École Nationale Supérieure d'Arts et Métiers

Compte Rendu

PJT

Robot Ping-pong

EL jakani youssef
Jarane aymen
Laroussi Amine
Lazar Mohamed

Enseignant :
Abdelmajid cheriffi

Table des matières

Table des figures

Liste des tableaux

Listings

I Introduction

1 Présentation générale du projet

Dans le cadre de l'unité d'enseignement PJT, nous avons été amenés à concevoir et développer un robot autonome capable de détecter, saisir et transporter des balles de ping-pong. Ce projet s'inscrit dans une démarche d'apprentissage pratique de la robotique, de l'électronique embarquée, et de la vision par ordinateur, en mobilisant à la fois des compétences techniques et méthodologiques.

2 Contexte et motivations

La robotique de service, notamment dans le domaine du sport et des loisirs, connaît un essor considérable. Le ramassage manuel des balles de ping-pong lors des entraînements représente une tâche fastidieuse et répétitive pour les joueurs. Automatiser ce processus présente un intérêt certain, tant pour le confort des utilisateurs que pour l'optimisation du temps d'entraînement. Ce contexte nous a motivés à concevoir un robot autonome, simple et efficace, pour effectuer cette tâche.

3 Objectifs du projet

L'objectif principal de ce projet est de réaliser un robot capable de :

- Détecter une balle de ping-pong grâce à une caméra intelligente et des algorithmes de vision.
- S'approcher de la balle de manière autonome
- saisir la balle à l'aide d'un bras robotisé.
- Revenir à une position de base une fois la balle récupérée, prêt à relancer le cycle.

4 Aperçu du fonctionnement du robot

Le robot suit un cycle de fonctionnement simple : il effectue une patrouille entre deux points fixes (positions A et B) à la recherche d'une balle. Dès qu'une balle jaune est détectée par la caméra, un module de traitement prend le relais pour centrer la balle dans le champ de vision. Le bras robotisé est ensuite activé pour effectuer la saisie de la balle. Une fois la balle récupérée, le robot retourne à sa position initiale et se prépare à recommencer le cycle.

II Conception du robot

1 Architecture générale du robot

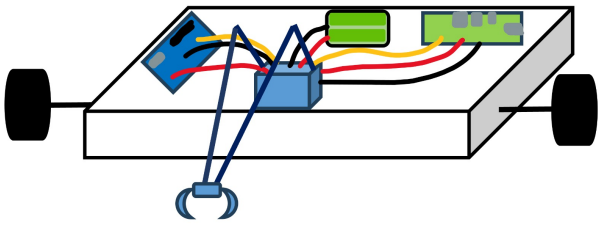

FIGURE II.1 – robot ramasseur de balles ping-pong

Le robot a été conçu selon une architecture modulaire, structurée autour de plusieurs sous-systèmes fonctionnels qui collaborent pour accomplir la mission principale : localiser, récupérer et transporter une balle de ping-pong de manière autonome. Chaque module est spécialisé dans une tâche précise, ce qui permet une meilleure organisation du développement, une facilité de test unitaire et une évolutivité du système en cas de modifications ou d'améliorations futures.

L'architecture repose sur quatre modules principaux, interconnectés à travers un système central de traitement :

- **Le module de perception** : Ce module est chargé de détecter la présence des balles de ping-pong dans l'environnement du robot. Il extrait des informations visuelles comme la position (coordonnées spatiales) de la balle par rapport au champ de vision du robot. Cette détection repose sur des technologies de vision, qui peuvent être comparées et optimisées selon différents critères (temps réel, précision, coût, etc.).
- **Le module de décision** : C'est le cerveau logique du robot. Il exploite les données fournies par la perception pour planifier et coordonner les actions du robot. Il décide notamment de la direction à suivre, de la nécessité d'activer le bras robotisé, ou encore du retour à une position initiale après saisie de la balle. Ce module exécute l'algorithme global de gestion des tâches du robot.
- **Le module de locomotion** : Il permet au robot de se déplacer dans l'environnement de manière contrôlée et réactive. À partir des instructions du module de décision, il gère la propulsion et l'orientation du robot en s'appuyant sur des moteurs et un système de pilotage adapté. La mobilité est essentielle pour ajuster dynamiquement la position du robot en fonction de la localisation de la balle.
- **Le module de manipulation** : Ce module intervient une fois la balle détectée et la position optimale atteinte. Il est composé d'un système mécanique articulé (généralement un bras robotisé) permettant de saisir la balle, de la stabiliser et de la transporter. Ce module nécessite une synchronisation fine avec les modules de perception et de décision pour assurer une prise précise.

