CentraleSupélec | université PARIS-SACLAY

PÔLE PROJET ROBOTIQUE
S7 PROJECT

ROBOT ARM INTERACTIVE CONTROL

Submitted by :

Gills-Roberto Belinga

Kamar El Hajj

Léo Dhalluin

Louis Hébrant

Luana Hartmann Franco da Cruz

Tarek Ouchouker

Supervised by :

Maria Makarov

30 janvier 2026

Table des matières

Chapitre 1

Introduction

1.1 Contexte et objectifs

L'évolution de la robotique moderne tend vers une interaction de plus en plus étroite entre l'homme et la machine, dépassant le cadre des robots industriels isolés pour aller vers la "cobotique" (robotique collaborative). Ce projet s'inscrit dans cette dynamique, répondant à un besoin croissant d'assistance technique pour des populations variées, qu'il s'agisse de personnes en perte d'autonomie (personnes âgées, situations de handicap) ou de professionnels (artisans) nécessitant une "troisième main". La finalité de ce projet est d'utiliser le bras robot pour permettre une interaction fluide lors d'un échange d'objet.

Le cœur du problème réside dans la confrontation entre un système robotique déterministe et un environnement opérationnel non structuré, caractérisé par la variabilité de l'opérateur humain. Contrairement à une chaîne de production standardisée, l'interaction homme-robot en milieu ouvert soulève plusieurs défis : l'opérateur n'est pas nécessairement formé à la robotique. Ses mouvements, sa vitesse d'approche et sa manière de présenter l'objet sont hétérogènes et non standardisés. De plus, le système doit être robuste face aux changements de luminosité, à la diversité des formes d'objets et aux arrière-plans changeants.

La problématique est donc la suivante : Comment garantir une interaction fiable, naturelle et sécurisée entre un robot et un opérateur humain non expert dans un environnement non contrôlé ? La solution développée vise à créer de la valeur par sa robustesse et son accessibilité.

Les résultats attendus se mesurent selon les critères suivants : le mouvement du robot doit être perçu comme non dangereux, anticipable et fluide par l'humain (prédictibilité du système). Le robot doit être opérationnel pour une démonstration finale, capable d'interagir avec succès avec l'ensemble des membres du groupe, mini-

misant le taux d'échec. Le système doit fonctionner indépendamment des variations mineures d'éclairage ou de la morphologie de l'utilisateur.

1.2 État de l'art

Le transfert d'objets homme-robot est bien plus qu'une simple transmission mécanique de coordonnées ou de forces. La littérature le définit comme une "action conjointe collaborative" complexe, qui nécessite une synchronisation fine entre deux agents distincts : le donneur (humain) et le receveur (robot). Contrairement aux environnements industriels classiques où les robots opèrent dans des cages isolées, le HRH impose une proximité physique et une interaction directe [1].

Le processus se divise conceptuellement en deux phases distinctes qui structurent la recherche actuelle : la phase de pré-handover qui couvre l'ensemble des processus cognitifs et physiques précédant le contact (détection de l'intention de l'humain, reconnaissance de l'objet, planification de la trajectoire d'approche et l'établissement d'un accord implicite sur le point d'échange) et la phase de handover physique qui débute au premier contact et se termine au relâchement complet de l'objet par le donneur (gestion des forces, stabilité de la prise et à la décision de fermeture de la pince) [1].

Le projet vise à traiter ces deux phases, avec une emphase particulière sur la phase de pré-handover dans les scénarios dynamiques, où la "négociation" du point d'échange est remplacée par une prédiction unilatérale de la trajectoire de l'objet par le robot.

Le fond du problème reste dans la gestion de la variabilité. L'opérateur humain n'est pas standardisé : sa taille, sa vitesse de mouvement, sa manière de tenir l'objet et son comportement en cas de stress varient. De plus, l'environnement "non structuré" implique des conditions d'éclairage changeantes et des arrière-plans encombrés (cluttered backgrounds) qui compliquent la tâche des algorithmes de vision par ordinateur traditionnels.

L'état de l'art actuel vis à vis de la reconnaissance des actions de l'humain montre une transition nette des méthodes classiques (segmentation par couleur, cinématique inverse analytique) vers des approches basées sur l'apprentissage (Deep Learning, Reinforcement Learning) pour gérer cette incertitude [2].

Dans le contexte du projet, où la réactivité est critique (surtout pour le scénario de l'objet mobile), les architectures "one-stage" comme YOLO s'imposent face aux architectures "two-stage" comme Faster R-CNN. Faster R-CNN, bien que très précis, sépare la génération de propositions de régions et la classification, induisant

une latence souvent incompatible avec le temps de vol d'un objet lancé (quelques centaines de millisecondes)[1].

YOLOv8 et ses successeurs représentent l'état de l'art actuel pour les applications embarquées sur des objets. Ces modèles traitent l'image entière en une seule passe réseau, prédisant simultanément les boîtes englobantes et les classes. YOLOv8 atteint un compromis vitesse/précision supérieur, capable de tourner à plus de 30-60 FPS sur du matériel standard (GPU) voire sur des cartes embarquées performantes[3] [4]. La précision cependant diminue avec la présence de plusieurs humains.

Un défi majeur pour un objet qui bouge est le flou cinétique (motion blur). Les détecteurs standard peuvent échouer si l'objet apparaît comme une traînée. Des variantes récentes, comme FMDS-YOLOv8 (proposé en 2025 pour la robotique agricole mais potentiellement applicable ici), intègrent des modules de fusion de caractéristiques multi-échelles et des mécanismes d'attention pour améliorer la détection de petits objets rapides ou flous. Ce modèle utilise une structure "FocalModulation spatial pyramid pooling" pour mieux capturer le contexte global, ce qui aide à distinguer la balle du fond même en cas de mouvement rapide [5].

Pour une balle rouge (objet sphérique), l'orientation n'a pas d'importance (symétrie infinie), ce qui simplifie le problème à une estimation de position 3D (x, y, z). Cependant, la perception de la profondeur sur des objets petits et courbes est souvent bruitée sur les caméras RGB-D (effets de bord, réflexions spéculaires). On peut opter pour une projection du centroïde du masque 2D vers le nuage de points 3D. C'est rapide mais sensible au bruit. On peut aussi utiliser des réseaux comme DenseFusion ou PoseCNN qui fusionnent les caractéristiques RGB et de profondeur au niveau pixel pour estimer la pose. Ces méthodes sont plus robustes aux occultations partielles (ex : main cachant une partie de la balle)[1].

Enfin, la sécurité est un impératif absolu mentionné dans votre projet : on veut un mouvement non-dangereux. L'interaction physique directe relève de normes strictes [6] :

ISO 13482 (Robots de soins personnels) : Elle définit les exigences pour les contacts physiques. Contrairement aux robots industriels isolés, le contact est autorisé mais l'énergie transférée doit être limitée.

ISO/TS 15066 (Robots collaboratifs) : Elle fournit des seuils biomécaniques précis (force de pression, énergie d'impact) pour chaque partie du corps humain. Pour un projet étudiant/recherche, respecter ces principes signifie implémenter des limites logicielles strictes sur les couples et les vitesses du QArm.

1.3 Hardware / Software architecture

Le projet est contraint par l'utilisation d'une plateforme matérielle spécifique, le robot manipulateur Quanser QArm, et d'un environnement logiciel imposé comprenant Matlab et Simulink, bien que l'ouverture vers Python soit envisagée pour les modules de vision avancée. Les critères de réussite sont stricts : le système doit être prédictible, déterministe, efficace face à la diversité des environnements et des comportements humains, et surtout, garantir une sécurité absolue et une perception de mouvement non dangereux pour l'opérateur.

1.3.1 Architecture Hardware

Le QArm est un manipulateur sériel 4 DOF (Degrés de Liberté) équipé d'une pince à tendons et d'une caméra RGB-D Intel RealSense.

Avec seulement 4 axes, le robot ne peut pas atteindre n'importe quelle position avec n'importe quelle orientation. La planification de trajectoire doit prendre cela en compte pour l'orientation de la pince (souvent maintenue horizontale ou verticale dans les cas qu'on va utiliser).

La communication via USB avec le robot et la caméra induit aussi des délais. Il peut être important de mesurer cette latence pour compenser la prédiction de trajectoire.

1.3.2 Architecture Software

Le projet impose Matlab/Simulink, mais l'utilisation du modèle YOLO se fait en Python. Quanser fournit des API Python complètes qui permettent de contrôler le robot et de lire les capteurs.

Avant de tester sur le robot réel (risques de collision, robot arrivé fin novembre), l'utilisation d'un environnement de simulation est indispensable. Quanser propose QLabs, un jumeau numérique haute fidélité du QArm. Cela permet de valider les algorithmes de vision (avec une caméra virtuelle) et de contrôle dans un environnement physique simulé (Unreal Engine), ce qui facilite le débogage des scénarios de "catching" sans risque matériel.

Chapitre 2

Déplacement du robot : mise en équation et simulation

2.1 Résumé des hypothèses principales

Pour le développement de l'architecture de contrôle, nous avons établi trois hypothèses fondamentales. Premièrement, la génération de trajectoire suppose une vitesse et une accélération nulles au début et à la fin de chaque segment, c'est-à-dire aux points de passage (*waypoints*). Deuxièmement, le délai de communication introduit par le protocole UDP entre le script Python (vision/planIfication) et Simulink (contrôleur) est considéré comme négligeable pour la stabilité de la boucle de contrôle de bas niveau. Enfin , le préhenseur (*gripper*) est modélisé comme un actionneur binaire (Ouvert/Fermé), où la commande logique est mise à l'échelle vers un signal PWM (0% à 100%) pour l'interface matérielle.

2.2 Description du système

Pour décrire le mouvement et la logique de contrôle, nous utilisons un ensemble de notations spécifiques. Les positions cibles pour la saisie et le dépôt de l'objet sont notées respectivement P_{pick} et P_{place}. Ces coordonnées sont dynamiques et transmises au contrôleur via le protocole UDP.

Afin d'éviter les collisions lors des déplacements verticaux, des positions d'approche, notées P_{appr}, sont calculées en amont (dans le script Python) en ajoutant un décalage vertical de sécurité (+15 cm) aux cibles. La position réelle de l'effecteur dans l'espace opérationnel, calculée à chaque pas de temps par le modèle de cinématique directe corrigé, est notée P_{meas}.

Enfin, la logique d'activation du préhenseur repose sur un seuil de tolérance de distance euclidienne, noté ϵ, fixé à 0.05 m pour ce projet. La génération de trajectoire, quant à elle, s'appuie sur une durée de mouvement T et des coefficients de splines

cubiques a_i calculés pour assurer la continuité de la vitesse.

2.3 Modèle de développement

Pour assurer un mouvement fluide entre les points de saisie et de dépôt, nous avons implémenté un générateur de trajectoire basé sur des **splines cubiques**. Cette approche mathématique permet de définir une trajectoire continue en position et en vitesse, évitant ainsi les sauts discontinus qui pourraient endommager les actionneurs.

Pour un segment de trajectoire commençant à $t = 0$ et finissant à $t = T$, la position $x(t)$ est définie par un polynôme de troisième degré :

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{2.1}$$

Les coefficients a_i sont déterminés en résolvant un système d'équations linéaires basé sur quatre contraintes aux limites : la position initiale x_0, la position finale x_f, la vitesse initiale $\dot{x}_0$ et la vitesse finale $\dot{x}_f$. En appliquant l'hypothèse de vitesses nulles aux points de passage, le système permet d'obtenir la matrice de coefficients suivante :

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \\ \frac{3(x_f - x_0)}{T^2} \\ \frac{-2(x_f - x_0)}{T^3} \end{bmatrix} \tag{2.2}$$

Ces coefficients sont calculés dynamiquement en temps réel pour chaque axe (x, y, z) dans l'espace opérationnel afin de générer la consigne de position.

