BUT GEII – Compte Rendu de Conception

Projet : Modernisation d’un Robot Aspirateur

Étudiant :BONILLA AZUAJE Rafael

Groupe: 404_B

Encadrant :

Établissement : IUT Lyon 1

Date : Avril 2025

Introduction

Ce projet vise à moderniser un robot aspirateur en lui intégrant une navigation autonome de haute précision, en remplaçant les capteurs de fin de course classiques par un LIDAR YDLIDAR X4 Pro. Ce capteur permet de cartographier l'environnement en temps réel, détecter les obstacles avec précision et planifier un trajet de nettoyage optimal.

Pour exploiter pleinement ces données, le robot fonctionnera avec un Lidar, dont le driver est exécuté sur une Raspberry Pi 5 avec Ubuntu, assurant ainsi un traitement en temps réel, une gestion centralisée des tâches et une grande extensibilité.

Le robot modernisé offrira les fonctionnalités suivantes :

- Contrôle des 5 moteurs (roues, brosses, aspiration)

- Cartographie en temps réel via LIDAR

- Planification d'itinéraire

- Surveillance de la batterie et gestion de la recharge

- Pilotage à distance via une interface utilisateur (bonus)

Cette configuration rendra le robot plus intelligent, réactif, autonome et évolutif.

Présentation Approfondie des Objectifs du Projet

Objectif 1 : Moderniser la navigation en remplaçant les capteurs de fin de course par le LIDAR

L'objectif principal de ce projet est de moderniser le système de navigation du robot aspirateur en remplaçant les capteurs de fin de course traditionnels par un LIDAR (Light Detection and Ranging), et ainsi permettre au robot de naviguer de manière autonome et plus intelligente dans son environnement.

Les capteurs de fin de course sont limités dans leur capacité à détecter les obstacles à distance. Le LIDAR, en revanche, permet une détection à longue portée et une cartographie continue de l'environnement, ce qui améliore la planification des déplacements et évite les zones déjà nettoyées.

Objectif 2 : Intégrer le driver pour le traitement des données et l'algorithme SLAM

L'intégration du driver permet d'exploiter les bibliothèques robotiques avancées et de traiter les données en temps réel via un algorithme SLAM (Simultaneous Localization and Mapping). Ce dernier permet au robot de se localiser tout en construisant une carte de son environnement, sans carte préexistante.

Grâce au driver, le robot peut également gérer les moteurs, l'évitement d'obstacles, et la coordination générale de manière centralisée.

Objectif 3 : Gérer les moteurs via une Raspberry Pi 5 et des Dual H Bridges

Tous les moteurs (roues, brosses, aspiration) sont pilotés depuis la Raspberry Pi 5 via des ponts en H (Dual H Bridge), ce qui permet de contrôler précisément vitesse, direction et synchronisation.

La centralisation via la Raspberry Pi 5, combinée à ROS, permet une cohérence entre la navigation et les actions mécaniques du robot.

Objectif 4 : Implémenter une gestion intelligente de la batterie et de la recharge

Le robot doit surveiller sa batterie en temps réel et retourner automatiquement à sa station de charge en cas de besoin. Après avoir rechargé, il reprend son itinéraire sans perte de progression. Cette autonomie énergétique permet un nettoyage sur de longues périodes sans supervision.

Objectif 5 : Ajouter une interface de pilotage à distance

Une interface utilisateur (web ou mobile) permet de démarrer/arrêter une session, définir des zones à éviter/nettoyer, ou modifier les paramètres. La commande peut se faire par Wi-Fi ou Bluetooth.

Objectifs et Réalisations du Semestre 3

- Intégration du LIDAR YDLIDAR X4 Pro sur le robot aspirateur

- Installation de ROS sur une Raspberry Pi 5 avec Ubuntu Core

- Mise en place de la cartographie en temps réel et détection des obstacles

- Développement initial des algorithmes de planification de trajet

Objectifs et Réalisations du Semestre 4

- Contrôle et synchronisation des 5 moteurs (roues, brosses, aspiration)

- Optimisation des algorithmes de planification d'itinéraire

État d'Avancement des Objectifs

1.**Contrôle des moteurs, objectif atteint :**

Travail réalisé : Après plusieurs essais et tests, la méthode idéale pour piloter les moteurs a été trouvée grâce à une interface intermédiaire utilisant un driver logique. Cette interface reçoit les commandes des pins de la Raspberry Pi, pour ensuite envoyer

les signaux électriques de commande aux Dual H Bridge, qui prenaient en charge l'avance, le recul ou l'arrêt des moteurs en fonction des informations reçues du Lidar. Plusieurs tests de continuité ont été réalisés et il a été possible de gérer le tout avec une alimentation séparée : 6.5V pour faire fonctionner les cartes électroniques intermédiaires et les Dual H Bridge, et 7 V+ nécessaires pour alimenter les moteurs.