Tous ces modules sont coordonnés par une unité centrale de traitement, qui gère la communication entre les sous-systèmes, la gestion des priorités, et le déclenchement des différentes actions selon une logique d'automatisation définie en amont. L'avantage de cette

organisation réside dans sa flexibilité : chaque module peut être développé, amélioré ou remplacé indépendamment des autres. Par exemple, il est possible de tester différentes solutions de vision (comme une caméra Pixy2 ou une caméra classique avec traitement via deep learning) sans modifier l'ensemble du système. Cette approche modulaire est donc parfaitement adaptée à un projet d'ingénierie évolutif et expérimental

2 Choix des composants

Le choix des composants matériels s'est inscrit dans une démarche d'ingénierie rationnelle, alignée sur l'architecture modulaire du robot. Cette approche, fondée sur la séparation fonctionnelle des sous-systèmes (perception, décision, locomotion, manipulation), vise à optimiser la conception, tout en garantissant une meilleure maintenabilité et une évolutivité du système.

La sélection des composants a été guidée par une analyse multicritère, prenant en compte les paramètres suivants :

- **Performance fonctionnelle :**les composants ont été retenus pour leur capacité à répondre efficacement aux exigences propres à chaque module, en termes de précision, de temps de réponse, et de robustesse en conditions réelles.
- **Facilité d'intégration :** une attention particulière a été portée à la compatibilité des éléments entre eux, notamment vis-à-vis de la plateforme de traitement (Raspberry Pi), des langages de programmation (Python), et des bibliothèques logicielles disponibles (OpenCV, TensorFlow, etc.).
- **Maîtrise des coûts :**dans le cadre d'un projet étudiant, les solutions choisies devaient rester accessibles financièrement, tout en maintenant un niveau de qualité suffisant pour garantir un fonctionnement fiable du système.
- **Évolutivité :**les composants retenus permettent une montée en gamme fonctionnelle, notamment par l'intégration future de méthodes avancées (intelligence artificielle, détection multi-objets, navigation autonome) ou de nouveaux capteurs complémentaires.

2.1 Module de perception

Le module de perception constitue l'un des éléments fondamentaux de l'architecture du robot. Il est chargé d'extraire, en temps réel, des informations pertinentes sur l'environnement à partir d'images capturées. Ces données visuelles sont ensuite analysées afin d'identifier non seulement la présence d'une balle de ping-pong, mais également d'autres éléments clés du champ d'action du robot tels que les obstacles, les repères spatiaux, ou encore les limites de la zone de navigation. Ainsi, ce module ne se limite pas à une simple détection d'objet, mais vise à fournir une compréhension contextuelle de l'environnement, indispensable pour une prise de décision fiable et dynamique. Deux approches ont été étudiées pour l'implémentation de ce module :

Caméra Pixy2 :

La caméra Pixy2 est un capteur de vision autonome doté d'un microcontrôleur intégré, capable de réaliser une détection rapide d'objets basés sur des signatures de couleur ou des formes simples (suivi de lignes, détection de codes-barres). Elle offre une solution embarquée légère avec un traitement en local, ce qui réduit la charge sur l'unité centrale de calcul.

FIGURE II.2 – Camera pixy2

Cependant, plusieurs limitations ont été observées lors des tests :

— **Capacité de détection restreinte :** la Pixy2 ne permet pas une reconnaissance simultanée combinant couleur et forme. Elle se limite à des scénarios simples, ce qui nuit à la robustesse de la détection dans un environnement complexe ou dynamique.

— **Sensibilité aux conditions d'éclairage :** les performances chutent considérablement en cas de variation lumineuse, générant des faux positifs ou des pertes de suivi.

— **Impossibilité d'extension par apprentissage machine :**la Pixy2 ne supporte pas l'intégration de réseaux de neurones convolutifs ou de modèles de deep learning, essentiels pour des tâches de détection avancées.

— **Faible contextualisation de l'environnement :**aucune analyse de la scène globale (obstacles, murs, repères) n'est possible avec cette solution.

Caméra Microsoft LifeCam Studio

FIGURE II.3 – Microsoft LifeCam Studio

La seconde solution repose sur l'utilisation d'une caméra USB haute définition, connectée au Raspberry Pi 5, et exploitant des bibliothèques de vision par ordinateur telles que OpenCV, TensorFlow ou YOLOv5. Cette approche permet de développer un système de

détection intelligent, adaptatif et capable d'interpréter des scènes visuelles complexes. Les avantages sont nombreux :

- **Détection multi-objets :** il devient possible de détecter simultanément la balle de ping-pong, les obstacles éventuels, les zones interdites ou les points de repère (balises de position, marquages au sol).
- **Utilisation du deep learning :** des modèles entraînés sur des bases de données personnalisées peuvent être intégrés pour améliorer la fiabilité, même en présence d'éléments perturbateurs (bruits visuels, ombres, reflets).
- **Traitement contextuel :** la caméra USB permet une interprétation complète de la scène, essentielle pour le mode de balayage du robot qui analyse l'environnement avant chaque action.
- **Compatibilité avec des algorithmes de suivi (tracking) :** pour maintenir un objet en vue lors des déplacements du robot.

Le choix s'est porté sur **la caméra USB HD**, principalement pour sa polyvalence, sa compatibilité logicielle avec le Raspberry Pi, et sa capacité à supporter des traitements avancés basés sur l'intelligence artificielle.

