



Rapport de projet : Robot transporteur

Intervenants :

Bidault Matthias
Colin Frédéric
Sauget Jean max

Sommaire

Présentation du projet :	3
Synoptique :	4
Opérateur Humain :	4
Interface Web (PWA) :	4
Robot transporteur :	4
Microcontrôleur : ESP32 ou raspberry pi pico	4
Modules Capteurs :	4
Modules Actionneurs :	4
Alimentation :	4
Communication :	5
Backend / Serveur de Supervision :	5
Infrastructure Physique :	5
Diagramme des cas d'utilisation :	6
Contraintes Imposées :	7
Suivi de Trajectoire Fiable :	7
Communication Robuste et Temps Réel :	7
Détection et Gestion des Obstacles :	7
Gestion Énergétique :	8
Intégration avec le Backend et le Back Office :	8
Composants utilisés :	9
Le microcontrôleur :	9
Les capteurs de distance :	10
Le Module NFC PN532 :	11
Robot mecanum :	12
Logiciel et application :	13
ClickUp :	13
Proteus :	13
Visual studio code :	14
Thonny :	14
Partie : Frédéric Colin	15
Programmation des capteurs TOF (VL53L0X) et NFC (PN532) :	15
Configuration des Tag NFC :	17
Site internet :	18
Partie : Matthias Bidault	19
Introduction :	19
Assemblage du robot :	19
Code :	20
Evolution :	21

Partie : Jean-Max SAUGET	22
Roues Mecanum	22
Moteurs à courant continu (DC) ou moteurs avec encodeurs	22
Batterie et support batterie	23
Choix microcontrôleur :	24
Réalisation PCB :	25

Présentation du projet :

Ce projet a pour objectif de concevoir un robot transporteur destiné à l'assistance à la manutention de charges. Capable de transporter des charges, ce robot améliore les conditions de travail tout en augmentant la productivité logistique.

Le robot se déplace à une vitesse maximale de 7 km/h, et livre les colis à une position sélectionnée via une interface web. Une fois la livraison effectuée, il retourne de lui-même à une zone de repos située hors des circuits logistiques.

Il est également capable de gérer les croisements avec d'autres robots , de détecter des obstacles et d'en informer l'opérateur via une interface de supervision.

Le système est conçu pour être plug and play, sans besoin de modifier l'infrastructure existante.

Nous avons un budget de 500 euros pour ce projet.

Synoptique :

Le système du robot transporteur peut être représenté sous forme de blocs fonctionnels interconnectés. Voici le détail :

Opérateur Humain :

Interaction avec le robot via :

- Une interface web (sélection du robot, choix de la destination).
- Retour d'informations depuis le robot (état, alertes).

Interface Web (PWA) :

Accessible par un ordinateur. Elle permet de :

- Choisir la destination de livraison.
- Afficher de l'état des robots (position, autonomie, activité)
- Communication avec le backend via MQTT.

Robot transporteur :

Composé de plusieurs sous-systèmes :

Microcontrôleur : ESP32 ou raspberry pi pico

Modules Capteurs :

- **ToF (VL53L0X)** : mesure de distance, détection d'obstacles.
- **PN532 NFC** : lecture de tags NFC placés sur le sol (positionnement, zones de croisement).

Modules Actionneurs :

- moteurs à courant continu avec contrôleurs de vitesse (PWM).
- Gestion des déplacements, changements de direction

Alimentation :

- Module de surveillance de la tension et du % d'autonomie

Communication :

- WiFi via ESP32.
- Protocole MQTT : publication de l'état, réception des commandes.

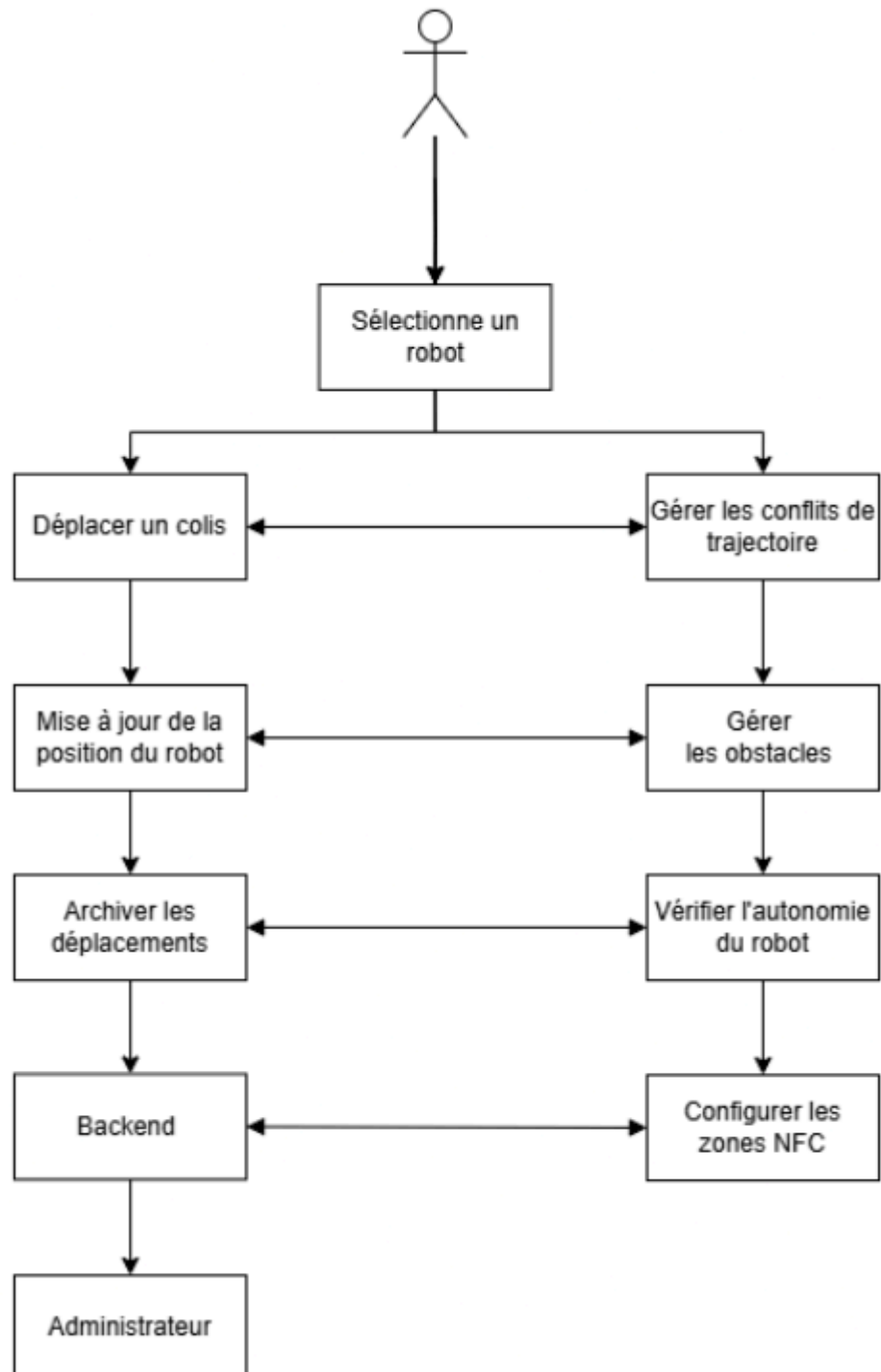
Backend / Serveur de Supervision :

