EraDev

RE12 - Rapport du prototype

Projet location de voiture sans conducteur

Table des matières

Introduction

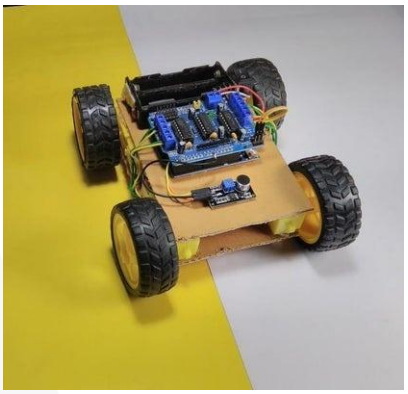
Dans le cadre du projet 2CSSIL : projet location de voiture sans conducteur, il nous a été demandé de réaliser un simulateur d'une voiture autonome avec Arduino et Raspberry.

Notre équipe ERADEV propose son prototype de voiture 'DoobyCar' qui est sensée être autonome et intelligente afin de parcourir des trajets demandés par le client sans avoir besoin d'un conducteur humain.

Notre vision au cours du développement de ce prototype était de présenter des versions incrémentales de notre produit pour mettre en avant chaque fonctionnalité à part. Nous avons conçu plusieurs solutions pour les divers problèmes rencontrés, ces solutions ont fait appel à différents outils et technologies que nous allons exposer à travers ce rapport.

DoobyCar X0

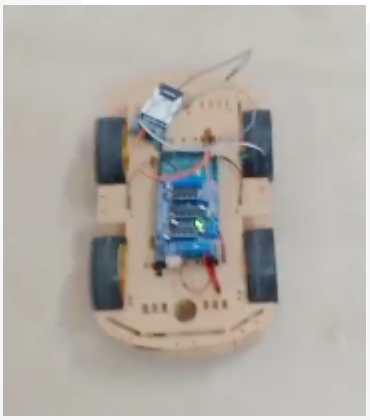
Naturellement, notre processus de conception a commencé avec les objectifs les plus basiques : Le suivi d'un trajet fixe à travers un script qui suit un langage et une grammaire précise (à détailler plus tard). Nous avons réalisé **DoobyCar X0** avec un châssis de voiture simple à 4 roues, 4 moteurs, un shield moteur et Arduino UNO. Le script est reçu à partir du Serial vers Arduino UNO pour passer vers un compilateur écrit en C.



Maintenant afin de libérer le véhicule de cette lecture à partir du Serial, on va passer à une version plus adaptée.

DoobyCar X1

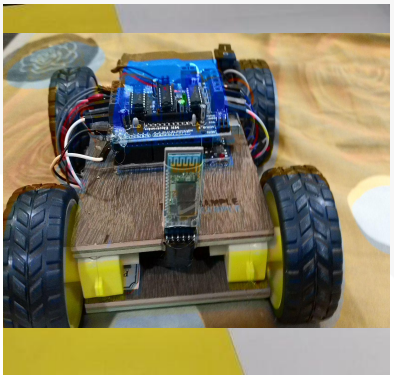
Avec cette version, nous avons remplacé Arduino UNO par Arduino MEGA pour l'avantage d'avoir plus de pins, et ainsi plus d'espace pour intégrer le module de la carte SD qui contient le fichier script en D#.



En conséquence, nous allons nous libérer du Serial et lire directement le script du trajet à partir de la carte SD. Néanmoins, cette solution présente un

inconvénient lors du changement de trajet. La carte SD doit être retirée pour modifier le fichier script et réinsérée pour la lecture.

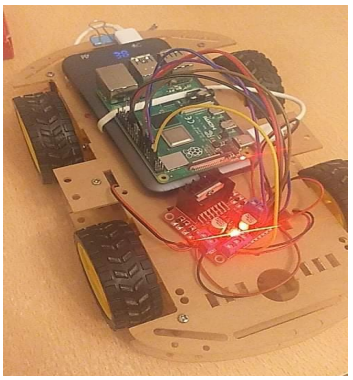
On passe ainsi à un modèle plus flexible qui utilise Bluetooth :



DoobyCar X2

Ce modèle intègre un module Bluetooth pour une communication plus fluide qui assure la facilité et flexibilité de changer un script du trajet.

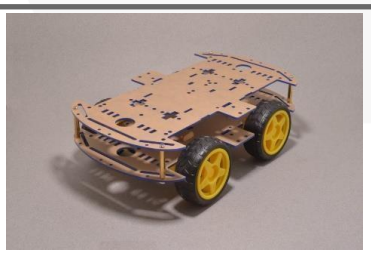
DoobyCar Y1

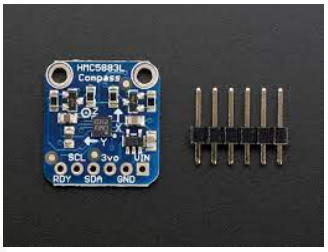


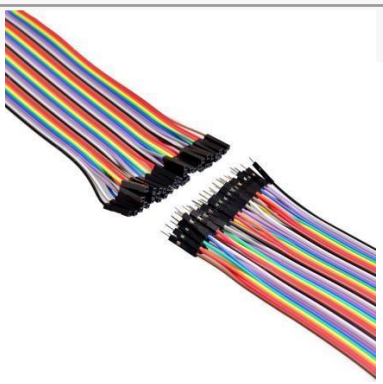
Pour pouvoir répondre aux autres objectifs du projet, il est préférable de prévoir une puissance de calcul supérieure à celle de Arduino. Nous avons donc fait appel à Raspberry Pi 4 Model B pour être le cerveau de DoobyCar. Ce fut un énorme changement dans les composants du véhicule (type d'alimentation, contrôleur des moteurs, conception du schéma et des programmes). D'ailleurs, nous avons ré-implémenté le compilateur du langage D# mais cette fois-ci en Python. (Code source inclus en annexe)

- A travers cette évolution, on a pu comparer toutes les solutions réalisables et choisir la meilleure pour notre cas : le modèle **Y1** à lequel nous allons ajouter le reste des fonctionnalités que nous allons détailler par la suite.

I. Liste des composants

Equipement	Description	Photo	Source
Châssis avec les roues	Le corps du véhicule		Enseignant
Moteurs	Pour le mouvement des roues		Enseignant
SparkFun Dual H-Bridge motor drivers L298	Contrôleur de moteurs		Achat personnel
Raspberry pi 4 modèle B 4 Go	Nano-ordinateur monocarte à processeur		FabLab
Capteur de distance à ultrasons	Mesurer la distance entre la voiture et un obstacle		Achat personnel

Carte SD 16GB	Pour Installation d'un système d'exploitation pour raspberry		FabLab
GPS module	La Géolocalisation du véhicule		FabLab
HMC5338l	magnétomètre à 3 axes		Achat personnel
Caméra USB			Achat personnel
Power bank (5V/3A)	L'alimentation du raspberry		Achat personnel
4 aa batteries lithium en série	L'alimentation des moteurs		Achat personnel

LED à deux couleurs			Enseignant
Câbles	Connecter les équipements		Enseignant

II. Liste des fonctionnalités intégrées

1. Suivi d'un trajet

a. Grammaire et Script D#

Pour écrire un trajet à suivre, nous avons proposé un langage qui s'appelle D#, avec une nature et syntaxe qui ressemblent à celle du langage anglais naturel. Voici les instructions principales de D# :

Forward arg ~~~~ pour avancer avec une vitesse de arg.
Backward arg ~~~~ pour reculer avec une vitesse de arg.
During arg ~~~~ pour garder l'état actuel du véhicule pour une durée arg.
Right arg ~~~~ pour tourner à droite avec une intensité de arg.
Left arg ~~~~ pour tourner à gauche avec une intensité de arg.