2.2 module de décision

Le module de décision constitue le cœur logique du robot, agissant comme un système de contrôle centralisé. Son rôle est de traiter les données issues du module de perception, d'interpréter la situation et de prendre des décisions en conséquence : orientation du robot, activation du bras robotisé, ou encore gestion des états (exploration, prise, retour à la position initiale...).

Pour cela, nous avons choisi d'utiliser un **Raspberry Pi 5**, un nano-ordinateur puissant et accessible, capable d'exécuter des scripts Python complexes tout en gérant les interactions avec les différents sous-systèmes via ses ports GPIO. Le Raspberry Pi permet également l'intégration de librairies de haut niveau comme NumPy, OpenCV, et TensorFlow, ce qui rend possible l'implémentation de comportements intelligents, adaptatifs, voire basés sur l'apprentissage automatique.

Toutefois, afin d'alléger la charge de calcul du Raspberry Pi et de garantir une commande temps réel précise du bras robotisé, la gestion de celui-ci a été déléguée à une carte Arduino, plus adaptée pour le contrôle de servomoteurs via des signaux PWM. Le Raspberry Pi envoie des instructions à l'Arduino via une liaison série (UART), qui les traduit en mouvements concrets du bras.

2.3 Module de locomotion

La locomotion du robot est assurée par un système à deux roues motrices indépendantes, également appelé configuration différentielle. Chaque roue est entraînée par un moteur à courant continu (DC), commandé via le driver de moteur **ADMH2407ND**. Ce driver permet une commande fine de la vitesse et du sens de rotation grâce à un pilotage en **PWM**, tout en assurant une protection électronique des moteurs.

Le **Raspberry Pi 5**, en tant qu'unité centrale de traitement, envoie les instructions de déplacement (avancer, tourner, reculer, s'arrêter) à un microcontrôleur Arduino. C'est l'Arduino qui, à son tour, interprète ces instructions et pilote le driver ADMH2407ND, en générant les signaux de contrôle nécessaires pour commander les moteurs.

Afin d'assurer une stabilité mécanique optimale, une roue folle a été ajoutée à l'arrière du châssis. Cette roue libre, non motorisée, permet au robot de rester stable tout en facilitant les manœuvres, sans friction excessive ni blocage.

2.4 Module de manipulation

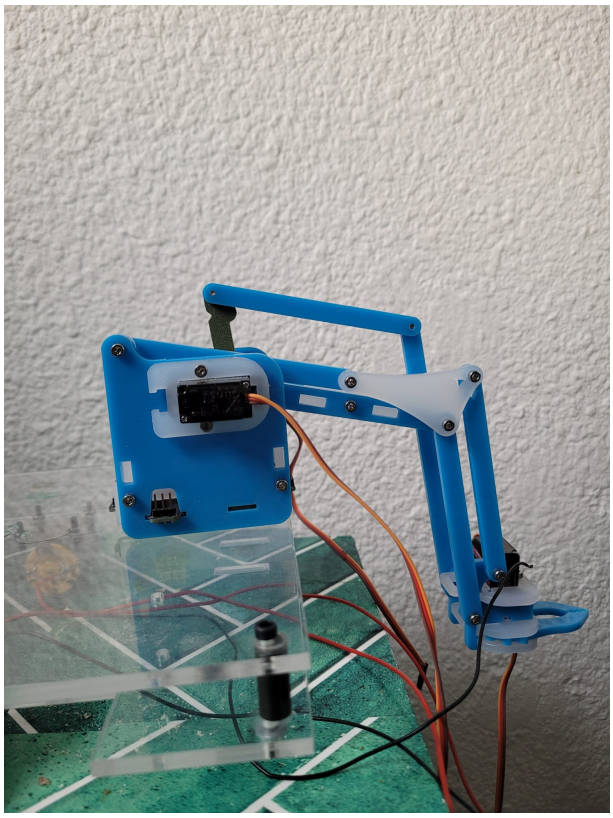

FIGURE II.4 – Bras du robot

Le module de manipulation est composé d'un bras robotisé articulé, spécialement conçu pour effectuer des mouvements précis afin de saisir une balle de ping-pong au sol, la soulever, puis la sécuriser pour son transport vers une zone de dépôt.

Ce bras présente trois degrés de liberté (DoF), permettant d'accomplir les actions nécessaires suivantes : - l'abaissement du bras vers la balle,

- la préemption (fermeture de la pince),
- la remontée de la balle,
- puis le repositionnement du bras à son état initial ou vers une autre position.

Chaque articulation est commandée par un servomoteur à retour de position, choisi pour sa précision angulaire, sa compacité et la simplicité de son pilotage via des signaux PWM. Les trois servomoteurs commandent respectivement :

1- l'épaule du bras (rotation verticale depuis le support),

2- le coude (pliage du bras),

3- la pince (fermeture pour attraper la balle).

