Université Euro Méditerranéenne Fès
EuroMed University of Fez

Ecole d'Ingénierie Digitale et d'Intelligence Artificielle
(EIDIA)

Projet de Fin de module

Filière : Robotique et Cobotique

Semestre : 7

Module : Méthodologies de Navigation et de Localisation des Robots

Thème :

Robot Mobile Autonome pour la Gestion Intelligente des Parkings

soutenu le 24/01/25,

Encadré par :

Pr. MRABTI Mostafa

Préparé par les étudiants:

- Bouarfa LAHMAR
- Mohamed BOFARHA
- Hiba MOUHSINE

Année Universitaire : 2024-2025

Table des Matières :

Année Universitaire : 2024-2025

1. Introduction

Contexte et objectifs du projet :

La gestion efficace des parkings est devenue un défi majeur dans les zones urbaines, principalement en raison de la croissance rapide du nombre de véhicules. Les conducteurs passent souvent un temps considérable à chercher des places libres, ce qui entraîne une augmentation de la consommation de carburant, des émissions de CO_2, et un stress accru pour les utilisateurs.

Ce projet a pour objectif de développer un robot mobile capable de :

- Localiser les places libres et occupées dans un parking.
- Naviguer de manière autonome tout en évitant les obstacles.
- Générer une carte dynamique représentant en temps réel l'état actuel du parking.

Les principaux objectifs spécifiques du projet sont les suivants :

- Automatiser le processus de détection des places de stationnement.
- Optimiser la gestion des parkings grâce à une cartographie dynamique et précise.
- Réduire l'impact environnemental lié à la recherche de places disponibles.
- Mettre en pratique les connaissances théoriques à travers la réalisation d'un projet concret.

Valeur ajoutée :

Le projet présente plusieurs avantages, notamment :

- **Une solution innovante** : Le système proposé améliore l'efficacité des parkings en automatisant la détection des places disponibles et en optimisant la gestion de l'espace.
- **L'utilisation de technologies accessibles** : Le recours à des outils comme Arduino, Raspberry Pi et divers capteurs assure une mise en œuvre économique et reproductible pour des projets similaires.
- **Une approche modulaire** : Le design du projet permet une flexibilité pour des améliorations futures, comme l'ajout de nouvelles fonctionnalités, notamment la reconnaissance automatique des plaques d'immatriculation ou l'intégration de technologies avancées.
- **Une alternative écologique** : En réduisant les trajets inutiles à l'intérieur des parkings, le projet contribue à diminuer les émissions de CO_2 et à favoriser une utilisation plus durable des ressources énergétiques.

2. Description et Concept du Projet :

2.1 Problématique :

Dans les zones urbaines densément peuplées, la gestion des parkings représente un défi majeur. Le manque d'automatisation et de suivi en temps réel des places disponibles engendre une mauvaise utilisation des espaces, augmentant ainsi le temps de recherche pour les conducteurs. Ce phénomène non seulement contribue à la congestion du trafic, mais participe également à l'augmentation des émissions de gaz à effet de serre. L'absence de solutions intelligentes pour optimiser le processus de stationnement entraîne un stress supplémentaire pour les conducteurs, ainsi qu'une inefficacité générale dans l'utilisation des infrastructures. L'introduction d'un système automatisé de détection et de gestion des places de parking pourrait considérablement améliorer l'efficacité, réduire le gaspillage énergétique et diminuer les impacts environnementaux associés à la recherche de stationnement.