Les règles de grammaire de D# :

```
<D#Program> ::= <D#Program> <D#Instruction> | epsilon
<D#Instruction> ::= Forward <D#arg> |
                    Backward <D#arg> |
                    Right <D#arg>    |
                    Left <D#arg>     |
                    During <D#arg>   |
                    Stop             |
<D#arg> ::= _INTEGER || _FLOAT
```

Parser du langage :

1. Parser en C :

   Utilisé avec Arduino, basé sur des conditions if else et une simple manipulation des chaînes de caractères en C.

2. Parser en Python :

Utilisé avec Raspberry Pi, écrit avec SLY (Sly Lex Yacc), effectue l'analyse syntaxique des programmes d'extension '.d', signale les erreurs de syntaxe et transforme les instructions D# en commandes qui ciblent les moteurs de la voiture. SLY offre une grande lisibilité, une syntaxe facile et une grande facilité de maintenance pour notre compilateur.

Le code source des deux versions de compilateurs ainsi que le reste du code utilisé pour la manipulation de DoobyCar peut être trouvé sur ce lien Github.

b. Navigation avec ROS

Robot Operating System (ROS), est un ensemble d'outils informatiques sous formes de logiciels libres open source, permettant de développer des logiciels pour la robotique.

ROS

La navigation du véhicule est assurée grâce au paquet Nav2 fourni sous ROS (Robot Operating System).

On commence par décrire notre robot (dimensions et formes) grâce à l'extension URDF. ROS dispose des librairies spécialisées pour représenter un modèle pour cette description et le visualiser sur Rviz dans un espace 3-dimensionnel qui représente la perception de notre véhicule pour l'espace qui l'entoure, la figure suivante montre une visualisation du modèle du véhicule sur Rviz:

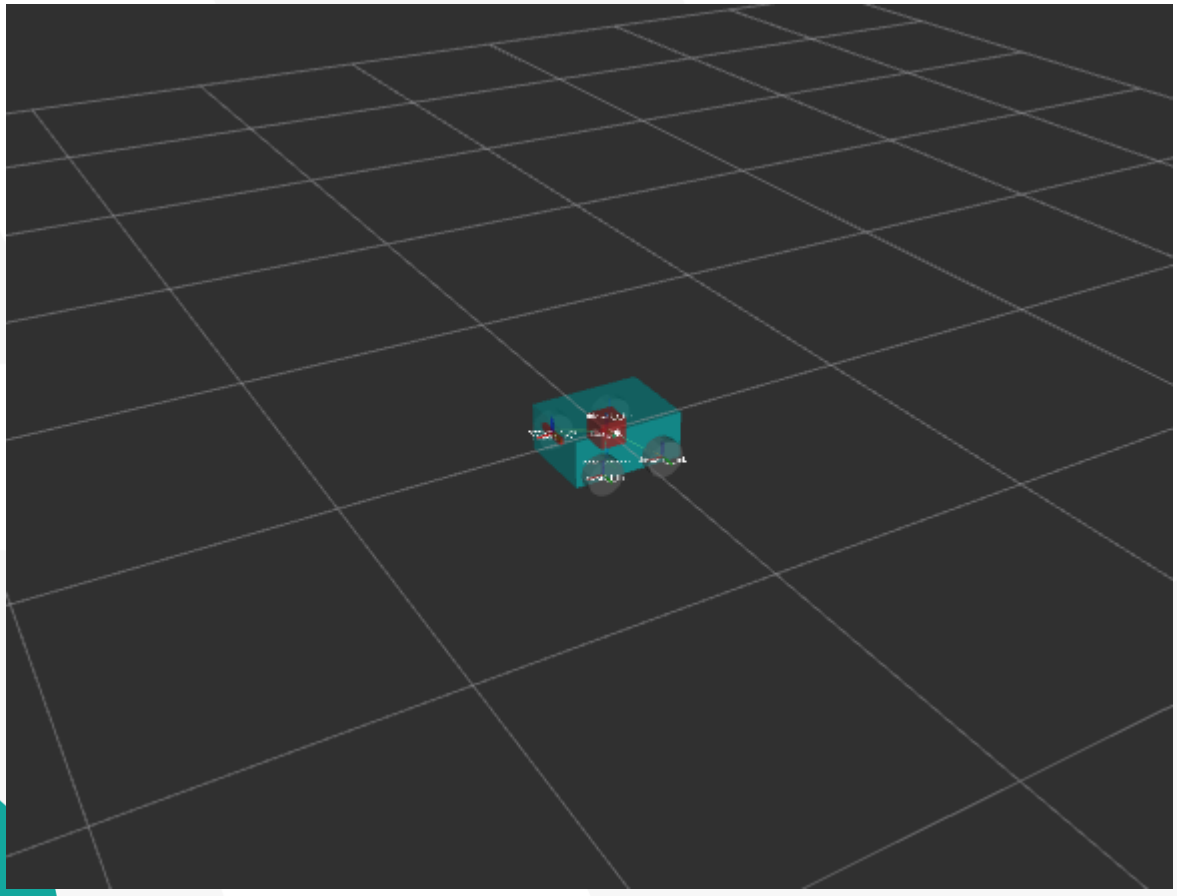


figure : représentation du véhicule en 3D

Ensuite, nous avons élaboré une carte géographique de l'école et l'avons intégrée dans Rviz, cette carte est utilisée comme référence par le paquet Nav2 pour calculer les itinéraires possibles et conduire le véhicule au but, ces itinéraires sont mises à jour au fur et à mesure que le véhicule avance vers son but et recalculés dans le cas ou un obstacle est détecté.