Le bras est fixé à une structure de support en acrylique, surélevée par rapport au sol pour permettre une course verticale suffisante. Il est dimensionné pour atteindre une balle

positionnée à une distance prédéfinie du robot, équivalente à la longueur totale du bras moins une marge d'approche de 8 cm.

Le contrôle des servomoteurs est assuré par une carte Arduino Uno, recevant des ordres envoyés par une Raspberry Pi 5 via communication série. Cette architecture permet de dissocier les tâches de détection (traitement image sur la Pi) et d'action (commande des moteurs par l'Arduino), assurant une coordination fluide et réactive entre la détection, la décision et l'exécution.

III Fonctionnement du robot

1 Fonctionnement globale

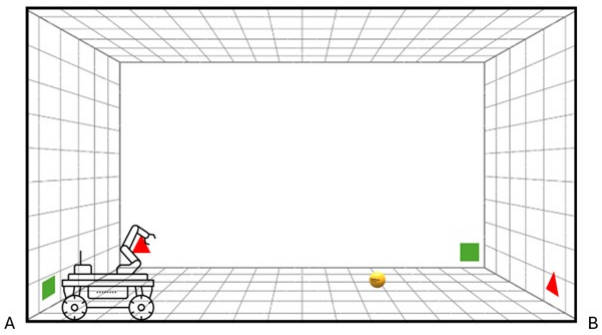

FIGURE III.1 – Environnement délimité simulant une salle de jeu

Le robot est conçu pour effectuer une mission simple mais complète : localiser, saisir, et rapporter une balle de ping-pong dans un espace défini. Son comportement suit un enchaînement logique d'étapes.

1.1 Initialisation :

Au démarrage, le robot se trouve en position A (indiquée par un carré vert). Au démarrage, le robot initialise ses composants (moteurs, caméras, bras manipulateur) puis commence la phase d'exploration à partir de sa position actuelle. Le robot vérifie également son orientation initiale et s'aligne, en tournant jusqu'à ce que le triangle rouge en B soit au centre du champs de vision, si besoin.

1.2 Balayage visuel :

Une fois prêt, le robot lance un balayage angulaire de l'environnement à l'aide de sa caméra. Il tourne sur lui-même progressivement pour analyser l'espace et tenter de localiser une balle de ping-pong, identifiable par sa couleur jaune et sa forme sphérique.

1.3 Détection et réaction :

Si une balle est détectée, le robot interrompt le balayage, s'oriente vers la cible, et procède à la saisie. Dans le cas contraire, il continue la rotation jusqu'à atteindre une limite définie (par un repère visuel comme un carré vert ou un triangle rouge).

1.4 Stratégie d'exploration :

Si aucune détection n'est réalisée dans la première zone (A ou B), le robot change de position et répète le processus. Ce déplacement fait partie d'une stratégie d'exploration efficace, qui lui permet de couvrir l'ensemble de la salle.

1.5 Retour :

Une fois la balle saisie, le robot retourne à la position A, marquant la fin de la mission

2 Processus de détection de la balle

Le robot utilise une caméra connectée à un système de vision basé sur OpenCV et un réseau de neurones pour détecter et localiser une balle de ping-pong dans un environnement réel. Cette détection suit plusieurs étapes de traitement d'image, de reconnaissance et d'analyse géométrique.

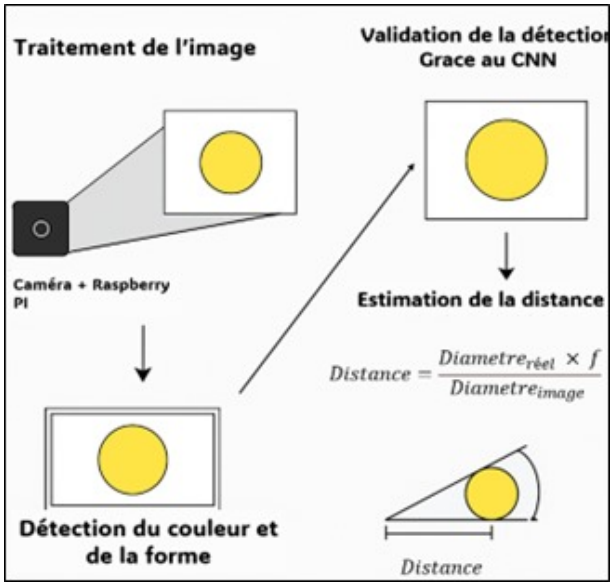

FIGURE III.2 – Processus de détection de la balle de ping-pong

2.1 acquisition de l'image :

La détection commence par la capture d'images en temps réel à partir d'une webcam (via cv2.VideoCapture). Chaque image est analysée individuellement pour y repérer la présence potentielle d'une balle.