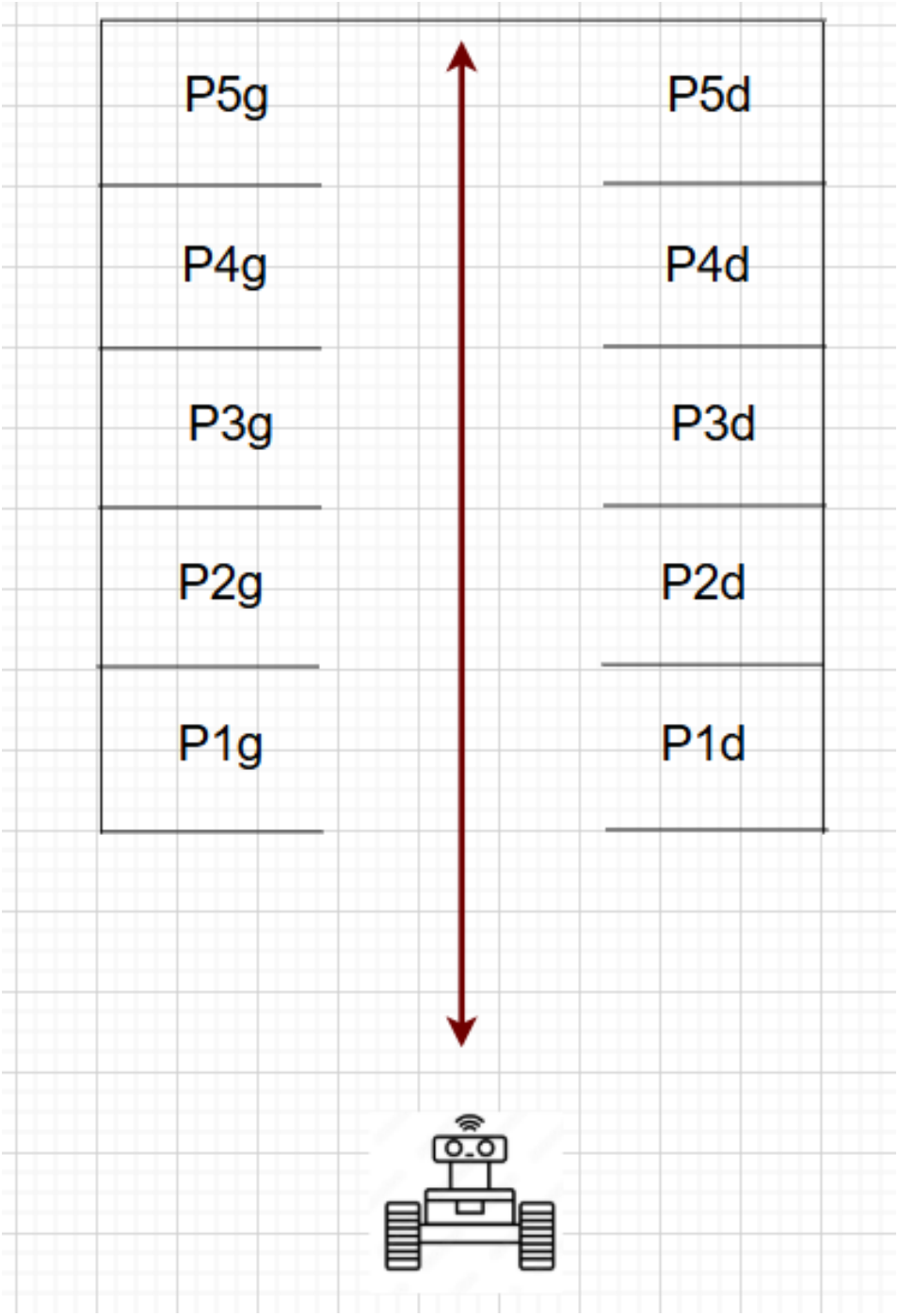
2.2 Concept du Robot Mobile :

Le projet propose un robot mobile capable de résoudre ces problématiques grâce à plusieurs fonctionnalités innovantes :

- **Détection des places libres et occupées** : Le robot utilise une caméra et des capteurs pour analyser les places de stationnement. Les données collectées permettent de mettre à jour en temps réel une carte dynamique du parking.
- **Évitement des obstacles** : Grâce à des capteurs ultrasoniques, le robot est capable de détecter et d'éviter les obstacles sur son chemin, garantissant un fonctionnement autonome et sûr.
- **Carte dynamique du parking** : Le robot génère et met à jour en temps réel une carte du parking indiquant les places disponibles et occupées. Ces informations peuvent être affichées sur une interface utilisateur ou intégrées à un système de gestion centralisé.

Cette approche modulaire et économique assure une solution adaptable à différents environnements, tout en offrant des opportunités de développement futur.

