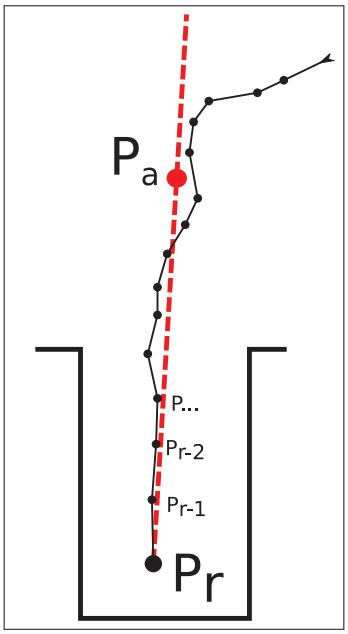


Figure 4.20 Point d'approche calculé à partir d'un PCA fait sur les points de la trajectoire d'insertion.

dans nos données lorsque ceux-ci seront projetés sur ce nouvel axe. Cet axe est connu comme étant l'axe de la première composante principale.

Pour trouver l'axe de la première composante principale, nous n'avons qu'à calculer le vecteur propre (eigenvector) de la matrice de covariance des points constituant le segment d'insertion. La covariance entre deux dimensions peut se calculer ainsi :

$$cov(X,Y) = \frac{\Sigma^{i}(X_i - \mu_X)(Y_i - \mu_Y)}{n-1} \tag{4.13}$$

où X et Y sont deux dimensions contenant les valeurs $x_1,...,x_n$ et $y_i,...,y_n$. Pour un cas où les données auraient plus de deux dimensions, comme dans notre cas, nous pouvons calculer la matrice de covariance ainsi :

$$C(X,Y,Z) = \begin{bmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{bmatrix} \tag{4.14}$$

Pour trouver les composantes principales de cette matrice de covariance, nous devons ensuite calculer les vecteurs propres de celle-ci (eigenvectors). Pour chacun de ces vecteurs propres calculés, dans notre cas 3 vecteurs puisque nous avons des données à 3 dimensions, nous pouvons calculer l'ordre dans lequel ils représentent les composantes principales en calculant les valeurs propres (eigenvalues) de chacun. Le vecteur obtenant la plus grande valeur propre représente alors l'axe le plus significatif ; c'est-à-dire la première composante principale. Le vecteur formant la première composante principale de nos données représente alors la ligne qui épouse le mieux nos données si nous calculions l'erreur du moindre carré entre les données originales et leurs projections sur cet axe.

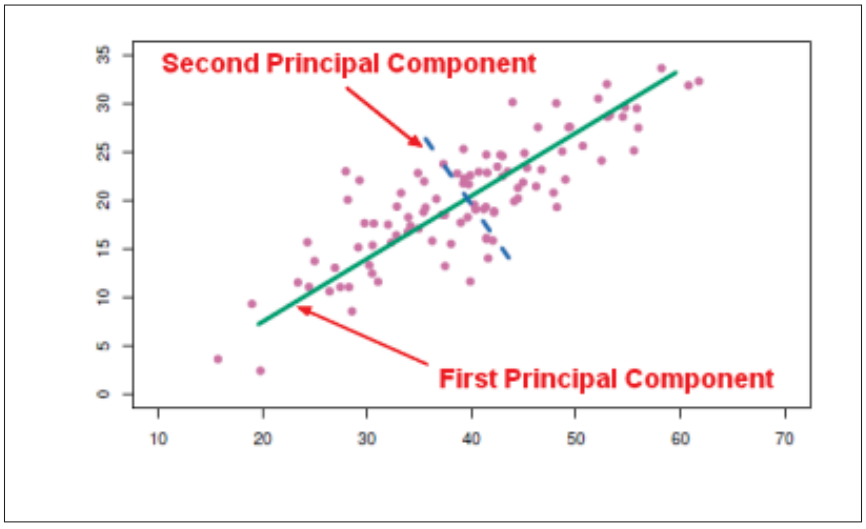


Figure 4.21 Exemple d'axe de composante principale et secondaire d'un ensemble de données. Tirée de https ://gerardnico.com/data_mining/pca

Bien que nous puissions calculer cet axe, nous devons préalablement déterminer les valeurs sur lesquelle nous calculerons cet axe. Nous voulons donc utiliser seulement les points conte-

nus dans l'axe de l'insertion. Pour déterminer ces points, nous partons du point de référence, c'est-à-dire le point étant la fin de la prédiction du segment d'insertion, et nous remontons la trajectoire générée sur une certaine distance θ et gardons ensuite tous les points de la trajectoire jusqu'à la distance déterminée pour calculer la première composante principale. Pour trouver le point à la distance θ, nous utiliserons la même technique que celle expliquée à la section 4.2.1.1.

Ensuite, après avoir calculé le vecteur représentant la première composante principale, nous pouvons trouver le point d'approche de l'insertion sur cet axe. Nous calculons tout d'abord la projection du point de référence sur l'axe déterminé pour ensuite suivre l'axe sur une distance de ε pour trouver notre point d'approche. Pour trouver la projection du point de référence P_r sur un axe AB, nous pouvons simplement projeter le vecteur AP_r et additionner cette projection au point A.

$$P_r^{'}(A,B) = A + (AP \cdot AB)/((AB \cdot AB) \times AB) \tag{4.15}$$

Dans cette équation (4.15), il est important de noter que la division ($/$) est une division par éléments.

Finalement, pour trouver le sens dans lequel nous devons nous déplacer sur l'axe d'une distance de ε pour obtenir notre point d'approche, nous projetons un point P_t contenu dans la trajectoire sur l'axe et calculons le vecteur entre ce nouveau point $P_t^{'}$ et notre point de référence projeté $P_r^{'}$. Le vecteur résultant nous donne la direction dans laquelle nous devons nous diriger pour obtenir notre point d'approche P_a.

La figure 4.22 montre une projection 2 dimensions de l'insertion effectuée précédemment à la figure 4.19. La trajectoire en rouge démontre la section utilisée pour trouver la première composante principale de notre insertion. Dans ce cas-ci, nous avons choisi une distance θ sur notre trajectoire de 2 cm. On peut apercevoir que l'axe généré par la première composante

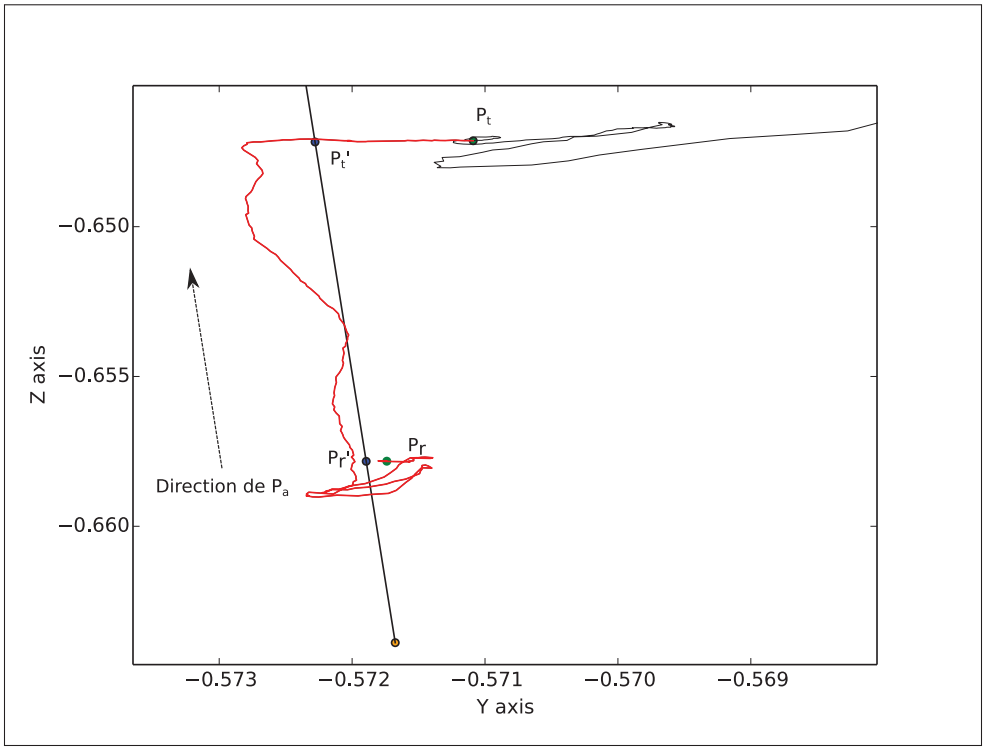


Figure 4.22 Résultat obtenu lors du calcul de l'axe de la composante principale sur les points de la trajectoire de notre démonstration exemple.

principale semble bien donner la direction de l'insertion. De plus, en regardant le vecteur $P_r^{'}P_t^{'}$, on peut aussi déduire que le point d'approche P_a sera dans la direction des Z+.

En utilisant l'axe généré, nous pouvons maintenant générer notre nouveau point d'approche pour notre tâche d'insertion. En gardant la même distance d'approche qu'auparavant (section 4.2.1.1), c'est-à-dire une distance de 5 cm du point de référence $P_r^{'}$, nous n'avons qu'à suivre le vecteur $P_r^{'}P_t^{'}$ sur une distance équivalente à la distance d'approche pour obtenir P_a.

La figure 4.23 représente la même démonstration qu'à la figure 4.19, mais cette fois-ci la méthode utilisant l'axe de la première composante principale est utilisée. Contrairement à la méthode basée sur la trajectoire de démonstration (section 4.2.1.1) où le point d'approche généré était décalé dans les axes orthogonaux à l'insertion, la méthode utilisant la PCA nous permet d'avoir un point d'approche qui est dans l'axe d'insertion malgré la mauvaise démonstration faite par l'utilisateur.

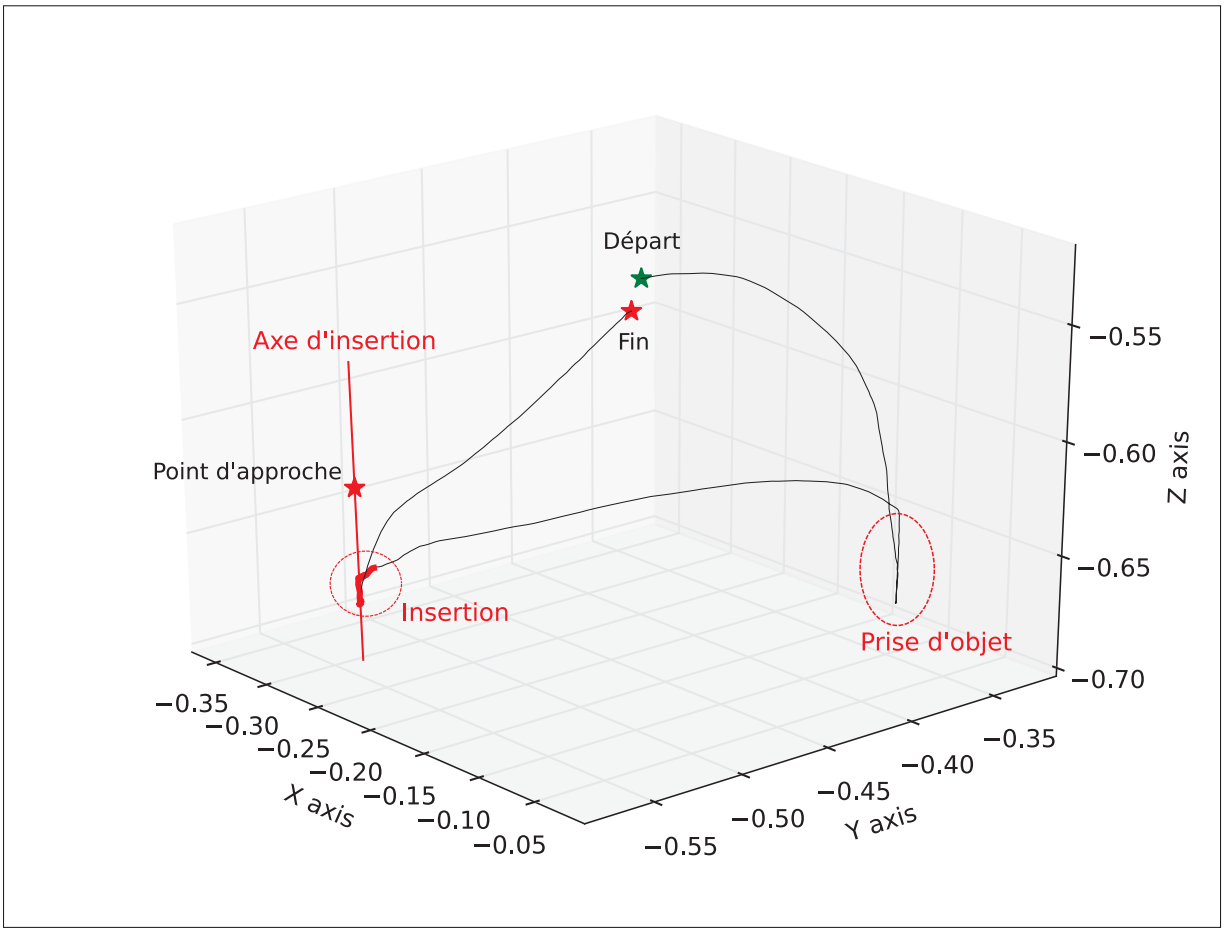


Figure 4.23 Point d'approche obtenu à l'aide de la technique utilisant le PCA sur notre démonstration exemple.

