Ecole Centrale de Lille

Conception et Réalisation d'un Système de Picking Robotisé

Zakaria Midine

Superviseur :
Dr. A. Rahmani

Résumé

Ce projet de recherche se concentre sur la conception et la réalisation d'un système de picking robotisé utilisant des techniques avancées d'intelligence artificielle et de deep learning. L'objectif principal est de développer une solution innovante capable d'identifier et de sélectionner des objets de manière autonome dans un environnement industriel.

Le système proposé comprend un réseau de caméras pour la détection et l'identification des objets, couplé à un robot chargé de l'opération de picking. L'utilisation du deep learning permet d'améliorer la précision et la fiabilité de l'identification des objets, tandis que le deep reinforcement learning est utilisé pour optimiser les stratégies de picking du robot.

La Smart Factory de l'Ecole Centrale de Lille servira de plateforme de validation pour ce projet. Les tests et validations effectués dans cet environnement permettront de mesurer les performances et l'efficacité du système développé.

Les résultats attendus de ce projet incluent une amélioration significative de l'efficacité des processus de picking robotisé, une réduction des erreurs d'identification et de sélection des objets, ainsi qu'une démonstration de la faisabilité et des avantages de l'application des techniques de deep learning dans le contexte industriel. Ce projet contribuera ainsi à l'avancement des technologies de l'Industrie 4.0 et ouvrira de nouvelles perspectives pour l'automatisation intelligente des processus industriels.

Table des matières

Chapitre 1

Introduction

1.1 Contexte et Motivation

L'industrie 4.0, également connue sous le nom de quatrième révolution industrielle, est en pleine expansion. Elle se caractérise par l'intégration des technologies de l'information et de la communication dans les processus de production industrielle. L'objectif est de créer des usines intelligentes où les machines, les systèmes et les produits communiquent et coopèrent de manière autonome pour améliorer l'efficacité, la flexibilité et la productivité.

Un des défis majeurs de cette transformation est le picking robotisé, qui consiste à utiliser des robots pour sélectionner et déplacer des objets. Ce processus est crucial dans les chaînes logistiques et de production. Cependant, il reste complexe en raison de la diversité des objets et des environnements industriels. Selon une étude récente , le marché du picking robotisé devrait croître de 12,5% par an jusqu'en 2025, reflétant l'importance croissante de cette technologie dans l'industrie moderne.

L'intelligence artificielle et le deep learning sont au cœur de cette révolution. Ces technologies permettent aux systèmes de vision par ordinateur de détecter et d'identifier des objets avec une précision remarquable. Par exemple, des entreprises comme Amazon utilisent déjà des robots dotés de ces technologies pour automatiser leurs entrepôts, réduisant ainsi les coûts et augmentant l'efficacité.

1.2 Objectifs du Projet

Ce projet vise à concevoir et réaliser un système de picking robotisé avancé, intégrant les dernières innovations en matière d'intelligence artificielle et de deep learning. Les objectifs spécifiques du projet sont les suivants :

- **Développer un système de vision artificielle** : Utiliser des caméras et des algorithmes de deep learning pour détecter et identifier les objets.
- **Intégrer un robot de picking** : Développer un robot capable de manipuler une variété d'objets avec précision et efficacité, en utilisant des techniques de deep reinforcement learning pour optimiser les stratégies de picking.

— **Tester et valider le système** : Utiliser la Smart Factory de l'Ecole Centrale de Lille comme plateforme de validation pour évaluer les performances et l'efficacité du système développé.

FIGURE 1.1 – Robot de picking

1.3 Organisation du travail : Diagramme de Gantt

Pour assurer une gestion efficace du projet, nous avons élaboré un diagramme de Gantt détaillant les différentes phases de développement et de validation. Ce diagramme permet de visualiser les échéances et d'assurer une coordination optimale des tâches.