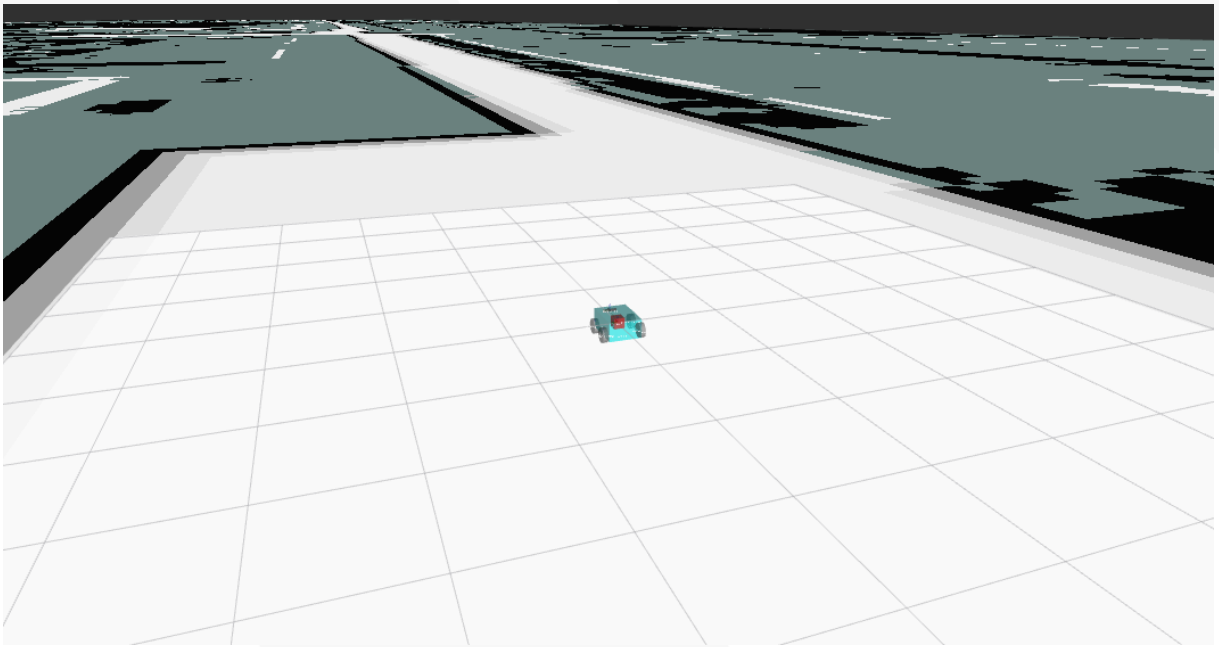


figure : simulation du véhicule dans la map

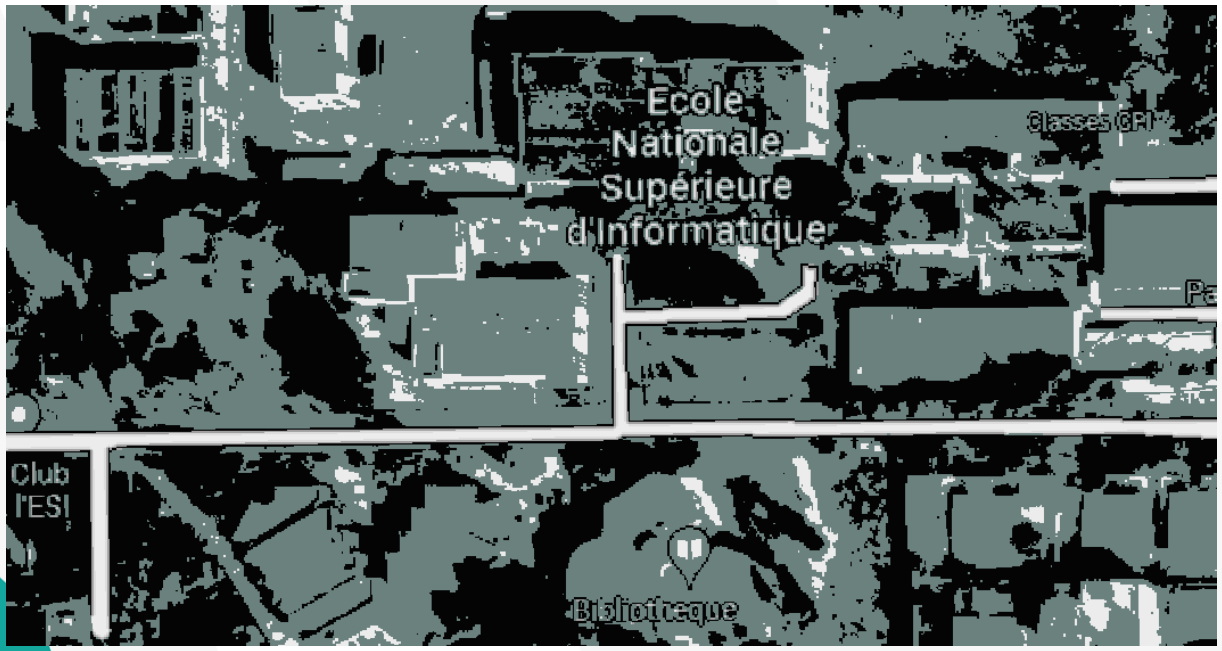


figure : Carte géographique de l'école (compatible avec ROS)

Nav2 communique avec le véhicule sur le réseau local via des messages standardisés pour le commander, ces messages représentent la vitesse et la direction que le véhicule doit suivre, ces messages lorsque reçus par le véhicule sont interprétés via des interfaces matériels implémentées pour les traduire en tensions de sortie sur les pins du raspberry pi et vers les moteurs.

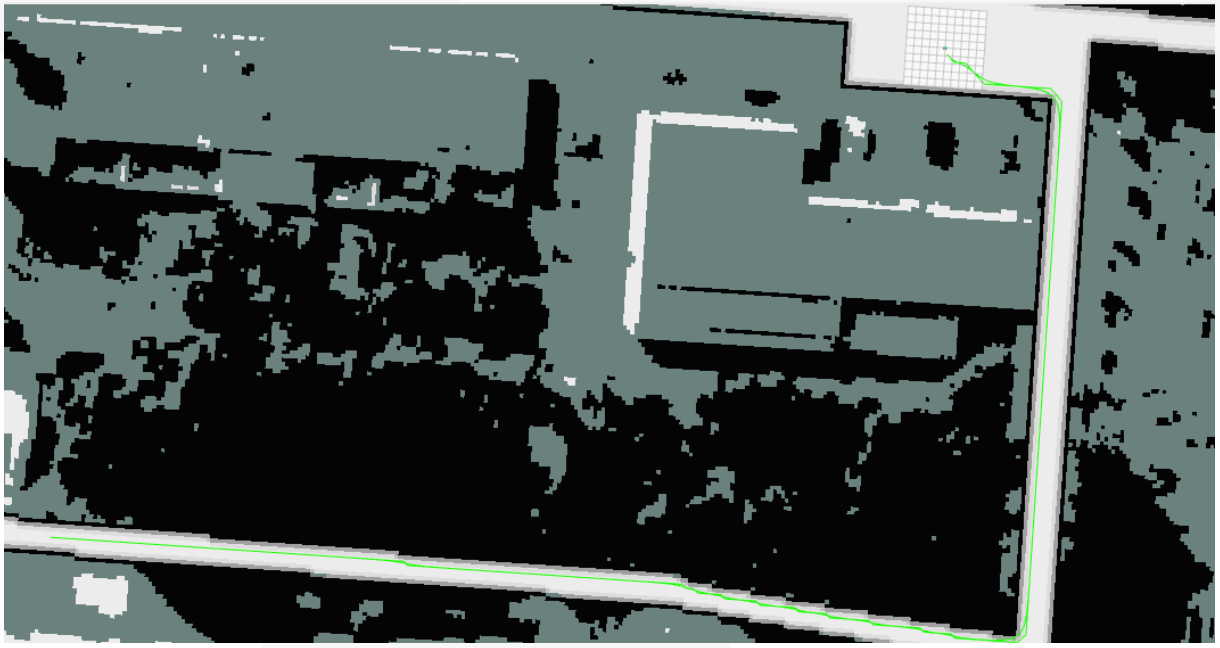


figure : Exemple d'un itinéraire menant du fablab à la direction

En revanche, le véhicule envoie plusieurs messages vers le module de navigation, notamment les lectures sur les senseurs (boussole, GPS, Ultrason).

Le module Nav2 peut être installé sur un pc connecté en réseau local, ou bien un serveur avec connexion vpn, les deux solutions ont été implémentées avec possibilité de basculement entre les deux en cas de besoin.

Tout le software utilisé et installé est décrit dans des fichiers Dockerfile ainsi que le déploiement de ces environnements dans des fichiers Docker-compose pour faciliter la reproduction de notre solution.

c. Navigation avec des points GPS

Pour passer d'un point à un autre en utilisant les coordonnées gps, notre solution était de suivre un ensemble de points(waypoints), entre le départ et la destination, généré à l'aide de l'api "Directions" fourni par Google Maps.

On a créé un parser en python qui fait le décodage du -polyline- obtenu en steps de waypoints et une action après chaque step.