Nous avons alors la position cartésienne du préhenseur pour l'approche de la tâche d'insertion. Par contre, nous n'avons pas déterminé l'orientation de celui-ci à ce point dans l'espace. Évidemment, puisque nous faisons nos expérimentations dans le but d'exécuter des insertions étroites et droites, l'orientation que le préhenseur aura eu pendant la démonstration de l'insertion est nécessairement celle que nous devons utiliser pour refaire la tâche. Nous utiliserons alors l'orientation finale du préhenseur lors de l'insertion comme orientation au point P_a ; l'orientation finale du préhenseur étant l'orientation au point de référence P_r.

4.3 Discussion sur la détection de tâche d'insertion et sur la segmentation complète d'une démonstration

Nous avons pu observer,grâce aux résultats obtenus avec la base de données de test, que notre système nous permet de bien de détecter des tâches d'insertion à l'intérieur de démonstrations complètes. Celui-ci a détecté et segmenté 86 des 91 tâches d'insertion avec succès et nous permet maintenant de développer une solution pour répéter des démonstrations contenant des tâches d'insertion.

Bien évidemment, le système développé contient quelques problèmes que nous avons préalablement soulevé. Un problème que nous pourrions aisément adresser est l'erreur que nous avons obtenu sur le temps de départ moyen de la localisation des tâches d'insertion, vu au tableau 4.2. Comme nous l'avions soulevé auparavant, le temps moyen de départ des segments prédits est beaucoup moins précis que le temps prédit de fin du segment et ce phénomène est dû à la précision de l'annotation du segment d'insertion pendant la prise de données. Effectivement, lors de la construction de la base de données, le démonstrateur devait appuyer sur un bouton lorsqu'une tâche était en cours et devait relâcher ce même bouton lorsque terminé. Par contre, aucune instruction n'a été émise sur la définition de la tâche d'insertion. Bien qu'il soit évident qu'une insertion est terminée lorsque la pièce insérée atteint le fond de l'insertion, il est libre à interprétation de savoir le début d'une tâche d'insertion. Est-ce lors de l'approche, lorsque la pièce à insérer entre en contact avec le récipient, lorsque nous somme à moins de 5 cm de l'insertion ? Est-ce relatif au type d'insertion à effectuer ? Lors d'une prochaine itération de génération de démonstrations pour l'entraînement d'un réseau, il sera nécessaire de définir préalablement et clairement ce qui constitue le début d'une tâche d'insertion pour diminuer la variance dans les données acquises et ainsi possiblement diminuer l'erreur introduite dans l'entraînement du CNN.

Nous croyons aussi que le CNN développé dans ce chapitre permet une évolution facile du système ; c'est-à-dire qu'il serait facile d'introduire de nouveaux types de tâches à détecter et à localiser. En regardant la figure 4.4 représentant notre réseau, nous pouvons observer que nous avons 4 sorties, soit : les niveaux de confiance que l'entrée est une insertion ou non (sorties

Softmax) et deux sorties symbolisant le début et la fin de la tâche d'insertion (si l'entrée est détectée comme une tâche d'insertion). Lors de notre entraînement, nous avons tout d'abord entraîné la partie classification sur toutes les données et ensuite nous avons entraîné la section de localisation sur les données contenant seulement des tâches d'insertion. Dans le même ordre d'idées, il serait facilement possible d'ajouter de nouvelles classes de données, par exemple des tâches de polissage, à notre système.

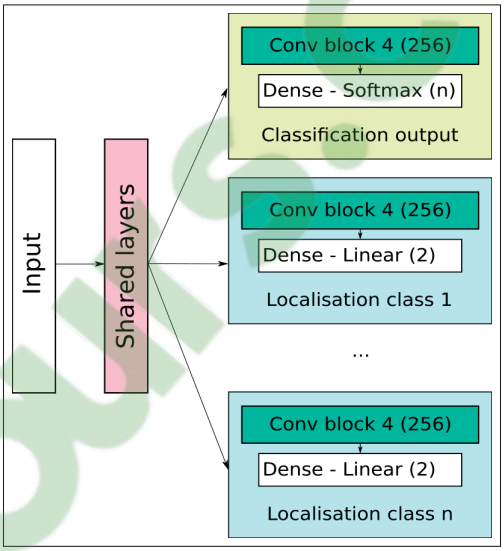


Figure 4.24 L'ajout de nouvelles classes à la sortie de notre CNN peut être facilement fait à l'aide d'un nouveau bloc de convolution en régression et une nouvelle sortie Softmax.

Pour ce faire, nous n'aurions qu'à ajouter le nombre voulu de classe au nombre de sortie en classification et entraîner de nouveaux blocs de localisation pour chacune des nouvelles classes générées. Évidemment, la section partagée du réseau devrait être entraînée à nouveau pour que celle-ci puisse connaître les caractéristiques importantes qui constituent ces nouvelles classes.

Dans ce chapitre, nous avons aussi développé deux méthodes pour générer un point d'approche pour la tâche d'insertion lorsque nous répétons la démonstration, soit une approche basée sur la trajectoire de démonstration et une approche basée sur l'analyse de la composante principale des points de l'insertion. La première nous permet de trouver rapidement un point d'approche qui sera obligatoirement sur la trajectoire de la démonstration, mais qui peut ne pas être dans

l'axe de l'insertion si la démonstration ne contient pas une approche adéquate à l'insertion. La deuxième approche sera obligatoirement dans l'axe d'insertion, mais est plus compliquée à utiliser dans une solution complète pour rejouer la démonstration. En effet, puisque le point d'insertion ne se trouve pas sur la trajectoire générée pendant la démonstration, nous ne pourrons pas simplement rejouer la démonstration pour atteindre le point d'approche. Nous discuterons de solutions possibles pour atteindre le point d'approche ainsi que de la façons d'effectuer la tâche d'insertion au chapitre 5.

CHAPITRE 5

DÉVELOPPEMENT D'UN SYSTÈME INTELLIGENT ET ROBUSTE POUR REJOUER UNE DÉMONSTRATION

5.1 Développement d'une solution robuste pour rejouer une tâche d'insertion

Puisque la cible première de ce travail est d'être capable de faire comprendre à un robot collaboratif que nous voulons exécuter une tâche d'insertion dans un cycle répété, il est primordial de s'attaquer à cette partie critique de la démonstration. Comme nous le savons déjà, ce travail vise à refaire une démonstration de tâche d'insertion dans un environnement qui ne serait pas nécessairement contrôlé. Recréer la tâche d'insertion enseignée devient alors particulièrement difficile compte tenu de plusieurs facteurs : l'incertitude de la position de la pièce dans le préhenseur du robot, l'incertitude de la position de l'assemblage ou même l'incertitude de la tolérance mécanique de l'insertion. Pour plusieurs raisons, la pièce à prendre par le robot ou alors l'endroit de l'insertion peut avoir bougé de quelques millimètres entre la démonstration et la répétition de la démonstration puisque nous visons à travailler dans un environnement qui n'est pas nécessairement fixe. Si un des acteurs de la tâche se déplace de quelques millimètres, il est évident qu'une tâche d'insertion qui requiert une bonne précision ne pourra pas être effectuée avec succès en répétant exactement la démonstration originale.

Heureusement, le problème de l'insertion est connu en robotique et plusieurs pistes de solution ont été développées depuis des années. Certaines recherches, telles que celles faites par Broenink & Tiernego (1996) ou encore par Kim *et al.* (1999), proposent d'approcher la tâche d'insertion avec la pièce dans une certaine inclinaison pour localiser le centre de l'insertion. D'autres recherches, telles que celles publiés par Jasim *et al.* (2014), proposent d'utiliser une méthode en spirale pour localiser la position du trou. Nous avons développé notre solution à partir de cette dernière technique.

La méthode spirale utilisée ici nous permet de faire la recherche d'un trou en balayant une surface de travail d'une manière itérative. Le robot recherche l'assemblage dans un certain

Clicours.com

rayon de la surface de travail et agrandit le rayon de recherche avec le temps si le trou n'est pas détecté. Cette méthode typique d'insertion fonctionne avec succès lorsque la location générale du trou d'insertion est connue. En sachant que la tâche d'insertion est dans un certain intervalle de distance, nous pouvons faire une recherche et connaître la position spécifique.

Dans le but de simplifier la tâche et puisque toutes les insertions effectuées avec le robot depuis le début de nos expérimentations se sont effectuées dans l'axe des Z du robot, nous assumerons ici que pour réaliser les insertions de notre démonstration, nous devons déplacer le robot dans l'axe des Z. Par contre, il est évident que nous pourrions générer une trajectoire similaire dans n'importe quel plan cartésien pour faire une insertion qui ne serait pas orthogonale au cadre de travail du robot. Nous n'aurions ici qu'à transformer les déplacements dans ce nouveau plan pour obtenir un résultat similaire.

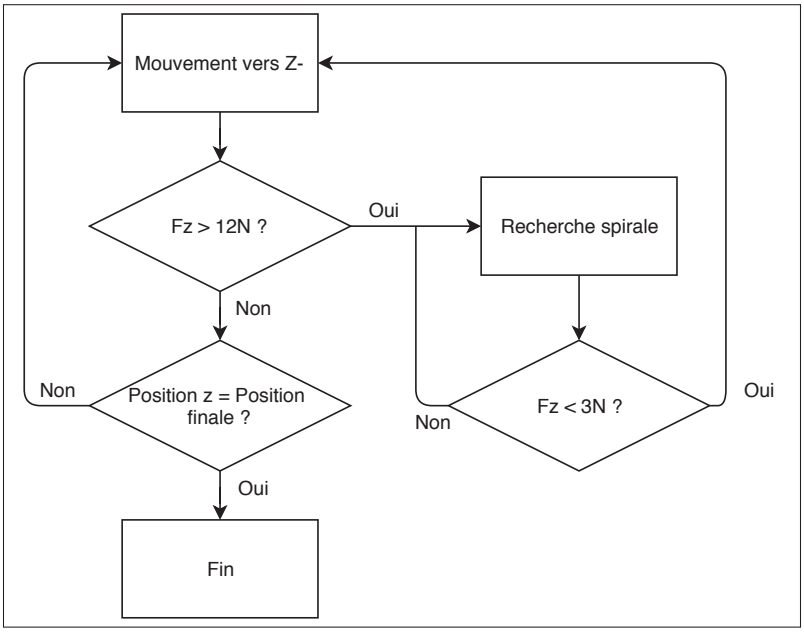


Figure 5.1 Algorithme de recherche spirale utilisé ici.

L'algorithme d'insertion utilisé dans ce travail peut se résumer comme suit :

1. Diriger la pièce vers les Z- jusqu'à ce qu'une force opposée de 12N soit lue par le capteur de force au poignet du robot.

2. Exécuter une recherche spirale jusqu'à ce que la force opposée diminue en dessous de 3 N (nous avons donc trouvé le trou).
3. Répéter les étapes 1 et 2 jusqu'à ce que le robot se retrouve à la même position dans l'axe Z que le point final de l'insertion démontrée.