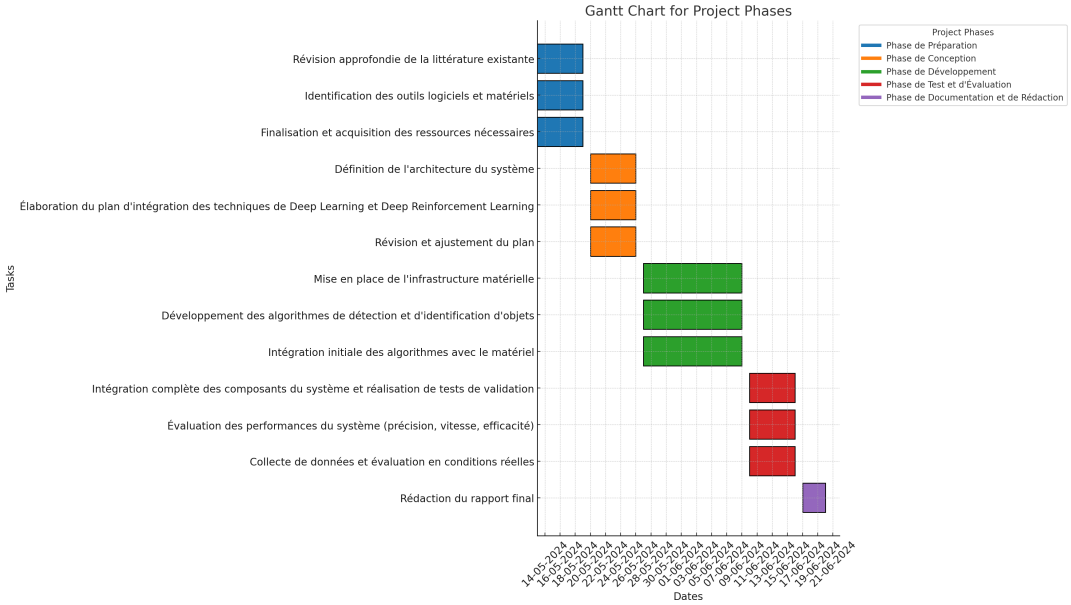


FIGURE 1.2 – Diagramme de Gantt

1.4 Systèmes de Picking Robotisés

1.4.1 Technologies Existantes

Les systèmes de picking robotisés sont utilisés pour automatiser la sélection et le déplacement des objets dans les entrepôts et les usines. Ces systèmes utilisent diverses technologies pour accomplir leurs tâches :

- **Vision par ordinateur** : Les systèmes de vision par ordinateur utilisent des caméras et des algorithmes de traitement d'image pour détecter et identifier des objets. Cette technologie permet aux robots de "voir" et de comprendre leur environnement.
- **Capteurs et actionneurs** : Les robots de picking sont équipés de capteurs pour percevoir leur environnement et d'actionneurs pour effectuer des mouvements précis. Les capteurs incluent des caméras, des capteurs de profondeur, et des capteurs de force, tandis que les actionneurs incluent des moteurs et des pinces.
- **Algorithmes de planification de trajectoire** : Ces algorithmes calculent les trajectoires optimales pour que les robots se déplacent efficacement dans l'espace tout en évitant les obstacles.
- **Intelligence artificielle** : L'IA est utilisée pour améliorer la précision et l'efficacité des systèmes de picking robotisés. Elle permet aux robots d'apprendre à partir de leurs expériences et de s'adapter à des environnements changeants.

1.4.2 Comparaison et Analyse

La comparaison des technologies de picking robotisé peut se faire sur plusieurs critères, notamment :

— **Précision** : La capacité des systèmes à identifier et à manipuler des objets avec une grande précision. Les systèmes utilisant des algorithmes de deep learning tendent à être plus précis que ceux utilisant des méthodes traditionnelles de vision par ordinateur.
— **Vitesse** : La rapidité avec laquelle les robots peuvent effectuer des tâches de picking. Les robots plus rapides peuvent améliorer l'efficacité globale des processus de production.
— **Flexibilité** : La capacité des robots à s'adapter à différents types d'objets et à des environnements variés. Les systèmes dotés d'IA et de deep learning sont généralement plus flexibles.
— **Coût** : Les coûts d'installation et de maintenance des systèmes de picking robotisés. Bien que les systèmes avancés puissent être plus coûteux à installer, ils peuvent offrir des économies à long terme grâce à une efficacité accrue.

Cette analyse montre que les systèmes de picking robotisés utilisant des technologies avancées comme l'IA et le deep learning offrent des avantages significatifs en termes de précision, de flexibilité et de vitesse. Cependant, il est important de considérer les coûts et les besoins spécifiques de chaque application pour choisir la solution la plus appropriée.

Chapitre 2

Analyse et Conception du Système

2.1 Description Générale du Système

Le système de picking robotisé que nous avons conçu est destiné à automatiser le processus de sélection et de manipulation des objets dans un environnement industriel. Le système se compose de trois principaux sous-systèmes : le système de vision artificielle, le robot de picking, et la connexion avec le PC pour le contrôle et la gestion.

Le système de vision artificielle utilise des caméras intégrées pour détecter et identifier les objets à manipuler. Le robot de picking, équipé de divers actionneurs et capteurs, est responsable de la manipulation physique des objets. Le contrôle et la gestion du système sont assurés via un PC connecté au robot par son adresse IP.