- Broker MQTT (ngrok) : Canal de communication entre interface web et robots.
- Back-office : Visualisation des robots en temps réel et la configuration des zones de livraison et des tags NFC.

Infrastructure Physique :

- Zone de dépôt : point de départ/arrivée des colis.
- Sol balisé avec tags NFC : chaque zone est identifiée par un tag lisible par le robot ainsi que par des lignes noires tracées au sol pour le suivi de ligne .
- Zone d'attente : emplacement hors du chemin de travail.

Diagramme des cas d'utilisation :



Contraintes Imposées :

Le projet du robot transporteur s'inscrit dans un cadre fonctionnel et technique strict afin d'assurer sa fiabilité, sa sécurité, et son efficacité dans un environnement logistique dynamique. Voici les principales contraintes qui ont guidé la conception et le développement du système :

Suivi de Trajectoire Fiable :

Le robot doit pouvoir suivre un tronçon défini sur le sol pour acheminer la commande jusqu'à la destination choisie. Ce suivi doit être précis afin d'éviter tout décalage pouvant entraîner des collisions ou des erreurs de livraison.

- La détection des zones de croisement est cruciale, car le robot doit être capable de changer de direction à ces intersections pour atteindre la bonne position.
- Le robot doit naviguer de manière fluide et continue, avec une vitesse maximale de 7 km/h, adaptée aux conditions de sécurité pour les opérateurs humains à proximité.

Communication Robuste et Temps Réel :

Le système repose sur une communication constante et fiable entre le robot, l'interface web et le backend de supervision.

- Un protocole de communication stable (tel que MQTT) doit être utilisé pour garantir l'échange rapide des commandes, des états, et des alertes.

Détection et Gestion des Obstacles :

Le robot doit être capable d'évaluer en temps réel la présence d'obstacles sur son trajet.

- Les capteurs de distance (ToF) détectent les objets fixes ou mobiles qui pourraient entraver la progression.
- En cas d'obstacle imprévu, le robot doit stopper sa progression et informer immédiatement l'opérateur via l'interface de supervision.
- L'évitement dynamique des obstacles n'est pas obligatoire mais peut être envisagé comme amélioration future.

Ergonomie et Facilité d'Utilisation :

Le système doit être plug and play, c'est-à-dire utilisable sans nécessiter de modifications lourdes sur l'infrastructure ou de compétences techniques avancées chez les opérateurs.

- L'interface web doit être intuitive pour sélectionner le robot et une destination.
- L'ensemble du système doit fonctionner de manière autonome, réduisant la charge cognitive des utilisateurs.

Gestion Énergétique :

La gestion de la batterie est une contrainte clé pour assurer l'autonomie opérationnelle du robot.

- Le système doit surveiller en continu l'état de charge et transmettre cette information à la supervision.
- Des alertes doivent être générées en cas de batterie faible, permettant au robot de retourner à une station de recharge ou de se mettre en mode veille.

Intégration avec le Backend et le Back Office :



Le robot communique avec un backend centralisé qui gère la supervision, la configuration des zones de livraison, et l'archivage des données.

- Le backend doit permettre la configuration des emplacements des tags NFC, essentiels pour la localisation et la navigation.
- La supervision doit offrir une vision en temps réel des positions, états, et alertes de tous les robots en activité.


Composants utilisés :

Le microcontrôleur :

Nous avons sélectionné les micro contrôleurs ESP 32 et le raspberry pi PICO pour qu'on puisse changer à notre guise.