Nous utilisons ici une méthode qui boucle entre recherche spirale et insertion puisque certaines tâches d'insertion comportent des formes irrégulières. Par exemple, une insertion ayant une marche à l'entrée pourrait causer problème si nous essayons d'aller directement au point final lorsque la force normale mesurée pendant la recherche spirale diminue. En ne forçant pas le robot à aller à la position finale de l'insertion, mais bien à simplement continuer à descendre, nous pouvons faire une deuxième recherche spirale lorsque la pièce rencontre la marche de l'insertion, comme illustré à la figure 5.2.

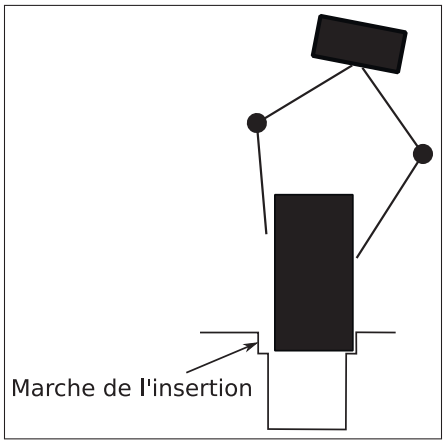


Figure 5.2 Cas où il est nécessaire de continuer à faire une recherche spirale. Plusieurs insertions peuvent contenir une marche ou un chanfrein.

5.2 Développement d'une solution de suivi de trajectoire pour la rejouabilité d'une démonstration

Dans le but de répéter la démonstration d'une manière optimale, en plus d'adresser le problème de la tâche d'insertion, nous devons aborder la question du suivi de trajectoires générées pendant la démonstration. Ici, plusieurs possibilités sont envisageables ; nous pourrions répéter

exactement la trajectoire apprise ou bien générer une nouvelle trajectoire complète dans le but d'optimiser le temps de la tâche.

Dans cette section, nous discuterons de plusieurs possibilités face à la trajectoire à emprunter pour rejouer la démonstration apprise. Nous expliquerons comment générer un nouveau segment de trajectoire pour atteindre des points d'approche ne faisant pas partie de la démonstration originale et nous démontrons notre approche pour auto-générer une trajectoire complète dans le but d'optimiser le temps d'exécution de la tâche effectuée.

5.2.1 Suivi de trajectoire

Lors de l'enregistrement de la démonstration faite par l'utilisateur, la trajectoire empruntée par celui-ci pour exécuter la tâche est sauvegardée dans le but de refaire cette trajectoire dans une éventuelle routine programmée pour le robot. Lors de l'enregistrement, la position cartésienne du préhenseur du robot ainsi que son orientation cartésienne sont sauvegardées à chaque cycle du robot, c'est-à-dire à chaque fois que le contrôleur du robot fait une boucle complète. Puisque nous utilisons un robot UR5 version CB2, la fréquence de ce cycle est de 125 hz. Nous sauvegardons alors la position du robot à toutes les 0.008 secondes.

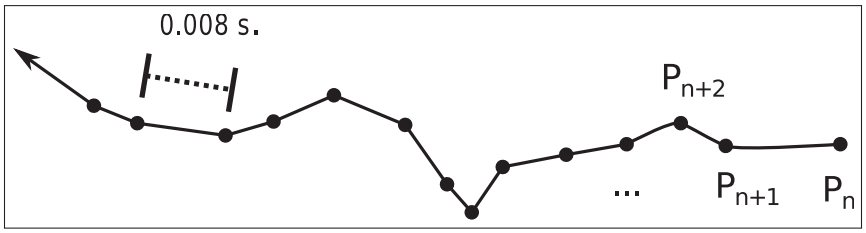


Figure 5.3 Suivi de trajectoire : l'espace entre chacun des points de l'enregistrement est de 0.008 s.

Dans le but de refaire la trajectoire apprise, nous n'avons qu'à donner les points enregistrés au robot pour que celui-ci rejoue la trajectoire enseignée. Il est possible, à l'aide de la commande "servoj" (UR (2015)), de commander un robot UR à se déplacer à un point dans l'espace dans un intervalle de temps fixe. En utilisant cette commande, nous pouvons envoyer chacun

des points enregistrés pendant la démonstration avec un temps de déplacement de 0.008 secondes (temps d'acquisition des points) pour obtenir l'exécution de la trajectoire exactement à la même vitesse qu'enseigné. Évidemment, il est possible de ralentir le suivi de trajectoire en augmentant le temps de déplacement de la commande servoj. Par exemple, pour obtenir une vitesse deux fois plus lente, nous pouvons donner un temps de 0.016 secondes à la commande servoj.

5.2.2 Modification d'une trajectoire dans le but de générer une approche à une tâche

Comme nous l'avons vu à la section 4.2.1.2, lorsqu'une insertion est détectée, nous pouvons générer un point d'approche qui se situera dans l'axe de l'insertion. Par contre, ce point d'approche ne se trouve généralement pas dans la trajectoire démontrée. Si l'utilisateur souhaite utiliser la trajectoire qu'il a démontré pour l'exécution de la tâche par le robot, il nous faut une méthode pour atteindre ce point d'approche sans avoir à redémontrer la tâche à faire et en modifiant le moins possible la trajectoire d'origine.

Dans le cas ci présent, nous opterons pour faire diverger la trajectoire démontrée pour que celle-ci atteigne le point d'approche, mais pour que l'exécution de la tâche s'exécute d'une manière naturelle, nous devons trouver une manière de générer une trajectoire qui fera une transition harmonieuse entre le point d'approche et la trajectoire originale.

Plusieurs solutions sont possibles pour faire ce travail. Par exemple, nous pouvons utiliser une des multiples sortes d'interpolation de type "spline" pour générer une trajectoire lisse entre deux points discontinus. Dans le cas ci-présent, nous avons utilisé la technique de spline Catmull-Rom, une technique populaire dans le domaine de l'infographie (Catmull & Rom (1974)).

Cette technique nous permet de générer une courbe entre deux points P_n, P_{n+1} à l'aide de quatre points, soient $P_{n-1}, P_n, P_{n+1}, P_{n+2}$. Cette courbe est définie par la relation suivante :

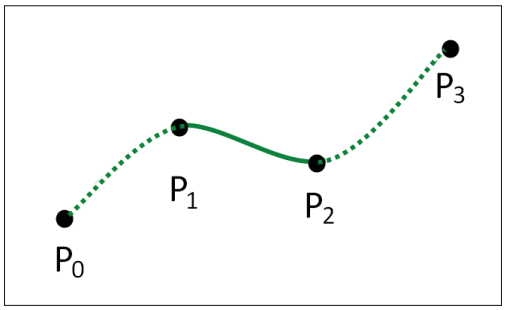


Figure 5.4 Exemple de spline Catmull-Rom utilisant 4 points pour générer une trajectoire.

$$C = \frac{t_2 - t}{t_2 - t_1} B_1 + \frac{t - t_1}{t_2 - t_1} B_2 \tag{5.1}$$

où B_1 et B_2 sont,

$$\begin{aligned} B_1 &= \frac{t_2 - t}{t_2 - t_0} A_1 + \frac{t - t_0}{t_2 - t_0} A_2 \\ B_2 &= \frac{t_3 - t}{t_3 - t_1} A_2 + \frac{t - t_1}{t_3 - t_1} A_3 \\ A_1 &= \frac{t_1 - t}{t_1 - t_0} P_{n-1} + \frac{t - t_0}{t_1 - t_0} P_n \\ A_2 &= \frac{t_2 - t}{t_2 - t_1} P_n + \frac{t - t_1}{t_2 - t_1} P_{n+1} \\ A_3 &= \frac{t_3 - t}{t_3 - t_2} P_{n+1} + \frac{t - t_2}{t_3 - t_2} P_{n+2} \end{aligned} \tag{5.2}$$

et où t_{i+1} est l'addition de la distance euclidienne entre les points t_{i+1} et t_i et la position de t_i.

$$t_{i+1} = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} + t_i \tag{5.3}$$

Dans notre cas, la courbe C générée par cette méthode est la trajectoire entre le point d'approche P_a de notre insertion et un point dans la trajectoire originale. Ce point P_t compris dans la trajectoire originale doit être un point qui se trouve à une distance d du point P_a. Nous déterminerons la position de P_t en accumulant la distance des points de la trajectoire depuis le début de la tâche d'insertion jusqu'à ce que nous atteignions une distance cumulative d, tel qu'expliqué précédemment à l'algorithme 4.1.

En connaissant le point P_n et P_{n+1}, qui sont respectivement P_t et P_a, il ne nous reste qu'à déterminer les points P_{n-1} et $Pn+2$. Le premier sera un point sur la trajectoire précédant P_t. Encore une fois, nous pouvons trouver un point à une distance de P_t en utilisant l'algorithme 4.1. Ici, nous utiliserons une distance de 1 cm pour déterminer notre nouveau point que nous appellerons P_{t-1}.

Pour déterminer la position du dernier point nécessaire P_{n+2}, nous calculerons la distance entre P_a et un autre point se trouvant sur l'axe d'insertion calculé à la section 4.2.1.2. Pour déterminer le point à utiliser, nous utiliserons à nouveau une distance de 1 cm. et nous nommerons ce nouveau point P_{a+1}.

Ensuite, après avoir calculé la courbe C à l'aide des points P_{t-1}, P_t, P_a et P_{a+1}, il est nécessaire de générer des points à l'aide de la courbe pour générer notre trajectoire. Pour que la transition entre la trajectoire originale et la courbe C soit invisible pour l'utilisateur, nous utiliserons la même densité de points sur le nouveau segment que la trajectoire précédant celui-ci. La densité de points d'un segment de trajectoire est la moyenne des distances entre les points composant ce segment. Puisque nous utilisons une vitesse constante entre les points tout au long de l'exécution de la tâche, comme expliqué à la section 5.2.1, la densité des points de la trajectoire est directement liée à la vitesse de déplacement du robot.

Pour calculer la densité λ de la trajectoire de notre nouveau segment C, nous prenons la moyenne des distances euclidiennes des points composant le segment $P_{t-1}Pt$. Finalement, nous générons les points nécessaires sur la courbe C avec une densité λ pour générer le segment de trajectoire liant P_t et P_a et ainsi liant notre démonstration au point d'approche généré.