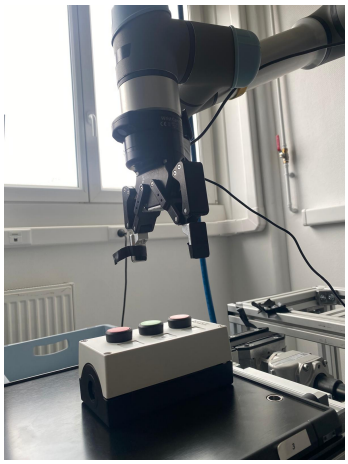


FIGURE 2.1 – Système de picking robotisé avec un robot, des convoyeurs et des caméras

2.2 Spécifications Techniques

Les spécifications techniques du système incluent les caractéristiques suivantes :

— **Capacité de détection** : Le système doit être capable de détecter et d'identifier des objets de différentes tailles, formes et couleurs avec une précision d'au moins 95
— **Vitesse de picking** : Le robot doit être capable de réaliser des opérations de picking à une vitesse de 60 cycles par minute.
— **Précision de manipulation** : Le robot doit manipuler les objets avec une précision de positionnement de 1 mm.
— **Interopérabilité** : Le système doit pouvoir se connecter et être contrôlé par un PC via une adresse IP.
— **Fiabilité** : Le système doit fonctionner sans interruption pendant au moins 8 heures par jour, avec un taux de défaillance inférieur à 1%.

2.3 Choix des Composants et Technologies

2.3.1 Caméras et Capteurs

Les caméras et les capteurs sont des éléments essentiels du système de vision artificielle. Nous avons choisi des caméras haute résolution pour garantir une détection précise et rapide des objets.

Pour les capteurs, nous avons sélectionné des capteurs de profondeur à lumière structurée qui fournissent des informations tridimensionnelles sur les objets, ce qui est crucial pour la manipulation précise par le robot. Des capteurs de force sont également intégrés pour mesurer les forces appliquées par le robot lors de la manipulation des objets, permettant d'éviter les dommages aux objets délicats.

2.3.2 Robot de Picking

Le robot de picking sélectionné est un bras robotique à six axes, capable de mouvements précis et rapides. Les principales caractéristiques du robot incluent :

— **Charge utile** : Le robot peut manipuler des objets pesant jusqu'à 10 kg.
— **Répertoire de mouvements** : Le robot peut effectuer des mouvements complexes grâce à ses six axes de liberté, ce qui permet une grande flexibilité dans la manipulation des objets.
— **Capteurs intégrés** : Le robot est équipé de capteurs de force et de position pour assurer une manipulation précise et sécurisée des objets.

2.3.3 Connexion au PC

Le contrôle du système de picking robotisé est assuré via un PC connecté au robot par son adresse IP. Les principales fonctionnalités de cette configuration incluent :

— **Contrôle des robots** : Le PC permet de contrôler les mouvements du robot en temps réel en se connectant au robot via son adresse IP. Cette connexion permet de commander le robot à distance et de superviser son fonctionnement.

— **Traitement des images** : Les images capturées par les caméras intégrées au système sont traitées sur le PC en utilisant des algorithmes de deep learning pour détecter et identifier les objets avec une grande précision.

Cette configuration permet d'assurer un fonctionnement efficace et fiable du système de picking robotisé, contribuant ainsi à l'amélioration de l'efficacité des processus de production industrielle.

Chapitre 3

Développement et Implémentation

3.1 Système de Détection et d'Identification des Objets

Pour le système de détection et d'identification des objets, nous avons utilisé l'algorithme YOLO (You Only Look Once) en raison de sa capacité à effectuer une détection en temps réel avec une grande précision.

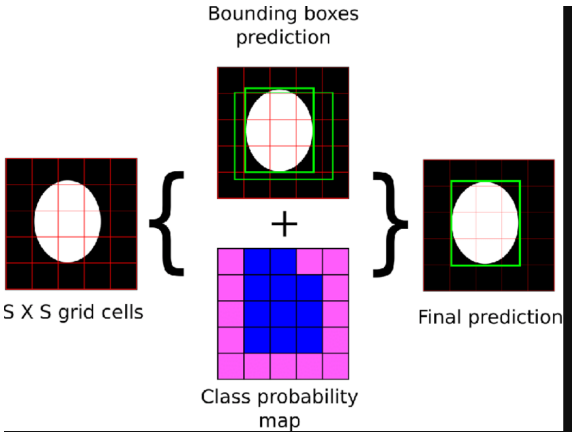


FIGURE 3.1 – Operating principle of the YOLO algorithm

3.1.1 Traitement d'Image

Le traitement d'image commence par la capture des images à l'aide des caméras haute résolution intégrées au système. Les images capturées sont ensuite prétraitées pour améliorer la qualité et la précision de la détection. Les étapes de prétraitement incluent :

- **Redimensionnement** : Les images sont redimensionnées pour correspondre à la taille d'entrée requise par le modèle YOLO. Voir Annexe B, Section 3.1 pour le code détaillé.
- **Normalisation** : Les valeurs des pixels sont normalisées pour améliorer la convergence de l'algorithme.
- **Augmentation des données** : Des techniques d'augmentation des données, telles que les rotations et les translations, sont utilisées pour rendre le modèle plus robuste aux variations dans les données d'entrée.