	 ESP32 wroom 32	 Raspberry pi pico
framework	arduino	arduino
GPIO	25	26
mémoire flash	4 Mo	2 Mo
prix	4 - 8 €	4 - 8 €

Les capteurs de distance :

			
	capteur TOF (VL53LXX)	Capteur Lidar (RPLiDAR A3)	Capteur ultrason (HC-SR04)
Principe	Envoyent un faisceau laser et mesurent le temps que met la lumière pour revenir	Balayage laser sur une grande zone, utilisé pour créer des cartes en 2D/3D.	Envoyent une onde sonore et mesurent le temps de retour de l'écho.
Avantages	Très précis, rapide, compacts et un angle de précision de 20° à 30°	Haute précision, utile pour la cartographie	Peu coûteux, pas sensible à la couleur ou à la lumière.
Inconvénients	Plus cher que mes capteurs ultrasons	cher et encombrant	Sensibles à la température, au vent, et aux surfaces molles ou absorbantes
Porté	Quelques cm à plusieurs mètres	environs 100 mètres et plus	2 cm à 5 m
Prix	4 - 6 €	400 à 500€	1 - 3 €

Le Module NFC PN532 :

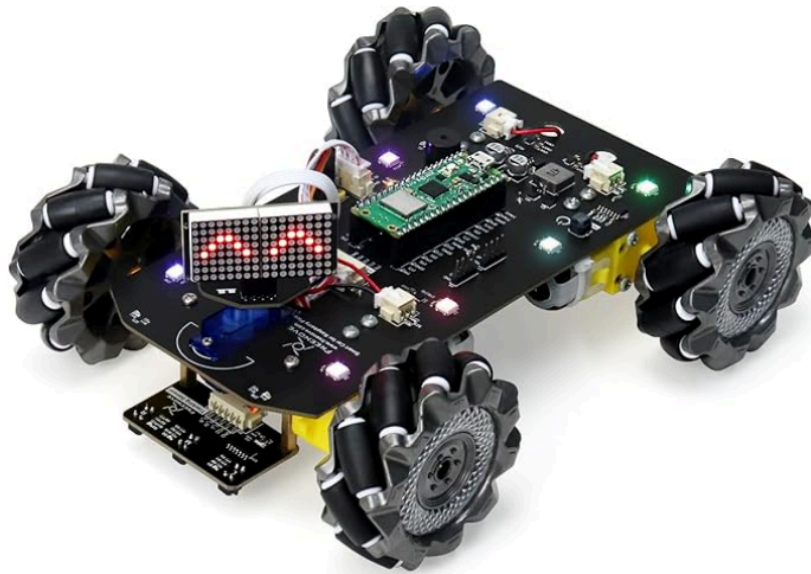
La technologie NFC est utilisée pour la localisation et la navigation du robot. Le module PN532 lit des tags NFC placés stratégiquement sur le sol ou les zones d'intérêt (zones de croisement, zones de dépôt, zone d'attente).

- Cette localisation par tags permet au robot d'identifier précisément sa position, de détecter les intersections et de valider les étapes de son parcours.
- Il nous a permis aussi de programmer des tags nfc pour le suivi de parcours.



Robot mecanum :

Nous avons pris un robot déjà réalisé pour nous permettre de gagner du temps sur le projet.



Logiciel et application :

ClickUp :

ClickUp est une plateforme tout-en-un de gestion de projet et de productivité qui permet aux équipes d'organiser leurs tâches, documents, communications et suivis de temps en un seul endroit. Elle propose des vues multiples (liste, Kanban, Gantt), des automatisations, des intégrations avec d'autres outils, et s'adapte à divers modes de travail (Scrum, Agile, etc.).



Proteus :

Proteus est un logiciel de CAO électronique (Conception Assistée par Ordinateur) utilisé pour concevoir, simuler et tester des circuits électroniques. Il permet de créer des schémas électriques, de concevoir des circuits imprimés (PCB) et de simuler le fonctionnement de microcontrôleurs (comme Arduino ou PIC) avec leurs composants associés. Très utilisé en électronique embarquée et en formation, il combine à la fois dessin, simulation et visualisation.



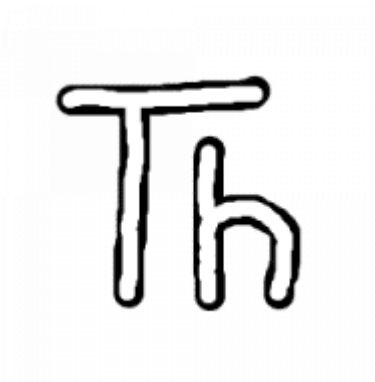
Visual studio code :

Visual Studio Code (Visual SC) est un éditeur de code source léger et gratuit développé par Microsoft. Il prend en charge de nombreux langages de programmation, propose l'autocomplétion intelligente, le debugging intégré, la gestion Git, et une large bibliothèque d'extensions. Polyvalent et personnalisable, il est largement utilisé pour le développement web, logiciel et embarqué.



Thonny :

Thonny est un environnement de développement intégré (IDE) léger pour Python, conçu spécialement pour les débutants. Il offre une interface simple, un débogueur visuel, l'affichage des variables, et une installation Python intégrée. Idéal pour l'apprentissage, il permet aussi de programmer des microcontrôleurs comme Raspberry Pi Pico grâce au support de MicroPython.



Partie : Frédéric Colin

Programmation des capteurs TOF (VL53L0X) et NFC (PN532) :

Ce programme permet de lire des tags nfc et d'avoir une lecture des 4 capteurs TOF en continu.