2.4 Formulation mathématique du problème

Le développement du modèle effectué dans la section précédente permet de définir la structure cinématique du robot QArm. Cette section formalise le problème mathématique de la commande, qui consiste à déterminer les variables articulaires nécessaires pour atteindre une pose désirée de l'effecteur, tout en respectant les contraintes géométriques et cinématiques.

Soit $\mathbf{q} \in R^4$ le vecteur des variables articulaires (angles moteurs) et $\mathbf{x}_{des} \in R^4$ le vecteur représentant la pose désirée de l'effecteur dans l'espace opérationnel (position cartésienne et orientation en lacet) :

$$\mathbf{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}, \quad \mathbf{x}_{des} = \begin{bmatrix} x_{des} \\ y_{des} \\ z_{des} \\ \gamma_{des} \end{bmatrix} \tag{2.3}$$

Le problème revient à inverser la fonction de cinématique directe $f(\cdot)$, établie précédemment, pour trouver la configuration $\mathbf{q}$ satisfaisant :

$$\mathbf{q} = f^{-1}(\mathbf{x}_{des}) \tag{2.4}$$

Cette solution doit satisfaire l'ensemble des contraintes mécaniques du robot, définies par les limites angulaires de chaque articulation i :

$$\theta_{i,min} \leq \theta_i \leq \theta_{i,max}, \quad \forall i \in \{1, \ldots, 4\} \tag{2.5}$$

La résolution analytique de ce système permet de déterminer les angles nécessaires pour positionner l'effecteur aux coordonnées de saisie (P_{pick}) et de dépôt (P_{place}) reçues via l'interface de communication.

2.4.1 Définition de l'espace d'état

Le système est décrit par la relation entre deux espaces vectoriels distincts :

— **L'espace articulaire (Joint Space) :** Il est défini par le vecteur des coordonnées généralisées $\mathbf{q} \in R^n$, où n est le nombre de degrés de liberté du robot.

$$\mathbf{q} = \begin{bmatrix} q_1 & q_2 & \ldots & q_n \end{bmatrix}^T \tag{2.6}$$

Chaque composante q_i correspond à la rotation angulaire de l'actionneur i.

— **L'espace opérationnel (Task Space) :** Il est défini par le vecteur pose $\mathbf{X}$ de l'effecteur (Tool frame) exprimé dans le repère de base (Base frame).

$$\mathbf{X} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T \tag{2.7}$$

Où (x, y, z) représentent la translation de l'origine du repère outil et (ϕ, θ, ψ) représentent l'orientation (rotations respectives autour des axes x, y, z illustrées en Figure 2.1).

2.4.2 Le modèle cinématique

Le problème direct (MGD) consiste à exprimer la pose de l'effecteur en fonction des coordonnées articulaires et des paramètres géométriques fixes du robot, tels que définis dans le Tableau 2.1 (notamment la distance verticale L_1).

$$\mathbf{X} = f(\mathbf{q}, L_1, \dots) \tag{2.8}$$

La fonction f est une fonction non-linéaire obtenue par la composition des matrices de transformation homogène successives. Par exemple, la coordonnée z de l'effecteur dépendra explicitement du paramètre L_1 (distance base-actionneur 1) et de la configuration des bras subséquents.

Inversement, le problème de commande (MGI) — nécessaire pour la simulation du déplacement — cherche à déterminer la configuration $\mathbf{q}$ pour une consigne $\mathbf{X}_{des}$ donnée :

$$\mathbf{q} = f^{-1}(\mathbf{X}_{des}) \tag{2.9}$$

2.4.3 Problème d'optimisation sous contraintes

La recherche de la solution $\mathbf{q}(t)$ sur l'intervalle de temps $[0, T]$ ne se réduit pas à une simple inversion algébrique. Elle doit être formulée comme un problème d'optimisation sous contraintes afin de garantir l'intégrité physique du robot.

Le problème mathématique se formule ainsi : trouver la trajectoire $\mathbf{q}(t)$ minimisant l'erreur de suivi $||\mathbf{X}_{des}(t) - f(\mathbf{q}(t))||$ sous les contraintes suivantes :

— **Contraintes de butées articulaires (Position Limits)**
Chaque articulation est limitée physiquement par sa conception mécanique :

$$q_{i,min} \leq q_i(t) \leq q_{i,max} \quad \forall t \in [0, T] \tag{2.10}$$

— **Contraintes cinématiques (Velocity Limits)**
Pour protéger les moteurs et la transmission, les vitesses articulaires sont bornées :

$$|\dot{q}_i(t)| \leq \dot{q}_{i,max} \tag{2.11}$$

— **Contraintes géométriques structurelles**
L'espace de travail atteignable (Workspace) est strictement délimité par les dimensions des liens (ex : L_1). Une solution $\mathbf{X}_{des}$ est valide si et seulement si elle appartient à ce sous-espace $\mathcal{W} \subset R^6$ engendré par la géométrie du robot.

2.5 Architecture générale

La solution conçue met en œuvre une architecture de système cyber-physique, découplant la planification de haut niveau de la commande de bas niveau. Le système opère à travers trois couches principales interconnectées.

Tout d'abord, la couche Perception Planification (Python) fait office de simulateur du système de vision. Elle calcule les vecteurs cibles (Prise, Dépose et Approche) à partir des entrées en coordonnées polaires et les encapsule dans une structure binaire. Ces données sont ensuite transmises à la couche Communication (UDP), qui transmet les données de trajectoire (15 valeurs en double précision) via un socket réseau local en utilisant un tampon à taille fixe pour garantir la stabilité en temps réel. Enfin, la couche Commande (Simulink) reçoit les données binaires, les désencapsule par transtypage mémoire, génère les points de passage de la trajectoire et exécute la cinématique inverse pour piloter le QArm.

Functional diagram(s)

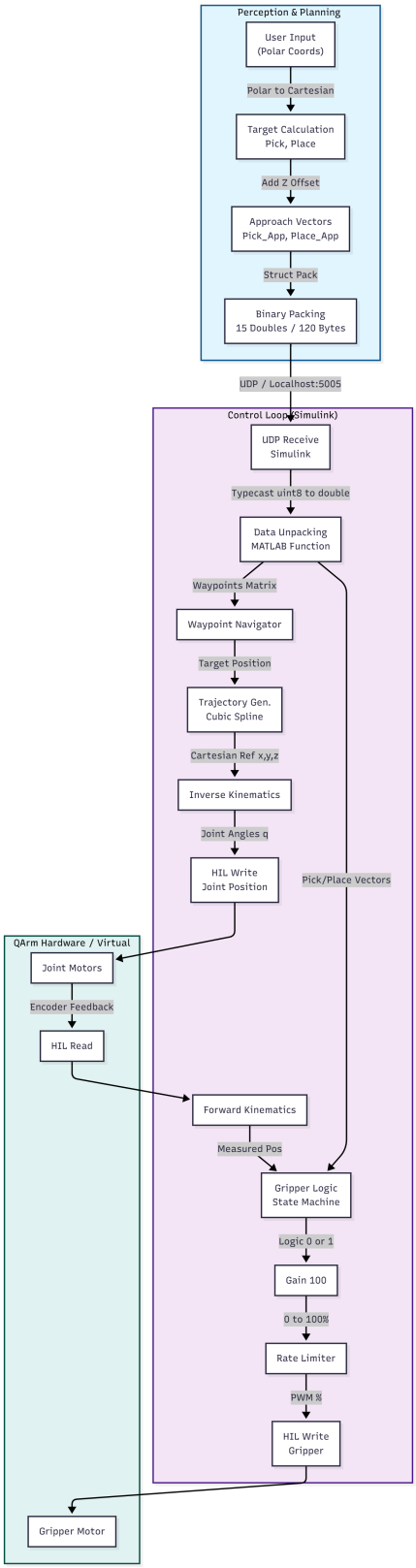


FIGURE 2.1 – diagramme de fonctions

2.6 Sous-système Hardware

Le sous-système matériel est le manipulateur série QArm (voir Fig. 2.2). Une analyse dimensionnelle a été réalisée pour valider son adéquation avec les exigences de la tâche de "Pick and Place".

La contrainte principale est l'accessibilité des cibles P_{pick} et P_{place}. L'espace de travail accessible est défini par l'enveloppe cinématique des articulations. Avec des longueurs de segments de $L_2 = 0.35$ m et $L_3 = 0.25$ m, la portée radiale maximale R_{max} est suffisante pour atteindre les cibles situées à un rayon $r \approx 0.5$ m, garantissant la condition $||P_{target}|| < R_{max}$.

Concernant l'effecteur final, la condition de saisie stable ("Force Closure") impose que l'ouverture maximale de la pince $W_{gripper}$ soit supérieure à la dimension de l'objet D_{obj} incluant une marge de sécurité δ pour l'erreur de positionnement ($W_{gripper} > D_{obj} + 2\delta$). Cette vérification dimensionnelle assure que les stratégies d'approche générées par le script Python sont physiquement réalisables.

FIGURE 2.2 – Vue isométrique du manipulateur QArm et de son effecteur final.

2.7 Sous-système Software

Le sous-système logiciel repose sur une architecture distribuée où la planification de haut niveau (Python) est découplée du contrôle temps réel (Simulink). Cette section détaille le flux de données, de la génération des coordonnées à l'exécution du mouvement.

2.7.1 Génération de Trajectoire et Interface Python

Un script Python agit comme un simulateur de système de vision. Il permet à l'utilisateur de définir les cibles (*Pick* et *Place*) en coordonnées polaires (rayon r, angle θ, hauteur z), plus intuitives pour l'opérateur.

Le script convertit ces entrées en coordonnées cartésiennes (x, y, z) pour le référentiel du robot. Une fonctionnalité de sécurité est intégrée au logiciel : le calcul automatique des vecteurs d'approche (P_{appr}) en ajoutant un décalage vertical ($z + 0.15m$) pour éviter les collisions lors des mouvements horizontaux.

Listing 2.1 – Conversion polaire-cartésienne implémentée en Python

```
def polar_to_cartesian(radius, angle_degrees, height):
    angle_radians = math.radians(angle_degrees)
    x = radius * math.cos(angle_radians)
    y = radius * math.sin(angle_radians)
    z = height
    return [x, y, z]
```

2.7.2 Protocole de Communication UDP et Encapsulation Binaire

Pour assurer une transmission fiable et rapide vers Simulink, nous avons opté pour une communication UDP binaire. Au lieu d'envoyer des chaînes de caractères (ce qui a posé des problèmes de *parsing* lors des tests initiaux), le script concatène les 5 vecteurs clés (Home, Pick, Pick_App, Place, Place_App) et les encapsule directement en octets.

Le module `struct` de Python est utilisé pour empaqueter 15 valeurs flottantes (`double`) dans un paquet de taille fixe (120 octets), garantissant l'intégrité des données reçues par le contrôleur.