Succès : Contrôle des moteurs avec une gestion efficace des moteurs balais, carte KiCad pour le driver logique, Raspberry Pi et Lidar intégrés pour une navigation contrôlée.

Difficultés : Une fuite de puissance a endommagé l'un des drivers. Malheureusement, nous avons dû nous passer de la carte KiCad intermédiaire et connecter directement la Raspberry Pi à la carte électronique des Dual H Bridge. Bien que cela ait fonctionné correctement, la sécurité fournie par la carte KiCad (avec isolation galvanique des drivers) a été perdue. De plus, bien que tous les moteurs aient été contrôlés, un moteur de roue a cessé de fonctionner, ce qui a limité le fonctionnement complet à 4 moteurs sur 5, tout en réussissant à faire fonctionner le moteur d'aspiration malgré des exigences de courant différentes.

2. Optimisation des Algorithmes Objectif atteint

Travail accompli :
Nous avons réussi à récupérer les données du LIDAR YDLIDAR X4 Pro en utilisant directement le driver fourni par le constructeur, sans passer par ROS 2. Cette approche plus directe nous a permis d'obtenir rapidement des mesures fiables de distance et d'angle, répondant aux besoins de notre projet sans complexifier inutilement l'architecture logicielle.

Résultats obtenus :
Grâce à cette méthode, nous avons pu commencer à générer des ébauches de plans simples de l'environnement. La qualité des données reçues a confirmé le bon fonctionnement du capteur et a permis de valider la première étape d'une cartographie.

Limites rencontrées :
La mise en œuvre complète d'un algorithme de SLAM n'a cependant pas été réalisée, faute de temps et en raison de la complexité du traitement associé. Le travail effectué s'est limité à une représentation visuelle basique de l'environnement, sans localisation dynamique ni navigation autonome.

Conclusion

Ce projet de robot aspirateur nous a permis de mettre en œuvre nos compétences en automatisme, en électronique et en programmation embarquée. À travers les différentes

étapes de conception, d'assemblage et de mise en service, nous avons pu appréhender les contraintes techniques liées à l'autonomie, à la détection d'obstacles et à l'efficacité de nettoyage. Les tests réalisés ont démontré la fiabilité du système dans un environnement contrôlé, bien que certaines améliorations sont envisageables, notamment en ce qui concerne la gestion de l'énergie et l'optimisation des algorithmes de navigation. Ce travail nous a non seulement permis de renforcer nos acquis théoriques, mais également de développer notre rigueur dans la gestion de projet et le travail en équipe, des compétences essentielles dans le domaine du génie électrique et de l'informatique industrielle.

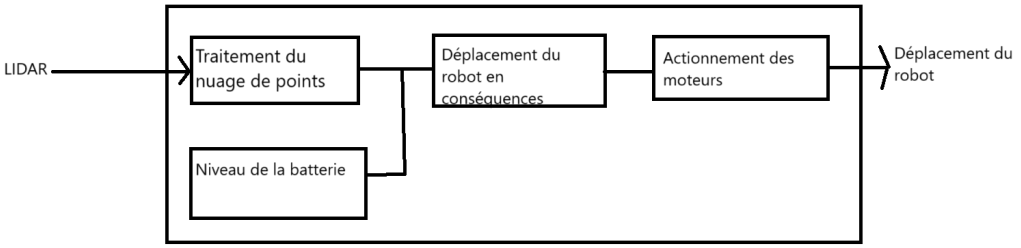
Malgré les contraintes rencontrées tout au long du projet, nous avons appris à les gérer. Même si le projet n'est pas entièrement terminé, nous serions désormais capables d'aborder d'autres projets, même d'un tout autre aspect, en sachant par où commencer et quelles approches adopter différemment grâce à l'expérience acquise lors de cette modernisation d'un robot aspirateur.