2.3 Schéma Explicatif du Fonctionnement :



Le robot commence son parcours à partir de sa position initiale, située dans une zone réservée (R). Il se dirige ensuite vers le parking pour effectuer sa mission de détection des places libres et occupées.

1. **Détection et Évitement d'Obstacles** :
    - Si le robot détecte un obstacle sur son trajet, il active son algorithme d'évitement. Il tourne à gauche pour contourner l'obstacle, puis revient à sa trajectoire initiale une fois l'obstacle dépassé.
2. **Analyse des Places de Parking** :
    - Le parking est organisé en deux colonnes de places : une colonne de places à gauche (notées P1g, P2g, P3g, P4g, P5g) et une colonne de places à droite (notées P1d, P2d, P3d, P4d, P5d). Chaque colonne contient 5 places, soit un total de 10 places à analyser.
    - Le robot s'arrête devant la première ligne de places (ligne 1). Il utilise le servomoteur pour orienter la caméra vers la droite et effectue une détection des places de cette colonne. Il détermine si chaque place est libre ou occupée et met à jour la carte dynamique en conséquence.
    - Ensuite, le robot oriente la caméra vers la gauche pour analyser les places de la colonne gauche. Il met à jour la carte dynamique avec les informations recueillies.
3. **Parcours Complet du Parking** :
    - Après avoir analysé la première ligne, le robot se déplace vers la ligne suivante (ligne 2) et répète le processus de détection pour les colonnes gauche et droite.
    - Ce processus est répété pour chaque ligne jusqu'à la dernière (ligne 5).
4. **Retour à la Position Initiale** :
    - Une fois toutes les places du parking analysées, le robot retourne à sa position initiale (R) et attend la prochaine période de vérification.

3. Matériel Utilisé :

3.1 Châssis Robot à 4 Roues :

Description : Un châssis robuste à 4 roues, conçu pour offrir une stabilité et une mobilité optimales dans un environnement de parking.
Rôle :

- Supporte tous les composants du robot.
- Permet un déplacement fluide et précis dans le parking.
- Facilite les manœuvres (avancer, reculer, tourner).

Spécifications :

- Matériau : Plastique renforcé.
- Moteurs : 4 x moteur à engrenages (1:48)
- Taille: 25x15 cm
- Charge maximale: environ 1 kg
- Quatre roues motrices
- Châssis: 2 châssis en plexiglas
- Courant: 80-100mA
- Vitesse de rotation à vide: 200 m/min

3.2 Driver pour les Moteurs – Contrôleur Shield L293D

Description :
Le Contrôleur Shield L293D est un circuit intégré destiné à contrôler les moteurs à courant continu, en modulant leur vitesse et leur direction. Ce module est couramment utilisé dans les systèmes robotiques pour piloter les moteurs du châssis avec une grande précision et fiabilité.

Rôle :

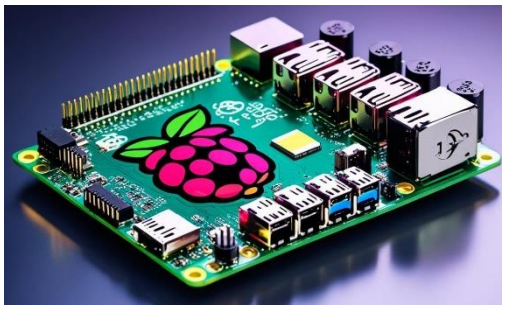
- Reçoit les signaux de commande de l'Arduino et les convertit en actions sur les moteurs (mouvements des roues).
- Permet de contrôler la direction des moteurs (avant/arrière) ainsi que la vitesse grâce à la modulation de largeur d'impulsion (PWM).

Spécifications :

- **Tension d'alimentation :** 4,5V à 36V.
- **Courant de sortie :** Jusqu'à 600 mA par canal (1,2 A en crête).
- **Nombre de moteurs :** Contrôle jusqu'à 4 moteurs à courant continu indépendants.

3.3 Carte Raspberry Pi

Description :
La carte Raspberry Pi 4 est un micro-ordinateur monocarte utilisé comme le "cerveau" du robot, permettant d'exécuter des tâches complexes telles que la navigation, la détection d'obstacles et la gestion de la cartographie.