Par exemple pour aller de (36.70474549715888, 3.1747571627627 48) vers (36.70497558929852, 3.173941771261235) on obtient:

step 1 :
start: lat= 36.7047442 , lng= 3.174756
end: lat= 36.7045197 , lng= 3.1742801
distance: 76 m
waypoints:
lat: 36.70474 , lng: 3.17476
lat: 36.7047 , lng: 3.17484
lat: 36.70458 , lng: 3.17483
lat: 36.70452 , lng: 3.17478
lat: 36.70457 , lng: 3.17456
lat: 36.70452 , lng: 3.17428
action: Head south

step 2 :
start: lat= 36.7045197 , lng= 3.1742801
end: lat= 36.7047331 , lng= 3.1736269
distance: 63 m
waypoints:
lat: 36.70452 , lng: 3.17428
lat: 36.70473 , lng: 3.17363
action: Turn right

step 3 :
start: lat= 36.7047331 , lng= 3.1736269
end: lat= 36.7049721 , lng= 3.1739403
distance: 47 m
waypoints:
lat: 36.70473 , lng: 3.17363
lat: 36.70495 , lng: 3.17372
lat: 36.70497 , lng: 3.17375
lat: 36.70498 , lng: 3.17378
lat: 36.70499 , lng: 3.1738
lat: 36.70499 , lng: 3.17388
lat: 36.70497 , lng: 3.17394
action: Turn right

2. Lecture des panneaux de signalisation

On s'attend bien à ce que notre DoobyCar respecte le code de la route et pour ce faire, elle doit avoir une sorte de 'vision' de son environnement pour détecter et reconnaître les panneaux de signalisation qui l'entourent. On fait donc appel à l'intelligence artificielle : un modèle de machine learning qui est entraîné sur un dataset de panneaux classifiés.

- Préparation du modèle et du jeu de données :

  Une bonne étape de lancement d'un mini-projet de modèle de classification est de trouver le bon jeu de données et des bons exemples des modèles qui ciblent des problèmes assez similaires.

  Pour la collecte de données, nous avons utilisé des images à partir de Google Images, des datasets disponibles sur Internet (Kaggle par exemple) et aussi des photos qu'on a prises.

  Notre modèle utilise la bibliothèque OpenCV (Open Computer Vision).

3. Détection des obstacles

La détection des obstacles se fait grâce au module ultrason qui détecte les objets dans un rayon de 3.0 mètres, ces données sont envoyées immédiatement vers le module Nav2 et ensuite vont être inscrites sur la map de navigation.

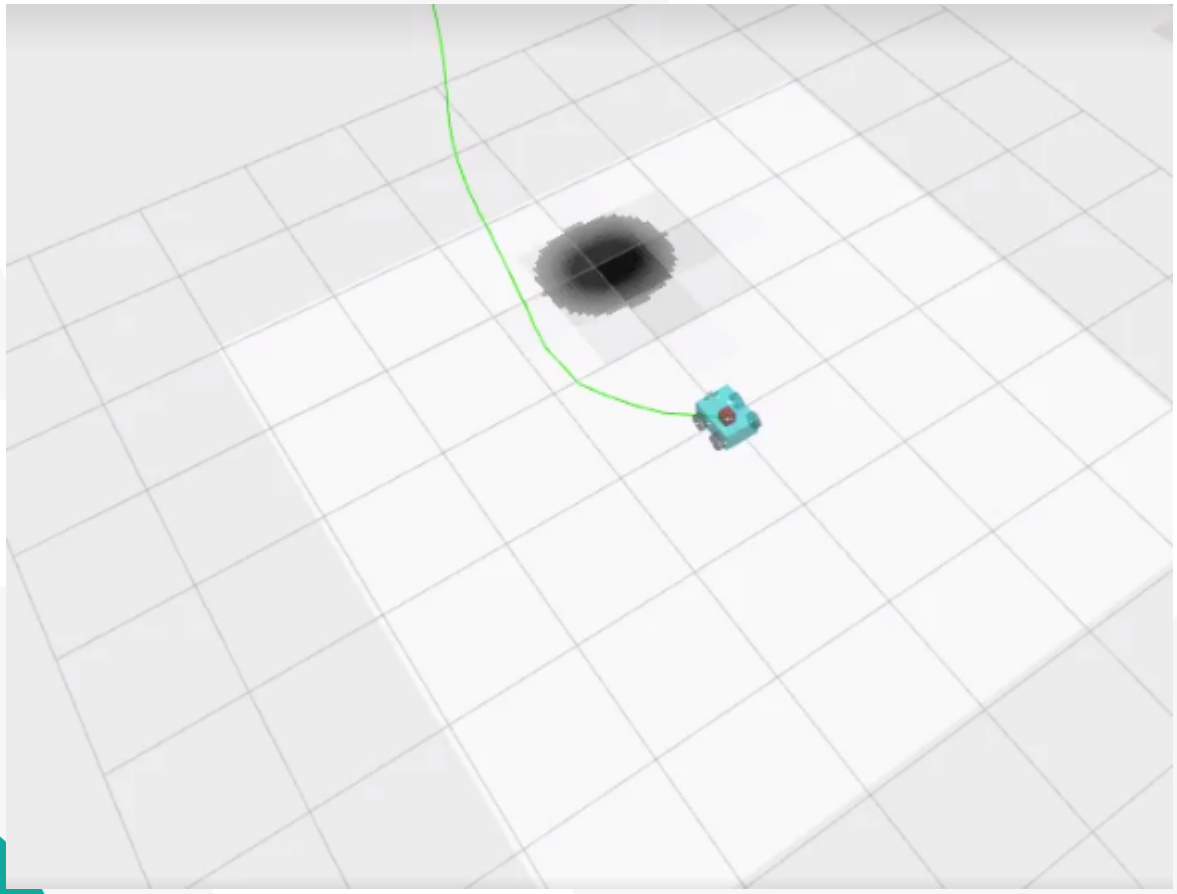
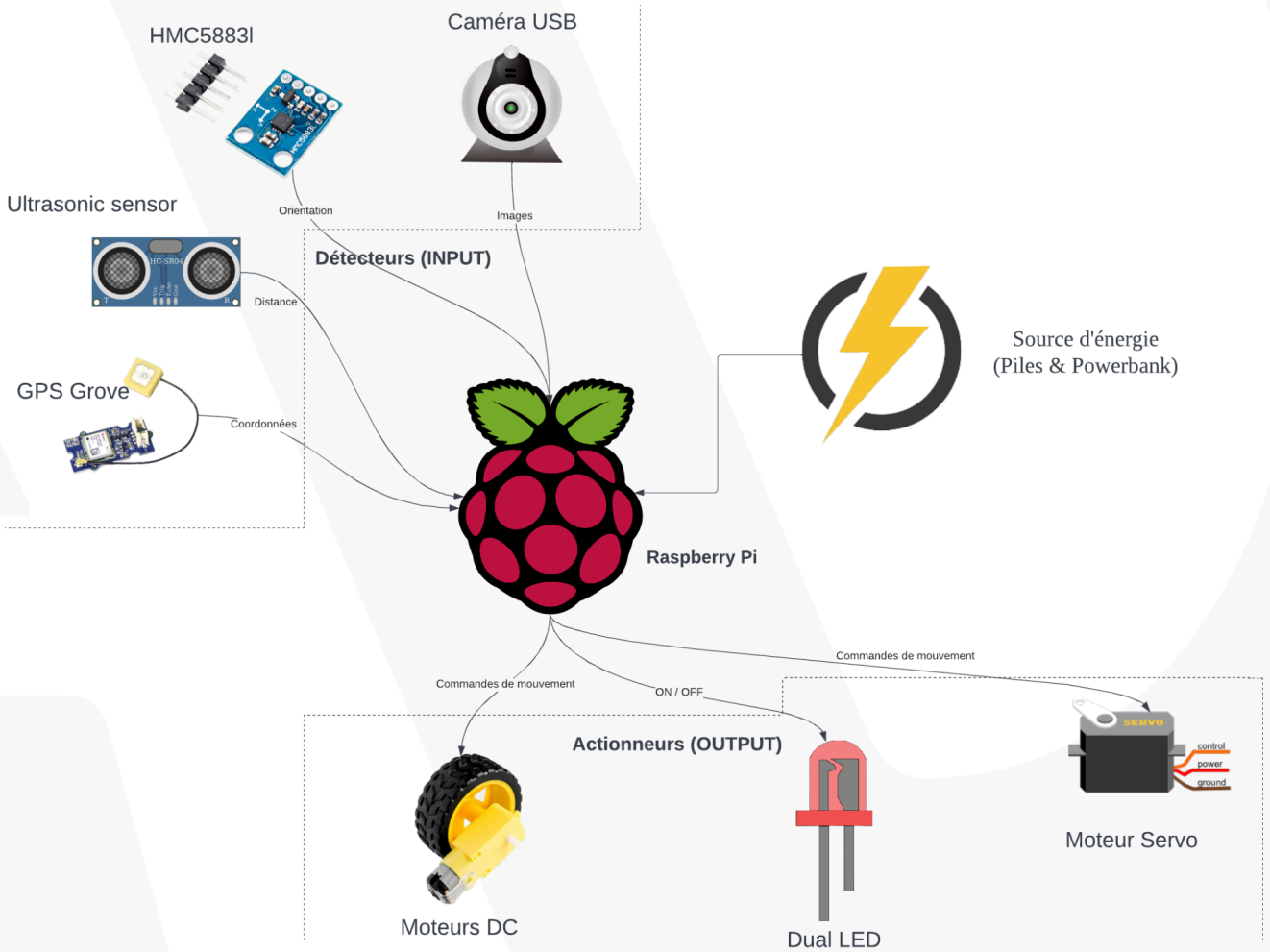