J'ai utilisé les librairies Adafruit_PN532.h et Adafruit_VL53L0X pour les capteurs.

```
#include <Wire.h>
#include <Adafruit_PN532.h>
#include <Adafruit_VL53L0X.h>
```

Initialisation des modules TOF et NFC.

```
digitalWrite(XSHUT_4, HIGH);
delay(20);
if (!sensor4.begin(0x33)) {
  Serial.println("Erreur capteur 4");
  while (1);
}
Serial.println("4 capteurs VL53L0X initialisés !");
```

```
uint32_t versiondata = nfc.getFirmwareVersion();
if (!versiondata) {
  Serial.println("PN532 non détecté !");
  while (1);
}

nfc.SAMConfig(); // Configure le PN532 en mode lecteur
Serial.println("Prêt à scanner une carte NFC...");
```

```
4 capteurs VL53L0X initialisés !
Prêt à scanner une carte NFC...
Capteur 1 : 24 mm
Capteur 2 : 59 mm
Capteur 3 : 29 mm
Capteur 4 : 25 mm
```

Mesure des distance en mm

```
sensor4.rangingTest(&measure, false);
uint16_t distance_mm4 = measure.RangeMilliMeter;
Serial.print("Capteur 4 : ");
Serial.println((measure.RangeStatus != 4) ? String(distance_mm4) + " mm" : "Hors portée");
```


Lecture des tags NFC ainsi que l'affichage de UID (unique identifier) et le nom du tag nfc inscrit dans le bloc 4.

```
if (nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength)) {  
    Serial.println("\nCarte NFC détectée !");  
  
    // Afficher l'UID  
    Serial.print("UID : ");  
    for (uint8_t i = 0; i < uidLength; i++) {  
        Serial.print(uid[i], HEX); Serial.print(" ");  
    }  
    Serial.println();  
  
    // Lecture bloc 4 du tag MIFARE Classic  
    uint8_t blockNumber = 4;  
    uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
```

Il y aura une pause de 0,2 secondes entre chaque de lecture de tag.

```
Carte NFC détectée !  
UID : A2 E4 44 67  
Contenu du bloc 4 : A3.....  
Pause de 0,2 secondes pour NFC  
Capteur 1 : 24 mm  
Capteur 2 : 61 mm  
Capteur 3 : 33 mm  
Capteur 4 : 23 mm
```

```
Carte NFC détectée !  
UID : 62 B6 3A 67  
✗ Échec de l'authentification du bloc 4 !  
Pause de 0,2 secondes pour NFC  
Capteur 1 : 19 mm  
Capteur 2 : 56 mm  
Capteur 3 : 31 mm  
Capteur 4 : 25 mm
```

Configuration des Tag NFC :

Ce programme permet d'écrire dans le bloc 4 le nom du tag nfc qui nous permettra de configurer un trajet via le site internet.

Exemple : REPOS, A1 > C3

Détection du PN532 :

```
void setup() {
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);
  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    Serial.println("PN532 non détecté !");
    while (1);
  }
}
```

Lecture d'un tag (UID) et écriture dans le bloc 4 :

```
nfc.SAMConfig();
Serial.println(" Écriture de 'REPOS' dans le bloc 4 ");
}

void loop() {
  Serial.println("Approchez un tag...");

  uint8_t uid[7];
  uint8_t uidLength;

  if (nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength)) {
    Serial.print("Tag détecté - UID : ");
    for (uint8_t i = 0; i < uidLength; i++) {
      Serial.print(uid[i], HEX); Serial.print(" ");
    }
    Serial.println();

    uint8_t blockNumber = 4;
```

Exemple : écriture de repos dans le bloc 4 avec confirmation.

```
uint8_t data[16] = {0};
const char* texte = "REPOS";
for (int i = 0; i < strlen(texte); i++) {
  data[i] = texte[i];
}

if (nfc.mifareclassic_WriteDataBlock(blockNumber, data)) {
  Serial.println("✅ Écriture réussie !");
} else {
  Serial.println("❌ Erreur d'écriture !");
}

delay(3000);
}
```

```
Écriture de 'REPOS' dans le bloc 4
Approchez un tag...
Tag détecté - UID : 62 B6 3A 67
✅ Écriture réussie !
Approchez un tag...
```

Site internet :

Le site permet de sélectionner un point de départ et un point d'arrivée. Le trajet du robot est visualisable grâce à l'affichage des cases rouges. À la fin de chaque trajet, si aucun nouveau point d'arrivée n'est sélectionné dans un délai de 10 secondes, le robot retourne automatiquement dans une zone d'attente prévue à cet effet.

Morpion - Sélection de Cases

A1	A2	A3
B1	B2	B3
C1	C2	C3

Zone d'attente

Zone d'attente

Lancer le trajet

Niveau de batterie : 78%

Sélectionnez une case de départ et une case d'arrivée.

Partie : Matthias Bidault

Introduction :

En ce qui concerne le sujet sur lequel mon équipe et moi avons travaillé, on peut dire qu'au niveau de mon implication, je m'occupais de la partie programmation de notre projet de robot transporteur.

La finalité du projet était que le robot puisse se déplacer sur une ligne noire qui était tracée au sol tout en étant capable d'interroger des badges RFID placés à certains points sur son parcours.

Ces deux fonctionnalités sont des préalables à l'autonomie du robot, et à sa capacité à interagir avec son environnement.

Assemblage du robot :

Avant la phase de programmation, j'ai participé au montage complet du robot. Cela comprenait l'installation des moteurs, des roues omnidirectionnelles, des capteurs infrarouges, du lecteur RFID, ainsi que le câblage sur la carte Arduino.

Cette étape m'a permis de bien comprendre la structure du robot et d'anticiper certaines contraintes techniques pour la suite du projet.