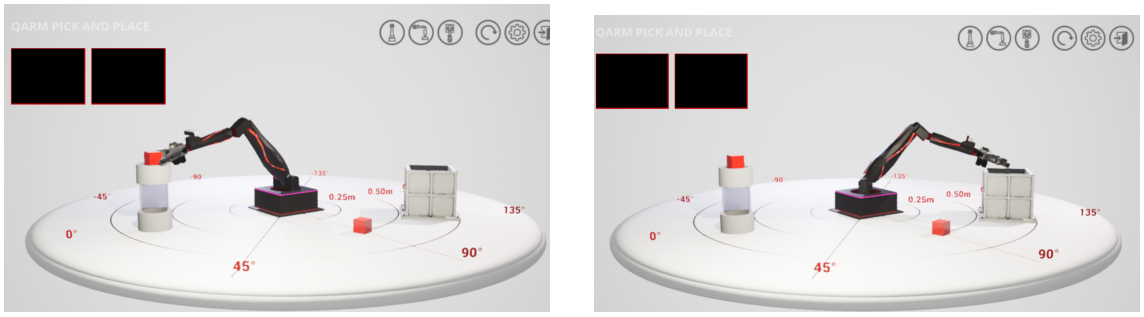
Listing 2.2 – Encapsulation binaire des données de trajectoire

```
# Concaténation des 5 vecteurs
all_values = (
    home_vector +
    pick_vector +
    pick_approach_vector +
    place_vector +
    place_approach_vector
)

# '15d' signifie : empaqueter 15 valeurs de type 'double' (8 octets)
message_bytes = struct.pack('15d', *all_values)

# Envoi du paquet binaire via UDP
sock.sendto(message_bytes, (UDP_IP, UDP_PORT))
```

2.7.3 Intégration Simulink et Résultats de Simulation

Côté réception, le modèle Simulink décode le flux binaire instantanément grâce à la fonction `typecast`. Les coordonnées sont ensuite injectées dans le générateur de trajectoire par splines cubiques et la machine à états du préhenseur.

Les figures ci-dessous illustrent le résultat final de cette chaîne logicielle, montrant le robot QArm atteignant précisément les positions cibles calculées par le script Python.



(a) Robot en position de saisie (Pick) (b) Robot en position de dépôt (Place)

FIGURE 2.3 – Validation de la chaîne de commande logicielle dans l'environnement Quanser Interactive Labs.

2.8 Manipulation de l'outil et cinématique différentielle

Dans le cadre de ce projet, nous avons également pris en compte les notions de *cinématique différentielle* et de *manipulation de l'outil*, telles que présentées dans le Lab 6 de Quanser. Contrairement aux premières approches basées uniquement sur la position articulaire, la cinématique différentielle permet de relier les vitesses articulaires aux vitesses de l'effecteur dans l'espace cartésien.

Cette formulation est essentielle lorsque l'on souhaite réguler non seulement la position de l'effecteur, mais aussi la manière dont il se déplace, par exemple pour garantir une vitesse uniforme dans des tâches comme la peinture, la soudure ou le dépôt d'un objet.

2.8.1 Cinématique différentielle

En dérivant la cinématique directe par rapport au temps, on obtient la relation :

$$V = J_0 \, \dot{\Theta},$$

où V représente les vitesses cartésiennes de l'effecteur, $\dot{\Theta}$ les vitesses articulaires, et J_0 le Jacobien du manipulateur exprimé dans le repère de base.

Cette relation, détaillée, permet d'analyser et de contrôler la vitesse de l'effecteur, ce qui n'est pas possible avec un simple contrôle en espace articulaire.

2.8.2 Utilisation de machines à états

Quanser recommande l'usage de *machines à états finis* (FSM) pour gérer les déplacements en espace opérationnel. Une FSM permet de structurer la logique de déplacement de l'effecteur en plusieurs étapes :

— initialisation : préparation du système ;
— déplacement vers la cible à vitesse cartésienne constante ;
— attente ou stabilisation une fois proche de la cible ;
— terminaison pour repasser à un état neutre ou enchaîner une nouvelle tâche.

Ce type d'architecture est simple à implémenter en pratique, via des structures `if-else`, `switch-case`, ou des blocs Simulink équivalents. Elle est particulièrement utile pour automatiser des tâches de manipulation où l'on doit garantir un mouvement fluide, reproductible et contrôlé de l'effecteur.

2.9 Identification des singularités

Lors du développement de l'architecture de contrôle du robot, nous avons pris en compte les singularités cinématiques du QArm. Une singularité correspond à une configuration articulaire dans laquelle l'effecteur perd la capacité de se déplacer dans une ou plusieurs directions cartésiennes, ce qui se traduit par une perte de rang dans la matrice jacobienne du manipulateur.

Selon la documentation Quanser, deux conditions principales de singularité affectent le QArm :

— lorsque l'effecteur est positionné directement au-dessus de la base, rendant la matrice jacobienne non inversible ;
— lorsque l'articulation 3 est alignée ou repliée (i.e. $\theta_3 = \pm 90°$), ce qui rend deux colonnes de la matrice jacobienne linéairement dépendantes.

Ces singularités réduisent la contrôlabilité de l'effecteur et doivent être évitées lors des déplacements en espace opérationnel.

2.10 Stratégie d'évitement des singularités

Nous avons adopté une stratégie simple d'évitement des singularités, inspirée de la zone cylindrique d'exclusion. L'idée est d'empêcher l'effecteur d'entrer dans un volume prédéfini autour de l'articulation de base, volume correspondant à l'une des régions singulières connues du QArm.

La stratégie consiste à :

— détecter lorsque la trajectoire commandée intersecte le cylindre d'évitement ;

— dévier la trajectoire pour suivre une trajectoire circulaire sur la frontière du cylindre ;
— reprendre la trajectoire nominale une fois que l'effecteur est sorti de la zone.

2.11 Contraintes techniques du cahier des charges

Le robot doit répondre aux spécifications techniques réelles suivantes, fondée sur la documentation fournie pour la simulation et en prenant en compte le comportement réel de l'appareil :

2.11.1 Caractéristiques Générales du Manipulateur

— **Degrés de Liberté (DoF) :** 4 DoF sériels, plus la capture. Configuration : Base (Roulis), Épaule (Tangage), Coude (Tangage), Poignet (Tangage).
— **Portée (Reach) :** 1m.
— **Charge Utile (Payload) :** 250 g à 750 g (selon la documentation).
— **Répétabilité (Repeatability) :** $\pm 0,05$ mm.
— **Masse du Manipulateur :** $8,25$ kg.

2.11.2 Plages et Vitesses Articulaires

Le tableau 2.1 détaille les plages et vitesses maximales des articulations :

TABLE 2.1 – Spécifications des Articulations du QArm

cccc

Articulation	**Plage Articulaire** (θ)	**Plage Articulaire** (rad)	**Vitesse Max.** (deg/s)
Base	$\pm 170°$	$\approx \pm 2,97$ rad	$\pm 90°/s$
Épaule	$\pm 85°$	$\approx \pm 1,48$ rad	$\pm 90°/s$
Coude	$-95°$ à $+75°$	$\approx -1,66$ rad à $+1,31$ rad	$\pm 90°/s$
Poignet	$\pm 167,5°$	$\approx \pm 2,92$ rad	$\pm 90°/s$

2.11.3 Effecteur et Capteurs

— **Gripper :** À deux doigts, basé sur tendon, permettant une saisie de l'objet en deux étapes (5 points de contact).
— **Capteur Visuel :** Caméra **RGBD** (e.g., Intel® RealSense™ D415) pour la profondeur et l'image couleur.
— **Capteurs Moteurs :** Surveillance de la position, de la vitesse et du courant de chaque articulation.

2.11.4 Contrôle et Interfaçage

— **Architecture :** Ouverte, permettant un accès aux signaux internes pour la création d'algorithmes de contrôle personnalisés.
— **Taux de Contrôle Externe :** 500 Hz.
— **Modes de Contrôle Internes :** Mode Position et Mode Courant.
— **Interfaces I/O Étendues :** Prise en charge des protocoles PWM, Analogique, I^2C, SPI, UART.
— **Connectivité :** Interface **QFLEX 2** (USB ou SPI).

Chapitre 3

Caméra et IA

3.1 Introduction

Ce chapitre détaille la chaîne de traitement mise en place pour permettre au bras robotique Quanser de détecter et saisir la balle de tennis. La solution repose sur une architecture de vision en temps réel (**YOLOv11s**) et une interface de communication réseau optimisée pour le contrôle bas-niveau. Les données de détection sont transmises au contrôleur Simulink/QUARC pour la planification de la trajectoire du bras.

3.2 Module de Vision : De l'Image à la Coordonnée

3.2.1 Choix du Modèle et Entraînement

Le module de détection repose sur l'architecture **YOLOv11s** (You Only Look Once, version Small), choisie pour son excellent compromis entre vitesse d'inférence et précision. Le modèle, nommé `my_model.pt`, a été entraîné spécifiquement sur des images de balles de tennis selon la méthodologie inspirée du tutoriel de référence, garantissant un apprentissage ciblé de l'objet.

Les étapes clés de la création du modèle sont :

1. **Collecte de Données :** Constitution d'un jeu de données varié d'images de balles de tennis sous divers éclairages, angles et arrière-plans.

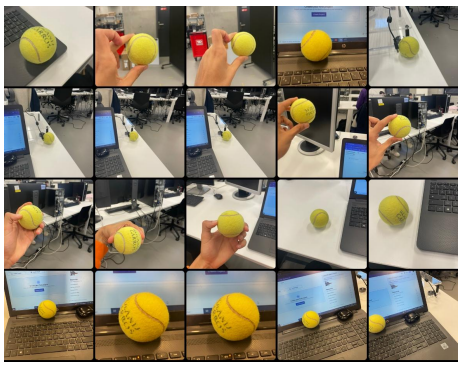


FIGURE 3.1 – Extrait du jeu de données

2. **Annotation des Données (Label Studio) :** L'outil *Label Studio* a été utilisé pour définir précisément les boîtes englobantes (bounding boxes) autour de chaque balle. Les annotations ont été exportées au format standard YOLO.
3. **Entraînement :** Le réseau a été entraîné en utilisant la librairie `ultralytics` (YOLOv11) sur un environnement accéléré (type Google Colab) [7] sur plusieurs époques, jusqu'à l'obtention du fichier de poids `my_model.pt`.

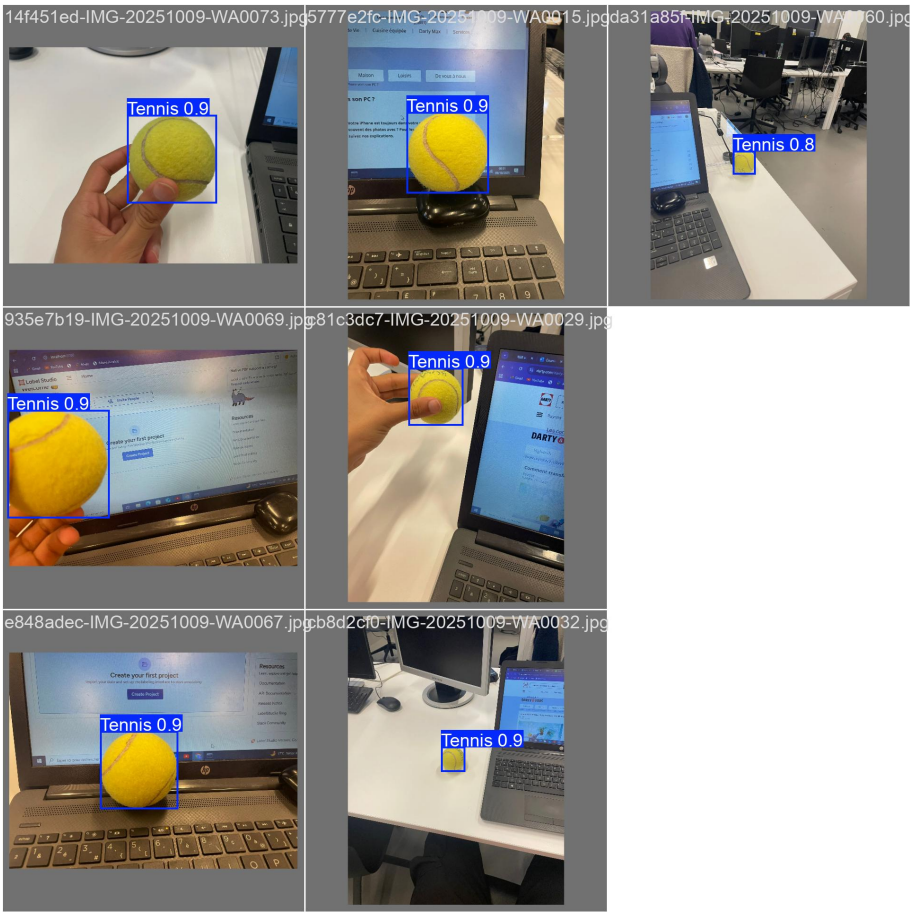


FIGURE 3.2 – Résultats du modèle YOLOv11s

3.2.2 Acquisition d'Image : Caméra Intel RealSense D415

L'acquisition des images est réalisée par la caméra de profondeur **Intel RealSense D415**, pilotée via la librairie `pyrealsense2`.