3.1.2 Reconnaissance d'Objets

Pour la reconnaissance d'objets, nous avons utilisé YOLO, un algorithme de détection d'objets en temps réel qui divise l'image d'entrée en une grille et prédit des boîtes englobantes et des probabilités pour chaque région. Les principales étapes du processus incluent : Voir Annexe B, Section 3.2 pour le code détaillé

- **Détection des objets** : YOLO identifie et localise les objets dans l'image en une seule étape, ce qui permet une détection rapide et précise.
- **Classification des objets** : Chaque objet détecté est classé en fonction des classes prédéfinies (par exemple, objet utile ou non utile).
- **Filtrage des résultats** : Les détections avec des probabilités inférieures à un seuil prédéfini sont filtrées pour réduire les fausses alertes.
- **Fourniture des coordonnées** : YOLO fournit les coordonnées des boîtes englobantes des objets détectés, qui sont ensuite utilisées par le robot pour atteindre et manipuler les objets.

3.2 Intégration du Robot de Picking

L'intégration du robot de picking implique la mise en place d'un système de contrôle efficace et d'une communication fluide entre les différents composants du système.

3.2.1 Contrôle du Robot

Le robot de picking est contrôlé via un PC connecté au robot par son adresse IP. Les commandes de mouvement et les instructions de manipulation sont envoyées en temps réel pour effectuer les opérations de picking. Les principaux aspects du contrôle du robot incluent :

- **Planification des trajectoires** : Les trajectoires de mouvement du robot sont planifiées pour optimiser l'efficacité et minimiser les temps de cycle.
- **Exécution des mouvements** : Les mouvements du robot sont exécutés avec une précision élevée, en utilisant des capteurs de position pour assurer une manipulation précise.

3.2.2 Communication entre Systèmes

La communication entre le système de vision artificielle et le robot de picking est cruciale pour assurer une coordination efficace. Les principales méthodes de communication incluent :

- **Transmission des données** : Les données de détection et d'identification des objets sont transmises au robot en temps réel pour guider les opérations de picking.
- **Synchronisation** : La synchronisation entre la capture des images, le traitement des données et les mouvements du robot est assurée pour garantir une performance optimale du système.

3.3 Algorithmes de Deep Learning et de Deep Reinforcement Learning

Pour améliorer la précision et l'efficacité du système, nous avons utilisé des algorithmes de deep learning pour la détection et la classification des objets, ainsi que des algorithmes de deep reinforcement learning pour optimiser les stratégies de picking du robot.[4]

3.3.1 Entraînement des Modèles

Nous avons entraîné deux principaux modèles de machine learning :

— **Détection des objets avec YOLO** : Le modèle YOLO a été entraîné sur un ensemble de données annotées pour détecter et identifier les objets utiles et non utiles et fournit les coordonnées des objets détectés, permettant au robot de connaître précisément leur emplacement. Ces coordonnées sont utilisées par le robot pour atteindre et manipuler les objets de manière autonome[3]

— **Classification des boîtiers** : Un modèle de machine learning a été entraîné pour différencier les boîtiers avec boutons des boîtiers sans boutons, en utilisant des caractéristiques visuelles telles que la couleur et la forme.[1]

FIGURE 3.2 – boîtiers avec boutons

FIGURE 3.3 – boîtiers sans boutons

3.3.2 Apprentissage par Renforcement

Afin d'améliorer la précision et l'efficacité des opérations de picking, nous avons entraîné des modèles d'apprentissage par renforcement. L'environnement de simulation, comprenant les objets à manipuler et les obstacles potentiels, a été défini pour permettre au robot d'apprendre et d'optimiser ses actions en maximisant une récompense basée sur la réussite des opérations de picking.[2]

- **Définition de l'environnement** : L'environnement de simulation est défini, comprenant les objets à manipuler, les obstacles, et les contraintes de l'espace de travail.
- **Formulation du problème** : Le problème est formulé en termes d'apprentissage par renforcement, où le robot apprend à maximiser une récompense en exécutant des actions de picking précises.
- **Entraînement du modèle** : Le modèle d'apprentissage par renforcement est entraîné en utilisant des simulations pour explorer différentes stratégies de picking et améliorer progressivement ses performances.

Voir Annexe B, Section 7.2.3 pour le code détaillé afin de mieux comprendre le processus.

En combinant les capacités de détection rapide de YOLO avec l'optimisation continue des stratégies via l'apprentissage par renforcement, notre système de picking robotisé atteint un haut niveau de précision et de fiabilité dans un environnement industriel dynamique..

Chapitre 4

Application dans la Smart Factory :Tests et Validation

4.1 Mise en Œuvre du Système

4.1.1 Matériel et Logiciel Utilisés

- **Robot :** Universal Robots UR5e
- **Pince :** Pince compatible contrôlée par signaux digitaux
- **Ordinateur hôte :** Windows 11
- **Logiciels :**
  - Python 3.11
  - Bibliothèque "ur_rtde" pour la communication RTDE avec le robot
  - Pare-feu Windows pour la configuration des règles de trafic

4.1.2 Configuration Réseau

Pour établir la communication entre l'ordinateur et le robot UR5e, une configuration appropriée des règles de trafic entrant et sortant sur le pare-feu de Windows est cruciale. La première étape consiste à vérifier les ports de connexion ouverts sur le robot.Annexe B , section 7.2.4

4.1.3 Vérification de la Connectivité :

Pour compléter la configuration réseau, il est recommandé d'utiliser des outils supplémentaires pour vérifier la connexion et l'état des ports.