Annexes

- Schémas KiCad de l'intégration électronique

- Documentation technique du LIDAR et des H Bridges

- Doc CR S3 (synoptique, organigramme, schéma structurelle détaillé, nomenclature)



- Doc CR S3 (synoptique, organigramme, schéma structurelle détaillé, nomenclature)

Moteurs et alimentation :

1. Moteurs des roues : DC, 3-6 V, 300 mA.

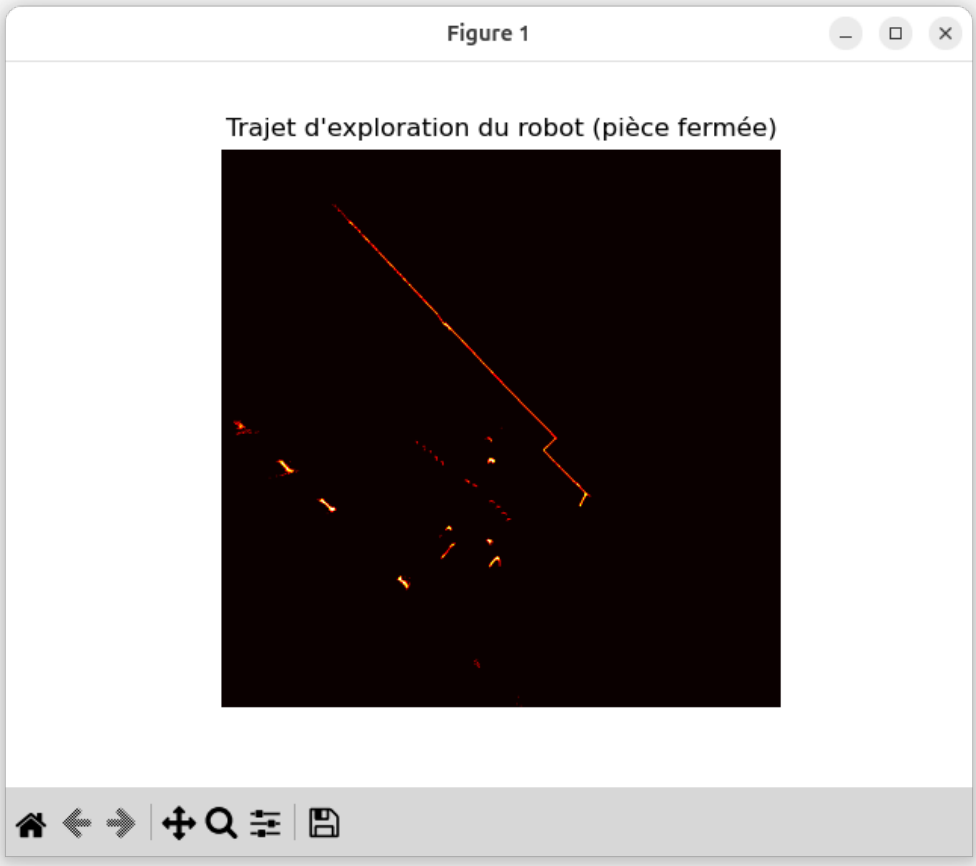
2. Moteurs des brosses : RC500-KN/13500, 3 V, 300 mA.

3. Moteur du ventilateur d'aspiration : Ariete Xclean, 6 V, 1-1.8 A.

Structure matérielle :

- Hacheur 1 : Pilote les moteurs des roues via GPIO (PWM) de la Raspberry Pi.
- Hacheur 2 : Contrôle les moteurs des brosses et du ventilateur d’aspiration.
- Raspberry Pi 5 : Gère les données du LIDAR, pilote les moteurs, et surveille la batterie.