2.2 Traitement d'image et détection de couleur :

2.2.1 - Conversion en HSV : L'image capturée est convertie de l'espace BGR vers HSV (`cv2.cvtColor`) afin de faciliter la détection de la couleur de la balle. Ce choix est pertinent car l'espace HSV est plus robuste aux variations d'éclairage.

```
import cv2
import numpy as np

# Plage HSV pour d tecter l'orange
lower_orange = np.array([10, 100, 100])
upper_orange = np.array([25, 255, 255])

# Conversion de l'image BGR en HSV
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

Listing 1 – Masque HSV pour la détection de couleur orange

2.2.2 - Filtrage par couleur. Une plage de couleur orange (valeurs HSV entre [5, 150, 150] et [25, 255, 255]) est utilisée pour créer un masque binaire :

```
import cv2
import numpy as np

mask = cv2.inRange(hsv, lower_orange, upper_orange)
```

Listing 2 – Application du filtre de couleur

2.2.3 - Amélioration de l'image (pré-traitement) Le contraste de l'image est amélioré avec CLAHE (Contrast Limited Adaptive Histogram Equalization), appliqué sur une version en niveaux de gris de l'image :

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_gray = clahe.apply(gray)
```

Listing 3 – Amélioration du contrast de l'image

2.2.4 - Réduction du bruit Le masque binaire est ensuite lissé avec un filtre Gaussien (cv2.GaussianBlur) afin de réduire les artéfacts avant l'analyse de forme :

```
import cv2
import numpy as np

blurred = cv2.GaussianBlur(mask, (9, 9), 2)
```

Listing 4 – Lissage du filtre de couleur

2.2.5 - Détection de forme (cercles) – HoughCircles. Pour améliorer la détection de la balle l'algorithme HoughCircles* est mis en place pour identifier les formes circulaires dans l'image binaire. Il s'agit d'un détecteur efficace pour les objets de forme sphérique comme une balle de ping-pong :

```
import cv2
import numpy as np

circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1.2, 50,
                           param1=50, param2=30, minRadius=10,
                           maxRadius=10)
```

Listing 5 – identification de la forme de la balle

Les paramètres contrôlent la précision et la sensibilité de la détection :

- param1 : seuil du détecteur de bord (Canny).
- param2 : seuil pour la détection circulaire.
- minRadius, maxRadius : pour limiter la taille des objets détectés.

2.2.6 - Calcul de la distance à la balle Une fois un cercle détecté, son rayon en pixels est utilisé pour calculer la distance réelle de la balle par la formule de la perspective :

$$\text{Distance} = \frac{\text{Diamètre}_{\text{réel}} \times f}{\text{Diamètre}_{\text{image}}}$$

Avec : - f : distance focale (FOCAL LENGTH) en pixels (585, à calibrer).

2.2.7 - Classification avec un réseau de neurones Pour valider la nature de l'objet détecté, un modèle d'intelligence artificielle (TensorFlow) est utilisé. Il s'agit d'un réseau de neurones convolutionnel (CNN) entraîné pour distinguer les balles de ping-pong d'autres objets.

```
model = load_model("model.h5")
```

Listing 6 – importation du modèle d'apprentissage

Chaque image extraite autour du cercle détecté est :

— Redimensionnée à 194 × 194

— Normalisée (/255.0)

— Passée au modèle pour classification

2.2.8 - Extraction des coordonnées Si la détection est validée :

— Les coordonnées (x, y) du centre du cercle sont extraites

— Elles permettent de centrer le robot sur la balle.

— Elles sont aussi utilisées pour afficher des informations en surimpression sur l'image.

En résumé :

Étape	Technique utilisée
Détection de couleur	HSV + `cv2.inRange()`
Amélioration d'image	CLAHE + `GaussianBlur`
Détection de forme	`HoughCircles`
Estimation de distance	Formule de la perspective
Validation de l'objet	Réseau de neurones pré-entraîné (TensorFlow)
Localisation	Extraction des coordonnées (x, y)

TABLE 1 – Techniques utilisées pour la détection et la localisation de la balle

3 Processus de prise et de manipulation de la balle

Lorsque la balle est détectée, le robot ajuste sa rotation pour centrer la balle dans le champ de vision. L'objectif est que la balle soit positionnée au centre de l'image, garantissant un alignement frontal avant l'approche. Il se dirige vers elle à une vitesse supérieure à celle utilisée pendant les déplacements standards. Une fois suffisamment proche, un mécanisme de bras articulé se déploie pour effectuer la prise, le bras descend jusqu'à la position estimée de la balle, saisit celle-ci à l'aide d'une pince, puis remonte pour sécuriser l'objet.

Lorsque le robot s'arrête à proximité d'une balle, il est positionné à une distance déterminée de celle-ci, que nous avons choisie égale à la longueur totale du bras moins une marge de 8 cm. Ainsi, la distance horizontale entre l'épaule du bras et la balle est définie comme :

x = (l1+l2) - 8 = 20 - 8 = 12 cm

où l1=8cm et l2 =12 cm sont respectivement les longueurs du premier et du deuxième segment du bras. La hauteur du centre de rotation de l'épaule par rapport au sol est obtenue en additionnant :

- le rayon des roues, soit 6,5 cm,

- et la hauteur du support du bras, soit 4,5 cm.