Rôle :

- Contrôle l'ensemble des capteurs et actionneurs du robot (caméra, LIDAR, servomoteur, etc.).
- Exécute les algorithmes de navigation, de détection et de cartographie en temps réel.
- Gère la communication entre les différents composants matériels et logiciels du robot.

Spécifications :

- **Modèle :** Raspberry Pi 4 (ou Raspberry Pi 3 selon la configuration).
- **Processeur :** Quad-core ARM Cortex-A72 à 1,5 GHz.
- **Mémoire RAM :** 8 Go.
- **Stockage :** Carte microSD (32 Go) pour le système d'exploitation et le stockage des données.

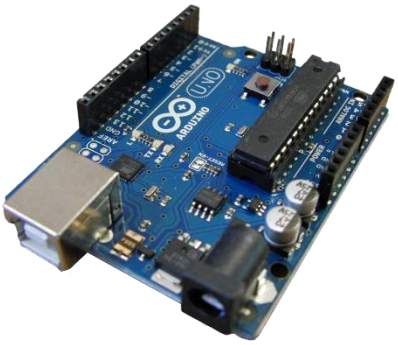
Alimentation : 5V DC via port USB-C.

3.4 Arduino Uno :

Description

L'**Arduino Uno** est une carte microcontrôleur polyvalente utilisée pour gérer les tâches de bas niveau dans le projet. Elle sert d'interface entre les capteurs, les actionneurs et la Raspberry Pi.

Rôle



- **Contrôle des moteurs** : L'Arduino Uno pilote les moteurs via le driver L293D, en envoyant des signaux de direction et de vitesse.
- **Gestion des capteurs** : Il traite les données des capteurs ultrasoniques et contrôle le servomoteur pour orienter la caméra.
- **Communication** : Il synchronise ses actions avec la Raspberry Pi pour assurer un fonctionnement coordonné du robot.

Spécifications

- **Microcontrôleur** : ATmega328P.
- **Tension d'alimentation** : 5V.
- **Entrées/sorties numériques** : 14 broches (dont 6 PWM).
- **Mémoire flash** : 32 Ko pour le stockage du programme.
- **Fréquence d'horloge** : 16 MHz (vitesse de traitement des instructions).

3.5 Caméra Raspberry Pi 3

- **Description** : Une caméra compacte connectée à la Raspberry Pi pour la capture d'images.
- **Rôle** :
    - Capture des images en temps réel pour détecter les places libres et occupées.
    - Utilisée avec des algorithmes de vision par ordinateur pour analyser les images.
- **Spécifications** :
    - Résolution : 8 mégapixels.
    - Connectivité : Interface CSI pour une connexion directe à la Raspberry Pi.
    - Logiciel : Compatible avec OpenCV pour le traitement d'images.

3.6 LIDAR TF Luna :

- **Description** : Un capteur LIDAR (Light Detection and Ranging) pour mesurer les distances avec précision.

- **Rôle** :
    - Détecte les obstacles et mesure les distances pour la navigation.
- **Spécifications** :
    - Portée : Jusqu'à 8 mètres.
    - Précision : ±3 cm.
    - Interface : Communication via UART ou I2C.

3.7 Servomoteur

- **Description** : Un moteur capable de tourner à un angle précis, utilisé pour orienter la caméra ou d'autres capteurs.
- **Rôle** :
    - Oriente la caméra à 180° pour balayer les places de parking à gauche et à droite.
    - Permet une détection complète des places sans déplacer le robot.
- **Spécifications** :
    - Angle de rotation : 0° à 180°.
    - Tension d'alimentation : 5V.
    - Couple : Suffisant pour supporter le poids de la caméra.

3.8. Piles Lithium :

- **Description** : Des batteries rechargeables à haute densité d'énergie.
- **Rôle** :
    - Alimentent le driver pour controller les moteurs.
    - Fournissent une source d'énergie stable et durable.
- **Spécifications** :
    - Tension : 3.7V par cellule (utilisées en série pour obtenir 7.4V).
    - Poids : Léger pour ne pas alourdir le robot.