— **Flux Vidéo :** Le script Python initialise et gère le flux couleur (RGB) à haute résolution (`1920x1080` @ 30 fps ou `1280x720` @ 90 fps).

— **Potentiel 3D :** La RealSense est un atout majeur, car elle permet, dans une phase ultérieure, l'ajout du flux de profondeur. Cette donnée (Z) est essentielle pour la **calibration et la localisation 3D** de l'objet de saisie par le bras Quanser.

3.3 Intégration et Communication avec l'Environnement QUARC

L'interface de contrôle du bras Quanser s'exécute dans l'environnement de modélisation Simulink, utilisant le package temps réel **QUARC**. La communication entre le script de vision (Python) et le contrôleur (Simulink) est assurée par le protocole UDP.

3.3.1 Protocole UDP pour le Temps Réel

Nous avons sélectionné le protocole **UDP** (User Datagram Protocol) pour garantir une latence minimale. Contrairement au TCP, l'UDP n'attend pas de confirmation de réception, privilégiant la transmission rapide des données les plus récentes. Dans le cadre d'un suivi d'objet en mouvement, il est préférable d'avoir une information légèrement moins fiable mais instantanée, plutôt que d'attendre la retransmission d'une information potentiellement obsolète.

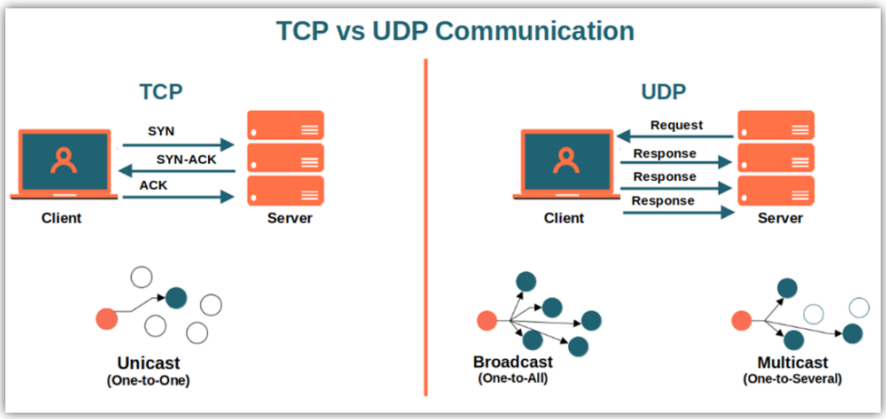


FIGURE 3.3 – Comparaison entre TCP et UDP (Source : https://cheapsslsecurity.com/blog/tcp-vs-udp-how-are-they-different/)

3.3.2 Structuration des Données Binaires pour Simulink

Afin que le bloc de réception de données QUARC puisse lire les informations de manière déterministe et efficace, les coordonnées calculées sont formatées en un paquet binaire de taille fixe.

Contrairement à une approche classique envoyant une boîte englobante 2D, notre script Python calcule la position spatiale et l'état de la pince avant l'envoi. La fonction `struct.pack` est utilisée pour sérialiser les données :

1. **Données :** Quatre valeurs flottantes sont transmises : la position 3D de la balle (x, y, z) et l'état logique de la pince ($grip\_state$).
2. **Format :** Le format d'encodage `'ffff'` garantit que ces quatre valeurs sont empaquetées en **16 octets** (4 float $\times$ 4 octets).

Ce paquet binaire est envoyé à l'adresse IP et au port du contrôleur Quanser. Dans Simulink, un bloc d'entrée (e.g., `UDP Receive`) est configuré pour décoder exactement ces quatre réels, fournissant ainsi la consigne directe pour la cinématique inverse.

Listing 3.1 – Extrait de l'envoi de données structurées par UDP (X, Y, Z, Grip)

```
# Les 4 valeurs à transmettre (Position 3D +  tat  Pince)
# point_to_send contient [x, y, z] calculés par déprojection
# grip_state est 0.0 (ouvert) ou 1.0 (fermé)
message_bytes = struct.pack(
    'ffff',
    point_to_send[0],
    point_to_send[1],
    point_to_send[2],
    float(grip_state)
)

# Envoi vers le contr leur
sock.sendto(message_bytes, (UDP_IP, UDP_PORT))
```

3.3.3 Algorithme de Perception et Stratégie de Saisie

L'architecture logicielle développée assure l'interface entre la vision par ordinateur et la commande du robot. Au-delà de la simple détection, nous avons implémenté une boucle de contrôle robuste conçue pour pallier les limitations physiques du capteur et les latences du système. Le processus décisionnel s'articule autour de trois étapes critiques :

1. **Reconstruction Spatiale (De la 2D à la 3D) :**
   À chaque itération, le réseau de neurones YOLOv11 isole la balle dans le flux RGB et fournit les coordonnées pixels (u, v) du centre de l'objet. Ces données

sont fusionnées avec la mesure brute de profondeur Z issue du capteur Intel RealSense. Une fonction de *déprojection* applique ensuite les paramètres intrinsèques de la caméra pour transformer le triplet (u, v, Z) en coordonnées cartésiennes métriques (x, y, z) dans le repère caméra, rendant la cible exploitable par le modèle cinématique du robot.

2. **Gestion de la Zone Morte (Blind Spot) :**
   Une contrainte majeure des caméras de profondeur est la saturation du capteur à très courte distance ($Z < 20\,\text{cm}$), phénomène critique lors de la phase finale d'approche. Pour éviter une perte de suivi au moment décisif, nous avons implémenté un mécanisme de persistance temporelle. Lorsque l'algorithme détecte une profondeur nulle ($Z = 0$), il bascule automatiquement sur une mémoire tampon conservant la **dernière position valide** connue. Cette substitution est maintenue pendant une fenêtre de tolérance de $0,5\,\text{s}$ (`TIMEOUT_THRESHOLD`), suffisante pour guider le robot "à l'aveugle" sur les derniers centimètres.

3. **Synchronisation Temporelle de la Saisie :**
   La fermeture de la pince n'est pas déclenchée par la simple position, mais par une machine à états temporelle compensant la latence. Dès que la balle pénètre dans la zone d'interaction, une temporisation asynchrone de $0,75\,\text{s}$ (`GRIP_DELAY`) est armée. Ce délai, calibré expérimentalement, permet de synchroniser l'actionnement mécanique de l'effecteur avec l'arrivée physique de l'objet, absorbant ainsi les délais de communication UDP et l'inertie du système pneumatique.

Cette logique permet de transformer une détection visuelle bruitée en une consigne de commande stable et déterministe, condition indispensable pour la réussite de la tâche de "Catching".

Performance Temps Réel : L'efficacité de cet algorithme a été validée expérimentalement sur les stations de travail de la salle de projet. Grâce à l'accélération matérielle fournie par les cartes graphiques **NVIDIA** installées sur ces machines, la boucle complète (acquisition, inférence YOLO, calcul 3D et envoi UDP) atteint une cadence stable comprise entre **45 et 55 FPS**.

3.4 Détection d'objet par couleur (baseline HSV)

Avant de mettre en place une détection par apprentissage profond (YOLO), nous avons commencé par une approche plus simple, entièrement basée sur la **couleur** de l'objet dans l'image. Cette méthode ne tient ni compte de la forme de l'objet ni d'un modèle appris : elle repose uniquement sur le seuillage d'une plage de couleur dans l'espace HSV, suivi d'opérations de filtrage et d'analyse de composantes connexes.

Cette approche s'appuie directement sur les laboratoires Quanser *Image Acquisition and Perception* (Lab 9) et *Object Detection* (Lab 10), qui introduisent la représentation des images comme matrices, les espaces de couleur RGB/HSV, les filtres gaussiens, les transformations morphologiques et l'analyse de blobs dans Simulink. [1]

3.4.1 Principe général

Le pipeline de traitement que nous utilisons peut être résumé en quatre étapes :

1. passage de l'image de l'espace RGB à l'espace HSV ;
2. lissage de l'image par un filtre gaussien pour réduire le bruit ;
3. seuillage min/max sur les trois canaux HSV pour obtenir un *masque binaire* ;
4. sélection de la plus grande composante connexe du masque comme objet détecté, et calcul de sa boîte englobante et de son centroïde.

3.4.2 Passage de RGB à HSV

Les images acquises par la caméra Intel RealSense sont d'abord représentées en RGB, sous la forme d'une matrice $m \times n \times 3$ où chaque pixel possède une intensité rouge, verte et bleue entre 0 et 255. Comme le rappellent les notes de concept Quanser, l'espace RGB mélange l'information de luminosité et de couleur, ce qui rend la segmentation sensible aux variations d'éclairage.

Nous convertissons donc l'image en espace HSV (Hue, Saturation, Value), où :

— la *teinte* (Hue) code la couleur pure (rouge, vert, bleu, etc.) et reste relativement stable quand l'éclairage change ;

— la *saturation* (Saturation) mesure la quantité de gris dans le pixel ;

— la *valeur* (Value) représente la luminosité globale du pixel.

3.4.3 Filtrage gaussien

Avant de segmenter par couleur, nous appliquons un **filtre gaussien** sur l'image (ou sur le canal de travail). Ce filtrage correspond à une convolution de l'image avec un noyau (kernel) gaussien de petite taille, par exemple un noyau 3×3 de la forme :

$$K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Ce type de noyau lisse les variations locales tout en donnant plus de poids au pixel central que l'average filter uniforme. Dans notre cas, il permet de

1. Voir les documents Quanser QArm–Image Acquisition and Perception et QArm–Object Detection.

réduire le bruit et d'éviter l'apparition de nombreux petits artefacts dans le masque binaire.

3.4.4 Segmentation en HSV et masque binaire

La segmentation par couleur consiste à sélectionner les pixels dont la valeur HSV appartient à une plage prédéfinie :

$$H_{\min} \leq H \leq H_{\max}, \quad S_{\min} \leq S \leq S_{\max}, \quad V_{\min} \leq V \leq V_{\max}.$$

Pour chaque pixel (x, y) de l'image, nous testons si son triplet (H, S, V) se trouve dans ces intervalles. On construit ainsi un **masque binaire** $M(x, y)$ tel que :

$$M(x,y) = \begin{cases} 1 & \text{si } (H,S,V) \in [H_{\min}, H_{\max}] \times [S_{\min}, S_{\max}] \times [V_{\min}, V_{\max}], \\ 0 & \text{sinon.} \end{cases}$$

3.4.5 Transformations morphologiques

Le masque binaire obtenu peut contenir du bruit (petits points isolés) ou des trous à l'intérieur de la région d'intérêt. Pour améliorer la qualité du masque, nous appliquons des **transformations morphologiques** classiques : érosion, dilatation et ouverture.

— L'*érosion* remplace chaque pixel par le minimum de son voisinage, ce qui élimine les petites taches de bruit mais amincit aussi les objets.
— La *dilatation* fait l'inverse : chaque pixel est remplacé par le maximum local, ce qui agrandit les régions blanches et comble les petits trous.
— L'*ouverture* consiste en une érosion suivie d'une dilatation, et permet de supprimer le bruit tout en conservant la taille de l'objet principal.