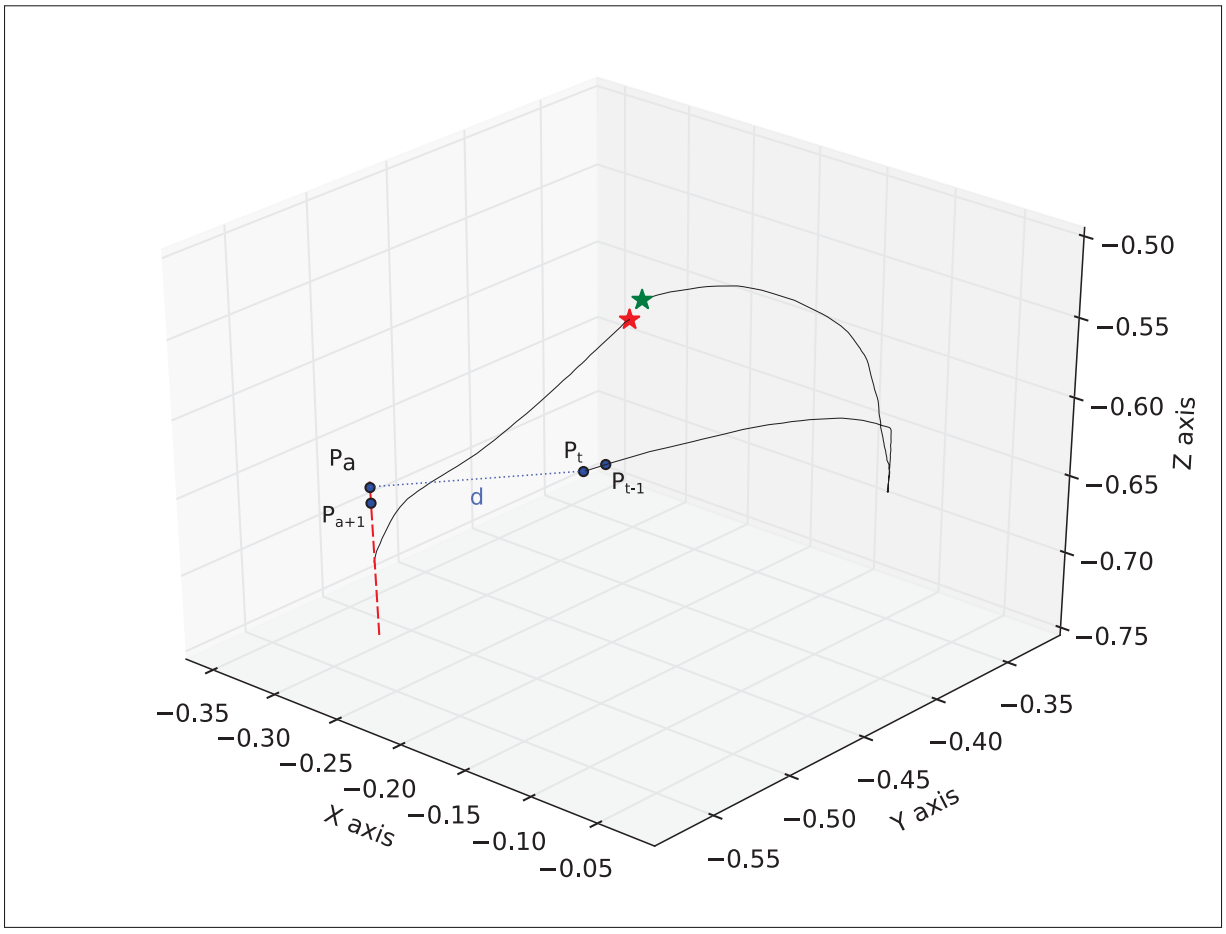


Figure 5.5 Segment de trajectoire que nous devons remplacer par une spline Catmull-Rom. La trajectoire sera entre P_a et P_t et sera définie par les points P_{a+1} et P_{t-1}

Lorsque appliqué sur la démonstration de la figure 5.5, nous obtenons une courbe C qui est affichée à la figure 5.6. On peut observer que la courbe s'ajuste de manière à bien atteindre les points de contrôle de la trajectoire et produit un résultat lisse.

Lorsque la courbe C est ajoutée à la trajectoire originale, nous obtenons alors une solution qui atteint le point d'approche P_a en changeant au minimum la trajectoire originale, comme démontré à la figure 5.6.

Pour atteindre l'orientation du préhenseur voulue au point P_a, nous utilisons une interpolation linéaire entre l'orientation θ_{pt} au point P_t et l'orientation θ_{pa} au point P_a tout au long du déplacement du robot sur la courbe C. Pour ce faire, nous calculons la différence entre l'orientation

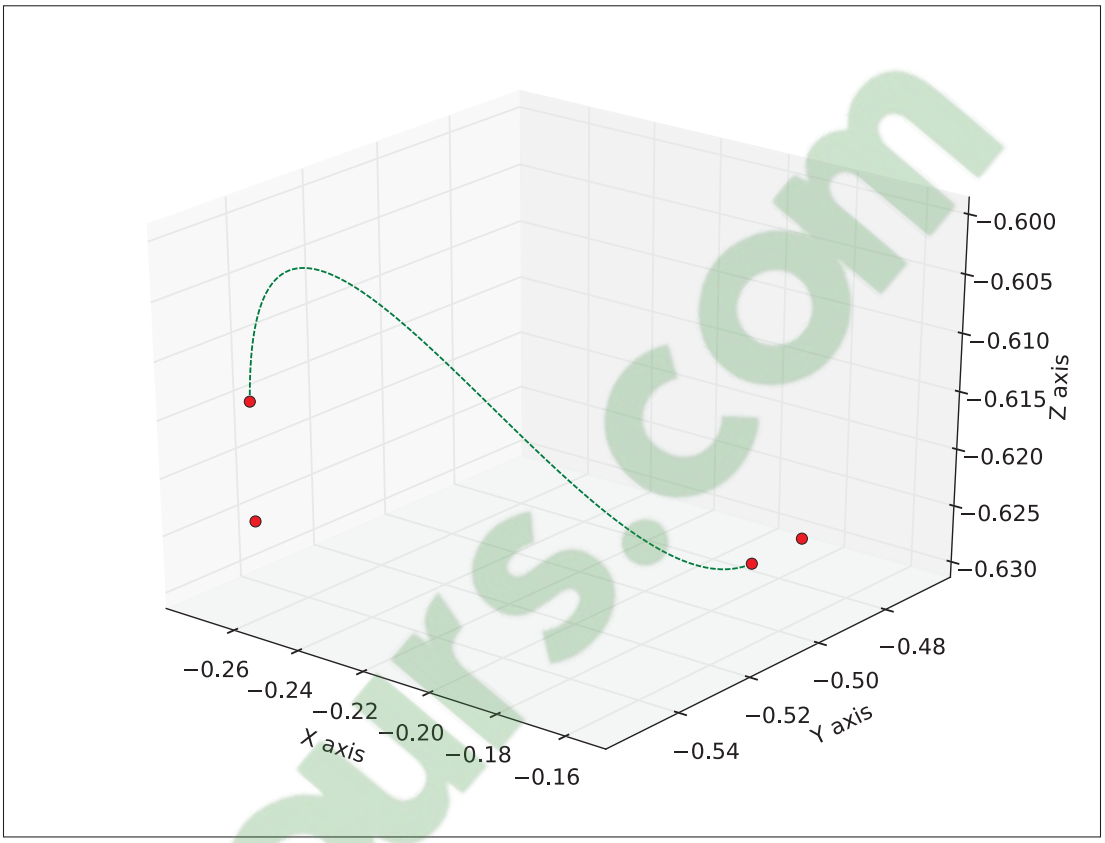


Figure 5.6 Trajectoire obtenue à l'aide des points P_{a+1}, P_a, P_t et P_{t-1} de la démonstration exemple.

du préhenseur au point d'approche et le point P_t et divisons le résultat par le nombre de points générés précédemment pour générer la trajectoire de la courbe C. En sachant le ratio de changement d'orientation pour chaque point de la courbe, nous pouvons déplacer le préhenseur du robot à chaque point de la courbe par ce ratio.

$$\theta_c(a) = \frac{(\theta_{pa} - \theta_{pt}) \times a}{n} + \theta_{pt} \tag{5.4}$$

Ici, $\theta_c(a)$ est l'orientation du préhenseur sur la courbe C au point $a \in \{0, ..., n\}$ et où n est le nombre de points générés pour créer la courbe C.

clicours.com

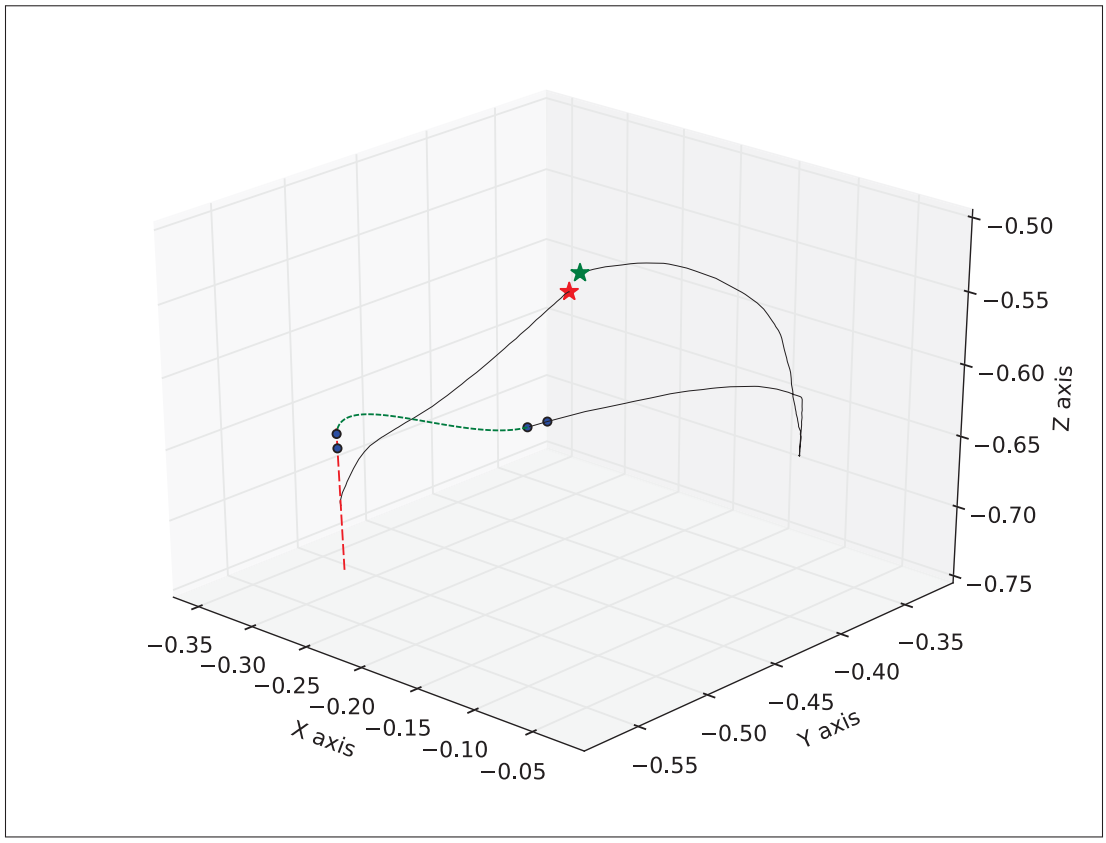


Figure 5.7 Résultat obtenu en introduisant la trajectoire d'approche générée dans la démonstration exemple. En rouge : l'axe d'insertion. En vert : la trajectoire générée pour atteindre le point d'approche de l'insertion.

5.2.3 Optimisation d'une démonstration : création d'un programme basé sur les tâches

Lors de la démonstration de la tâche à faire par l'utilisateur, celui-ci démontre plusieurs tâches telles que : prendre un objet, déposer un objet, faire une insertion, etc. Par contre, puisque le centre d'intérêt de l'utilisateur est d'exécuter des tâches, celui-ci peut négliger les trajectoires générées entre les tâches à effectuer et ainsi faire une démonstration qui ne serait pas optimale en matière de temps. Dans le but d'optimiser le temps global de cycle du robot, nous proposons de faire un nouveau programme à partir de la démonstration originale en ne gardant que les endroits de la démonstration qui concernent des tâches spécifiques. En ne gardant que ces segments, nous pouvons générer des trajectoires plus optimisées entre les tâches pour accélérer le temps de cycle du robot.

Puisque nous nous sommes attardé longuement sur les tâches d'insertions tout au long de ce mémoire, nous ne définirons que quelques tâches pour la construction de notre programme, soient les suivantes.

- **L'insertion d'objets** : cette action est automatiquement détectée par notre système lors de l'étude de la démonstration originale.
- **La prise d'objets** : cette action sera caractérisée par la fermeture du préhenseur pendant la démonstration originale.
- **Le dépôt d'objets** : cette action sera caractérisée par une détection négative d'insertion lors de l'ouverture du préhenseur pendant la démonstration originale.

Étant donné que nous voulons exécuter ces tâches sans utiliser la trajectoire de la démonstration, nous définissons des points d'approche et de retrait qui seront à 10 cm dans l'axe des Z par rapport à la prise ou le dépôt des objets. Pour le reste de ce chapitre, nous assumerons que toutes les tâches effectuées se font dans l'axe des Z du robot pour simplifier le développement de la solution. Il est bien entendu qu'il serait possible de développer une méthode d'automatisation de trajectoire avec des tâches effectuées dans plusieurs orientations dans de futurs développements.

Pour générer une trajectoire automatiquement à partir d'une démonstration, nous regarderons le nombre de tâches à effectuer et générerons des points de trajectoire et des sous-routines pour exécuter notre solution globale. Les étapes à suivre pour générer la solution globale sont les suivantes :

- **Initialisation de la routine** - Afin de connaître toutes les tâches à effectuer lors de la génération de la routine, nous devons tout d'abord lister les tâches effectuées pendant la démonstration originale.
    - Nous utilisons alors notre système de détection d'insertion pour lister le nombre d'insertion, leur temps et leur position.