- Map de la pièce scannée par le LIDAR



- Code Python pour la gestion des Moteurs avec le Lidar

```
# Importation des bibliothèques nécessaires
```

```
import ydlidar
import time
import gpiod

# Définition du GPIO utilisé
GPIOCHIP = "gpiochip0"

# Définition des broches pour le moteur gauche et droit du déplacement
MOTOR_LEFT_FORWARD = 17
MOTOR_LEFT_BACKWARD = 18
MOTOR_RIGHT_FORWARD = 22
MOTOR_RIGHT_BACKWARD = 23

# Définition des broches pour les moteurs des balais
BALAIS_LEFT_FORWARD = 24
BALAIS_LEFT_BACKWARD = 25
BALAIS_RIGHT_FORWARD = 5
BALAIS_RIGHT_BACKWARD = 6

# Définition du GPIO pour le moteur supplémentaire qui tourne en continu
MOTEUR_SUPPLEMENTAIRE = 27

# Initialisation de l'accès aux lignes GPIO
chip = gpiod.Chip(GPIOCHIP)
motor_pins = [
    MOTOR_LEFT_FORWARD, MOTOR_LEFT_BACKWARD,
    MOTOR_RIGHT_FORWARD, MOTOR_RIGHT_BACKWARD,
    BALAIS_LEFT_FORWARD, BALAIS_LEFT_BACKWARD,
    BALAIS_RIGHT_FORWARD, BALAIS_RIGHT_BACKWARD,
    MOTEUR_SUPPLEMENTAIRE
]
lines = chip.get_lines(motor_pins)
lines.request(consumer="motor_control", type=gpiod.LINE_REQ_DIR_OUT)

# Fonction pour avancer le robot
def avancer():
    print("Le robot avance")
    lines.set_values([
        1, 0,  # Moteur gauche avance
        1, 0,  # Moteur droit avance
        1, 0,  # Balai gauche tourne
        1, 0,  # Balai droit tourne
        1      # Moteur supplémentaire activé
    ])

# Fonction pour faire tourner le robot à droite
def tourner_droite():
    print("Le robot tourne à droite")
    lines.set_values([
        1, 0,  # Moteur gauche avance
```

```
        0, 1,  # Moteur droit recule
        0, 0,  # Balai gauche arrêté
        0, 0,  # Balai droit arrêté
        1      # Moteur supplémentaire reste activé
    ])

# Fonction pour arrêter tous les moteurs sauf le moteur supplémentaire
def arreter():
    print("Le robot s'arrête")
    lines.set_values([
        0, 0,  # Moteur gauche arrêté
        0, 0,  # Moteur droit arrêté
        0, 0,  # Balai gauche arrêté
        0, 0,  # Balai droit arrêté
        1      # Moteur supplémentaire reste activé
    ])

# Configuration du port série du LiDAR
LIDAR_PORT = "/dev/ttyUSB0"
LIDAR_BAUDRATE = 128000

# Initialisation du LiDAR
lidar = ydlidar.CYdLidar()
lidar.setlidaropt(ydlidar.LidarPropSerialPort, LIDAR_PORT)
lidar.setlidaropt(ydlidar.LidarPropSerialBaudrate, LIDAR_BAUDRATE)
lidar.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TRIANGLE)
lidar.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.YDLIDAR_TYPE_SERIAL)
lidar.setlidaropt(ydlidar.LidarPropScanFrequency, 7.0)
lidar.setlidaropt(ydlidar.LidarPropSampleRate, 5)
lidar.setlidaropt(ydlidar.LidarPropSingleChannel, True)
lidar.setlidaropt(ydlidar.LidarPropAutoReconnect, True)

# Démarrage du LiDAR
if lidar.initialize() and lidar.turnOn():
    print("Demarrage Lidar\n")

    try:
        while True:
            # Récupération d'un scan LiDAR
            scan = ydlidar.LaserScan()

            if lidar.doProcessSimple(scan):
                distance_a_0 = None
                # Recherche de la distance face au robot (0°)
                for point in scan.points:
                    if abs(point.angle) < 0.5:
                        distance_a_0 = point.range
                        break

                if distance_a_0 is not None:
```

```
                print(f"Distance à 0° : {distance_a_0:.2f} m")

                # Si pas d'obstacle proche, le robot avance
                if distance_a_0 > 0.30:
                    avancer()
                else:
                    # Si obstacle détecté, arrêt du robot puis rotation
                    while True:
                        tourner_droite()
                        scan = ydlidar.LaserScan()
                        if lidar.doProcessSimple(scan):
                            distance_a_0 = None
                            for point in scan.points:
                                if abs(point.angle) < 0.5:
                                    distance_a_0 = point.range
                                    break
                            # Quand le chemin est à nouveau libre, arrêt de la rotation
                            if distance_a_0 is not None and distance_a_0 > 0.30:
                                break
            else:
                print("Pas donnees a 0°.")

            # Attente avant le prochain traitement
            time.sleep(0.5)

    except KeyboardInterrupt:
        # Arrêt propre du programme en cas d'interruption par l'utilisateur
        print("\nArrêt du programme.")
        arreter()

    # Arrêt du LiDAR
    lidar.turnOff()
else:
    print("Échec Lidar")

# Libération des ressources GPIO
lines.release()
chip.close()

# Déconnexion propre du LiDAR
lidar.disconnecting()
```