Dans notre implémentation, nous utilisons principalement une ouverture avec un noyau 5×5 sur le masque binaire. Dans le modèle Quanser, cette étape est réalisée dans la section "Filtering" où les commutateurs manuels permettent de tester l'effet des différentes opérations morphologiques sur la sortie *Mask Video Display*.

3.4.6 Analyse de blobs et extraction de l'objet

Une fois le masque propre, la dernière étape consiste à extraire *l'objet le plus probable* à partir des composantes connexes. Nous utilisons pour cela un bloc de *blob analysis* (Image Find Objects) qui :

— parcourt le masque à la recherche de régions de pixels connectés ;
— calcule pour chaque composante son aire (nombre de pixels), son centroïde

et sa boîte englobante ;

— applique un seuil minimal sur l'aire pour ignorer les régions trop petites ;

— peut exclure les objets "coupés" par le bord de l'image.

Dans notre configuration, nous sélectionnons la composante connexe dont l'aire est maximale et nous utilisons son centroïde et sa boîte englobante comme estimation de la position de l'objet dans l'image.

On obtient 3.4 en appliquant ce modèle à une image issue de la caméra du robot pour détecter un objet bleu et renvoyer la distance :

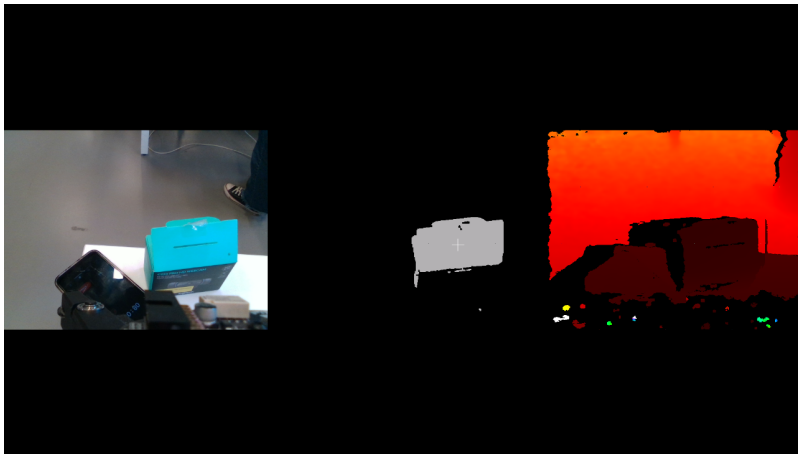


FIGURE 3.4 – Détection d'un objet bleu (images gauche et centre) et la distance (image de droite)

3.4.7 Comparaison entre la détection HSV et la détection YOLO

Dans le cadre de ce projet, nous avons implémenté deux approches complémentaires pour la détection de l'objet : une méthode classique basée sur la couleur en HSV, et une méthode d'apprentissage profond de type YOLOv11. Ces deux techniques présentent des caractéristiques différentes en termes de robustesse, de généralisation et de performance.

Méthode HSV (baseline). La détection par couleur en HSV constitue une **approche de base** pour localiser un objet simple, tel qu'une balle de couleur vive. Elle offre plusieurs avantages :

— Simplicité d'implémentation : la méthode repose uniquement sur un seuillage des canaux HSV et peut être directement mise en œuvre ;

— Coût de calcul très faible : elle fonctionne aisément en temps réel, même sur une machine peu puissante ;

— Aucun entraînement : la méthode ne requiert ni dataset ni phase d'apprentissage.

Cependant, cette approche présente plusieurs limites majeures :

— Sensibilité à l'éclairage : des variations de luminosité ou de saturation peuvent déplacer la plage HSV optimale ;
— Manque de robustesse : tout objet de couleur similaire peut être détecté à la place de la balle ;
— Aucune compréhension visuelle avancée : la méthode ne tient pas compte de la forme, du contexte ou de l'arrière-plan.

Méthode YOLOv11 (deep learning). À l'inverse, la détection par apprentissage profond via YOLOv11 offre une vision beaucoup plus robuste et généralisable :
— Détection basée sur l'apparence complète : la forme, la texture et le contexte visuel sont pris en compte ;
— Forte robustesse aux variations d'éclairage, de fond, d'orientation de l'objet et même aux légères occultations ;
— Adaptation grâce à l'entraînement : le modèle peut être spécialisé pour détecter précisément un type d'objet.
Ses principales limites sont :
— Coût de calcul plus élevé : l'inférence nécessite une carte graphique ou une optimisation logicielle pour atteindre le temps réel ;
— Besoin d'un dataset annoté : la phase d'entraînement impose la création et l'annotation d'un ensemble d'images.

Conclusion comparative. La méthode HSV constitue une solution **rapide, simple et efficace** pour un environnement contrôlé où l'objet possède une couleur facilement isolable. Elle sert de **baseline** pour valider les premières étapes de la chaîne de vision.

La méthode YOLOv11, plus coûteuse mais aussi beaucoup plus robuste, permet d'obtenir une détection fiable dans des conditions réelles : variations d'éclairage, arrière-plans complexes, changements de vitesse ou de forme apparente de l'objet. Elle représente donc la solution finale privilégiée pour notre système d'interaction homme–robot.

Chapitre 4

Implémentation et Contrôle sur le Robot Réel

4.1 Commande et asservissement du Pick and Place

L'implémentation de la loi de commande a été réalisée sous l'environnement Simulink, interfacé avec le robot QArm via la couche logicielle QUARC de Quanser. Cette section détaille l'architecture logicielle déployée pour assurer le cycle de "Pick and Place" de manière robuste et répétable.

4.1.1 Architecture Globale de la Commande

Le schéma de contrôle global, illustré par la Figure 4.1, repose sur une répartition des responsabilités. En effet, nous avons segmenté le problème en trois blocs fonctionnels distincts : la supervision de la mission, la génération de trajectoire et la conversion cinématique.

Cette structure permet de valider chaque sous-système indépendamment avant l'intégration complète sur le matériel (HIL - Hardware In the Loop).

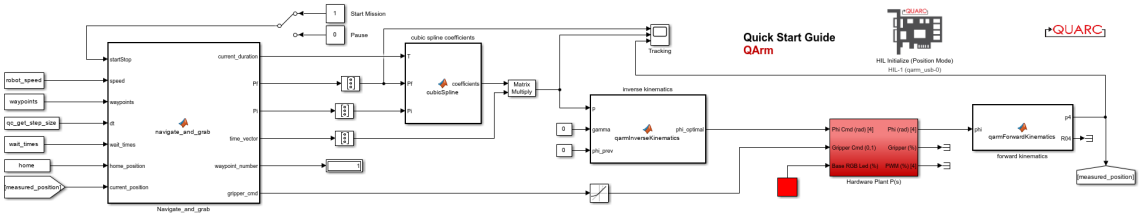


FIGURE 4.1 – Schéma bloc Simulink complet de la commande du QArm. On distingue la chaîne : Planification → Trajectoire → Cinématique Inverse → HIL.

Le flux d'information est le suivant :

1. Le bloc **Navigate_and_grab** détermine la prochaine cible cartésienne (X, Y, Z) et l'état de la pince en fonction de l'avancement du cycle.

2. Le bloc **Cubic Spline** lisse cette consigne pour éviter les discontinuités de vitesse.
3. Le bloc **Inverse Kinematics** traduit cette position lissée en quatre angles moteurs $(\theta_1, \theta_2, \theta_3, \theta_4)$.
4. Le bloc **HIL Initialize** envoie les commandes aux servomoteurs et récupère les mesures des encodeurs.

4.1.2 Génération de Trajectoire par Interpolation Cubique

L'envoi direct d'une consigne de position sous forme d'échelon (step) est proscrit car cela demande une accélération théorique infinie, provoquant des chocs mécaniques et la saturation des actionneurs.

Pour garantir un mouvement fluide, nous avons implémenté un générateur de trajectoire polynomial de degré 3 (Spline Cubique). L'équation horaire de la position est définie par :

$$P(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \quad \text{pour} \quad t \in [0, T_{trajet}] \tag{4.1}$$

Les coefficients a_i sont recalculés à chaque changement de waypoint pour satisfaire les conditions aux limites : position initiale P_i, position finale P_f, vitesse initiale V_i et vitesse finale V_f.

Une particularité de notre implémentation est l'introduction d'un facteur d'**agressivité** (variable `aggressiveness`). Plutôt que d'imposer une vitesse initiale nulle (démarrage très lent), nous autorisons une vitesse initiale proportionnelle à la vitesse moyenne du trajet. Cela permet de dynamiser le cycle tout en restant dans les limites physiques du robot.

Listing 4.1 – Fonction de calcul des coefficients du profil de vitesse

```
1  function coefficients = cubicSpline(T, Pf, Pi)
2      % Facteur empirique déterminant la vivacité du démarrage
           (0.2 = doux)
3      aggressiveness = 0.2;
4
5      % 1. Calcul de la vitesse moyenne nécessaire
6      if T > 0.001
7          V_avg = (Pf - Pi) / T;
8      else
9          V_avg = [0;0;0]; % Protection contre la division par z
               éro
10     end
```

```
11
12      % 2. Définition des conditions aux limites
13      % Vitesse initiale non-nulle pour réduire l'inertie
            apparente au démarrage
14      Vi = V_avg * aggressiveness;
15      % Vitesse finale nulle pour garantir l'arrêt précis sur la
            cible
16      Vf = [0;0;0];
17
18      % 3. Résolution du système d'équations pour le polyn me
            cubique
19      a0 = Pi;
20      a1 = Vi;
21      % Accélération requise
22      a2 = (3*(Pf - Pi) - 2*Vi*T - Vf*T) / (T^2);
23      % Jerk (à-coup) requis
24      a3 = (-2*(Pf - Pi) + (Vi + Vf)*T) / (T^3);
25
26      coefficients = [a0, a1, a2, a3];
27  end
```

4.1.3 Modélisation Cinématique et Optimisation

Le pilotage dans l'espace opérationnel nécessite de résoudre le problème géométrique inverse (MGI). Le bras QArm présente une architecture anthropomorphe avec un décalage latéral (offset) au niveau du coude, modélisé par l'angle $\beta = \arctan(L_3/L_2)$.

Stratégie de résolution du MGI (Modèle Géométrique Inverse)

Le problème inverse admet généralement plusieurs solutions mathématiques pour une même position de l'effecteur (par exemple, configuration "coude en haut" ou "coude en bas"). Pour assurer la continuité du mouvement et éviter que le robot ne tente de changer brusquement de configuration (ce qui serait physiquement impossible), nous avons développé un algorithme de sélection de solution.

À chaque pas de calcul, la fonction `qarmInverseKinematics` :

1. Calcule les 4 solutions théoriques possibles ϕ_{sol}.
2. Compare ces solutions à la configuration angulaire précédente ϕ_{prev}.
3. Sélectionne la solution $\phi_{optimal}$ qui minimise la norme euclidienne : $\|\phi_{new} - \phi_{prev}\|$.

De plus, une fonction de sécurité `check_joint_limits` vérifie que la solution choisie ne viole pas les butées logicielles du robot (ex : $\pm 170°$ pour la base).

Listing 4.2 – Extrait de la sélection de la solution optimale (MGI)

```
% Comparaison itérative pour trouver la configuration la plus
    proche
phi_optimal = phi(:,1);

% Test si la solution 2 est plus proche de l'état précédent
    que la solution 1
if (norm(phi_optimal - phi_prev) > norm(phi(:,2) - phi_prev))
    phi_optimal = phi(:,2);
end
% (Les tests continuent pour les solutions 3 et 4...)

% Sécurité ultime : Si la solution optimale est hors limites,
    on arrête le robot
if (check_joint_limits(phi_optimal))
    phi_optimal = [0;0;0;0];
end
```

Validation par Cinématique Directe

En parallèle, un bloc de Cinématique Directe (MGD) est utilisé sur le retour des encodeurs. Il permet de reconstruire la position réelle (X, Y, Z) de l'outil pour la comparer à la consigne et quantifier l'erreur de poursuite.