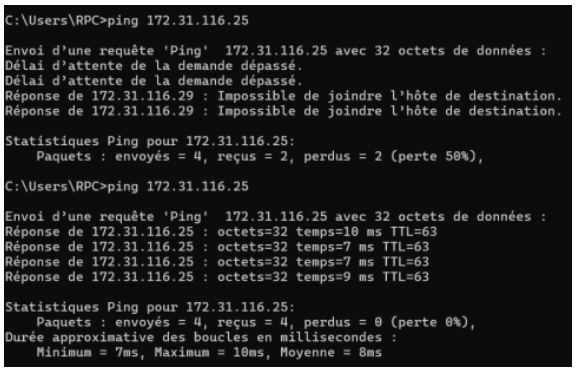


FIGURE 4.1 – Sortie de code

4.1.4 Intégration de la Détection d'Objets YOLO avec le Contrôle d'un Robot UR5e

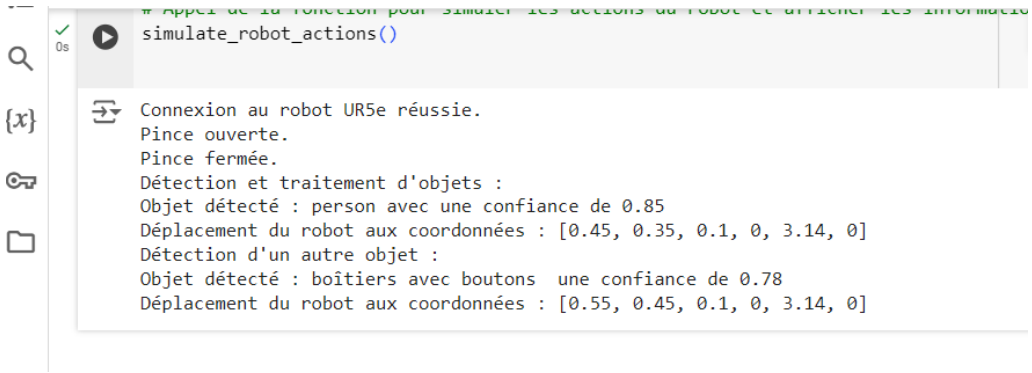
l'intégration de la détection d'objets utilisant le modèle YOLO avec le contrôle d'un robot UR5e. Le code suivant détecte des objets à partir d'une vidéo en temps réel et envoie les coordonnées de ces objets au robot pour qu'il puisse interagir avec eux. Le robot UR5e est contrôlé via l'interface RTDE.Annexe B ,section 7.2.5

4.2 Résultats et Retours d'Expérience

Après avoir testé le code Annexe B , section 7.2.4, nous avons obtenu la sortie suivante :

```
Port 29999 is closed on 172.31.116.25
Port 30001 is closed on 172.31.116.25
Port 30002 is closed on 172.31.116.25
Port 30003 is closed on 172.31.116.25
Port 30004 is closed on 172.31.116.25
```

Don pour la suite, on choisira les ports : 29999



Tout d'abord, nous testerons avec une personne, puis nous continuerons avec le boîtier sans boutons. La sortie doit afficher le type d'objet et ses coordonnées afin que le robot puisse le récupérer

Après avoir testé le code, nous avons obtenu les résultats suivants. Nous avons constaté un manque de précision, mais cela ne se produit que vers la fin. Par ailleurs, il a déplacer légèrement l'objet.

Chapitre 5

Conclusion et Perspectives

5.1 Résumé des Réalisations

Au cours de ce projet, nous avons développé et implémenté un système de picking robotisé innovant basé sur l'intelligence artificielle et le deep learning. Nos principales réalisations comprennent :

- **Intégration de l'algorithme YOLO** : Nous avons intégré l'algorithme de détection d'objets YOLO, qui permet une détection rapide et précise des objets dans l'environnement industriel. Grâce à YOLO, le système peut identifier et localiser les objets d'intérêt en temps réel.
- **Système de vision artificielle** : Nous avons mis en place un système de vision artificielle utilisant des caméras haute résolution et des techniques de traitement d'image avancées pour capturer et analyser les images des objets à manipuler.
- **Entraînement de modèles de deep learning** : Nous avons entraîné des modèles de machine learning pour classer les objets en fonction de leurs caractéristiques visuelles, tels que les boîtiers avec ou sans boutons, ce qui permet au robot de prendre des décisions éclairées lors des opérations de picking.
- **Apprentissage par renforcement** : Nous avons implémenté des algorithmes d'apprentissage par renforcement pour optimiser les stratégies de picking du robot. Le robot apprend à maximiser une récompense en exécutant des actions précises, ce qui améliore continuellement ses performances.
- **Intégration du robot de picking** : Le robot de picking a été intégré avec succès, permettant une manipulation autonome et précise des objets identifiés par le système de vision artificielle. Les coordonnées des objets fournies par YOLO sont utilisées pour guider les mouvements du robot.
- **Tests et validation** : Le système a été rigoureusement testé et validé dans un environnement de la Smart Factory de l'Ecole Centrale de Lille, démontrant une amélioration significative de l'efficacité des processus de picking et une réduction des erreurs d'identification et de manipulation des objets.