3.9 Powerbank :

- **Description** : Une batterie portable utilisée pour alimenter la Raspberry Pi.
- **Rôle** :
    - Fournit une alimentation de secours ou supplémentaire.
    - Permet une plus grande autonomie pour le robot.
- **Spécifications** :
    - Capacité : 20000 mAh ou plus.
    - Ports de sortie : USB-C pour connecter la Raspberry Pi.
    - Tension de sortie : 5V pour la Raspberry Pi.

4. Modèle Mathématique et Algorithmes

4.1 Détection des Places et Obstacles

- **Modélisation de la Matrice des Places :**
    - Le parking est représenté par une matrice M où chaque ligne correspond à une rangée de places et chaque colonne représente une place spécifique (gauche ou droite).
    - Les valeurs de la matrice sont binaires :
        - 0 : Place libre.
        - 1 : Place occupée.

$$M = \begin{pmatrix} P5g & P5d \\ P4g & P4d \\ P3g & P3d \\ P2g & P2d \\ P1g & P1d \end{pmatrix}$$

    - La matrice est initialisée à zéro et mise à jour en temps réel lors du parcours du robot.
- **Gestion des Obstacles avec Capteur TF LUNa :**
    - Les capteurs ultrasoniques détectent les obstacles à courte portée (par exemple, un véhicule mal garé ou un objet sur le chemin).
    - Si un obstacle est détecté à une distance inférieure à un seuil prédéfini ($r < 40cm$), le robot s'arrête ($v = 0$, $\omega = 0$).
    - Le robot choisit la direction gauche pour contourner l'obstacle en fonction des données du LIDAR, puis reprend sa trajectoire initiale.

4.2 Algorithme de Détection des Voitures

- **Utilisation de la Caméra pour Identifier les Véhicules** :
  - La caméra Raspberry Pi capture des images en temps réel des places de parking.
  - Des algorithmes de vision par ordinateur (avec OpenCV) sont utilisés pour détecter la présence de véhicules :
    - Analyse des contours et des formes.
    - Détection des couleurs ou des textures spécifiques aux voitures.
  - Si une voiture est détectée, la place correspondante est marquée comme occupée dans la matrice **M**.
- **Mise à Jour de la Carte Dynamique** :
  - Les données capturées par la caméra et les capteurs sont utilisées pour mettre à jour la carte dynamique du parking.
  - La carte est affichée en temps réel, montrant les places libres (en vert) et occupées (en rouge).
  - La mise à jour est continue, permettant une visualisation précise de l'état du parking à tout moment.

5. Programmation

5.1 Langages de Programmation Utilisés

- **C++ pour Arduino** :
  - **Rôle** : Le langage C++ est utilisé pour programmer l'Arduino Uno, qui contrôle les moteurs, les capteurs ultrasoniques et le servomoteur.

- **Python pour Raspberry Pi** :
  - **Rôle** : Le langage Python est utilisé pour programmer la Raspberry Pi, qui gère la caméra, le LIDAR, et la génération de la carte dynamique.

5.2 Bibliothèques et Leur Rôle

Bibliothèques Utilisées

1. **time** :
   - **Rôle** : Gère les délais et les temporisations dans le programme.
   - **Utilisation** : Introduire des pauses entre les actions (par exemple, attendre que le servomoteur termine son mouvement).
2. **RPi.GPIO** :
   - **Rôle** : Permet de contrôler les broches GPIO (General Purpose Input/Output) de la Raspberry Pi.
   - **Utilisation** : Lire les données des capteurs connectés (ultrasoniques, LIDAR) et envoyer des signaux aux actionneurs (moteurs, servomoteur).

3. **serial** :
    - **Rôle** : Facilite la communication série entre la Raspberry Pi et l'Arduino ou d'autres périphériques (comme le LIDAR).
    - **Utilisation** : Envoyer et recevoir des données via les ports série (UART).
4. **signal** :
    - **Rôle** : Gère les signaux système, comme l'arrêt propre du programme lors de l'interruption par l'utilisateur.
    - **Utilisation** : Assurer que le robot s'arrête correctement en cas d'urgence.
5. **sys** :
    - **Rôle** : Fournit des fonctions pour interagir avec le système d'exploitation.
    - **Utilisation** : Gérer les arguments de ligne de commande ou quitter le programme en cas d'erreur.
6. **picamera2** :
    - **Rôle** : Permet de contrôler la caméra Raspberry Pi pour capturer des images.
    - **Utilisation** : Capturer des images en temps réel pour la détection des véhicules.
7. **numpy** :
    - **Rôle** : Fournit des fonctions pour manipuler des tableaux et matrices numériques.
    - **Utilisation** : Traiter les images capturées (par exemple, conversion en niveaux de gris, détection de contours).