On obtient donc :

y = 6,5 + 4,5 = 11 cm

Connaissant les coordonnées du point cible (x,y), ainsi que les longueurs des deux segments du bras (l1 et l2), nous pouvons appliquer les équations de cinématique inverse pour déterminer les deux angles nécessaires :

q2, l'angle entre les deux segments du bras (au niveau du coude) :

$$q_2 = \pm \arccos\left(\frac{x^2 + y^2 - (l_1^2 + l_2^2)}{2 l_1 l_2}\right) \tag{1}$$

q1, l'angle entre l'horizontale et le premier segment du bras (entre l'épaule et le coude) :

$$q_1 = \arctan\left(\frac{y\,(l_1 + l_2\cos(q_2)) - x \cdot l_2 \sin(q_2)}{x\,(l_1 + l_2\cos(q_2)) + y \cdot l_2 \sin(q_2)}\right) \tag{2}$$

Avec les valeurs numériques précédentes, on obtient :

— q2 = 72,73

— q1 = -2,23

Ces deux angles nous permettent de positionner précisément le bras pour qu'il atteigne la balle avec la pince.

4 Résumé et organigramme.

Afin de mieux visualiser le fonctionnement global du robot, l'organigramme ci-dessous présente les différentes étapes clés du processus, depuis l'initialisation jusqu'au retour à la position de départ après la détection et la récupération de la balle.

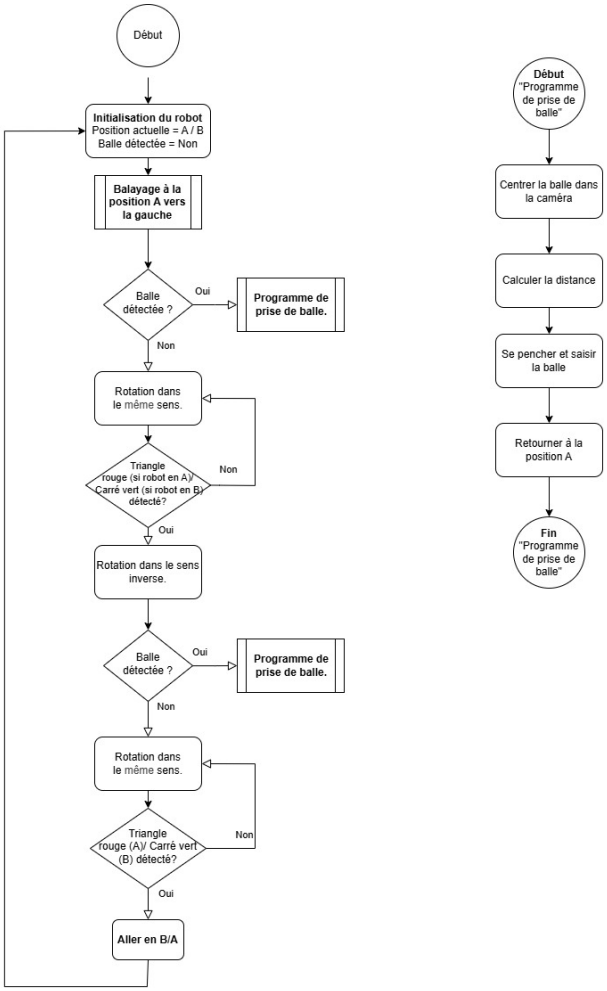

FIGURE III.3 – Organigramme de fonctionnement du robot

IV Conclusion et perspectives

1 Bilan du projet

Ce projet a permis de concevoir, programmer et tester un robot autonome capable de détecter, localiser et manipuler une balle de ping-pong dans un environnement contrôlé. À travers l'intégration de composants mécaniques (châssis, bras, moteurs), électroniques (capteurs, contrôleurs), et logiciels (algorithmes de détection et de navigation), le robot a atteint les objectifs fonctionnels définis. L'ensemble des modules a pu être validé individuellement puis testé en conditions réelles, ce qui a mis en évidence la cohérence globale de l'architecture développée. Le système réagit de manière autonome aux obstacles, identifie correctement la balle grâce à la vision, et exécute les séquences de prise et de déplacement avec fiabilité.

2 Limitations et améliorations possibles

Malgré les résultats satisfaisants, certaines limitations persistent. Le système de vision, basé uniquement sur la reconnaissance de la couleur et de la forme, reste sensible aux variations de luminosité ou à des objets perturbateurs similaires. De plus, la précision du

bras manipulateur dépend fortement de l'alignement du robot avec la balle, ce qui peut entraîner des échecs de saisie si la balle est légèrement décentrée.

Par ailleurs, la stratégie de recherche reste déterministe et ne prend pas en compte des comportements adaptatifs. Le déplacement du robot est aussi limité à un plan fixe sans prise en compte dynamique de l'environnement. Une amélioration future pourrait intégrer un système de cartographie (SLAM), une intelligence artificielle embarquée pour la décision, ou encore un retour haptique pour sécuriser la saisie de la balle.