Clicours.COM

- Nous regardons les fermetures du préhenseur et listons ceux-ci comme des prises d'objet. Nous listons les temps dans la démonstration et leur position.
- Nous regardons les ouvertures du préhenseur et listons celles-ci comme des prises d'objet. Puisque les tâches d'insertion terminent évidemment par l'ouverture du préhenseur, nous vérifions que la position de l'ouverture n'est pas à moins de 1 cm d'une tâche d'insertion pour chaque ouverture ; si c'est le cas, nous ne tenons pas compte de cette ouverture puisque nous considérons que celle-ci fait partie d'une tâche d'insertion déjà listée. Nous listons les temps dans la démonstration et les positions des autres ouvertures.

La liste des tâches est ensuite réorganisée pour contenir les tâches à effectuer dans l'ordre chronologique. Nous générons ensuite les points d'approche P_a de chacune de ces tâches. Si la tâche est une insertion, nous générons un point d'approche en utilisant la technique vue à la section 4.2.1.2. Si un dépôt ou une prise d'objet est la tâche à effectuer, nous générons un point d'approche P_a qui sera à 10 cm au-dessus de la tâche dans l'axe des Z. ($P_a = P_{tache} + [0,0,0.01]$). Finalement, nous initialisons la vitesse v de déplacement du robot voulue lors de l'exécution de la routine. Cette vitesse sera utilisée pour générer la distance entre les points sur les trajectoires de façon à obtenir la vitesse constante, comme expliqué à la section 5.2.1.

- **Départ et exécution de la tâche initiale** - Puisque le point de départ P_d de notre routine ne possède pas de point précédent pour utiliser la technique de spline de Catmull-Rom, nous générons un point P_{d-1} en soustrayant une fraction du vecteur allant de P_{depart} au point d'approche P_a de la première tâche.

  Nous générons ensuite la courbe C entre P_{depart} et P_a en utilisant les points $P_{d-1}, P_{depart}, P_a, P_{tache}$ et générons des points sur cette courbe pour obtenir une vitesse v. Ensuite, si la tâche à effectuer est une insertion, nous n'avons qu'à utiliser la technique d'insertion développée précédemment dans ce document pour atteindre le point final de la tâche. Si la tâche n'est pas une insertion, nous créons une trajectoire linéaire entre P_a et P_{tache} et générons des points sur cette ligne avec une vitesse de $v/3$ puisque nous ne voulons pas approcher la tâche à vitesse rapide. Nous ouvrons/fermons ensuite la pince selon le type de tâche à effectuer et

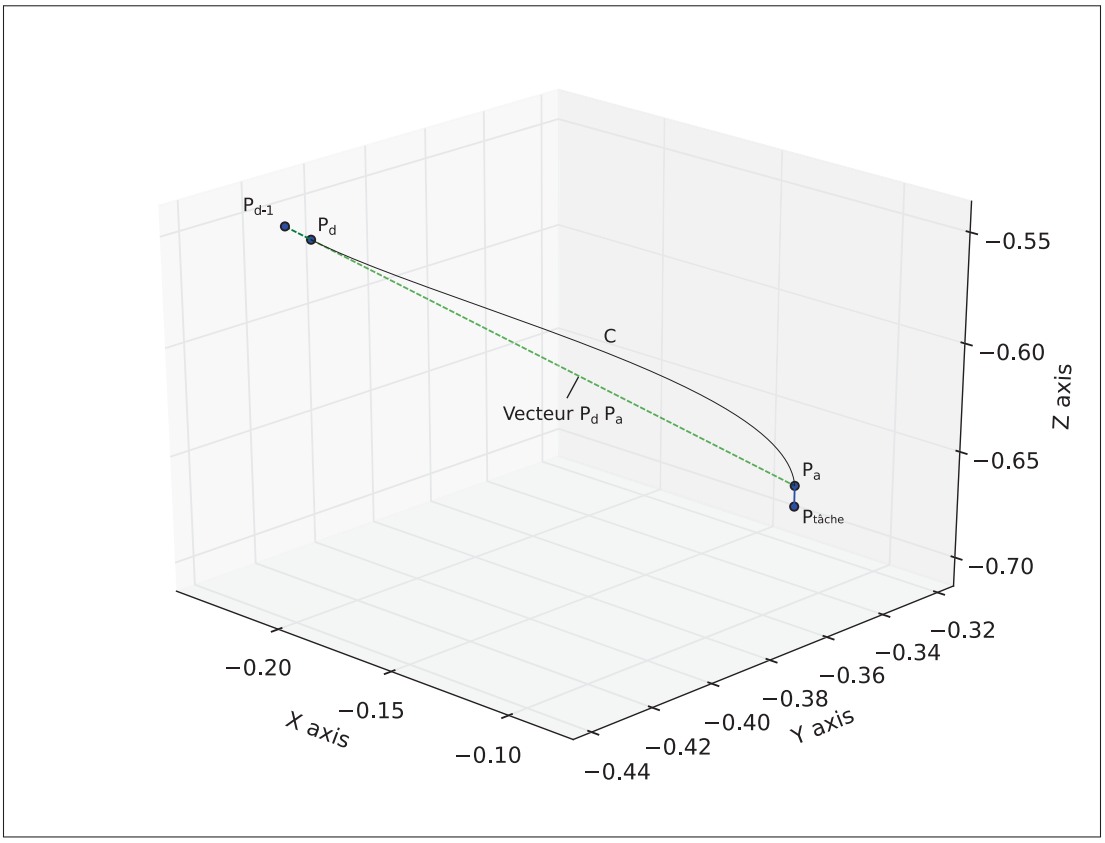


Figure 5.8 Point précédant le point de départ généré à l'aide de l'axe entre le point de départ P_d et le point d'approche de la première tâche P_a

créons une trajectoire linéaire entre P_{tache} et P_a et générons des points sur cette ligne avec une vitesse de $v/3$, puisque nous ne voulons pas nous rétracter de la tâche à vitesse rapide. Nous terminons alors la première tâche au point P_a.

- **Pour chaque tâche restante** - Nous générons la courbe C entre P_{a-1} et P_a en utilisant les points $P_{tache-1}, P_{a-1}, P_a, P_{tache}$ et générons des points sur cette courbe pour obtenir une vitesse v. Ensuite, si la tâche à effectuer est une insertion, nous n'avons qu'à utiliser la technique d'insertion développée précédemment dans ce document pour atteindre le point final de la tâche. Si la tâche n'est pas une insertion, nous créons une trajectoire linéaire entre P_a et P_{tache} et générons des points sur cette ligne avec une vitesse de $v/3$ puisque nous ne voulons pas approcher la tâche à vitesse rapide. Nous ouvrons/fermons ensuite la pince selon le type de tâche à effectuer et créons une trajectoire linéaire entre P_{tache} et P_a, puis

générons des points sur cette ligne avec une vitesse de $v/3$ puisque nous ne voulons pas nous rétracter de la tâche à vitesse rapide. Nous terminons alors la première tâche au point P_a.

- **Terminer la routine au point final** - Puisque le point de départ P_{finale} de notre routine ne possède pas de point suivant pour utiliser la technique de spline de Catmull-Rom, nous générons un point $P_{finale+1}$ en ajoutant une fraction du vecteur allant de P_{a-1} à P_{finale}, de la même manière que pour générer le gradient du point de départ de la routine. Nous générons ensuite la courbe C entre P_{a-1} et P_{finale} en utilisant les points $P_{tache-1}, P_{a-1}, P_{finale}, P_{finale+1}$ et générons des points sur cette courbe pour obtenir une vitesse v.

En utilisant toujours la même démonstration exemple qu'auparavant, nous obtenons les tâches, les points d'approche ainsi que les trajectoires illustrés à la figure 5.9.

Dans cet exemple, deux tâches étaient à effectuer pour compléter la routine et ainsi, trois courbes C_1, C_2, C_3 ont été générées. Le segment vert de la trajectoire représente l'approche/-rétraction de la prise de l'objet et le segment rouge de la trajectoire représente la rétraction du robot après la tâche d'insertion. Il est à noter que la trajectoire de la tâche d'insertion n'est pas affichée, puisque, comme expliqué à la section 5.1, notre méthode de recherche en spirale est une composante active et non statique de notre routine.

Avec cette méthode, nous croyons qu'il serait facile pour un utilisateur de modifier rapidement la routine en ajoutant de nouvelles tâches, en effaçant une tâche existante ou encore même en ajoutant des points de passage obligatoires pour modifier les trajectoires générées. De plus, nous croyons que cette nouvelle routine permettra d'atteindre des temps de cycle plus rapides que ceux que nous aurions utilisant la trajectoire originale ou bien la routine générée précédemment à la section 5.2.2. Nous comparerons les temps d'exécution ainsi que les taux de succès de ces méthodes dans la prochaine section.

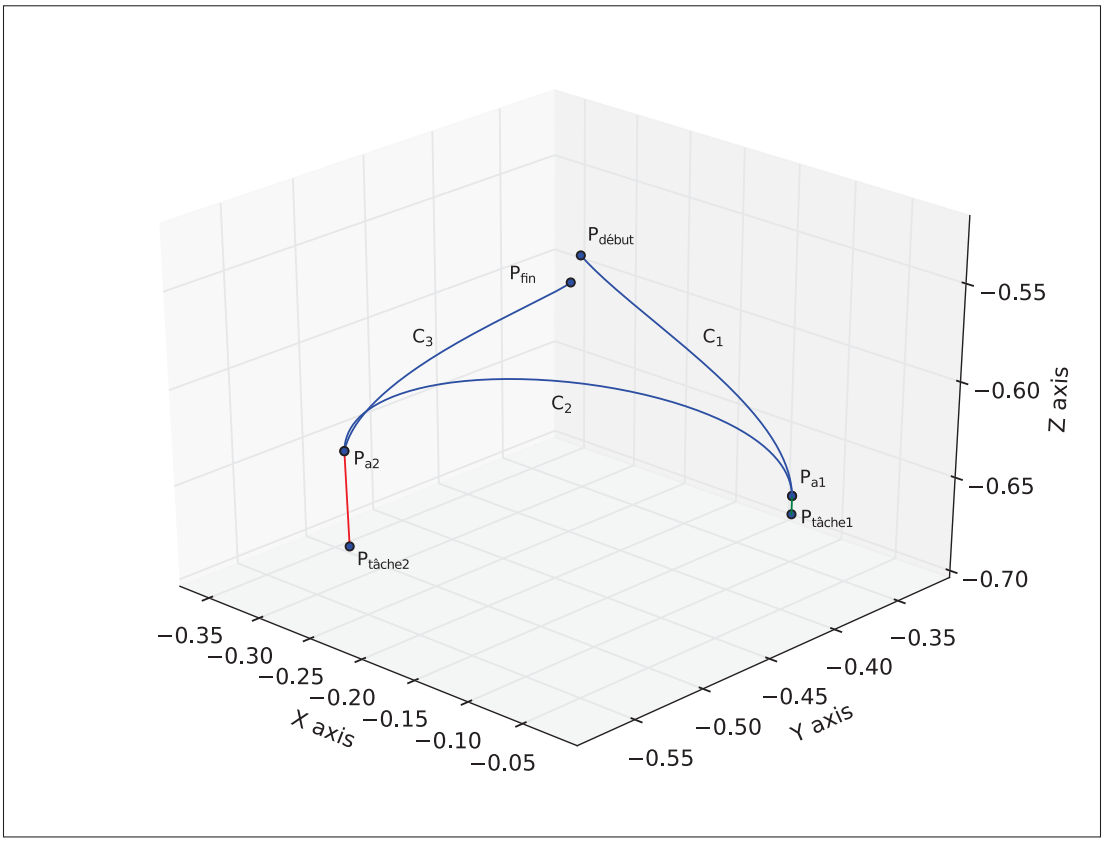
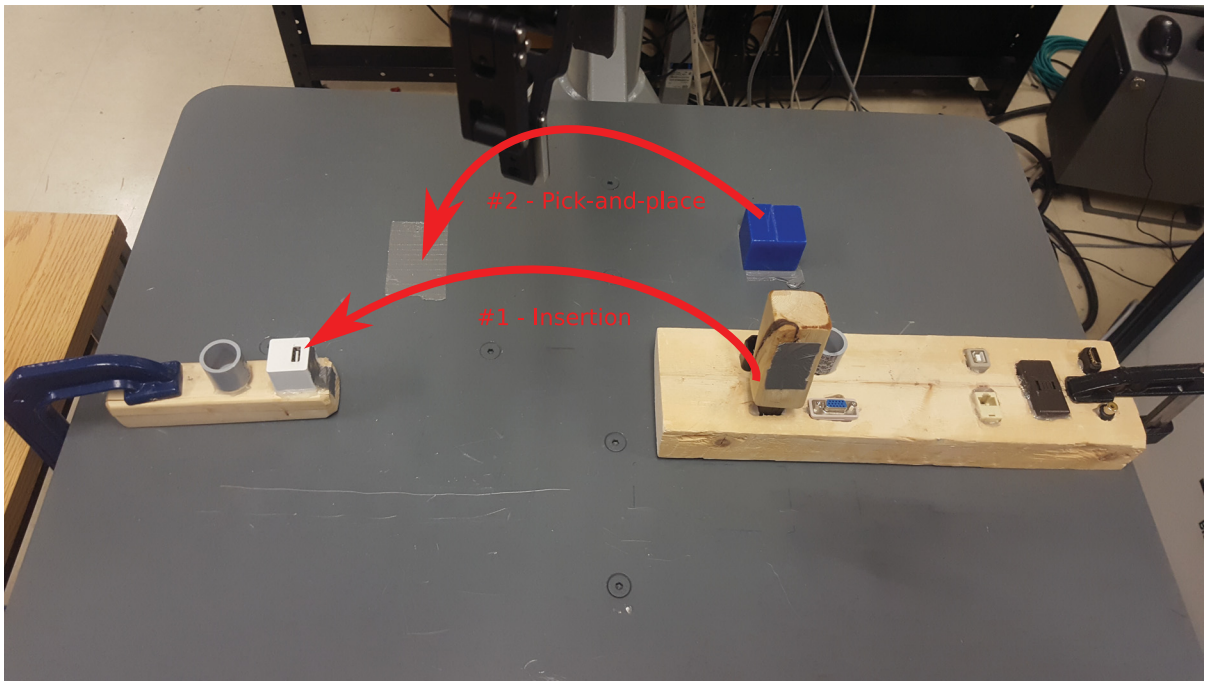