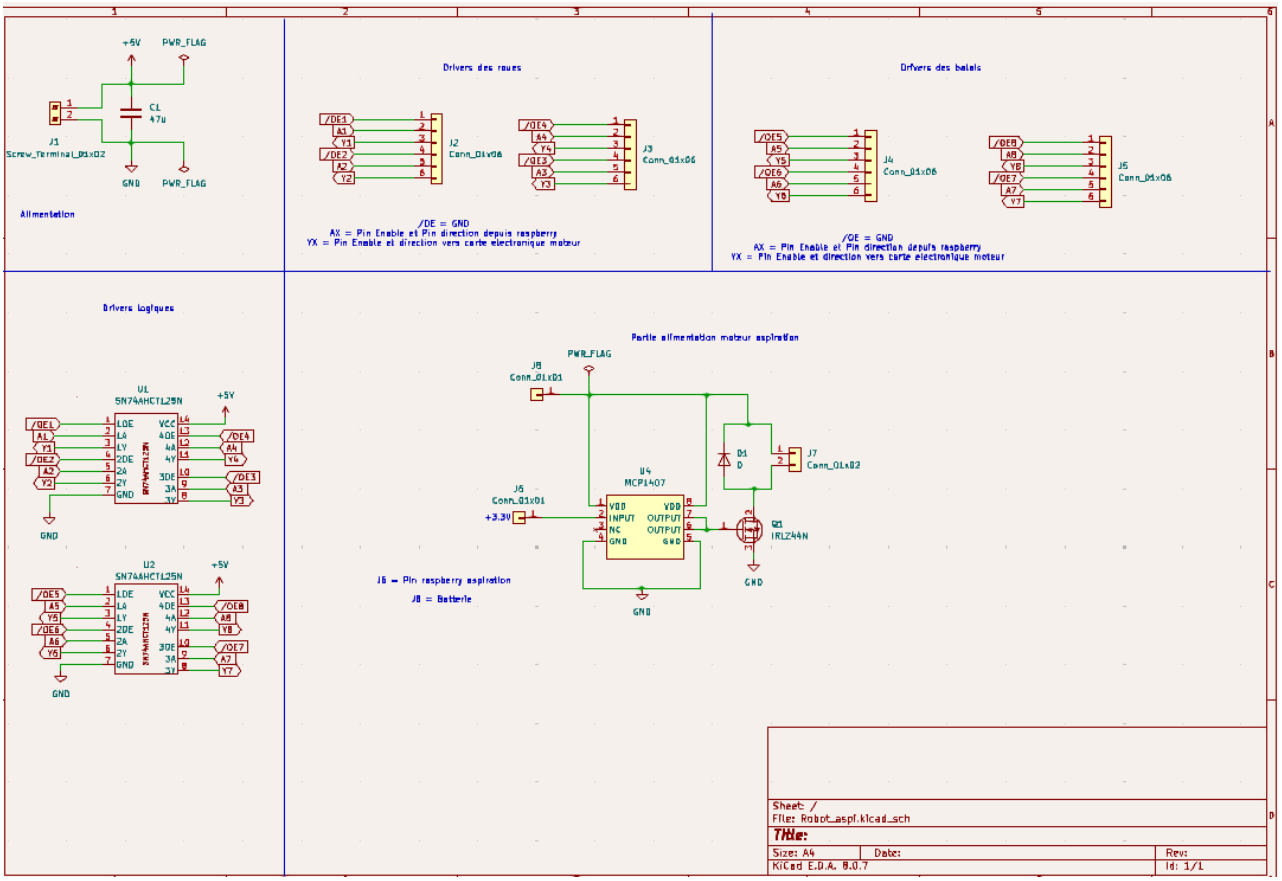
- Code Python pour la création d’une carte mentale avec le Lidar.