figure : représentation des obstacles

Le module de navigation ensuite va conduire le véhicule en adaptant l'itinéraire en temps réel pour éviter les obstacles, il envoi en retour des informations sur l'état et le progrès de la tâche et en cas d'échec annule l'opération et envoi une notification au serveur.

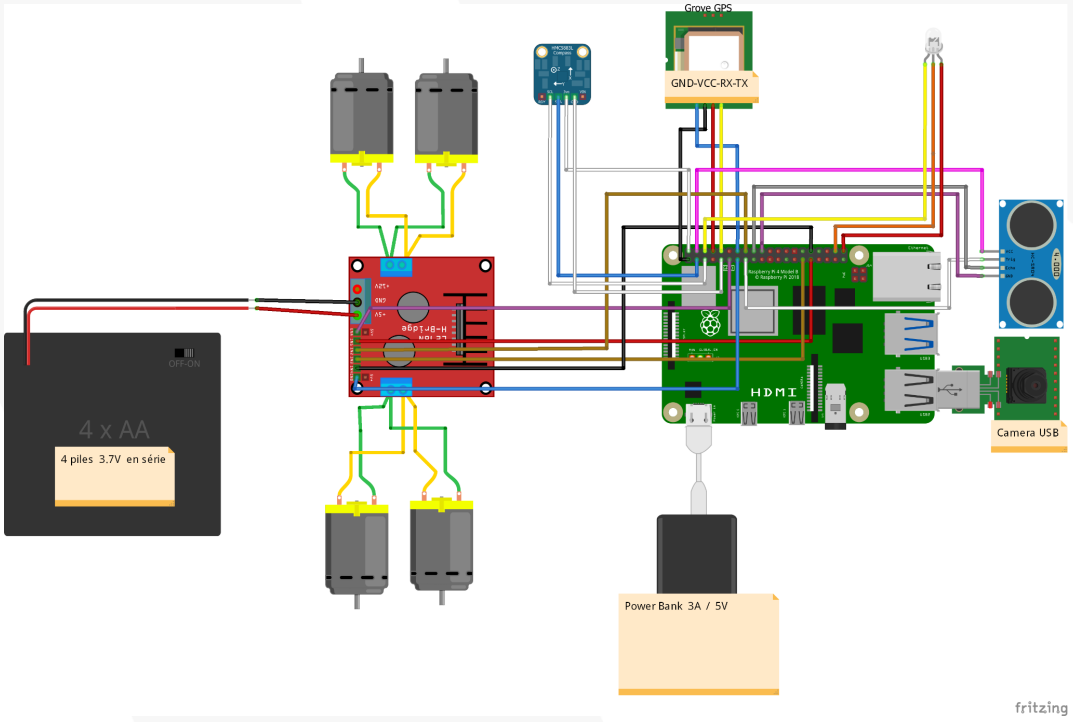
III. Schéma logique du montage

Notre Raspberry Pi est alimenté par une source d'énergie (Powerbank) et reçoit des données à partir des détecteurs reliés, un traitement est effectué au niveau du Raspberry Pi et les actions générées sont envoyées aux actionneurs appropriés.



IV. Schéma physique du montage

1. Schéma



2. Liaison des composants

Composant	Port	Pin
GPS	GND	PIN14
	VCC	PIN1
	RX	PIN8(UATRO TX)
	TX	PIN10(UATRO RX)
HMC	VCC	PIN2
	GND	PIN9
	SDA	PIN3(GPIO2)
	SCL	PIN5(GPIO3)
ULTRASON	VCC	PIN4

LED	GREEN	PIN40(GPIO21)
	RED	PIN38(GPIO20)
	GND	PIN6
SHIELD	IN1	PIN31(GPIO6)
	IN2	PIN16(GPIO23)
	IN3	PIN29(GPIO5)
	IN4	PIN32(GPIO12)
	ENA	PIN11(GPIO17)
	ENB	PIN13(GPIO27)

	GND	PIN20
	TRIG	PIN15(GPIO22)
	ECHO	PIN18(GPIO24)

V. Liste du software installé

1. Paquets installés

Paquets pour les interfaces matériel:

rpi.gpio

mpu6050-raspberrypi

gpsd-py3

python3-smbus

gpsd gpsd-clients

Paquets pour la vision des ordinateurs:

libopencv-dev

python3-opencv

opencv-python-headless

ros-galactic-cv-bridge

ros-galactic-vision-opencv

Paquets pour ROS :

ros-galactic-rmw-cyclonedds-cpp

ros-galactic-joint-state-publisher-gui

ros-galactic-xacro

ros-galactic-robot-localization

ros-galactic-gazebo-ros-pkgs

ros-galactic-navigation2

ros-galactic-nav2-bringup

ros-galactic-slam-toolbox

ros-galactic-nav2-amcl

ros-galactic-rosbridge-server

2. Noeuds et architecture ROS

Voici l'architecture des dossiers de notre solution ROS:

|__ ROS:

|__ ros2_ws

“”” contient le code source de notre solution basée sur ROS”””

|__ dooby_interface

“”” les pilotes matériels ainsi que les publishers pour les différent données lu apartir des senseurs”””

|__ dooby_ai

“”” dossier pour les fonctionnalités ai du véhicule (détection des panneaux signalisation) “””

|__ dooby_simulation

“”” dossier pour les fonctionnalités de simulation de l'espace 3D “””

|__ dev-node

“”” contient l'environnement d'un noeud de développement équipé de toute l'installation de ROS avec les paquets nécessaires”””