Figure 5.9 Trajectoire complètement générée par la technique vue ici à partir de la démonstration exemple.

5.3 Expérimentations et résultats

Dans le but de vérifier les algorithmes de génération de routines présentées dans la section précédente, nous avons conduit une expérience portant sur la démonstration d'une tâche/trajectoire par un utilisateur. L'expérience a été conduite à l'aide de 5 sujets différents. Il était demandé aux sujets de démontrer deux tâches différentes à l'intérieur d'une même démonstration, soit de prendre une clé USB pour l'insérer dans une prise du même type et prendre un bloc pour le déposer sur un endroit précis de l'espace de travail. L'espace de travail ainsi que les objets utilisés peuvent être observés à la figure 5.10.

Suite à la démonstration effectuée, nous avons généré trois routines dans le but de reproduire la démonstration : une routine reproduisant exactement la démonstration, une routine utilisant



end

Figure 5.10 Démonstration faite par les sujets lors de cette expérience. 1- Prendre la clé USB avec le robot. 2- Effectuer l'insertion de la clé USB dans le socle. 3- Prendre le bloc bleu avec le robot. 4- Déposer le bloc bleu sur la section argentée de la surface de travail.

les trajectoires utilisées avec l'ajout d'une technique d'insertion (section 5.2.2) et finalement une routine utilisant une trajectoire optimisée (section 5.2.3). Nous avons ensuite utilisé ces routines pour rejouer les démonstrations effectuées par les sujets et ainsi comparer les méthodes utilisées selon le temps d'exécution de la tâche et la robustesse de l'exécution. Les résultats obtenus peuvent être observés au tableau 5.1.

En regardant les résultats moyens obtenus, nous pouvons observer plusieurs choses importantes.

- Rejouer la démonstration originale, bien que la routine utilisée contienne la trajectoire exacte générée par l'utilisateur, ne garantit pas le succès de tâche. En effet, puisque l'environnement utilisé par le robot n'est pas entièrement contrôlé, c'est-à-dire que certaines

Tableau 5.1 Temps requis pour effectuer la démonstration pour chaque sujet ainsi que les temps requis pour refaire la démonstration exacte, la démonstration exacte avec solution d'insertion et la démonstration complète à l'aide de notre solution complète.

	Expérience 1 : position initiale					
	Temps démo.	**Temps rejoué**	**Temps rejoué +spirale**	**Temps pour effectuer la spirale**	**Temps rejoué solution complète**	**Temps pour effectuer la spirale**
S1	40.7	38.1	58.7	25.0	45.2	23.8
S2	52.4	**Échec**	1 :14.3	13.9	44.3	20.3
S3	42.3	43.2	43.7	5.2	31.8	11.2
S4	50.2	**Échec**	55.7	16.4	31.5	10.8
S5	56.51	48.2	1 :21.1	35.2	59.0	36.9
Moy.	48.4±6.7	-	62.7±15	19.1±11.4	42.4±11.4	20.6±10.7

pièces ne sont pas exactement dans la position initiale enseignée (exemple : clé USB un peu en angle), la tâche d'insertion risque de ne pas être répétable.

- La solution partielle utilisant la trajectoire modifiée pour ajouter une recherche spirale permet d'assurer l'insertion de la clé USB dans son socle, mais entraîne un temps d'exécution plus lent.
- La solution complète permet non seulement d'assurer le succès de la tâche d'insertion, mais permet aussi d'accélérer le temps d'exécution de la routine originale. Puisque cette solution remplace complètement la trajectoire empruntée par l'utilisateur, celle-ci ne contient pas plusieurs sections de la trajectoire non-optimale générée telle qu'une pause dans le mouvement du robot pendant la démonstration.

Cette expérience nous a permis de tester rapidement le temps d'exécution de chacune des solutions proposées ainsi que leur robustesse face à une tâche d'insertion.

Dans un deuxième temps, nous avons conduit une deuxième expérience pour vérifier l'adaptabilité de nos solutions face à des imprévus. Pour ce faire, nous avons bougé le socle d'insertion de la clé USB de 2 mm dans l'axe X de la base du robot et nous avons réutilisé les routines

générées précédemment sur le nouvel environnement. Les résultats obtenus sont reportés au tableau 5.2.

Tableau 5.2 Temps requis pour refaire la démonstration avec un décalage dans la position de l'insertion.

	Expérience 2 : translation de l'insertion de 2 mm					
	Temps démo.	**Temps rejoué**	**Temps rejoué +spirale**	**Temps pour effectuer la spirale**	**Temps rejoué solution complète**	**Temps pour effectuer la spirale**
S1	40.7	**Échec**	40.4	5.7	40.2	19.0
S2	52.4	**Échec**	1 :33.1	35.4	1 :07.6	44.8
S3	42.3	**Échec**	1 :04.8	24.9	47.9	22.7
S4	50.2	**Échec**	53.27	13.8	34.8	12.4
S5	56.51	**Échec**	54.0	12.4	37.9	15.4
Avg.	48.4 $\pm$6.7	-	61.1 $\pm$20.0	18.4 $\pm$11.7	45.7 $\pm$13.2	22.9 $\pm$12.9

On peut observer rapidement qu'aucune des routines utilisant la trajectoire exacte de la démonstration n'a été capable de réaliser la tâche d'insertion, bien évidemment parce que l'insertion n'était plus au même endroit. Par contre, les solutions utilisant une solution d'insertion ont toutes terminées avec succès en plus d'avoir terminé avec des temps moyens similaires aux résultats obtenus lors de la première expérience.

Nous croyons que les résultats obtenus ici nous indiquent que l'utilisation d'une technique d'insertion lors de la ré-exécution d'une démonstration nous permet d'assurer la réussite de la tâche. De plus, nous croyons que la solution mise de l'avant ici utilisant des trajectoires optimisées semble très prometteuse dans un contexte de production où le temps d'exécution de la tâche est critique. Les démonstrations effectuées dans le cadre de cette expérience peuvent être retrouvées à l'annexe I.

CONCLUSION ET RECOMMANDATIONS

Les outils développés pour la robotique collaborative durant les dernières années ont permis à l'industrie d'automatiser des procédés plus facilement que jamais. La facilité d'installation et d'utilisation de cette technologie a permis à un plus grand public de profiter de la robotique dans tous les milieux. De plus, grâce à l'apprentissage kinesthésique fait à l'aide de contrôle par impédance, la démonstration d'une tâche se fait rapidement et sans nécessiter une connaissance accrue de la robotique comme prérequis. Par contre, comme nous l'avons constaté dans les expériences effectuées dans le cadre de ce projet, il peut être difficile de démontrer certaines tâches requérant une grande précision en position ou encore en force. Nous avons plus précisément parlé de la problématique d'enseigner une tâche d'insertion à l'aide d'une démonstration en position.

Dans l'optique de garder le déploiement d'un robot collaboratif facile, nous avons développé une solution d'enseignement de tâche d'insertion qui ne demande pas à l'utilisateur de programmer la séquence manuellement. En analysant les forces appliquées sur le robot par l'utilisateur et par l'espace de travail du robot pendant la démonstration, notre système basé sur un réseau de neurones convolutifs permet de détecter dans le temps l'endroit où une tâche d'insertion a été effectuée. En traitant cette partie de la démonstration avec une solution de recherche spirale, nous avons prouvé que nous sommes capables de rejouer la démonstration avec un taux de succès plus élevé que la méthode traditionnelle de suivi de trajectoire.

Nous avons mis en place une méthode d'entraînement de CNN en plusieurs étapes nous permettant d'entraîner un réseau à classifier des signaux de force pour la présence de tâches d'insertion pour ensuite être capable d'utiliser les couches entraînées pour créer une deuxième sortie pour localiser précisément la tâche d'insertion dans un segment de signaux de force. Nous croyons que cette méthode d'entraînement nous permet de détecter une tâche d'insertion avec une plus grande précision. De plus, cette technique nous permet d'ajouter facilement au

clicours.com

système la détection d'un autre type de tâche en utilisant les mêmes couches entraînées du réseau. Nous n'aurions qu'à entraîner les dernières couches de sortie du réseau, c'est-à-dire la couche de classification ainsi qu'une nouvelle sortie en régression, pour être capable d'ajouter une nouvelle tâche à localiser.

Nous avons aussi mis en place une technique de contrôle par impédance nous permettant de réduire les instabilités lorsque le robot entre en contact avec des surfaces rigides. En effet, à l'aide de simples vérifications, tel que vérifier le gradient de la force appliquée au préhenseur du robot, nous avons développé une façon de changer dynamiquement l'amortissement du système pour ainsi faciliter les tâches d'insertion pour l'utilisateur.

Bien sûr, certaines questions n'ont pas été répondu au cours de ce projet. Notre solution globale permet de rejouer une démonstration à l'aide des positions enregistrées, mais il n'est toujours pas possible d'utiliser les forces appliquées pendant la démonstration. De plus, nous n'avons utilisé ici qu'une seule technique d'insertion, la recherche spirale. Il est évident que cette solution ne peut pas être utilisée pour tous les types d'insertion possibles.

Pour obtenir un système plus complet et plus proche des réalités de l'industrie, certains module restent à être développés. Par exemple, la détection automatique de solution optimale pour une tâche d'insertion pourrait laisser le choix au système d'utiliser une technique plus appropriée à l'environnement. Il serait aussi bien intéressant de mesurer les performances du CNN présenté ici en ajoutant plusieurs autres tâches à classifier, telles que du polissage ou bien un suivi de trajectoire précis. Même si notre architecture de réseau de neurone semble être adéquate pour la détection de tâche d'insertion, il serait intéressant d'étudier l'utilisation d'architecture plus typiquement utilisée avec des signaux temporels, tels que les LSTMs. Un autre projet futur qui pourrait aider grandement à l'automatisation de l'enseignement de tâches d'insertion serait d'utiliser un système d'apprentissage par renforcement visant à trouver la manière optimale de faire une insertion.

Certes, une des conclusions de ce projet est sans aucun doute l'efficacité impressionnante des réseaux de neurones dans des domaines encore inexplorés.