Code :

```
void initialiser_moteurs() {
    pinMode(M1_IN1, OUTPUT); pinMode(M1_IN2, OUTPUT);
    pinMode(M2_IN1, OUTPUT); pinMode(M2_IN2, OUTPUT);
    pinMode(M3_IN1, OUTPUT); pinMode(M3_IN2, OUTPUT);
    pinMode(M4_IN1, OUTPUT); pinMode(M4_IN2, OUTPUT);

    // Assurer que tous les moteurs sont arrêtés au démarrage
    analogWrite(M1_IN1, 0); analogWrite(M1_IN2, 0);
    analogWrite(M2_IN1, 0); analogWrite(M2_IN2, 0);
    analogWrite(M3_IN1, 0); analogWrite(M3_IN2, 0);
    analogWrite(M4_IN1, 0); analogWrite(M4_IN2, 0);
}
```

Cette fonction permet d'initialiser les broches des moteurs en sortie et de s'assurer qu'ils sont bien arrêtés au démarrage. Cela évite des mouvements imprévus dès la mise sous tension du robot.

```
if (capteurGauche == NON_DETECTE && capteurMilieu == DETECTE && capteurDroit == NON_DETECTE) {
    if (etatActuel != dernierEtatCapteurs && Serial) Serial.println("AVANCE");
    avancerToutDroit();
}
else if ((capteurGauche == NON_DETECTE && capteurDroit == DETECTE) ||
         (capteurMilieu == DETECTE && capteurDroit == DETECTE)) {
```

Le robot utilise trois capteurs infrarouges pour suivre une ligne noire. Selon l'état des capteurs (gauche, milieu, droite), il ajuste sa trajectoire en appelant des fonctions de correction. En cas de perte totale de la ligne, un délai de récupération est prévu avant l'arrêt de sécurité.

```
if (nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength, 50)) {
    if (Serial) {
        Serial.print("[NFC] Tag: ");
        for (uint8_t i = 0; i < uidLength; i++) {
            if (uid[i] < 0x10) Serial.print("0");
            Serial.print(uid[i], HEX);
            Serial.print(" ");
        }
    }
}
```

Chaque badge contient un identifiant unique (UID), et les données sont lues sur un bloc spécifique après authentification. Ces informations peuvent être utilisées pour déclencher des comportements conditionnels du robot.

```
void corrigerGaucheLent() {  
    analogWrite(M1_IN1, 0);  
    analogWrite(M2_IN1, VITESSE_CORRECTION);  
    analogWrite(M3_IN1, 0);  
    analogWrite(M4_IN1, 0);  
    analogWrite(M1_IN2, VITESSE_CORRECTION);  
    analogWrite(M2_IN2, 0);  
    analogWrite(M3_IN2, VITESSE);  
    analogWrite(M4_IN2, VITESSE);  
}
```

```
void loop() {  
    suiviligne(); // Priorité au suivi de ligne  
  
    unsigned long maintenant = millis();  
    if (maintenant - derniereLectureNFC >= INTERVALLE_NFC) {  
        lireNFC();  
        derniereLectureNFC = maintenant;  
    }  
  
    delay(10); // Délai minimal pour stabilité  
}
```

La boucle principale donne la priorité au suivi de ligne, qui est essentiel au déplacement. La lecture NFC est effectuée à intervalles réguliers pour ne pas perturber la stabilité du mouvement.

Evolution :

Je suis finalement partie sur Thonny, car pour mettre en place le serveur web, la gestion des librairies était beaucoup plus simple, notamment avec le Raspberry Pi Pico que j'utilise comme microcontrôleur.

Thonny est particulièrement adapté pour ce type de carte, ce qui m'a permis d'installer les modules nécessaires sans difficulté et d'avancer plus efficacement sur le projet.

```
def tourner_vers_direction(direction_cible):
    global direction_actuelle
    if direction_actuelle == direction_cible:
        return
    index_actuel = directions.index(direction_actuelle)
    index_cible = directions.index(direction_cible)
    diff = (index_cible - index_actuel) % 4
    if diff == 1:
        print(f"[NAVIGATION] Tourne à droite de {direction_actuelle} vers {direction_cible}")
        tourne_droite()
    elif diff == 3:
        print(f"[NAVIGATION] Tourne à gauche de {direction_actuelle} vers {direction_cible}")
        tourne_gauche()
    elif diff == 2:
```

Cette fonction `tourner_vers_direction(direction_cible)` permet d'orienter un robot dans une direction donnée (`direction_cible`) en fonction de sa direction actuelle (`direction_actuelle`).

Elle compare la direction actuelle à la direction cible à l'aide de leurs positions dans une liste de directions (par exemple : `["nord", "est", "sud", "ouest"]`).

```
def calculer_chemin(depart, arrivee):
    if depart == arrivee:
        return []
    pos_depart = grille_positions[depart]
    pos_arrivee = grille_positions[arrivee]
    chemin = []
    pos_actuelle = pos_depart
    while pos_actuelle != pos_arrivee:
        if pos_actuelle[1] < pos_arrivee[1]:
            direction_necessaire = "EST"
            pos_actuelle = (pos_actuelle[0], pos_actuelle[1] + 1)
        elif pos_actuelle[1] > pos_arrivee[1]:
            direction_necessaire = "OUEST"
            pos_actuelle = (pos_actuelle[0], pos_actuelle[1] - 1)
        elif pos_actuelle[0] < pos_arrivee[0]:
            direction_necessaire = "SUD"
            pos_actuelle = (pos_actuelle[0] + 1, pos_actuelle[1])
        elif pos_actuelle[0] > pos_arrivee[0]:
            direction_necessaire = "NORD"
            pos_actuelle = (pos_actuelle[0] - 1, pos_actuelle[1])
        chemin.append(direction_necessaire)
    return chemin
```

La fonction `avancer_si_possible()` permet au robot d'avancer d'une case.

Elle regarde dans quelle direction le robot est tourné (nord, sud, est ou ouest), puis elle vérifie si la case devant lui est dans la grille et libre.