6. Mise en Œuvre et Résultats

6.1 Étapes de Conception et de Fabrication

1. **Étude Préliminaire** :
    - **Définition du Projet** : Poser l'idée du projet et identifier les objectifs principaux (détection des places, évitement d'obstacles, génération de carte dynamique).
    - **Fonctionnement Exact** : Définir le fonctionnement global du robot, y compris les interactions entre les composants matériels et logiciels.
2. **Conception** :
    - **Choix des Composants Matériels** :
        - Sélection du châssis à 4 roues pour une stabilité optimale.
        - Choix des moteurs, des capteurs (LIDAR, ultrasoniques, caméra), et des cartes électroniques (Raspberry Pi, Arduino).

- **Conception des Algorithmes** :
  - Développement des algorithmes de détection des places libres/occupées.
  - Conception des algorithmes de navigation et d'évitement d'obstacles.
  - Modélisation de la matrice des places pour la génération de la carte dynamique.

3. Fabrication :

- **Assemblage du Châssis** :
  - Montage des moteurs sur le châssis et installation des roues.
  - Fixation des supports pour les capteurs et les cartes électroniques.
- **Conception du Châssis du Robot sous SolidWorks:**

La conception du châssis du robot a été réalisée sous SolidWorks, un logiciel de CAO, pour modéliser en 3D l'ensemble des composants et optimiser leur disposition. Le châssis a été conçu pour accueillir la Raspberry Pi, l'Arduino, les capteurs (caméra, LIDAR), les moteurs et les batteries, tout en assurant stabilité et modularité. Des supports spécifiques ont été modélisés pour chaque composant, et des simulations de mouvement ont validé la faisabilité technique. Cette étape a permis de générer des plans détaillés pour la fabrication, réduisant les erreurs et garantissant une structure robuste et fonctionnelle.

- **Intégration des Capteurs et Cartes Électroniques** :
  - Installation des capteurs (LIDAR, ultrasoniques, caméra) et des cartes (Raspberry Pi, Arduino).
  - Connexion des composants selon les schémas électriques définis.
- **Vérification des Circuits Électriques** :
  - Test de chaque connexion pour s'assurer de la bonne transmission des signaux.
  - Vérification de l'alimentation électrique pour éviter les surcharges.

4. Programmation :

- **Test des Composants Individuels** :
  - Programmation et test de chaque composant (moteurs, capteurs, caméra) pour vérifier leur bon fonctionnement.
  - Exemple : Test des moteurs avec l'Arduino, test de la caméra avec la Raspberry Pi.
- **Assemblage des Composantes** :
  - Intégration des programmes individuels dans un système unifié.

- Synchronisation des actions entre l'Arduino (contrôle des moteurs et capteurs) et la Raspberry Pi (traitement des images et génération de la carte).

- **Implémentation de l'Algorithme Final** :
  - Déploiement des algorithmes de détection, de navigation, et de cartographie.
  - Test du système complet pour valider le fonctionnement global.

6.2 Limitations dues à l'Absence des Encodeurs

- **Problème** : L'absence d'encodeurs sur les moteurs limite la précision de la localisation du robot.
- **Impact** :
    - **Erreurs de Positionnement** : Sans encodeurs, le robot ne peut pas mesurer avec précision la distance parcourue, ce qui entraîne une accumulation d'erreurs de positionnement au fil du temps.
    - **Navigation Moins Précise** : La localisation dans le parking repose principalement sur les données du LIDAR et des capteurs ultrasoniques, ce qui est moins précis que des encodeurs.