4.1.4 Supervision de la Mission : Machine à États

L'intelligence séquentielle du robot est centralisée dans la S-Function `navigate_and_grab`. Ce bloc agit comme un superviseur qui gère les transitions entre les phases d'approche, de manipulation et de repli.

La gestion du temps est ici critique. Plutôt que de définir des durées fixes pour chaque mouvement, le bloc calcule dynamiquement le temps de trajet T_{travel} nécessaire pour maintenir une vitesse linéaire constante $V_{consigne}$, quelle que soit la distance à parcourir :

$$T_{travel} = \frac{\|P_{cible} - P_{actuel}\|}{V_{consigne}} \tag{4.2}$$

Gestion de la tolérance et synchronisation du Gripper

Un point délicat du "Pick and Place" est la synchronisation entre l'arrêt du bras et l'activation de la pince. Si la pince se ferme alors que le bras bouge encore légèrement (phase de stabilisation), l'objet peut être renversé.

Nous avons introduit une variable `POSITION_TOLERANCE` fixée à 40 mm. La machine à états ne valide l'arrivée au point (et donc l'activation du gripper)

que si deux conditions sont réunies :

— Le temps alloué au trajet est écoulé ($t_{clock} \geq T_{travel}$).

— L'erreur de position réelle est inférieure à la tolérance.

L'utilisation de variables `persistent` est indispensable ici pour conserver l'état du système (numéro du waypoint, état de la pince, horloge interne) entre deux appels de la fonction par le solveur Simulink.

Listing 4.3 – Logique de la machine à états pour le séquencement

```
case 1 % ---  TAT  MOUVEMENT ---
    t_clock = t_clock + dt;
    current_error = norm(current_position - p_end);

    % Condition de transition stricte : Temps écoulé ET
       Position atteinte
    if t_clock >= t_travel && current_error <
       POSITION_TOLERANCE
        state = 2; % Passage à l'état PAUSE (Action du Gripper
           )
        t_clock = 0;
        p_start = p_end; % Mise à jour du point de départ pour
            le prochain mouvement

        if target_idx == 1
            % Arrivée au point de Prise (Pick) -> Fermeture
               Pince
            t_wait = TIME_PICK;
            grip_state = 1;
        else
            % Arrivée au point de Dépose (Place) -> Ouverture
               Pince
            t_wait = TIME_PLACE;
            grip_state = 0;
        end
    end
```

Cette architecture garantit que le robot ne passe jamais à l'étape suivante tant que l'étape courante n'est pas physiquement validée, assurant un taux de réussite élevé sur la tâche de manipulation.

4.1.5 Stratégie d'Asservissement et Réglage du PID

Une fois les angles articulaires de consigne calculés par le modèle géométrique inverse, ils sont transmis au contrôleur bas niveau du robot.

Bien que le QArm dispose d'un asservissement par défaut, nos premiers essais ont révélé une erreur de positionnement de l'ordre du centimètre au niveau de l'effecteur. Cette imprécision provient de deux sources conjuguées :

— **Incertitudes du modèle géométrique :** Les longueurs de bras ($L_1, L_2...$) utilisées dans l'algorithme d'inversion sont les valeurs théoriques constructeur. Nous n'avons pas réalisé de calibration métrologique pour vérifier les longueurs réelles, ce qui induit une dérive cartésienne naturelle.

— **Erreur de traînage et gravité :** Sous l'effet de son propre poids (notamment bras tendu), le robot subit une erreur statique si la raideur de l'asservissement est insuffisante.

Pour compenser ces effets sans recalibrer mécaniquement le robot, nous avons pris la main sur le réglage du PID interne via le bloc `HIL Set Property` en mode *Position Control*.

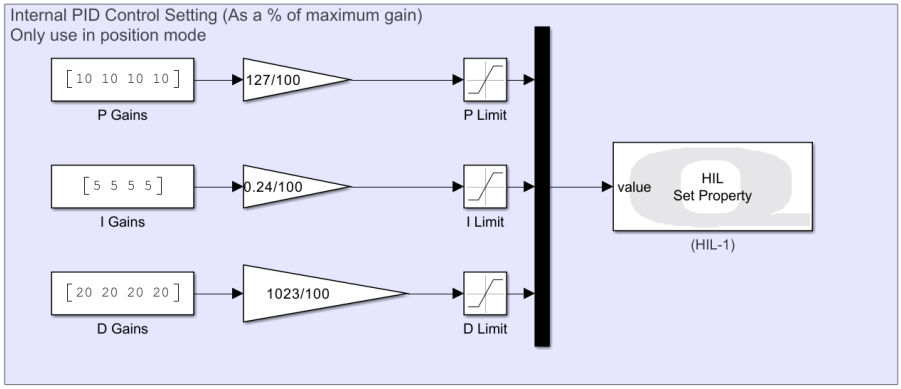


FIGURE 4.2 – Configuration des gains du PID interne via l'interface HIL. On note l'injection de gains vectoriels pour piloter les 4 axes simultanément.

Comme illustré sur la Figure 4.2, nous avons ajusté les gains pour "rigidifier" le comportement du bras :

1. **Gain Proportionnel (P) :** Augmenté pour accroître la réactivité et le couple de rappel immédiat.
2. **Gain Intégral (I) :** L'ajout d'une composante intégrale est ici crucial. Elle permet d'accumuler l'erreur persistante (due au poids du bras et aux frottements) et d'augmenter la commande moteur jusqu'à l'annulation totale de l'écart angulaire.
3. **Gain Dérivé (D) :** Ajusté pour amortir les oscillations induites par l'augmentation du gain P, garantissant un arrêt net sans dépassement.

Pour chiffrer, voici deux essais avec une même commande (x,y,z)=(-0.2 ; -0.5 ; 0.5) sans correction du PID et avec correction du PID.

Cette stratégie de "sur-correction" par le PID permet d'atténuer significativement les erreurs statiques en forçant les moteurs à respecter strictement la consigne angulaire, malgré les perturbations externes. On obtient une erreur statique nulle et une amélioration du temps de réponse de 0.5 s.

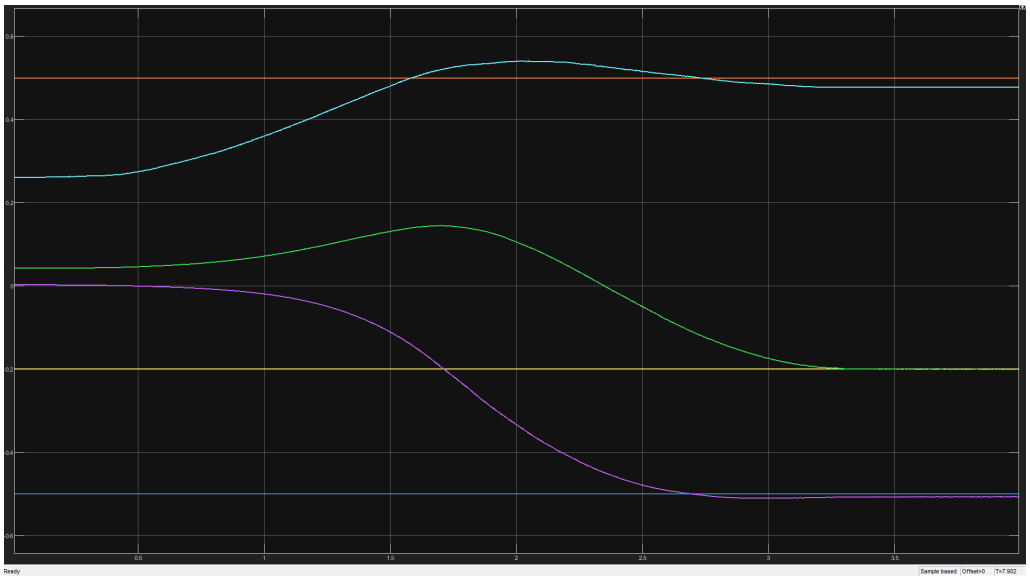


FIGURE 4.3 – (x,y,z)=(-0.2 ; -0.5 ; 0.5) avec PID sans correction

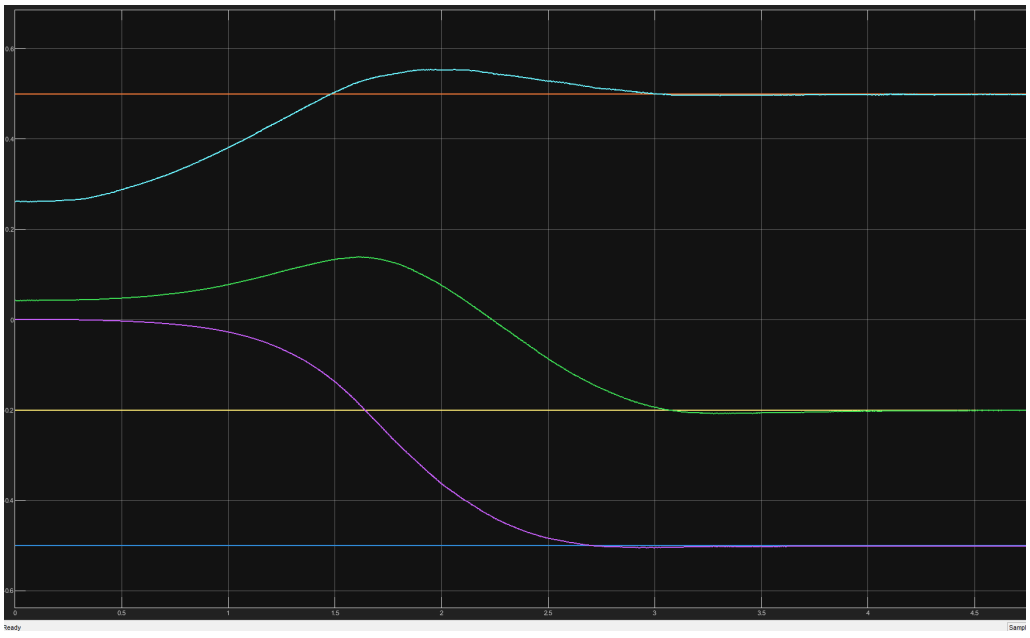


FIGURE 4.4 – (x,y,z)=(-0.2 ; -0.5 ; 0.5) PID avec correction

Chapitre 5

Prédire la trajectoire

5.1 Introduction et Contexte

Cette section est une piste de réflexion pour une autre stratégie que le tracking de la main de l'utilisateur avec l'objet.

5.2 Chaîne de traitement

Afin de permettre au robot d'anticiper le mouvement de la balle, nous avons mis en place une chaîne de traitement qui transforme une information visuelle brute en un modèle physique exploitable. (Voir figure 4.1 pour un schéma bloc résumé)

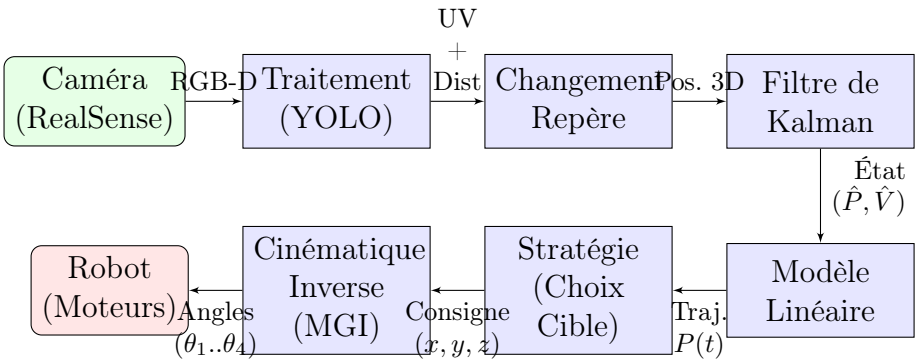


FIGURE 5.1 – Schéma bloc de l'architecture de contrôle incluant la cinématique inverse.