Si c'est le cas, le robot avance d'une case dans cette direction en mettant à jour sa position.

```
def executer_chemin():
    global chemin_en_cours, index_chemin, attend_commande, direction_actuelle
    if not chemin_en_cours or index_chemin >= len(chemin_en_cours):
        print("[NAVIGATION] Trajet terminé!")
        attend_commande = False
        chemin_en_cours = []
        index_chemin = 0
        return
    direction_cible = chemin_en_cours[index_chemin]
    print(f"[NAVIGATION] Étape {index_chemin + 1}/{len(chemin_en_cours)}: {direction_cible}")
    tourner_vers_direction(direction_cible)
    direction_actuelle = direction_cible # Correction ici
    avancer_intersection()
    index_chemin += 1
    if index_chemin < len(chemin_en_cours):
        print("[NAVIGATION] Prochaine étape dans 2 secondes...")
    else:
        print("[NAVIGATION] Destination atteinte!")
        attend_commande = False
        chemin_en_cours = []
        index_chemin = 0
```

La fonction `executer_chemin()` permet au robot de suivre un itinéraire. Elle commence par vérifier si le chemin est terminé.

Si c'est le cas, elle réinitialise les variables de navigation et indique que le trajet est fini.

Sinon, elle récupère la prochaine direction à suivre, oriente le robot dans cette direction grâce à la fonction `tourner_vers_direction()`, puis le fait avancer jusqu'à la prochaine intersection. À chaque étape, elle affiche des informations sur la progression.

```
def lire_nfc():
    global nfc_en_cours, attend_commande, dernier_tag_lu, position_actuelle
    if nfc_en_cours:
        return
    nfc_en_cours = True
    uid = nfc.read_passive_target(timeout=0.05)
    if uid:
        uid_str = ":".join([f"{i:02X}" for i in uid])
        tag_name = uid_to_tag.get(uid_str, f"Inconnu ({uid_str})")
        print(f"[NFC] UID: {uid_str} → {tag_name}")
        stop_all()
        if tag_name in grille_positions:
            position_actuelle = tag_name
            print(f"[POSITION] Position mise à jour: {position_actuelle}")
            attend_commande = True
            dernier_tag_lu = tag_name
        nfc_en_cours = False
```

La fonction `lire_nfc()` détecte un badge NFC, récupère son identifiant, puis met à jour la position du robot si le badge correspond à une position connue. Elle arrête les mouvements et prépare le robot à recevoir une nouvelle commande.


```
def suivi_ligne():
    global dernier_etat_capteurs
    cap_g = CAPTEUR_GAUCHE.value
    cap_m = CAPTEUR_MILIEU.value
    cap_d = CAPTEUR_DROIT.value
    pattern = (cap_g << 2) | (cap_m << 1) | cap_d
    if pattern != dernier_etat_capteurs:
        print(f"[CAPTEURS] Pattern: {bin(pattern)} → ", end='')
        if pattern == 0b010:
            print("AVANCE")
            avancer()
        elif pattern in [0b001, 0b011]:
            print("CORRECTION GAUCHE")
            corriger_gauche_lent()
        elif pattern in [0b100, 0b110]:
            print("CORRECTION DROITE")
            corriger_droite_lent()
        elif pattern == 0b111:
            print("INTERSECTION")
            avancer()
        elif pattern == 0b000:
            print("PLUS DE LIGNE - STOP")
            stop_all()
        else:
            print("ÉTAT INCONNU")
    dernier_etat_capteurs = pattern
```

La fonction `suivi_ligne()` lit les capteurs pour détecter la ligne. Selon le signal reçu, elle fait avancer, corrige la trajectoire à gauche ou à droite, détecte une intersection ou s'arrête si la ligne est perdue.

```
# --- Point d'accès Wi-Fi ---
ssid = "RobotTransporteur"
password = "12345678"
print("Démarrage point d'accès Wi-Fi...")
wifi.radio.start_ap(ssid=ssid, password=password)
time.sleep(2)
print("Point d'accès actif:", ssid)

pool = socketpool.SocketPool(wifi.radio)
server = pool.socket(pool.AF_INET, pool.SOCK_STREAM)
server.bind(("0.0.0.0", 80))
server.listen(5)
server.settimeout(None)

print("Serveur HTTP démarré")
```

Cette partie du code crée un point d'accès Wi-Fi appelé « RobotTransporteur » avec le mot de passe « 12345678 ».

Ensuite, il démarre un petit serveur web sur le port 80, prêt à recevoir des connexions. Ce serveur permettra de communiquer avec le robot via le réseau Wi-Fi créé.

```

destination = None
if "GET /?destination=" in request:
    start = request.find("destination=") + 12
    end = request.find(" ", start)
    if end == -1:
        end = request.find("&", start)
    if end == -1:
        end = len(request)
    destination = request[start:end]

if destination and destination in grille_positions:
    print(f"[COMMANDE] Navigation vers {destination} depuis {position_actuelle}")
    chemin_en_cours = calculer_chemin(position_actuelle, destination)
    index_chemin = 0
    print(f"[CHEMIN] Calculé: {chemin_en_cours}")
    attend_commande = False

action = None
if "GET /?action=left" in request:
    action = "left"
elif "GET /?action=right" in request:
    action = "right"
elif "GET /?action=forward" in request:
    action = "forward"

if attend_commande and action:
    if action == "left":
        print("Commande manuelle : tourner à gauche")
        tourne_gauche()
    elif action == "right":
        print("Commande manuelle : tourner à droite")
        tourne_droite()
    elif action == "forward":
        print("Commande manuelle : avancer")
        avancer_intersection()
    attend_commande = False