3 Perspectives d'évolution

Pour faire évoluer ce projet, plusieurs pistes peuvent être explorées. D'abord, l'ajout de capteurs plus précis comme des lidars ou des caméras stéréoscopiques permettrait une meilleure perception 3D de l'environnement. L'implémentation d'un apprentissage automatique (par exemple via du deep learning pour la détection d'objets) pourrait aussi rendre le robot plus robuste face à la variabilité des scènes. D'un point de vue mécanique, un bras manipulateur plus articulé ou assisté par des capteurs de pression permettrait une prise plus délicate et fiable. Enfin, étendre l'algorithme de navigation à une exploration adaptative ou collaborative (multi-robots) ouvrirait la voie à des applications plus complexes, comme le tri d'objets ou la logistique en environnement semi-structuré.

V Annexes

```python
import cv2
import os
import numpy as np

# Définition du dossier de sauvegarde
save_dir = r"C:\Users\Aymen\Documents\birse\PJT_PINGPONG_ROBOT\BDD_BALLE"
if not os.path.exists(save_dir):
    os.makedirs(save_dir)

def capture():
    cap = cv2.VideoCapture(0)  # Ouvrir la caméra une seule fois
    img_counter = 0  # Compteur d'images

    cv2.namedWindow('Détection Balle Ping-Pong')  # Créer une seule fenêtre

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Erreur : Impossible de capturer l'image")
            break

        # Conversion en HSV pour la détection de couleur
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Définition des plages de couleur pour une balle de ping-pong (orange/blanc)
        lower_orange = np.array([5, 150, 150])
        upper_orange = np.array([25, 255, 255])

        mask = cv2.inRange(hsv, lower_orange, upper_orange)  # Filtrage des pixels dans la plage

        # Détection de contours et de cercles
        blurred = cv2.GaussianBlur(mask, (9, 9), 2)
        circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1.2, 50, param1=50, param2=30, minRadius=10, maxRadius=100)

        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                # Dessiner le cercle détecté
                cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)

                if cv2.waitKey(1) & 0xFF == ord('c'):  # Sauvegarder l'image avec 'c'
```

```
                    img_name = os.path.join(save_dir, f"balle_{img_counter}.jpg")
                    roi = frame[i[1] - i[2]:i[1] + i[2], i[0] - i[2]:i[0] + i[2]]  # Extraire la balle
                    if roi.size > 0:
                        cv2.imwrite(img_name, roi)
                        print(f"Image enregistr e : {img_name}")
                        img_counter += 1

        cv2.imshow('D tection Balle Ping-Pong', frame)  # Une seule fen tre

        if cv2.waitKey(1) == ord('q'):  # Quitter avec 'q'
            break

    cap.release()
    cv2.destroyAllWindows()  # Fermer toutes les fen tres proprement

capture()
```

Listing 7 – programme pour enregistrement d'image

```python
import cv2
import numpy as np
from tensorflow.keras.models import load_model

# === Param tres ===
MODEL_PATH = r"C:\Users\Aymen\Documents\birse\PJT_PINGPONG_ROBOT\Programmes\model.h5"
IMAGE_DIM = 194  # Taille attendue par le mod le
FOCAL_LENGTH = 585
KNOWN_DIAMETER = 40  #

# Charger le mod le
model = load_model(MODEL_PATH)

def detect_red_triangles(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red1 = np.array([0, 100, 100])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([160, 100, 100])
    upper_red2 = np.array([180, 255, 255])

    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
    red_mask = cv2.bitwise_or(mask1, mask2)

    red_only = cv2.bitwise_and(frame, frame, mask=red_mask)
    gray = cv2.cvtColor(red_only, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        epsilon = 0.04 * cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, epsilon, True)
        if len(approx) == 3 and cv2.contourArea(cnt) > 500:
            cv2.drawContours(frame, [approx], 0, (0, 255, 0), 3)
            cv2.putText(frame, 'Triangle Rouge', tuple(approx[0][0]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

def detect_green_squares(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_green = np.array([40, 70, 70])
    upper_green = np.array([80, 255, 255])
    green_mask = cv2.inRange(hsv, lower_green, upper_green)

    green_only = cv2.bitwise_and(frame, frame, mask=green_mask)
```

```
    gray = cv2.cvtColor(green_only, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        epsilon = 0.04 * cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, epsilon, True)
        if len(approx) == 4 and cv2.contourArea(cnt) > 500 and cv2.isContourConvex(approx):
            cv2.drawContours(frame, [approx], 0, (0, 255, 255), 3)
            cv2.putText(frame, 'Carre+ Vert', tuple(approx[0][0]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

def detect_ping_pong_ball(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_orange = np.array([5, 150, 150])
    upper_orange = np.array([25, 255, 255])
    mask = cv2.inRange(hsv, lower_orange, upper_orange)

    blurred = cv2.GaussianBlur(mask, (9, 9), 2)
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1.2, 50, param1=50, param2=30, minRadius=10, maxRadius=100)

    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            x, y, radius = i[0], i[1], i[2]
            diameter_pixels = radius * 2
            distance = (KNOWN_DIAMETER * FOCAL_LENGTH) / diameter_pixels if diameter_pixels > 0 else 0

            roi = frame[max(0, y - radius):min(y + radius, frame.shape[0]),
                        max(0, x - radius):min(x + radius, frame.shape[1])]

            if roi.size > 0:
                roi_resized = cv2.resize(roi, (IMAGE_DIM, IMAGE_DIM))
                roi_resized = roi_resized.astype("float32") / 255.0
                img_array = np.expand_dims(roi_resized, axis=0)

                prediction = model.predict(img_array, verbose=0)
                predicted_class = np.argmax(prediction)

                if predicted_class == 0 and prediction[0][0] >= 0.9:
```

```
                    cv2.circle(frame, (x, y), radius, (255, 0, 0), 2)
                    cv2.putText(frame, f"X: {x}  Y: {y}", (x + 35, y - radius + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0, 0), 2)
                    cv2.putText(frame, f"Diam: {diameter_pixels}px", (x + 35, y - radius + 35), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0, 0), 2)
                    cv2.putText(frame, f"Dist: {distance:.2f} mm", (x + 35, y - radius + 55), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0,0), 2)
                    cv2.putText(frame, "Balle de Ping-Pong", (x + 35, y - radius), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2)

def main():
    cap = cv2.VideoCapture(0)
    cv2.namedWindow('D tection Mixte')

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Erreur : Impossible de capturer l'image")
            break

        detect_red_triangles(frame)
        detect_green_squares(frame)
        detect_ping_pong_ball(frame)

        cv2.imshow('D tection Mixte', frame)

        if cv2.waitKey(1) == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

main()
```