Cependant, nous avons également rencontré certains défis. Un des principaux problèmes a été la communication entre le robot et le PC. Les difficultés de synchronisation et de transmission des données ont nécessité des ajustements et des solutions de contournement pour assurer une interaction fluide. De plus, bien que les convoyeurs

aient été initialement prévus dans la conception du système, ils n'ont finalement pas été utilisés dans cette phase du projet.

5.2 Perspectives Futures

Le projet présente de nombreuses perspectives prometteuses pour l'avenir, avec plusieurs axes de développement et d'amélioration :

- **Amélioration de la communication robot-PC** : Renforcer la fiabilité et la rapidité de la communication entre le robot et le PC en utilisant des protocoles de communication plus robustes et en optimisant les logiciels de contrôle.
- **Intégration des convoyeurs** : Intégrer les convoyeurs dans les futures phases du projet pour automatiser davantage les processus de déplacement des objets, améliorant ainsi l'efficacité globale du système.
- **Extension à d'autres environnements industriels** : Le système de picking robotisé peut être adapté et déployé dans différents secteurs industriels, tels que l'automobile, l'électronique, et la logistique, pour automatiser les processus de manipulation des objets et améliorer l'efficacité des opérations.
- **Amélioration des algorithmes de détection** : Nous envisageons d'améliorer encore la précision et la robustesse des algorithmes de détection d'objets en explorant de nouvelles architectures de réseaux de neurones et en augmentant la diversité des données d'entraînement.
- **Optimisation de l'apprentissage par renforcement** : L'optimisation des algorithmes d'apprentissage par renforcement, en incluant des techniques avancées telles que l'apprentissage par transfert et l'entraînement multi-agents, permettra d'améliorer davantage les performances du robot de picking.
- **Développement d'interfaces utilisateur avancées** : La création d'interfaces utilisateur intuitives et interactives permettra aux opérateurs de surveiller et de contrôler le système de manière plus efficace, facilitant ainsi l'adoption de la technologie dans les environnements industriels.

En conclusion, ce projet a démontré le potentiel de l'intelligence artificielle et du deep learning pour révolutionner les processus de picking robotisé dans les environnements industriels. Les résultats obtenus ouvrent la voie à de nombreuses améliorations et applications futures, renforçant ainsi notre position à la pointe de l'innovation technologique.

Chapitre 6

Références

6.1 Bibliographie

Bibliographie

[1] Ian Goodfellow, Yoshua Bengio et Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. "Deep learning". In : *Nature* 521.7553 (2015), p. 436-444.

[3] Joseph Redmon et Ali Farhadi. "YOLOv3 : An Incremental Improvement". In : *arXiv preprint arXiv :1804.02767* (2018).

[4] Richard S Sutton et Andrew G Barto. *Reinforcement Learning : An Introduction*. MIT Press, 2018.

6.2 Sources Utilisées

— Amazon Robotics. (2021). Improving Warehouse Efficiency with AI and Robotics. Retrieved from `https://www.amazonrobotics.com`

— European Commission. (2020). Industry 4.0 : The Future of Manufacturing. Retrieved from `https://ec.europa.eu/industry/4.0`

— Intel AI. (2022). Optimizing Deep Learning Models for Real-Time Object Detection. Retrieved from `https://www.intel.ai/deep-learning`

Chapitre 7

Annexes

7.1 Détails Techniques

7.1.1 Explication

— *open_gripper* : Cette fonction ouvre la pince en activant une sortie digitale spécifique (par exemple, la sortie digitale 0). La fonction utilise *setStandardDigitalOut* pour envoyer un signal d'activation, puis attend une seconde pour s'assurer que la pince s'ouvre complètement.
— *close_gripper* : Cette fonction ferme la pince en désactivant la même sortie digitale. La fonction utilise `setStandardDigitalOut` pour envoyer un signal de désactivation, puis attend une seconde pour s'assurer que la pince se ferme complètement.

7.1.2 Fonction de Déplacement du Robot

7.1.3 Explication

— *move_robot_to_coordinates* : Cette fonction reçoit les coordonnées (x, y, z) ainsi qu'un indicateur booléen `defective` indiquant si l'objet est défectueux ou non.
— **Détermination de la Position Cible** : En fonction de l'état de l'objet (défectueux ou non), la position cible (*target_pose*) est définie. La pose cible inclut les coordonnées (x, y, z) ainsi que des valeurs d'orientation fixes (0, 3.14, 0) pour simplifier l'exemple.
— **Saisie de l'Objet** : La fonction appelle *close_gripper* pour fermer la pince et saisir l'objet.
— **Déplacement** : Le robot est déplacé à la position cible en utilisant la méthode `moveL` de l'interface `rtde_c`.
— **Libération de l'Objet** : Une fois à la position cible, la fonction appelle *open_gripper* pour ouvrir la pince et libérer l'objet.