```

Le programme fonctionne en boucle pour gérer le robot. Quand un chemin est en cours et qu'il n'attend pas de commande, il avance selon ce chemin. Sinon, il suit la ligne au sol et lit les tags NFC pour connaître sa position.

```

destination = None
if "GET /?destination=" in request:
    start = request.find("destination=") + 12
    end = request.find(" ", start)
    if end == -1:
        end = request.find("&", start)
    if end == -1:
        end = len(request)
    destination = request[start:end]

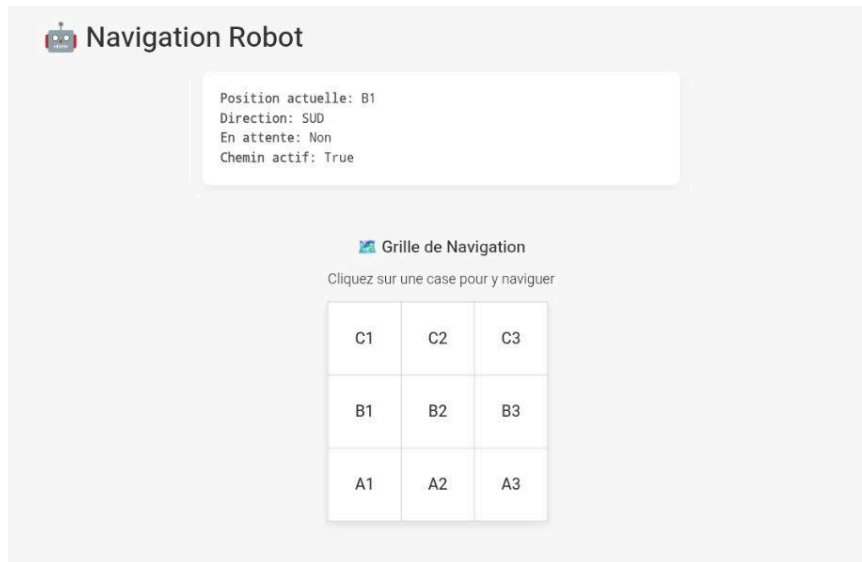
if destination and destination in grille_positions:
    print(f"[COMMANDE] Navigation vers {destination} depuis {position_actuelle}")
    chemin_en_cours = calculer_chemin(position_actuelle, destination)
    index_chemin = 0
    print(f"[CHEMIN] Calculé: {chemin_en_cours}")
    attend_commande = False

action = None
if "GET /?action=left" in request:
    action = "left"
elif "GET /?action=right" in request:
    action = "right"
elif "GET /?action=forward" in request:
    action = "forward"

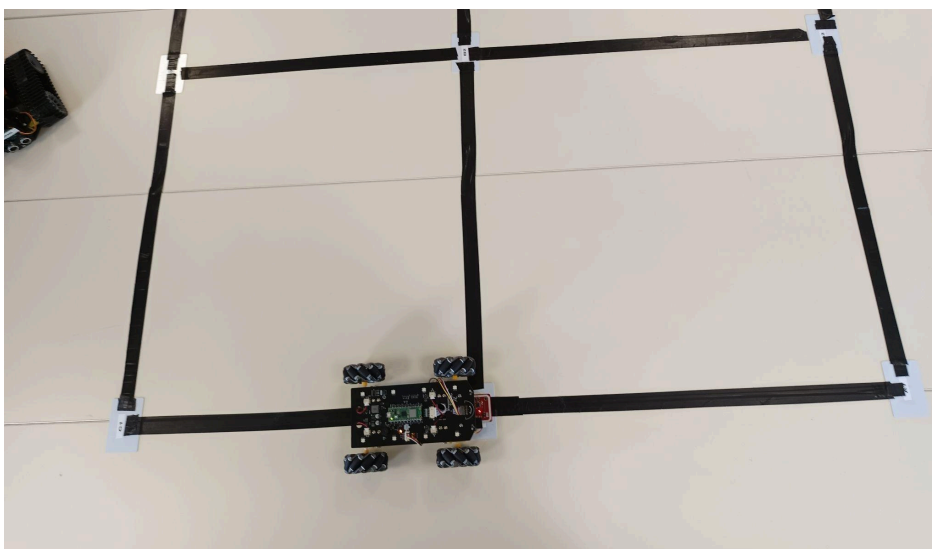
if attend_commande and action:
    if action == "left":
        print("Commande manuelle : tourner à gauche")
        tourne_gauche()
    elif action == "right":
        print("Commande manuelle : tourner à droite")
        tourne_droite()
    elif action == "forward":
        print("Commande manuelle : avancer")
        avancer_intersection()
    attend_commande = False

```

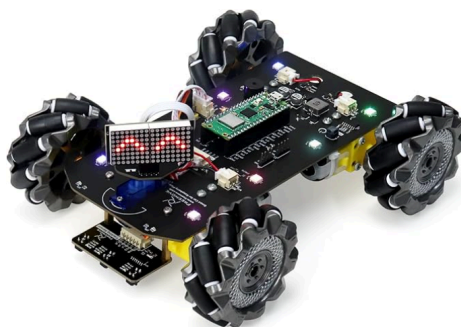
En parallèle, il écoute les connexions au serveur web. Lorsqu'un client envoie une requête, le robot peut répondre avec son état actuel, calculer un nouveau chemin vers une destination demandée



Grâce à mon code, j'ai créé une nouvelle interface en HTML accessible via le serveur web du robot. Cette interface permet de visualiser l'état du robot en temps réel (position, direction, etc.) et d'envoyer des commandes à distance, comme choisir une destination



Ainsi, selon l'interface HTML, on peut voir en temps réel où se situe le robot sur la grille.



Partie mécanique :

Roues Mecanum

Les roues Mecanum sont la caractéristique principale de ce robot. Ce sont des roues omnidirectionnelles qui permettent un mouvement dans toutes les directions sans tourner le châssis.