6.3 Tests et Validation des Fonctionnalités

- **Tests des Moteurs** :
    - **Contrôle de Vitesse et Direction** : Les moteurs ont été testés pour vérifier leur capacité à avancer, reculer, et tourner avec précision.
    - **Résultat** : Les moteurs répondent correctement aux commandes, mais la précision du déplacement est limitée sans encodeurs.
- **Tests des Capteurs** :
    - **Capteurs Ultrasoniques** : Testés pour détecter les obstacles à courte portée.
    - **LIDAR** : Testé pour la détection des obstacles à plus longue portée et la localisation du robot.
    - **Caméra** : Testée pour la détection des véhicules dans différentes conditions de luminosité.
    - **Résultat** : Les capteurs fonctionnent comme prévu, mais la détection des véhicules peut être affectée par une faible luminosité.
- **Tests de la Carte Dynamique** :
    - **Mise à Jour en Temps Réel** : La carte est mise à jour en fonction des données capturées par la caméra et les capteurs.
    - **Résultat** : La carte est générée avec succès, mais des erreurs mineures peuvent survenir en raison des limitations de localisation.

6.4 Analyse des Résultats Obtenus

- **Détection des Places** :
    - **Précision** : La caméra et les algorithmes de vision par ordinateur permettent de détecter les véhicules avec une précision satisfaisante dans des conditions normales de luminosité.
    - **Limites** : La détection est moins fiable dans des conditions de faible éclairage ou en présence de reflets.

- **Navigation** :
    - **Évitement d'Obstacles** : Le robot parvient à détecter et à éviter les obstacles grâce aux capteurs ultrasoniques et au LIDAR.
    - **Localisation** : La localisation est moins précise en l'absence d'encodeurs, ce qui affecte la génération de la carte dynamique.
- **Carte Dynamique** :
    - **Fonctionnalité** : La carte est générée en temps réel et mise à jour avec les données capturées.
    - **Précision** : La carte est globalement précise, mais des erreurs peuvent survenir en raison des limitations de localisation.

7. Discussion :

Points Forts

- **Fonctionnalités Complètes** : Le robot intègre avec succès la détection des places libres et occupées, l'évitement d'obstacles, et la génération d'une carte dynamique en temps réel.
- **Modularité** : L'utilisation de la Raspberry Pi et de l'Arduino permet une grande flexibilité dans le développement et l'ajout de nouvelles fonctionnalités.
- **Coût Maîtrisé** : Les composants utilisés sont économiques tout en offrant des performances suffisantes pour le projet.
- **Innovation** : Le projet propose une solution intelligente et automatisée pour améliorer la gestion des parkings, réduisant ainsi le temps de recherche de places et l'impact environnemental.

Limitations et Perspectives d'Amélioration

- **Absence d'Encodeurs** :
    - **Problème** : La localisation du robot est moins précise, ce qui affecte la qualité de la carte dynamique.
    - **Amélioration** : Ajouter des encodeurs pour mesurer avec précision la distance parcourue et améliorer la localisation.
- **Limites du LIDAR** :
    - **Problème** : Le LIDAR est efficace pour la détection d'obstacles, mais il peut être coûteux et énergivore.
    - **Amélioration** : Explorer des alternatives moins coûteuses ou optimiser l'utilisation du LIDAR pour réduire la consommation énergétique.
- **Détection des Véhicules** :
    - **Problème** : La détection par caméra est affectée par les conditions de luminosité.
    - **Amélioration** : Intégrer des capteurs infrarouges ou des algorithmes d'IA pour améliorer la détection dans des conditions de faible éclairage.

8. Conclusion

Synthèse des Réalisations

- Le projet a permis de concevoir et de réaliser un robot mobile autonome capable de détecter les places libres et occupées dans un parking, d'éviter les obstacles, et de générer une carte dynamique en temps réel.
- Les fonctionnalités principales ont été validées avec succès, démontrant la faisabilité d'une solution automatisée pour la gestion des parkings.
- Les algorithmes de détection, de navigation, et de cartographie ont été implémentés avec précision, bien que certaines limitations techniques aient été identifiées.