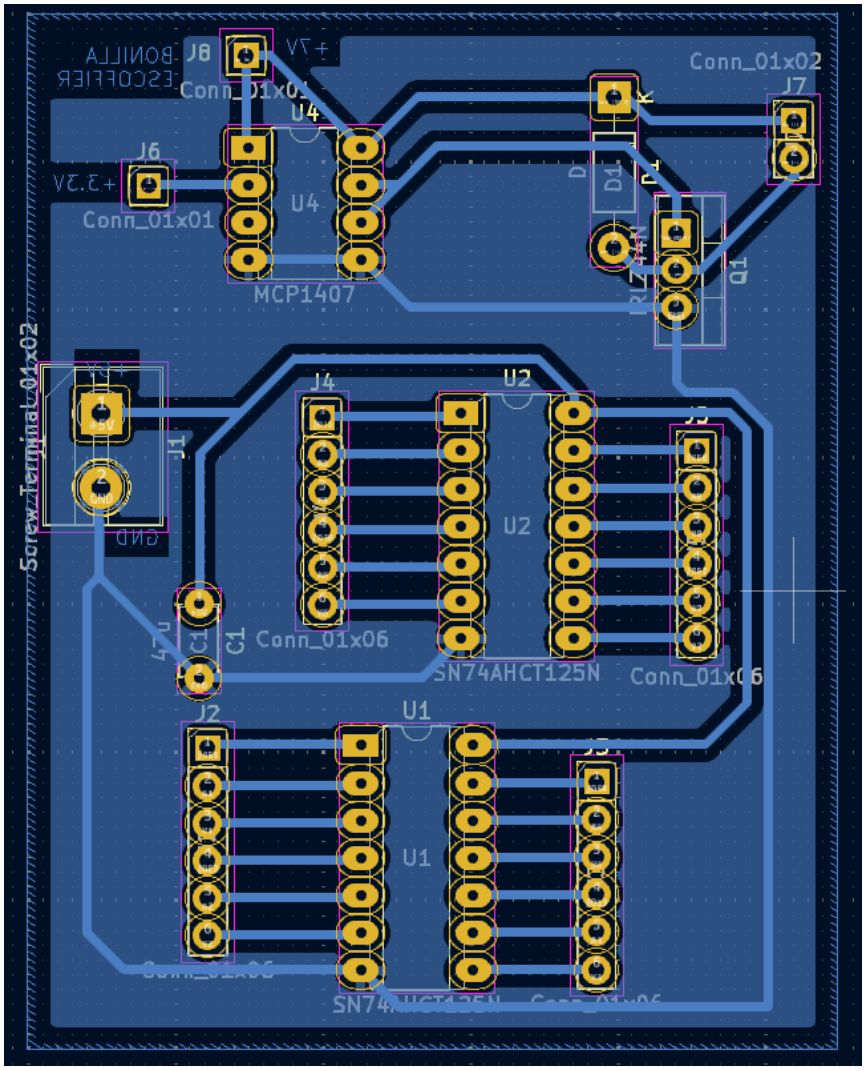
```
# Importation des bibliothèques nécessaires
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2

# Chargement de la carte mentale
# La carte est convertie en niveaux de gris pour simplifier le traitement
image = Image.open("carte_mentale.png").convert('L')
carte = np.array(image)

# Application d'un seuillage pour isoler les murs
# Les murs seront les pixels dont la valeur est supérieure à 20
_, binaire = cv2.threshold(carte, 20, 255, cv2.THRESH_BINARY)

# Détection des contours des obstacles sur la carte binaire
# Seuls les contours externes sont récupérés
contours, _ = cv2.findContours(binaire, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Création d'un masque vide pour représenter la pièce remplie
masque_piece = np.zeros_like(binaire)

# Remplissage des contours détectés pour obtenir une pièce entièrement fermée
cv2.drawContours(masque_piece, contours, -1, 255, thickness=cv2.FILLED)
```

```
# Création de la carte d'occupation
# 0 : espace libre
# 1 : obstacle (murs)
grille = np.where(masque_piece == 255, 0, 1)


# Définition du point de départ pour l'exploration
# Le robot commence approximativement au centre de la carte
h, w = grille.shape
for offset in range(h):
    if grille[h // 2 + offset, w // 2] == 0:
        start_y = h // 2 + offset
        start_x = w // 2
        break


# Génération du chemin d'exploration en zigzag
# Le robot parcourt chaque ligne de gauche à droite puis de droite à gauche successivement
trajet = np.zeros_like(grille, dtype=np.uint8)
sens = 1  # Variable permettant d'alterner le sens de parcours des lignes


for y in range(h):
    if sens == 1:
        ligne = range(w)
    else:
        ligne = range(w - 1, -1, -1)
```

```
    for x in ligne:
        if grille[y, x] == 0:
            trajet[y, x] = 255  # Marquage du chemin suivi par le robot

    sens *= -1  # Changement de sens pour la ligne suivante

# Affichage et sauvegarde de la trajectoire calculée
plt.imshow(trajet, cmap='hot')
plt.title("Trajet d'exploration du robot (pièce fermée)")
plt.axis('off')
plt.savefig("trajectoire_robot.png", bbox_inches='tight', pad_inches=0)
plt.show()
```

-Schémas Kicad, carte intermédiaire entre Raspberry et les hacheurs





Gerber de la carte électronique