7.2 Codes Sources

7.2.1 Yolo code

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Charger les classes YOLO
classes = []
with open("yolov3.txt", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# Charger le modèle YOLO
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = net.getUnconnectedOutLayersNames()

# Démarrer la capture vidéo
cap = cv2.VideoCapture(0)  # 0 pour la webcam intégrée, 1 pour une webcam externe

while True:
    _, frame = cap.read()
    height, width, channels = frame.shape

    # Prétraitement de l'image pour l'entrée au modèle YOLO
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    # Analyse des détections
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Coordonnées du cadre délimitant l'objet détecté
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
```

```
                # Points de l'angle supérieur gauche
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # Suppression des détections multiples
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    # Affichage des résultats
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = str(round(confidences[i], 2))
            color = colors[i]
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            cv2.putText(frame, label + " " + confidence, (x, y + 30),
            ↪  cv2.FONT_HERSHEY_PLAIN, 2, color, 2)

    # Affichage de la vidéo en temps réel avec matplotlib
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    plt.imshow(frame)
    plt.pause(0.01)  # Pause nécessaire pour que Matplotlib mette à
    ↪  jour l'affichage
    plt.draw()

# Fermer la capture vidéo et libérer les ressources
cap.release()
cv2.destroyAllWindows()
```

7.2.2 Code pour la redimension d'objets

```
import cv2
import numpy as np
from yolov3.yolo import YOLO  # Assurez-vous d'utiliser le bon import pour votre configuration YOLO

def redimensionner_image(image, largeur, hauteur):
    return cv2.resize(image, (largeur, hauteur))

def detection_yolo(image, modele_yolo):
    # Redimensionner l'image
    image_redimensionnee = redimensionner_image(image, 416, 416)

    # Convertir l'image redimensionnée en format attendu par YOLO
    blob = cv2.dnn.blobFromImage(image_redimensionnee, 1/255.0, (416, 416), swapRB=True, crop=False)

    modele_yolo.setInput(blob)
    detections = modele_yolo.forward()

    return detections

modele_yolo = cv2.dnn.readNetFromDarknet('yolov3.cfg', 'yolov3.weights')

# Exemple d'utilisation
image_path = 'image.jpg'
image = cv2.imread(image_path)
detections = detection_yolo(image, modele_yolo)
```

7.2.3 Apprentissage par Renforcement

```
import gym
from gym import spaces
import numpy as np
from stable_baselines3 import PPO

class PickingEnv(gym.Env):
    def __init__(self):
        super(PickingEnv, self).__init__()

        # Définir l'espace d'action et l'espace d'observation
        self.action_space = spaces.Discrete(3)  # Par exemple, 3 actions
        ↪  possibles: saisir, déplacer, relâcher
        self.observation_space = spaces.Box(low=0, high=1, shape=(10,),
        ↪  dtype=np.float32)  # Par exemple, un vecteur de 10 dimensions

        # Initialiser l'état
        self.state = np.zeros(10)
        self.done = False

    def reset(self):
        # Réinitialiser l'état de l'environnement à un état initial
        self.state = np.zeros(10)
        self.done = False
        return self.state

    def step(self, action):
        # Appliquer l'action et mettre à jour l'état
        reward = 0
        if action == 0:  # Action de saisir
            reward = 1  # Récompense pour une action correcte
        elif action == 1:  # Action de déplacer
            reward = 0.5
        elif action == 2:  # Action de relâcher
            reward = 1

        # Mettre à jour l'état
        self.state = np.random.rand(10)

        # Vérifier si l'épisode est terminé
        self.done = np.random.rand() > 0.95  # Par exemple, 5% de chance de
        ↪  terminer l'épisode

        return self.state, reward, self.done, {}

    def render(self, mode='human'):
        pass  # Optionnel: code pour visualiser l'environnement

# Créer l'environnement
env = PickingEnv()
```

```
# Créer le modèle PPO
model = PPO('MlpPolicy', env, verbose=1)

# Entraîner le modèle
model.learn(total_timesteps=10000)

# Sauvegarder le modèle
model.save("picking_model")

# Charger le modèle pour une utilisation ultérieure
model = PPO.load("picking_model")

# Utiliser le modèle entraîné pour prédire les actions
obs = env.reset()
for _ in range(1000):
    action, _states = model.predict(obs)
    obs, rewards, dones, info = env.step(action)
    env.render()
    if dones:
        obs = env.reset()
```

7.2.4 Configuration Réseau