Clicours.COM

ANNEXE I

EXPÉRIENCES RÉALISÉES LORS DE LA SECTION 5.3

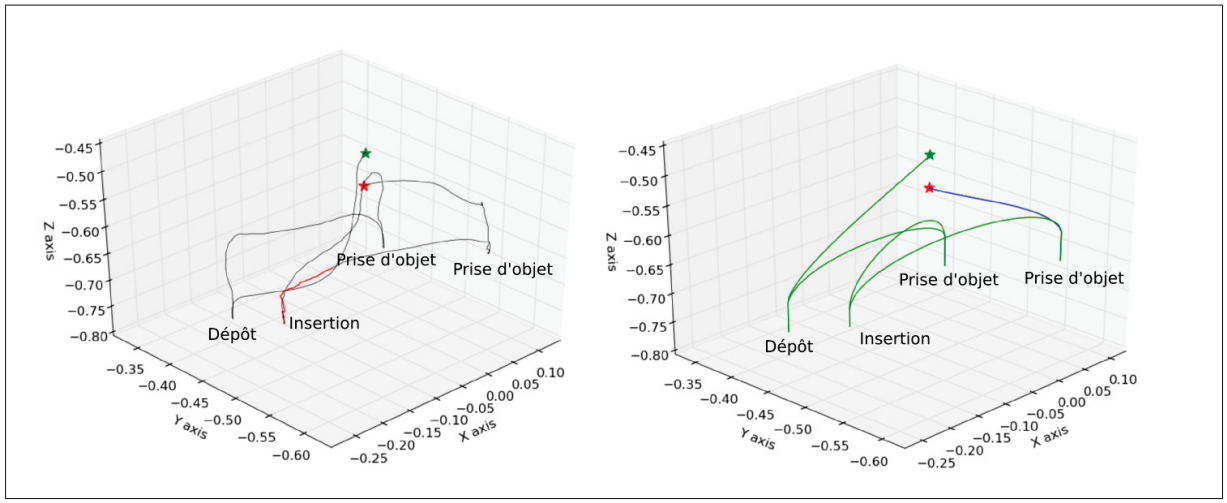


Figure-A I-1 Sujet 1 - Gauche : chemin emprunté pendant la démonstration par le sujet. Droite : trajectoire générée par notre algorithme. Étoile rouge : position de départ de la démonstration. Étoile verte : position finale de la démonstration.

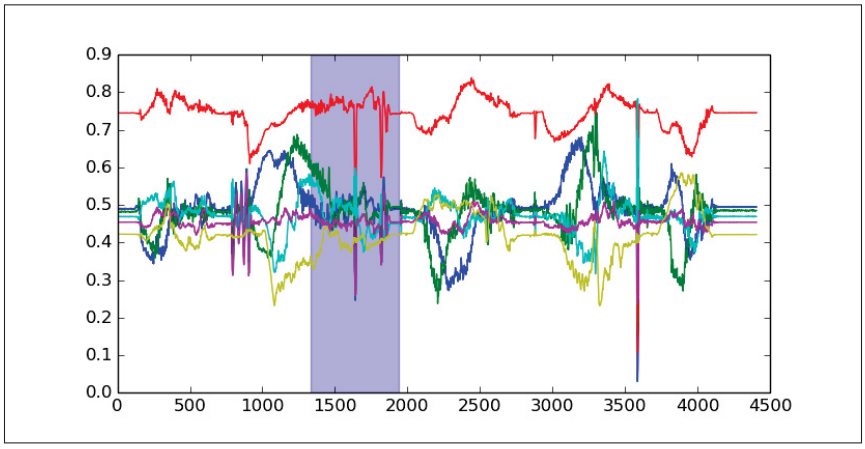


Figure-A I-2 Sujet 1 - Forces normalisées appliquées sur le poignet du robot pour exécuter la démonstration. Zone bleu : zone détectée par notre algorithme comme étant une tâche d'insertion.

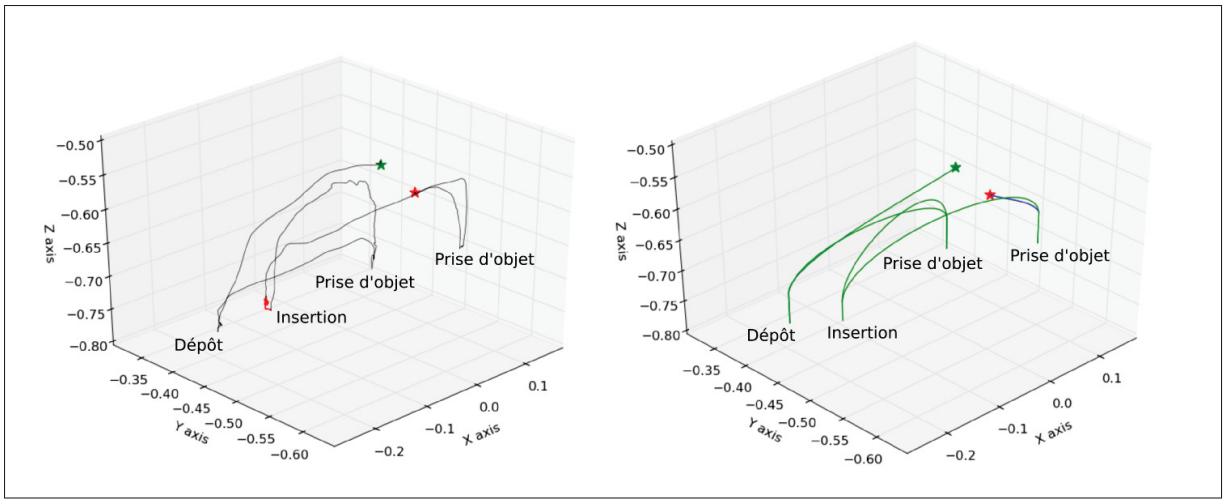


Figure-A I-3 Sujet 2 - Gauche : chemin emprunté pendant la démonstration par le sujet. Droite : trajectoire générée par notre algorithme. Étoile rouge : position de départ de la démonstration. Étoile verte : position finale de la démonstration.

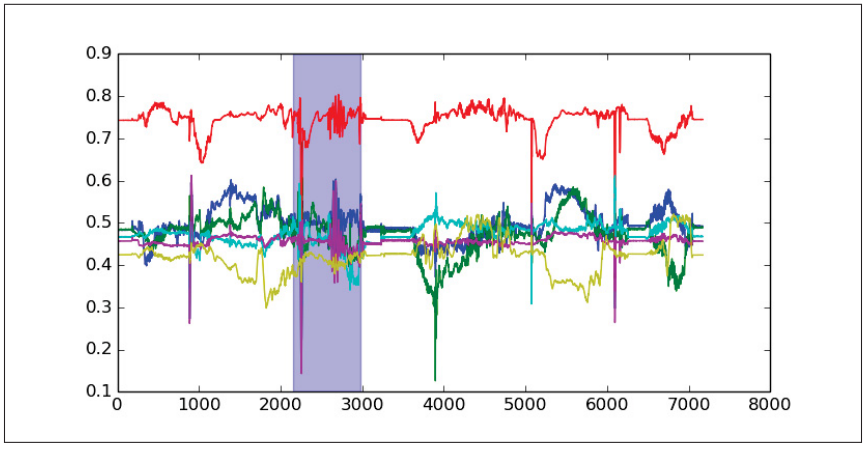


Figure-A I-4 Sujet 2 - Forces normalisées appliquées sur le poignet du robot pour exécuter la démonstration. Zone bleu : zone détectée par notre algorithme comme étant une tâche d'insertion.

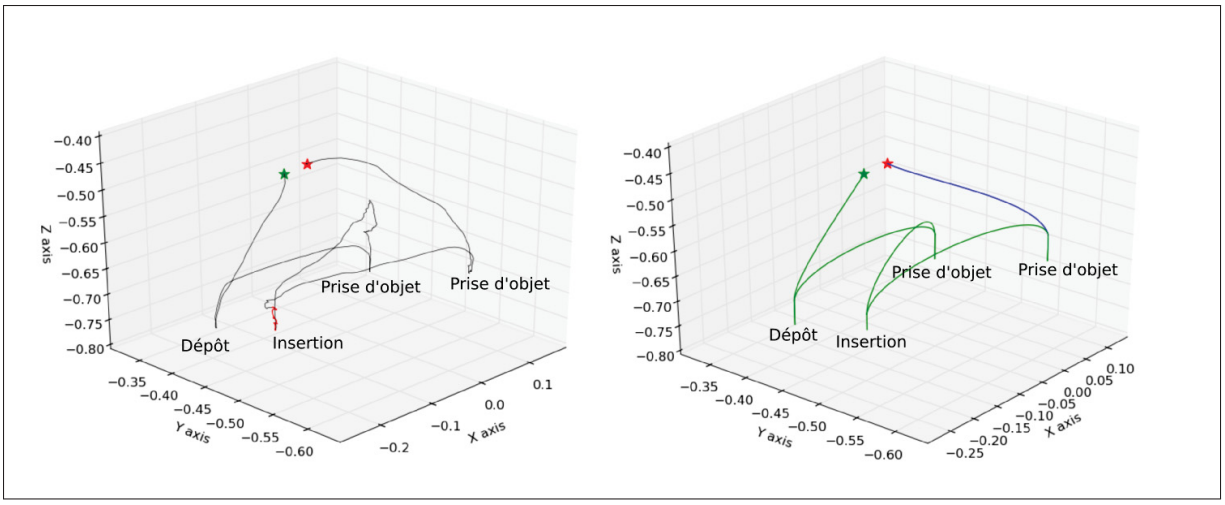


Figure-A I-5 Sujet 3 - Gauche : chemin emprunté pendant la démonstration par le sujet. Droite : trajectoire générée par notre algorithme. Étoile rouge : position de départ de la démonstration. Étoile verte : position finale de la démonstration.

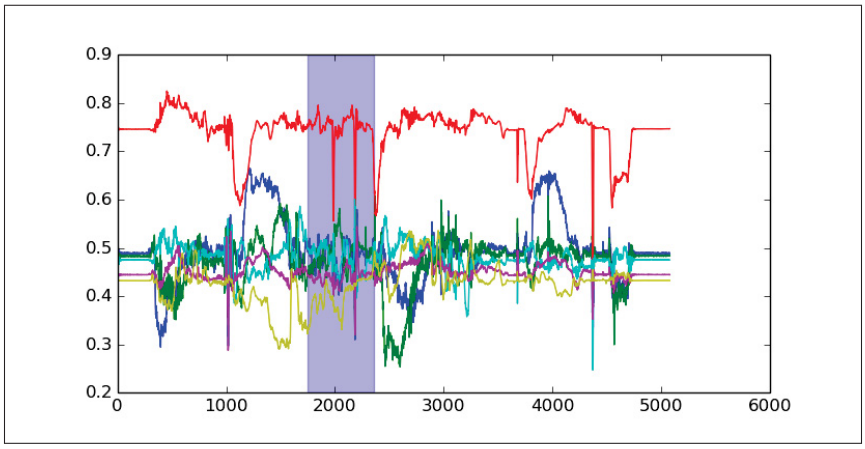


Figure-A I-6 Sujet 3 - Forces normalisées appliquées sur le poignet du robot pour exécuter la démonstration. Zone bleu : zone détectée par notre algorithme comme étant une tâche d'insertion.