- **Structure :**
 - Chaque roue est équipée de plusieurs galets montés en diagonale (45°) par rapport à l'axe de la roue.
 - Le montage des galets sur les roues permet des vecteurs de force orientés, rendant possible les déplacements latéraux.
- **Fonctionnement :**
 - Selon le sens de rotation de chaque roue, le robot peut avancer, reculer, tourner sur place ou se déplacer en diagonale/latéralement.
 - Le contrôle des roues est souvent assuré via des algorithmes de cinématique inverse.

Moteurs à courant continu (DC) ou moteurs avec encodeurs

Chaque roue Mecanum est fixée à un moteur DC ou à un moteur avec encodeur (pour un meilleur contrôle).

- **Caractéristiques :**

- Tension typique : 6–12 V

- Couple suffisant pour supporter le poids total du robot
- Les moteurs peuvent être à réduction pour augmenter le couple au détriment de la vitesse
- **Fixation :**
 - Les moteurs sont fixés sur les plaques latérales du châssis avec des supports métalliques ou des pinces motorisées.
 - Chaque moteur est connecté à une roue Mecanum par un axe ou un moyeu.

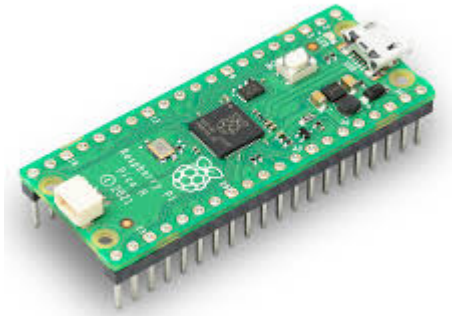
Batterie et support batterie

Une batterie rechargeable (souvent Li-ion ou LiPo) alimente les moteurs et les composants électroniques.

- Le support est souvent fixé au centre du châssis pour équilibrer le poids.
- Un interrupteur général permet d'activer ou désactiver le robot.

Elle nous permet donc de rajouter les capteurs dont on a besoin.

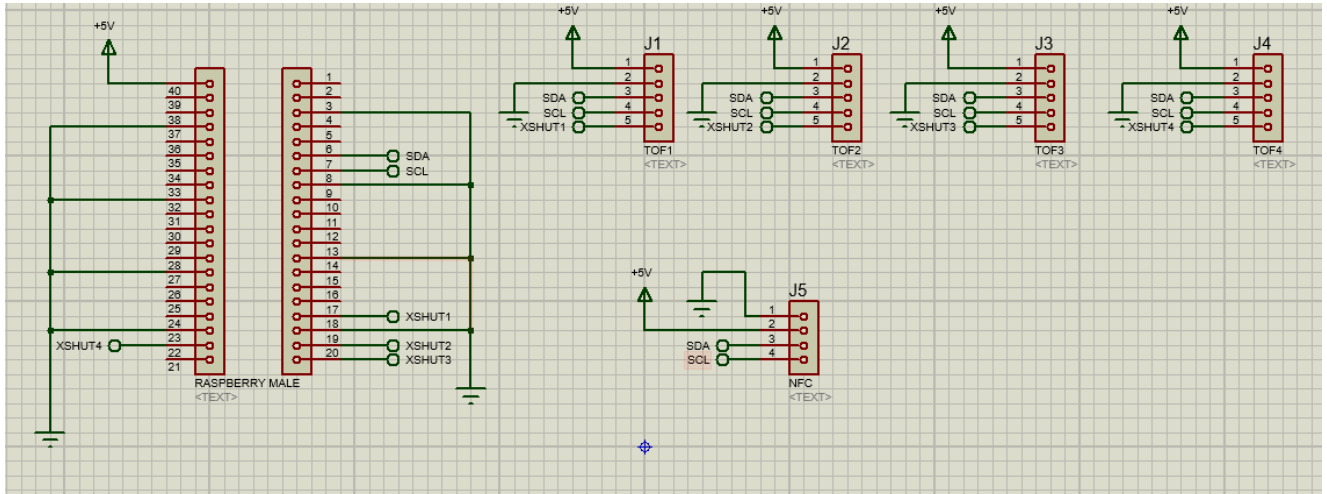
Choix microcontrôleur :



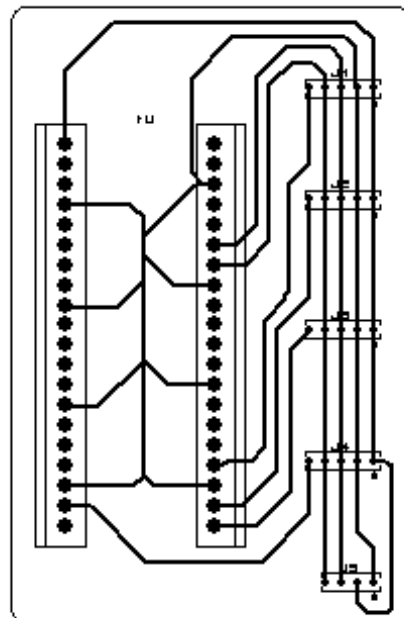
Nous avons choisi d'utiliser le raspberry Pi pico car il est directement implanté sur le robot et nous permet de faire un PCB en format "shield" à implanter directement sur le robot sans interfaçage raspberry pi pico pour un autre micro contrôleur.

Réalisation PCB :

Etant donné que pour nos besoins on a ajouté 4 capteurs ToF et un capteur NFC notre schéma structurel ressemble donc à ça.



Pour des soucis de facilité et de gain de temps notre PCB prendra la forme d'un shield à ajouter sur le PCB déjà existant du robot.



Les machines qui nous permettent d'usiner notre PCB nous permet seulement de faire une carte en simple face

Les pistes sont en 25 centièmes.