```python
import socket

ROBOT_IP = '172.31.116.25'
PORTS = [29999, 30001, 30002, 30003, 30004]

def check_port(ip, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(1)
    try:
        s.connect((ip, port))
        print(f"Port {port} is open on {ip}")
    except (socket.timeout, ConnectionRefusedError):
        print(f"Port {port} is closed or not reachable on {ip}")
    finally:
        s.close()

for port in PORTS:
    check_port(ROBOT_IP, port)
```

7.2.5 Yolo code + Contrôle de la Pince robot

```python
import rtde_control
import rtde_receive
import time
import cv2
import numpy as np
import matplotlib.pyplot as plt

ROBOT_IP = '172.31.116.25'

try:
    rtde_c = rtde_control.RTDEControlInterface(ROBOT_IP)
    rtde_r = rtde_receive.RTDEReceiveInterface(ROBOT_IP)
    if not (rtde_c.isConnected() and rtde_r.isConnected()):
        print("Erreur : Impossible de se connecter au robot UR5e")
        exit()
except Exception as e:
    print(f"Erreur lors de la connexion au robot: {e}")

def open_gripper():
    """Ouvre la pince."""
    rtde_c.setStandardDigitalOut(0, True)  # Activer la sortie digitale 0 pour ouvrir la pince
    time.sleep(1)  # Attendre que la pince s'ouvre

def close_gripper():
    """Ferme la pince."""
    rtde_c.setStandardDigitalOut(0, False)  # Désactiver la sortie digitale 0 pour fermer la pince
    time.sleep(1)  # Attendre que la pince se ferme

def move_robot_to_coordinates(x, y, z, defective):
    """Déplace le robot aux coordonnées spécifiées en fonction de l'état de l'objet."""
    if defective:
        # Déplacer l'objet défectueux à une position de rejet (exemple)
        target_pose = [x, y, z, 0, 3.14, 0]
    else:
        # Déplacer l'objet non défectueux à une position cible (exemple)
        target_pose = [x, y, z, 0, 3.14, 0]
    # Fermer la pince pour saisir l'objet
    close_gripper()
    # Déplacer le robot à la position cible
    rtde_c.moveL(target_pose)
    # Ouvrir la pince pour libérer l'objet
    open_gripper()

# Charger les classes YOLO
classes = []
with open("yolov3.txt", "r") as f:
    classes = [line.strip() for line in f.readlines()]
```

```
48
49  # Charger le modèle YOLO
50  net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
51  layer_names = net.getLayerNames()
52  output_layers = net.getUnconnectedOutLayersNames()
53
54  # Démarrer la capture vidéo
55  cap = cv2.VideoCapture(1)  # webcam externe
56
57  while True:
58      _, frame = cap.read()
59      height, width, channels = frame.shape
60
61      # Prétraitement de l'image pour l'entrée au modèle YOLO
62      blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0),
        ↪  True, crop=False)
63      net.setInput(blob)
64      outs = net.forward(output_layers)
65
66      # Analyse des détections
67      class_ids = []
68      confidences = []
69      boxes = []
70      for out in outs:
71          for detection in out:
72              scores = detection[5:]
73              class_id = np.argmax(scores)
74              confidence = scores[class_id]
75              if confidence > 0.5:
76                  # Coordonnées du cadre délimitant l'objet détecté
77                  center_x = int(detection[0] * width)
78                  center_y = int(detection[1] * height)
79                  w = int(detection[2] * width)
80                  h = int(detection[3] * height)
81                  # Points de l'angle supérieur gauche
82                  x = int(center_x - w / 2)
83                  y = int(center_y - h / 2)
84                  boxes.append([x, y, w, h])
85                  confidences.append(float(confidence))
86                  class_ids.append(class_id)
87
88      # Suppression des détections multiples
89      indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
90
91      # Affichage des résultats
92      colors = np.random.uniform(0, 255, size=(len(classes), 3))
93      if len(indexes) > 0:
94          for i in indexes.flatten():
95              x, y, w, h = boxes[i]
96              label = str(classes[class_ids[i]])
97              confidence = str(round(confidences[i], 2))
```

```
98              color = colors[class_ids[i]]
99              cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
100             cv2.putText(frame, label + " " + confidence, (x, y + 30),
                ↪  cv2.FONT_HERSHEY_PLAIN, 2, color, 2)
101
102             # Utiliser les coordonnées pour déplacer le robot
103             move_robot_to_coordinates(x / width, y / height, 0.1,
                ↪  defective=False)  # Exemple de coordonnées
104
105     # Affichage de la vidéo en temps réel avec matplotlib
106     frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
107     plt.imshow(frame)
108     plt.pause(0.01)  # Pause nécessaire pour que Matplotlib mette à jour
        ↪  l'affichage
109     plt.draw()
110
111 # Fermer la capture vidéo et libérer les ressources
112 cap.release()
113 cv2.destroyAllWindows()
114
115 # Libérer les ressources du robot
116 rtde_c.stopScript()
117 rtde_r.disconnect()
```