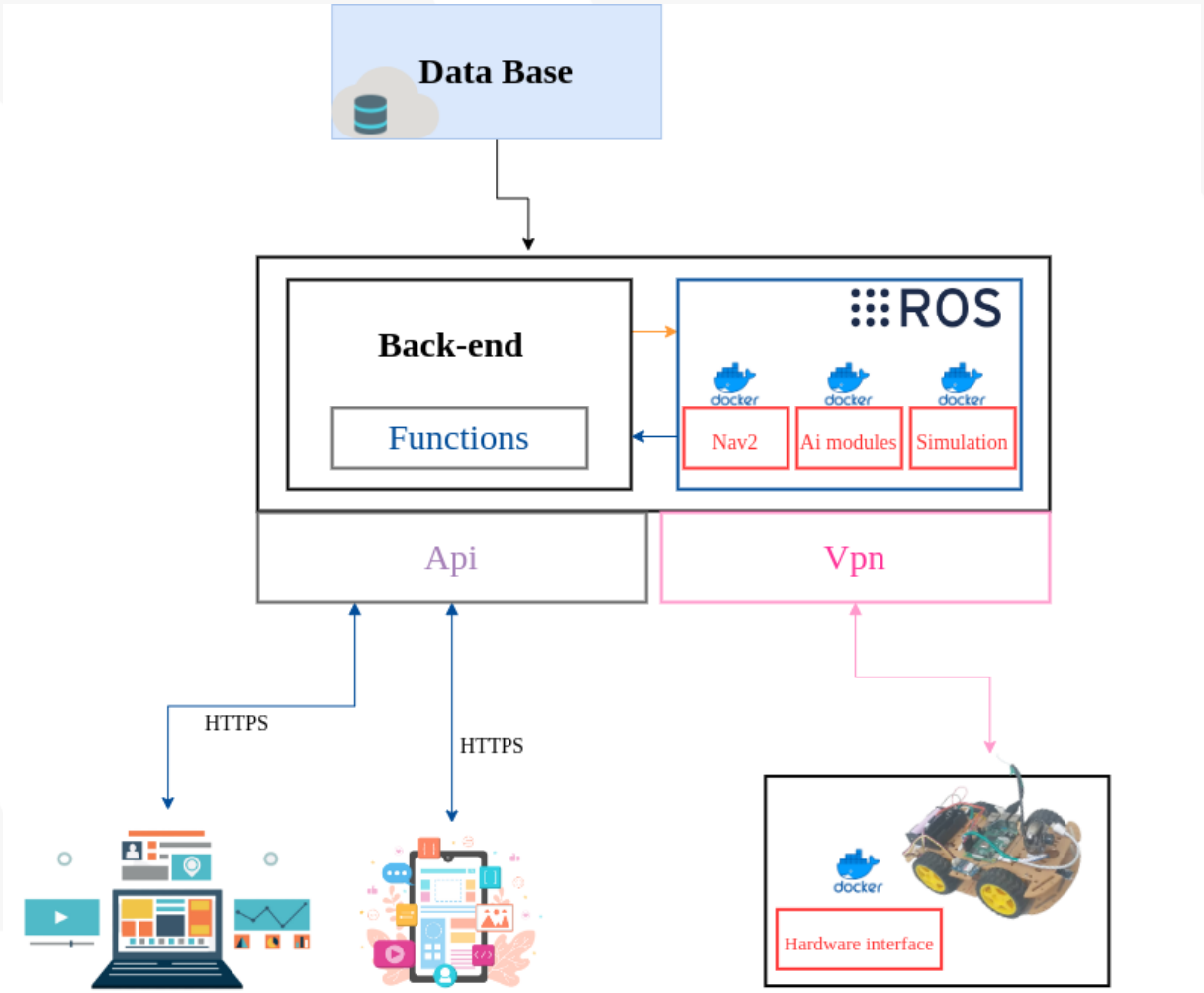
|__ pi-node

“”” l'environnement exécuté sur le système imbriqué, emballé dans un conteneur Docker pour la facilité de redeploiement”””

|__ server-node

“”” l'environnement du serveur qui exécture le module de Navigation ainsi que l'api REST de communication avec les application cliente”””

Voici la vue globale de notre solution :

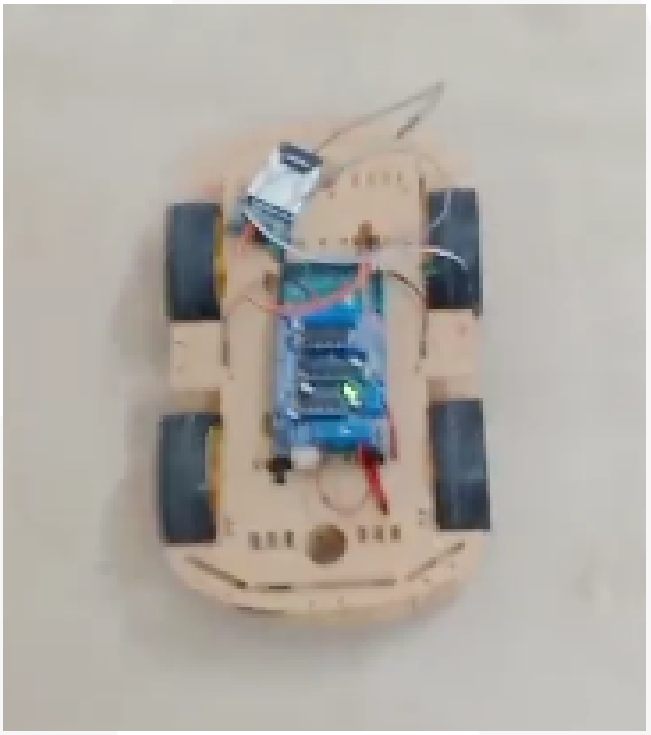


Le VPN utilisé est appelé Husarnet, il est peer to peer c-à-d la connexion se fait directement entre les nœuds sans unité centralisée. Parmi les avantages de husarnet est qu'il est facile à mettre en place avec peu de configuration, de plus il a une bonne documentation avec beaucoup d'exemples d'usage.