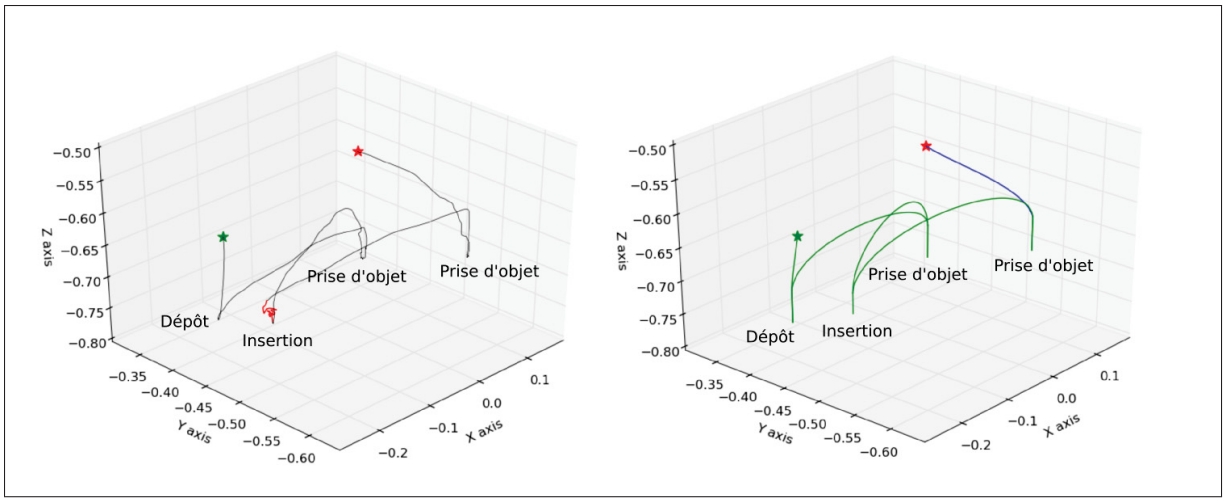


Figure-A I-7 Sujet 4 - Gauche : chemin emprunté pendant la démonstration par le sujet. Droite : trajectoire générée par notre algorithme. Étoile rouge : position de départ de la démonstration. Étoile verte : position finale de la démonstration.

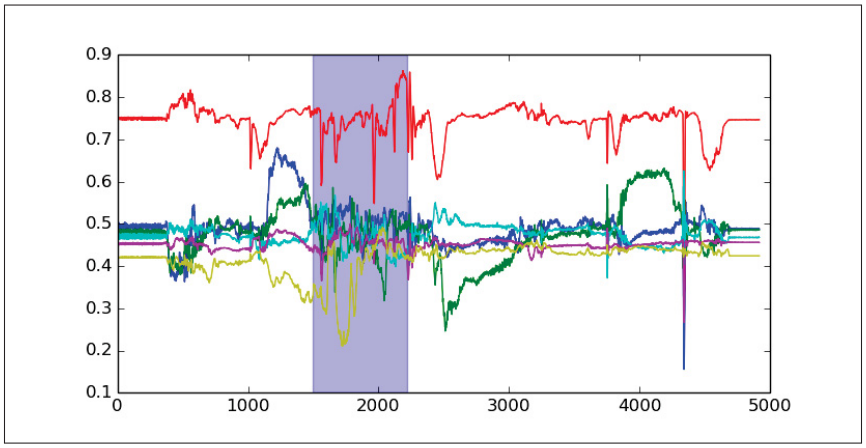


Figure-A I-8 Sujet 4 - Forces normalisées appliquées sur le poignet du robot pour exécuter la démonstration. Zone bleu : zone détectée par notre algorithme comme étant une tâche d'insertion.

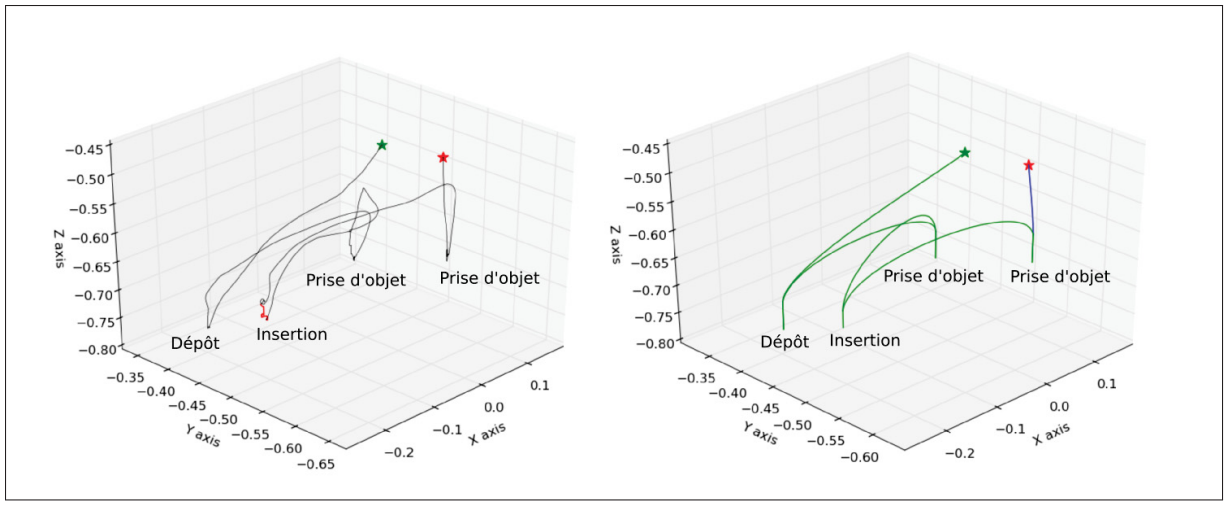


Figure-A I-9 Sujet 5 - Gauche : chemin emprunté pendant la démonstration par le sujet. Droite : trajectoire générée par notre algorithme. Étoile rouge : position de départ de la démonstration. Étoile verte : position finale de la démonstration.

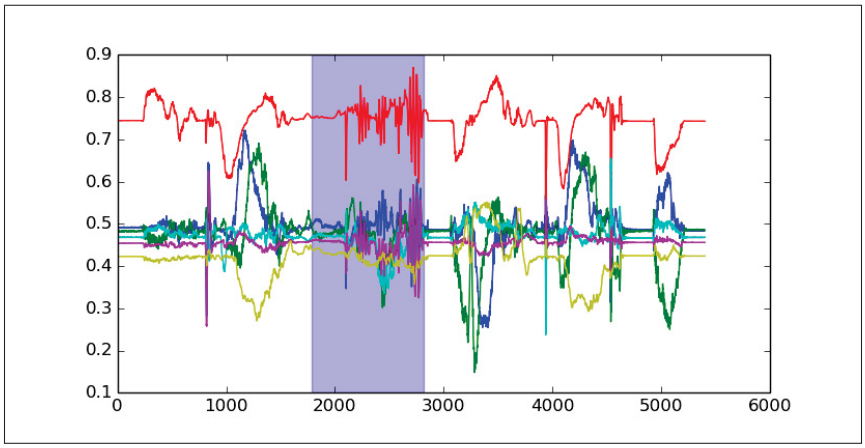


Figure-A I-10 Sujet 5 - Forces normalisées appliquées sur le poignet du robot pour exécuter la démonstration. Zone bleu : zone détectée par notre algorithme comme étant une tâche d'insertion.

BIBLIOGRAPHIE

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G. & Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10), 1533–1545.

Alexe, B., Deselaers, T. & Ferrari, V. (2012). Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11), 2189–2202.

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5), 469–483.

Billard, A., Calinon, S., Dillmann, R. & Schaal, S. (2008). Robot programming by demonstration. Dans *Springer handbook of robotics* (pp. 1371–1394). Springer.

Broenink, J. F. & Tiernego, M. L. (1996). Peg-in-hole assembly using impedance control with a 6 dof robot. *Proceedings of the 8th European Simulation Symposium*, pp. 504–508.

Catmull, E. & Rom, R. (1974). A class of local interpolating splines. Dans *Computer aided geometric design* (pp. 317–326). Elsevier.

Chhatpar, S. R. & Branicky, M. S. (2001). Search strategies for peg-in-hole assemblies with position uncertainty. *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 3, 1465–1470.

De Gregorio, D., Zanella, R., Palli, G., Pirozzi, S. & Melchiorri, C. (2018). Integration of robotic vision and tactile sensing for wire-terminal insertion tasks. *IEEE Transactions on Automation Science and Engineering*, (99), 1–14.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). Imagenet : A large-scale hierarchical image database. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255.

Deterding, S., Sicart, M., Nacke, L., O'Hara, K. & Dixon, D. (2011). Gamification. using game-design elements in non-gaming contexts. *CHI'11 extended abstracts on human factors in computing systems*, pp. 2425–2428.

Doya, K. (2000). Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current opinion in neurobiology*, 10(6), 732–739.

Duchaine, V. & Gosselin, C. M. (2007). General model of human-robot cooperation using a novel velocity based variable impedance control. *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint*, pp. 446–451.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Gosselin, C. M. (2006). Mécanique des Manipulateurs GMC-64388 (pp. 116-117). Université Laval.

Graves, A., Mohamed, A.-r. & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649.

Hamner, B., Koterba, S., Shi, J., Simmons, R. & Singh, S. (2010). An autonomous mobile manipulator for assembly tasks. *Autonomous Robots*, 28(1), 131.

Hersch, M., Guenter, F., Calinon, S. & Billard, A. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6), 1463–1467.

Hikosaka, O., Nakamura, K., Sakai, K. & Nakahara, H. (2002). Central mechanisms of motor skill learning. *Current opinion in neurobiology*, 12(2), 217–222.

Hogan, N. (1985). Impedance control : An approach to manipulation : Part II—Implementation. *Journal of dynamic systems, measurement, and control*, 107(1), 8–16.

Ikeura, R., Monden, H. & Inooka, H. (1994). Cooperative motion control of a robot and a human. *Robot and Human Communication, 1994. RO-MAN'94 Nagoya, Proceedings., 3rd IEEE International Workshop on*, pp. 112–117.

Imamizu, H., Miyauchi, S., Tamada, T., Sasaki, Y., Takino, R., PuÈtz, B., Yoshioka, T. & Kawato, M. (2000). Human cerebellar activity reflecting an acquired internal model of a new tool. *Nature*, 403(6766), 192.

Ioffe, S. & Szegedy, C. (2015). Batch normalization : Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv :1502.03167*.

Jasim, I. F., Plapper, P. W. & Voos, H. (2014). Position identification in force-guided robotic peg-in-hole assembly tasks. *Procedia Cirp*, 23, 217–222.

Kim, I.-W., Lim, D.-J. & Kim, K.-I. (1999). Active peg-in-hole of chamferless parts using force/moment sensor. *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, 2, 948–953.

Kingma, D. P. & Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

Kwiatkowski, J., Cockburn, D. & Duchaine, V. (2017). Grasp stability assessment through the fusion of proprioception and tactile signals using convolutional neural networks. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 286–292.

Lee, H., Pham, P., Largman, Y. & Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. *Advances in neural information processing systems*, pp. 1096–1104.

Li, T. L., Chan, A. B. & Chun, A. (2010). Automatic musical pattern feature extraction using convolutional neural network. *Proc. Int. Conf. Data Mining and Applications*.

Myers, D. R., Pritchard, M. J. & Brown, M. D. (2001). Automated programming of an industrial robot through teach-by showing. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 4, 4078–4083.

Nguyen, K., Fookes, C., Ross, A. & Sridharan, S. (2018). Iris recognition with off-the-shelf CNN features : A deep learning perspective. *IEEE Access*, 6, 18848–18855.

Peternel, L., Petrič, T. & Babič, J. (2015). Human-in-the-loop approach for teaching robot assembly tasks using impedance control interface. *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 1497–1502.

Rad, N. M., Bizzego, A., Kia, S. M., Jurman, G., Venuti, P. & Furlanello, C. (2015). Convolutional neural network for stereotypical motor movement detection in autism. *arXiv preprint arXiv :1511.01865*.

Roberge, J.-P., Rispal, S., Wong, T. & Duchaine, V. (2016). Unsupervised feature learning for classifying dynamic tactile events using sparse coding. *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 2675–2681.

Robotiq. (2018). Force copilot. Repéré à https://robotiq.com/products/force-copilot.

Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. Dans *Adaptive motion of animals and machines* (pp. 261–280). Springer.

Scherer, D., Müller, A. & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. Dans *Artificial Neural Networks–ICANN 2010* (pp. 92–101). Springer.

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), 1929–1958.

Tsumugiwa, T., Yokogawa, R. & Hara, K. (2002). Variable impedance control based on estimation of human arm stiffness for human-robot cooperative calligraphic task. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, 1, 644–650.

Clicours.COM

Uijlings, J. R., Van De Sande, K. E., Gevers, T. & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2), 154–171.

UR. (2015). *The URScript Programming Language* [3.1]. Repéré à http://www.sysaxes.com/manuels/scriptmanual_en_3.1.pdf.

Zinn, M., Khatib, O., Roth, B. & Salisbury, J. K. (2004). Playing it safe [human-friendly robots]. *IEEE Robotics & Automation Magazine*, 11(2), 12–21.