5.2.1 Acquisition et Fusion des données (RGB-D)

Le point de départ est la caméra Intel RealSense D415. Contrairement à une caméra classique qui ne fournit qu'une image 2D, ce capteur nous permet d'obtenir deux flux d'informations synchronisés :

— Flux RGB : Il permet la détection visuelle de l'objet grâce aux algorithmes de vision par ordinateur (modèle YOLOv11 ou seuillage HSV). Cela nous donne les coordonnées (u, v) du centre de l'objet dans l'image (en pixels).
— Flux de Profondeur (Depth) : Il associe à chaque pixel une distance d (en mètres).

En fusionnant ces données, nous reconstruisons la position spatiale de l'objet dans le repère de la caméra, notée $P_{cam} = [x_c, y_c, z_c]$. Cette section n'a pas encore été implémantée.

5.2.2 Transformation de repères et reconstruction 3D

Pour que le QArm puisse se déplacer vers un objet détecté dans l'image, il est nécessaire de convertir les coordonnées pixels issues de la caméra en coordonnées 3D exprimées dans le repère du robot.

Paramètres intrinsèques de la caméra. La caméra Intel RealSense D415 est modélisée par une matrice intrinsèque K, qui relie les coordonnées physiques normalisées (X_p, Y_p) aux coordonnées pixels (u, v) :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_p \\ Y_p \\ 1 \end{bmatrix}.$$

Cette matrice, calibrée à l'aide d'une mire, dépend du point principal et des focales en pixels.

Alignement RGB–Depth. Les images RGB et profondeur ne sont pas alignées par défaut, ce qui fausse l'échantillonnage de la profondeur sur le centroïde détecté. L'alignement des deux images est donc indispensable avant de reconstruire la position 3D de l'objet.

Reconstruction 3D En combinant les coordonnées pixels (u, v) de l'objet avec la valeur de profondeur Z, on obtient sa position dans le repère de la caméra :

$$P_c = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}.$$

Changement de repère caméra → poignet. Une rotation de 90° autour de l'axe z permet d'aligner les axes :

$$R_c^w = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Prise en compte de la translation. Le décalage géométrique entre la caméra et le poignet est modélisé par un vecteur $t_c^w = [d_x\ d_y\ d_z]^T$. L'ensemble forme la matrice homogène :

$$T_c^w = \begin{bmatrix} R_c^w & t_c^w \\ 0 & 1 \end{bmatrix}.$$

Conversion finale. La position de l'objet exprimée dans le repère du poignet est donnée par :

$$P_w = T_c^w\, P_c.$$

Cette position peut ensuite être utilisée pour la commande visuelle ou pour générer une trajectoire de saisie.

5.2.3 Estimation de vitesse : Dérivée naïve vs Filtre de Kalman

Une fois la position P_{mes} obtenue à chaque instant t, nous devons obtenir l'information de vitesse de cet objet pour prédire sa position future.

Une première approche, dite "naïve", consiste à calculer la dérivée discrète de la position entre deux images successives : $V = (P_k - P_{k-1})/\Delta t$. Bien que simple, cette méthode n'a pas été retenue pour l'implémentation finale. En effet, elle présente un défaut majeur : elle amplifie considérablement le bruit de mesure. Avec une caméra de profondeur dont la précision fluctue (bruit de quantification en Z), la dérivée brute devient inexploitable pour une prédiction fine.

Pour résoudre ce problème et estimer proprement la vitesse, nous utilisons un Filtre de Kalman. Cet algorithme récursif permet de fusionner le modèle physique du mouvement (prédiction) avec les mesures réelles (correction) pour minimiser l'erreur quadratique.

Formulation théorique et modélisation stochastique

Nous considérons le problème de la reconstruction de l'état en présence de bruits. Le mouvement de l'objet est modélisé par un système dynamique linéaire invariant en temps discret.

Nous définissons le vecteur d'état X_k à l'instant k, contenant la position et la vitesse :

$$X_k = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \tag{5.1}$$

L'équation d'état et l'équation de mesure s'écrivent sous la forme standard

suivante :

$$\begin{cases} X_k = FX_{k-1} + v_k \\ Z_k = HX_k + w_k \end{cases} \tag{5.2}$$

Dans notre cas, l'entrée de commande u est nulle (l'objet est en mouvement libre ou porté). Les termes v_k et w_k représentent respectivement le bruit de processus (incertitudes sur le modèle de vitesse constante) et le bruit de mesure (imprécision des capteurs). Conformément à la théorie de l'estimation optimale, nous faisons l'hypothèse que v_k et w_k sont des bruits blancs centrés gaussiens non corrélés, caractérisés par leurs matrices de covariance :

— $E\{v_k v_k^T\} = V$, où V est la matrice de covariance du bruit de processus (confiance dans le modèle).

— $E\{w_k w_k^T\} = W$, où W est la matrice de covariance du bruit de mesure (confiance dans les mesures).

Définition des matrices du modèle : Dans l'hypothèse d'un mouvement à vitesse constante entre deux acquisitions séparées par un temps d'échantillonnage Δt (correspondant à l'inverse du framerate de la caméra), la matrice de transition F et la matrice d'observation H (ne mesurant que la position) s'écrivent :

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I_3 & \Delta t \cdot I_3 \\ 0_3 & I_3 \end{bmatrix} \tag{5.3}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} I_3 & 0_3 \end{bmatrix} \tag{5.4}$$

L'objectif est de trouver l'estimée $\hat{X}_k$ qui minimise la variance de l'erreur d'estimation $\Sigma_k = E\{(X_k - \hat{X}_k)(X_k - \hat{X}_k)^T\}$.

Analyse de convergence et stabilité

Pour garantir que notre estimateur fonctionne correctement sur la durée et ne diverge pas, nous devons vérifier les propriétés structurelles du système. D'après le cours sur la commande par retour d'état avec observateur, l'existence et la stabilité de l'observateur dépendent de la propriété d'observabilité.

Nous appliquons le critère de rang de Kalman sur la paire (F, H). La

matrice d'observabilité s'écrit :

$$\mathcal{O} = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix} \tag{5.5}$$

En calculant les deux premiers blocs de cette matrice avec nos définitions de F et H, nous obtenons :

$$\begin{bmatrix} H \\ HF \end{bmatrix} = \begin{bmatrix} I_3 & 0_3 \\ I_3 & 0_3 \end{bmatrix} \begin{bmatrix} I_3 & \Delta t \cdot I_3 \\ 0_3 & I_3 \end{bmatrix} = \begin{bmatrix} I_3 & 0_3 \\ I_3 & \Delta t \cdot I_3 \end{bmatrix} \tag{5.6}$$

Puisque le temps d'échantillonnage $\Delta t \neq 0$, cette matrice est de rang plein (rang = 6). Dans notre modèle cinématique, la mesure de la position suffit donc à reconstruire l'intégralité de l'état, y compris la vitesse. La paire (F, H) est observable.

Bien que le formalisme théorique du cours de Commande des Systèmes Dynamiques de 2ème année soit souvent présenté en temps continu, les résultats fondamentaux concernant la commandabilité, l'observabilité et la convergence de l'estimateur s'étendent directement au cas discret. Ainsi, en vertu du théorème de convergence du filtre de Kalman, puisque la paire (F, H) est détectable (car observable) et la paire $(F, V^{1/2})$ est stabilisable :

1. La variance de l'erreur d'estimation Σ_k converge vers une matrice constante unique Σ_∞, solution définie positive de l'équation algébrique de Riccati.
2. Le gain de Kalman converge vers une valeur constante L_∞.

Cette convergence théorique est fondamentale pour notre projet. Elle signifie que nous pouvons utiliser un *Filtre de Kalman Asymptotique* (gain constant) après une courte période transitoire.

Implémentation sous MATLAB

Pour la mise en œuvre pratique, nous utilisons le bloc *Kalman Filter* de l'environnement MATLAB/Simulink. Ce bloc réalise les opérations suivantes à chaque pas de temps k (le temps utilisé est fixe) :

1. **Prédiction :** Il utilise le modèle interne (matrices F et B) pour estimer l'état a priori à partir de l'état précédent.
2. **Correction :** Il calcule l'innovation (la différence entre la position mesurée par la caméra Z_k et la position prédite). Il multiplie cette erreur par le gain de Kalman L (calculé à partir des matrices de covariance V et W) pour corriger l'estimation d'état.

Le résultat de ce filtrage est un vecteur d'état lissé $\hat{X}_k$ contenant une estimation propre de la vitesse $V = [\dot{x}, \dot{y}, \dot{z}]$, utilisable pour l'extrapolation linéaire de la trajectoire.

5.2.4 Modélisation cinématique (Approche linéaire)

Disposant désormais de la position P_0 et de la vitesse V_0 à l'instant t_0, nous pouvons anticiper la position de l'objet à un instant futur t.

Dans notre scénario, l'objet est déplacé par l'utilisateur vers le robot. Nous faisons donc l'hypothèse simplificatrice que le mouvement est rectiligne et uniforme sur la courte fenêtre de temps de la prédiction et de l'interception avec le bras du robot (accélération nulle). La trajectoire anticipée suit donc les équations linéaires suivantes :

$$\begin{cases} x(t) = x_0 + \dot{x}_0 t \\ y(t) = y_0 + \dot{y}_0 t \\ z(t) = z_0 + \dot{z}_0 t \end{cases} \tag{5.7}$$

5.3 Stratégies d'Interception

Une fois la trajectoire future connue, le robot doit prendre une décision : où et quand intercepter l'objet ? Nous avons identifié deux stratégies distinctes.

5.3.1 Stratégie 1 : Restriction au Plan (Approche 2D)

Cette première approche consiste à simplifier le problème en fixant le mouvement du robot dans un plan vertical prédéfini.

Principe de fonctionnement : Le robot ne cherche pas à suivre la balle dans les trois dimensions. L'algorithme calcule mathématiquement l'intersection entre le modèle cinématique de la balle et ce plan virtuel fixe. Le problème se réduit alors à déterminer deux coordonnées sur ce plan.

Avantages et Inconvénients :

— Facilité d'implémentation : Cette méthode réduit la complexité des calculs de cinématique inverse et ne nécessite pas une synchronisation temporelle parfaite entre le bras du robot et la trajectoire de la main avec la balle. Le robot peut se rendre au point d'impact estimé et attendre que la balle arrive.

— Contrainte d'usage : Elle impose à l'utilisateur de viser correctement pour que la trajectoire coupe le plan dans la zone atteignable par le robot (facile selon le cas d'usage du robot). C'est une solution robuste mais moins

flexible.

5.3.2 Stratégie 2 : Interception Dynamique 3D (Approche Optimale)

La seconde stratégie vise à exploiter l'intégralité de l'espace de travail du robot (*Workspace*) sans contrainte géométrique préalable.

Principe de fonctionnement : L'algorithme analyse l'ensemble des points de la trajectoire future de la balle. Il identifie tous les points qui se trouvent à l'intérieur du rayon d'action du robot et sélectionne le point d'interception envisageable ou optimal (en terme de temps). Le robot doit alors coordonner son mouvement pour arriver à ce point précis (x, y, z) exactement au moment t où la balle y passera.

Avantages et Inconvénients :

— Efficacité réelle : Cette approche est beaucoup plus naturelle pour l'utilisateur ("vraie vie").