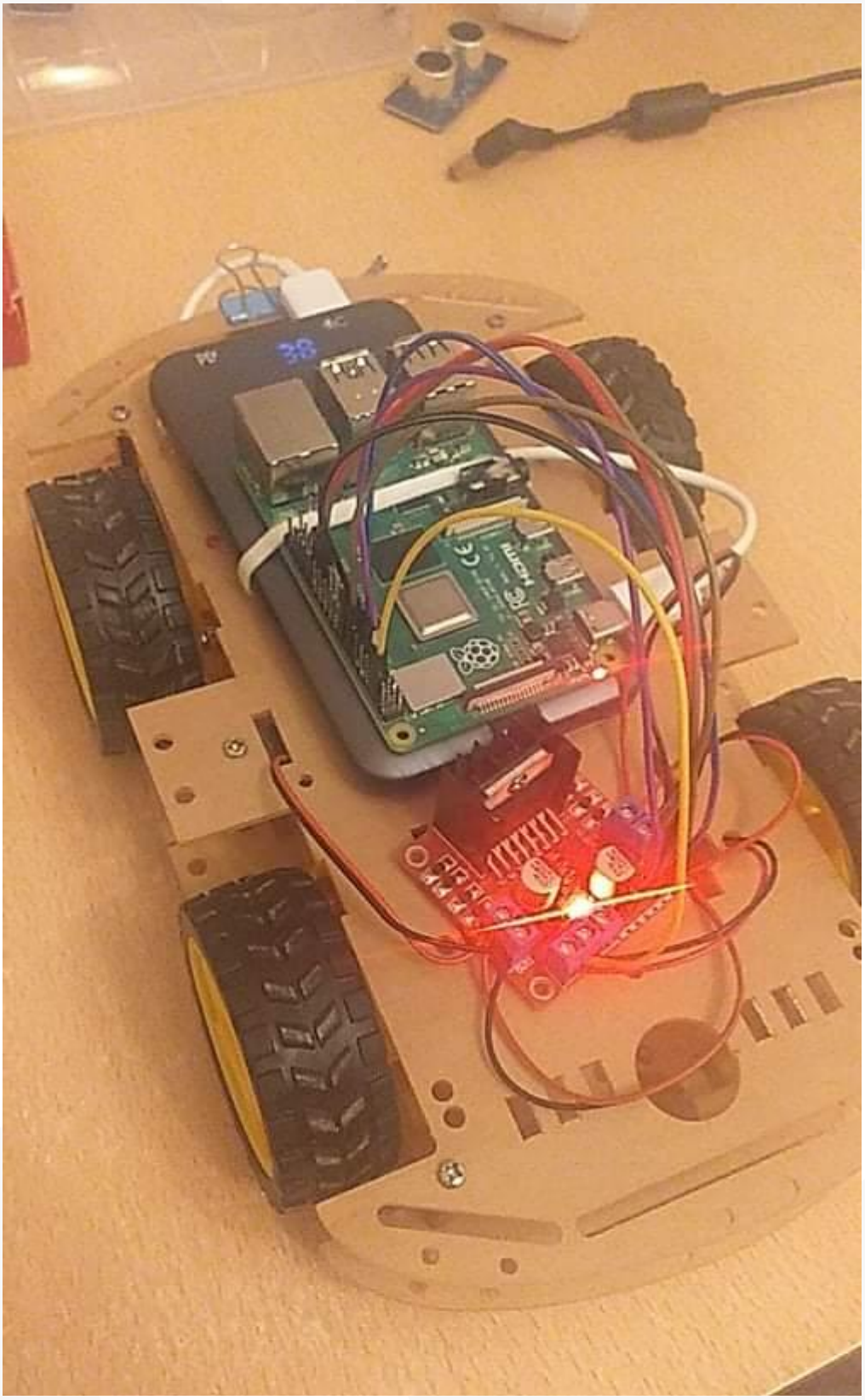
VI. Photos de DoobyCar

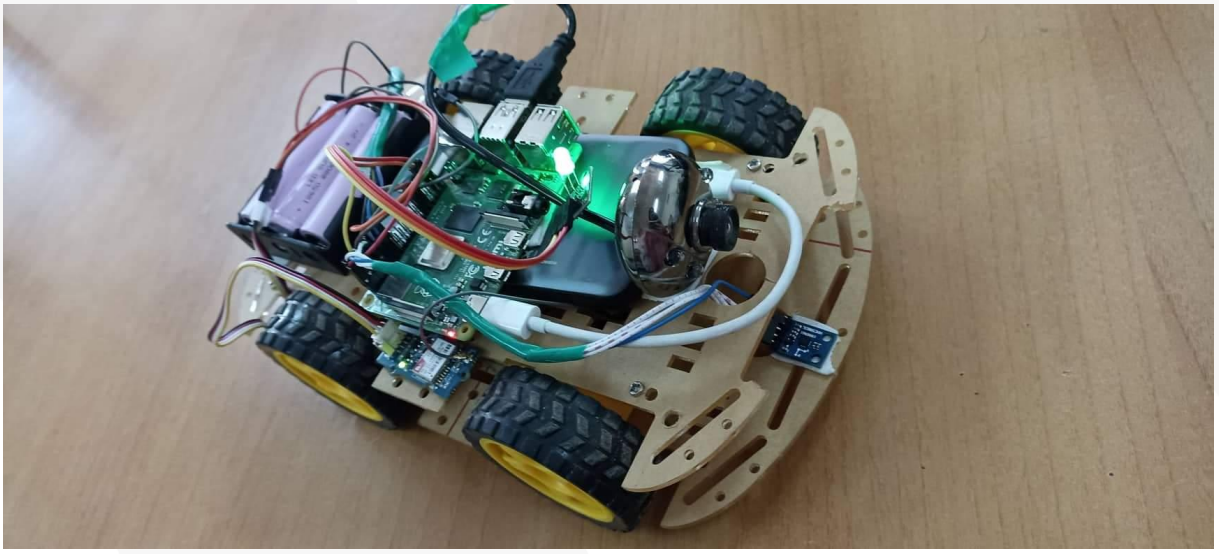
Trouvez plus de photos et vidéos ici.