Listing 8 – programme de détection

```
# In[ ]
import joblib
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import cv2
import warnings
import glob
import numpy as np
from sklearn.model_selection import train_test_split

warnings.filterwarnings('ignore')

# Initialisation des listes pour les images et labels
X = []
y = []
image_dim = 194  # Taille des images

# Chargement des images pour chaque employ
def load_images_from_folder(folder_path, label):
    images = glob.glob(folder_path + '/*.jpg')  # Chargement des fichiers d'images
    for image_file in images:
        img = cv2.imread(image_file)
        img = cv2.resize(img, (image_dim, image_dim))
        X.append(img)
        y.append(label)

# Chargement des images des employ s
load_images_from_folder(r"C:\Users\Aymen\Documents\birse\PJT_PINGPONG_ROBOT\BDD_BALLE", 0)

# Conversion des listes en tableaux NumPy
X = np.array(X, dtype=np.float32)
y = np.array(y)
print (X)
print(y)
# In[]
# Division des donn es en ensemble d'entra nement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalisation des images (valeurs entre 0 et 1)
X_train /= 255.0
X_test /= 255.0

# Cr ation du mod le CNN
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(194, 194, 3)),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(3, activation='softmax')
])

# Affichage de l'architecture du mod le
model.summary()

# Compilation du mod le
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
   from_logits=True),
              metrics=['accuracy'])


# Entra nement du mod le
history = model.fit(X_train, y_train, epochs=15, validation_data=(
   X_test, y_test))
joblib.dump(history, "reconnaissance_faciale.pkl")#
# Affichage de l' volution  de l'accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
#  valuation  finale du mod le
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc:.4f}")
# In[ ]
from tensorflow.keras.models import save_model
save_model(model,r"C:\Users\Aymen\Documents\birse\
   PJT_PINGPONG_ROBOT\Programmes\model.h5")
```

Listing 9 – entrainnement du modele

```
// Inclure la biblioth que Servo
#include <Servo.h>

// D claration des servomoteurs
Servo servo1; // Servo du coude
Servo servo2; // Servo de   l  paule
Servo servo3; // Servo de la pince

// Position initiale des servos
int angleInitial = 100;  // Position initiale pour le coude
int angleInitialC = 90;  // Position initiale pour   l  paule

// Angles cibles pour d placer le bras robotis  (r sultats du mod le
    cin matique)
int angle1 = 252.73;  // Angle pour le coude (servo1)
int angle2 = 87.77;   // Angle pour   l  paule   (servo2)
int angle3 = 180;     // Angle pour la pince (servo3)

void setup() {
  servo1.attach(11);  // Servo1 (coude)
  servo2.attach(6);   // Servo2 ( paule )
  servo3.attach(5);   // Servo3 (pince)

  servo1.write(angleInitial);
  servo2.write(angleInitialC);
  servo3.write(90);
  delay(1000);

  servo1.write(180);
  servo2.write(90);
  delay(1000);

  servo1.write(angle1);
  servo2.write(angle2);
  delay(1500);

  servo3.write(180);
  delay(1500);

  servo1.write(angleInitial);
  servo2.write(angleInitial);
}

void loop() {
  // Le mouvement est ex cut  une seule fois dans setup()
}
```

Listing 10 – Controle du bras robotisé