— Complexité : Elle exige une excellente réactivité du système et une modélisation précise du temps de mouvement du robot. La moindre latence non compensée peut entraîner un échec de la saisie et nuire à la sécurité de l'utilisateur, ce qui est inconcevable.

Conclusion

Ce projet a permis de réaliser une architecture complète pour l'interaction entre un humain et le robot QArm. Nous avons d'abord validé toutes les étapes en simulation sur QLabs avant de passer au robot physique. Le système final est désormais capable de détecter un objet et de le saisir de manière fluide.

Sur le plan du contrôle, nous avons réussi à programmer des mouvements doux grâce aux splines cubiques. Les modèles mathématiques de cinématique inverse permettent au robot d'atteindre précisément les cibles envoyées par l'ordinateur. Nous avons également intégré une stratégie pour éviter les zones de singularités qui pourraient bloquer le bras.

Pour la vision, nous avons deux possibilités : HSV ou modèle Yolov11. Nous avons opté pour le modèle IA. Ce choix rend le robot beaucoup plus robuste face aux changements de lumière et aux arrière-plans complexes. La communication entre Python et Simulink se fait en temps réel via le protocole UDP, ce qui garantit une très faible latence.

Concernant les résultats, les tests finaux confirment que le robot réussit la majorité de ses tentatives de saisie en faisant du tracking.

Une piste de réflexion a été développée durant ce rapport avec la prédiction de trajectoire mais non mise en place.

Enfin, ce projet démontre qu'il est possible de créer une interaction naturelle et sécurisée entre un opérateur humain et un bras robotique. Le système respecte les limites de vitesse et de force pour garantir la sécurité de l'utilisateur. Cette solution ouvre la voie à des applications concrètes d'assistance technique ou de cobotique.

Axes à développer pour un prochain projet sur Interact

— Implanter la machine d'états finis sur le robot
— Changer le modèle et faire une prédiction de trajectoire au lieu du tracking
— Développer le tracking avec le modèle HSV

Annexe A

Liste du matériel et des logiciels (versions comprises)

A.1 Description Hardware

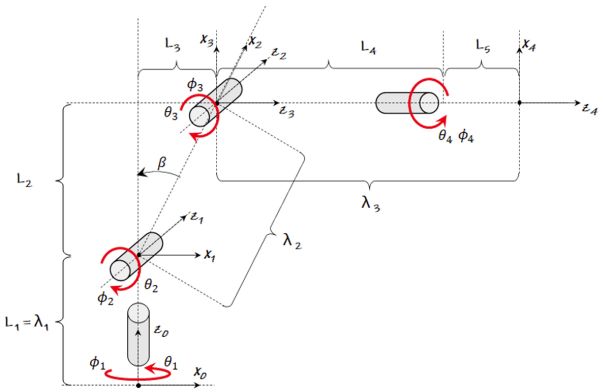


Figure 1 Frame diagram for the Quanser Arm manipulator

FIGURE A.1 – Schéma montrant la structure du robot et les notations utilisées

A.2 Paramétrisation DH et correspondance entre les espaces d'articulations

La Figure 1.1 représente le manipulateur en position "home". À cette position, le vecteur des angles articulaires selon la convention de Denavit–Hartenberg est donné par :

$$\tilde{\Theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} 0 \\ \beta - \dfrac{\pi}{2} \\ -\beta \\ 0 \end{bmatrix}.$$

Les codeurs et actionneurs du robot sont calibrés dans cette configuration. Ainsi, une commande articulaire $[0\ 0\ 0\ 0]^T$ amène le manipulateur en position "home", et les capteurs internes lisent également cette configuration. Cela définit l'espace des angles articulaires physiques, noté ϕ.

Le lien entre les variables articulaires physiques ϕ et les variables DH $\tilde{\Theta}$ est résumé dans le Tableau A.2. Cette transformation permet de travailler en espace DH sans porter explicitement les décalages constants. Par exemple, une commande de $\phi_2 = 0$ correspond à

$$\theta_2 = \beta - \frac{\pi}{2},$$

c'est-à-dire à la configuration "home" pour l'articulation 2.

Les longueurs L_2, L_3, L_4, L_5 servent uniquement à définir les transformations géométriques (rotations et translations élémentaires) et sont également regroupées dans ce tableau.

Nom	**Mesuré depuis**	**Vers**	**Autour / Le long de**
a_i	z_{i-1}	z_i	x_i
α_i	z_{i-1}	z_i	x_i
d_i	x_{i-1}	x_i	z_{i-1}
θ_i	x_{i-1}	x_i	z_{i-1}

TABLE A.1 – Résumé des paramètres DH

Nouveau paramètre	**Paramètre original**
λ_1	L_1
λ_2	$\sqrt{L_2^2 + L_3^2}$
λ_3	$L_4 + L_5$
β	$\tan^{-1}(L_3/L_2)$
Nouveau paramètre	**Paramètre original**
ϕ_1	θ_1
ϕ_2	$\theta_2 + \frac{\pi}{2} - \beta$
ϕ_3	$\theta_3 + \beta$
ϕ_4	θ_4

TABLE A.2 – Transformation linéaire pour simplifier les équations de cinématique

Annexe B

Organisation

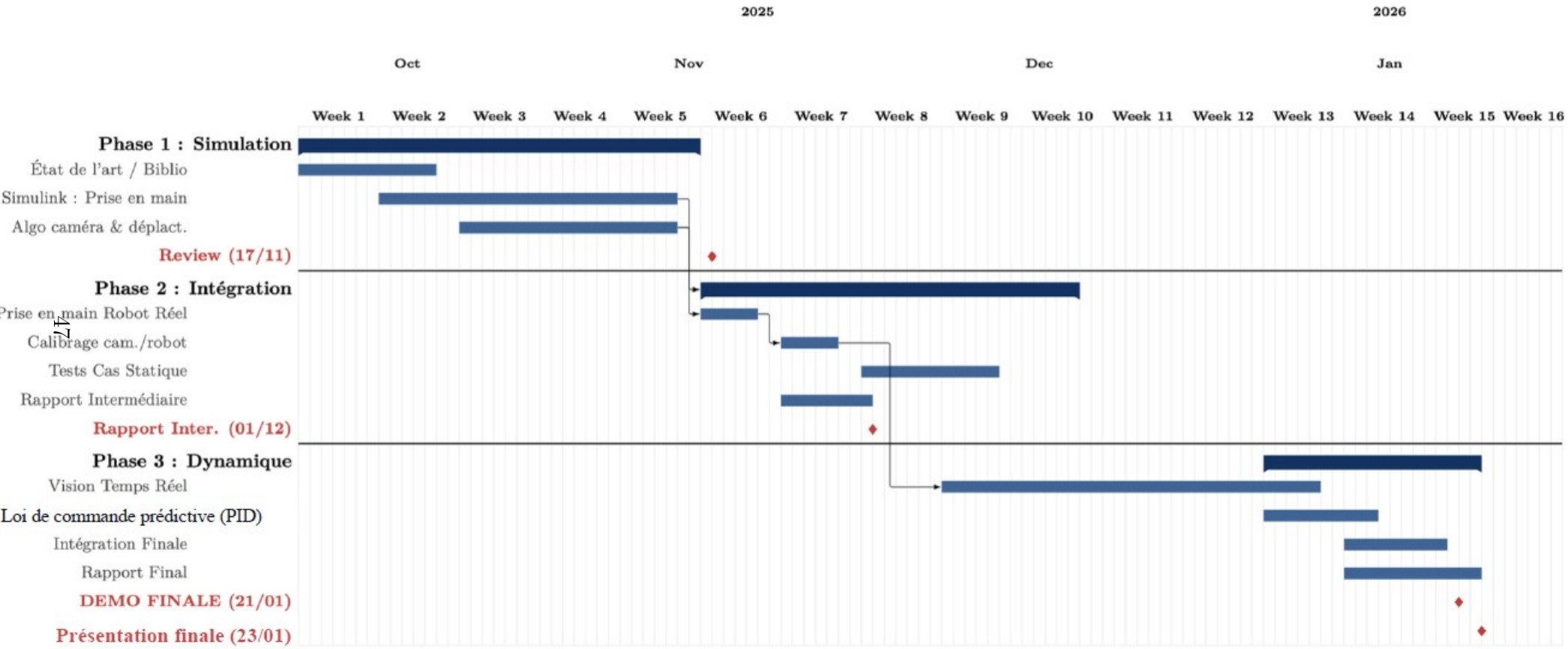
B.1 Diagramme de Gantt

B.2 Répartition des tâches

Chaque tâche/secteur avait un responsable dédié et des membres. Il y avait trois responsabilités :

— Commande
— Modèle Yolo
— HSV
— Interaction/Perception
— Coordination

On avait un responsable pour chacun de ces secteurs qui changeait au fur et à mesure du projet. Tout le monde a été responsable à un moment d'un secteur. A des moments, il n'y avait pas besoin de travailler dans un secteur. Ainsi, on se répartissait dans les autres secteurs. Tout le monde a travaillé dans différents secteurs. Le secteur coordination a été essentiel pour gérer les différents secteurs et la coordination dans les livrables.


2025
2026
Oct
Nov
Dec
Jan
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Phase 1 : Simulation
État de l'art / Biblio
Simulink : Prise en main
Algo caméra & déplact.
Review (17/11)
Phase 2 : Intégration
Prise en main Robot Réel
Calibrage cam./robot
Tests Cas Statique
Rapport Intermédiaire
Rapport Inter. (01/12)
Phase 3 : Dynamique
Vision Temps Réel
Loi de commande prédictive (PID)
Intégration Finale
Rapport Final
DEMO FINALE (21/01)
Présentation finale (23/01)

Bibliographie

[1] M.-G. Kim, Y.-L. Kim, H.-Y. Jeong, H.-C. Park, and J.-H. Park, "Human-to-robot handover control of an autonomous mobile robot based on hand-masked object pose estimation," in *2024 21st International Conference on Ubiquitous Robots (UR)*. IEEE, 2024, pp. 1–8. [Online]. Available : https://ieeexplore.ieee.org/document/10597657

[2] M. Al-Faris, J. Chiverton, D. Ndzi, and A. I. Ahmed, "A review on computer vision-based methods for human action recognition," *Journal of Imaging*, vol. 6, no. 6, p. 46, 2020. [Online]. Available : https://pmc.ncbi.nlm.nih.gov/articles/PMC8321068/

[3] B. Malobický, M. Hruboš, J. Kafková, J. Krško, M. Michálik, R. Pirník, and P. Kuchár, "Towards seamless human–robot interaction : Integrating computer vision for tool handover and gesture-based control," *Applied Sciences*, vol. 15, no. 7, p. 3575, 2025. [Online]. Available : https://www.mdpi.com/2076-3417/15/7/3575

[4] A. Schakkal, G. Bellegarda, and A. Ijspeert, "Dynamic object catching with quadruped robot front legs," 2024, preprint, accepted to IROS 2024. [Online]. Available : https://arxiv.org/abs/2410.08065

[5] Y. Zhuang, K. Xu, Z. Liu, J. Li, L. Shen, and J. Wang, "Design and experimental investigation of the grasping system of an agricultural soft manipulator based on fmds-yolov8," *Frontiers in Plant Science*, vol. 16, p. 1683380, 2025. [Online]. Available : https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2025.1683380/full

[6] Standard Bots, "Collaborative robot safety standards you must know," Standard Bots, 2025. [Online]. Available : https://standardbots.com/blog/collaborative-robot-safety-standards

[7] Edje Electronics, "Train YOLO Models in Google Colab (YOLOv11, YOLOv8, YOLOv5)," Google Colab Notebook, 2025, consulté le 29/11/2025. [Online]. Available : https://colab.research.google.com/github/EdjeElectronics/Train-and-Deploy-YOLO-Models/blob/main/Train_YOLO_Models.ipynb