Impacts Potentiels sur la Gestion des Parkings

- **Efficacité** : Le robot permet une gestion plus efficace des parkings, réduisant le temps de recherche de places pour les conducteurs.
- **Sécurité** : L'évitement d'obstacles et la localisation précise améliorent la sécurité dans les parkings.
- **Environnement** : En optimisant l'utilisation des places, le projet contribue à réduire la circulation et les émissions de gaz à effet de serre.
- **Innovation** : Ce projet ouvre la voie à des solutions intelligentes et automatisées pour la gestion des espaces urbains, avec des applications potentielles dans les centres commerciaux, les aéroports, et les zones résidentielles.

9. Annexes :

Codes Source du Projet

```python
import time
import RPi.GPIO as GPIO
import serial
import signal
import sys

SERVO_PIN = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(SERVO_PIN, GPIO.OUT)
servo = GPIO.PWM(SERVO_PIN, 50)
servo.start(0)

def set_servo_angle(angle):
    duty_cycle = 2 + (angle / 18)
    servo.ChangeDutyCycle(duty_cycle)
    time.sleep(1)
    servo.ChangeDutyCycle(0)

arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(2)

def send_command(command):
    if arduino.is_open:
        arduino.write((command + "\n").encode())
        print(f"Commande envoye : {command}")
    else:
        print("Erreur : le port srie n'est pas ouvert.")

def forward_sequence():
    send_command("FORWARD")
    time.sleep(0.6)
    send_command("STOP")

def stop_sequence():
    send_command("STOP")
    for angle in [0, 90, 180, 90]:
        set_servo_angle(angle)
        time.sleep(2.5)

ser = serial.Serial("/dev/serial0", 115200, timeout=0)

def read_tfluna_data():
    while True:
        counter = ser.in_waiting
        if counter > 8:
            bytes_serial = ser.read(9)
            ser.reset_input_buffer()
            if bytes_serial[0] == 0x59 and bytes_serial[1] == 0x59:
                distance = bytes_serial[2] + bytes_serial[3] * 256
                strength = bytes_serial[4] + bytes_serial[5] * 256
                temperature = bytes_serial[6] + bytes_serial[7] * 256
                temperature = (temperature / 8.0) - 256.0
                return distance / 100.0, strength, temperature
```

```
def signal_handler(sig, frame):
    send_command("STOP")
    print("Arrt des moteurs")
    arduino.close()
    ser.close()
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

def avoid_obstacle():
    send_command("STOP")
    time.sleep(0.5)
    send_command("LEFT")
    time.sleep(0.5)
    send_command("FORWARD")
    time.sleep(1)
    send_command("STOP")
    time.sleep(0.5)
    send_command("RIGHT")
    time.sleep(0.5)
    send_command("FORWARD")
    time.sleep(1)

    send_command("STOP")
    time.sleep(0.5)
    send_command("RIGHT")
    time.sleep(0.5)
    send_command("FORWARD")
    time.sleep(1)
    send_command("STOP")
    time.sleep(0.5)
    send_command("LEFT")
    time.sleep(0.5)
    send_command("FORWARD")
    time.sleep(1)

    send_command("FORWARD")
    time.sleep(2)
```

```
def robot_navigation():
    try:
        last_distance = float('inf')
        while True:
            distance, strength, temperature = read_tfluna_data()
            print(f"Distance: {distance:2.2f} m, Strength: {strength:2.0f} /
65535, Temperature: {temperature:2.1f} C")

            if distance < 0.4:
                print("Obstacle d moins de 40 cm, changement de
direction...")
                avoid_obstacle()

                time.sleep(1)
                while True:
                    new_distance, _, _ = read_tfluna_data()
                    if abs(new_distance - distance) > 0.2:
                        break
                    print("En attente que la zone soit dgage...")
                    time.sleep(0.5)

                last_distance = new_distance
            else:
                send_command("FORWARD")
                time.sleep(0.3)
                last_distance = distance
            time.sleep(0.1)
    except KeyboardInterrupt:
        pass

def main():
    try:
        for _ in range(5):
            forward_sequence()
            stop_sequence()

        robot_navigation()

    finally:
        send_command("STOP")
        if arduino.is_open:
            arduino.close()
        if servo:
            servo.stop()
        GPIO.cleanup()

if _name_ == "_main_":
    main()
```