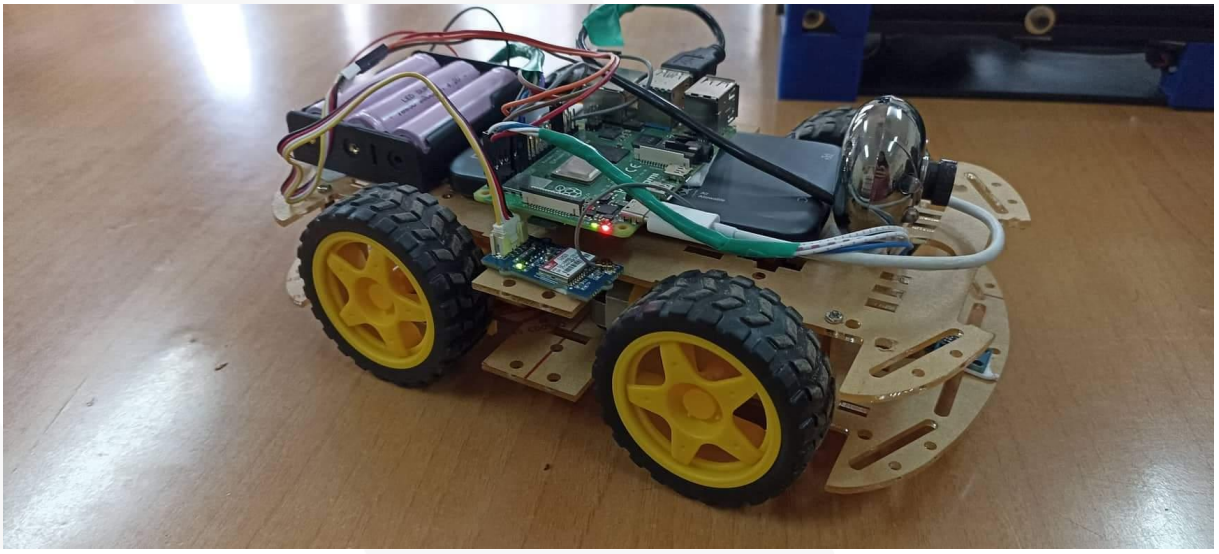
Prise le 23 - 06 - 2022

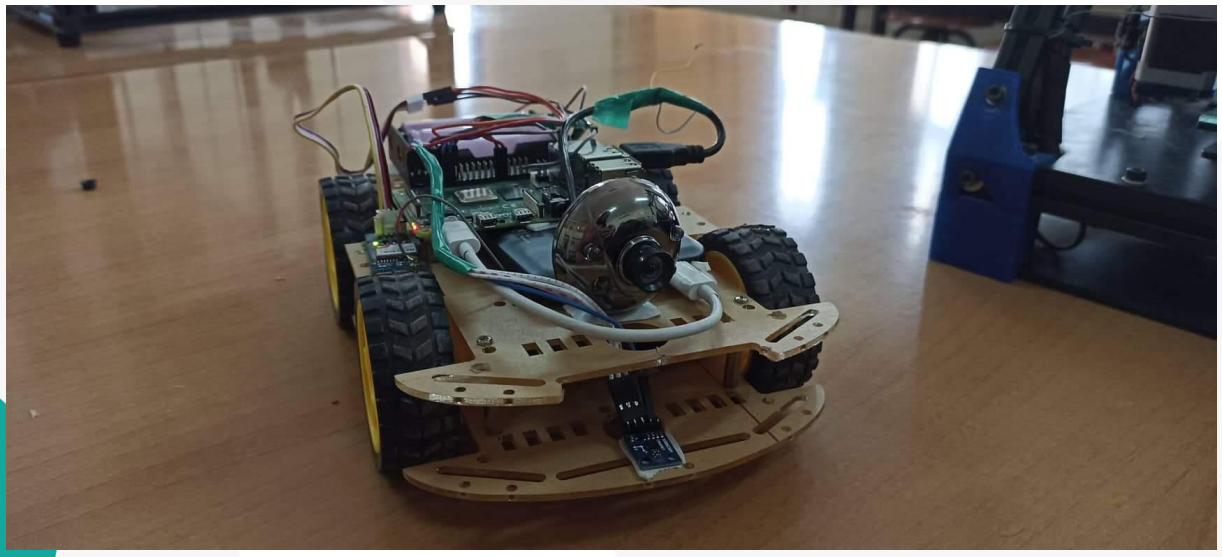


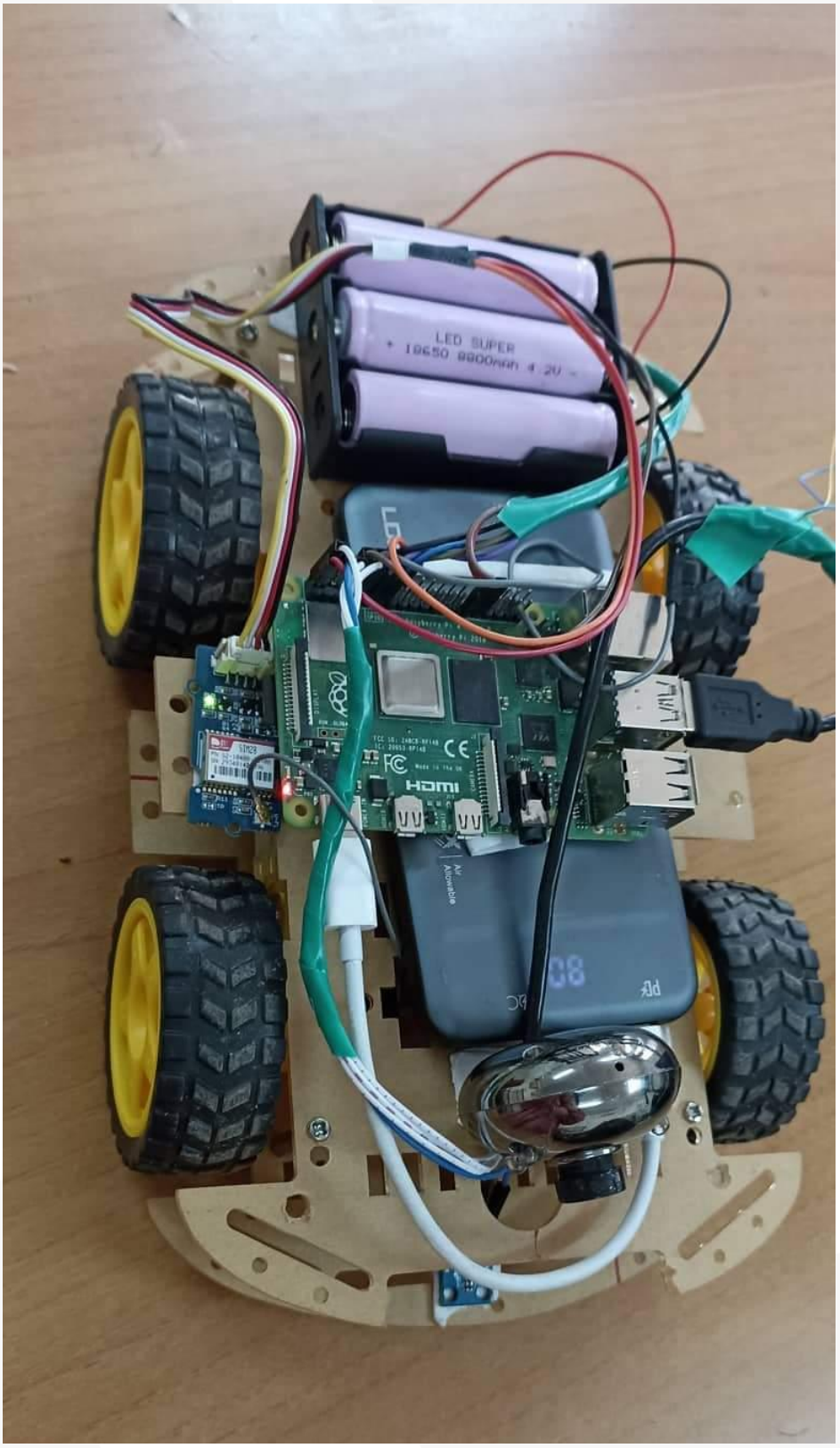
Prise le 29 - 03 - 2022

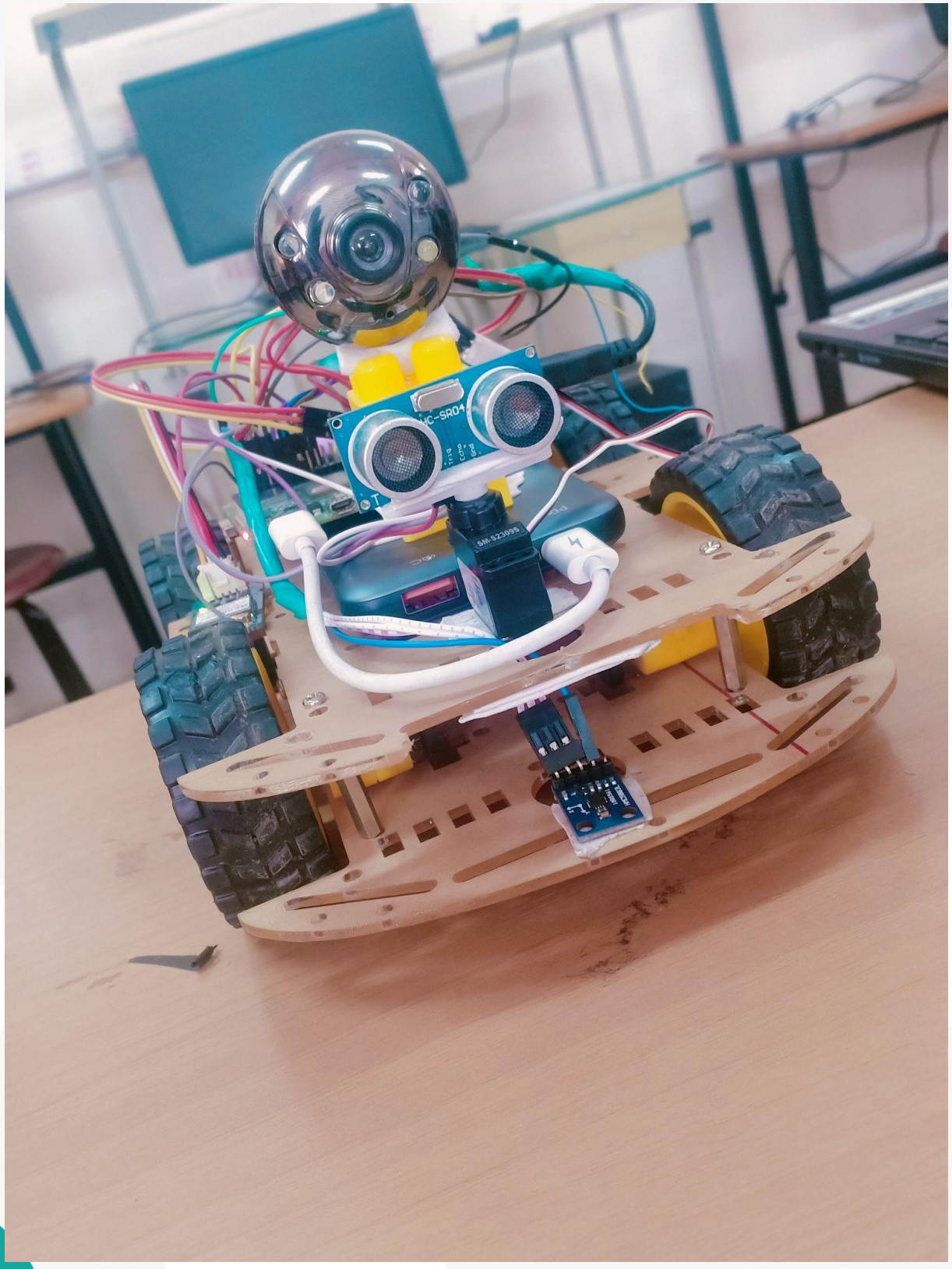


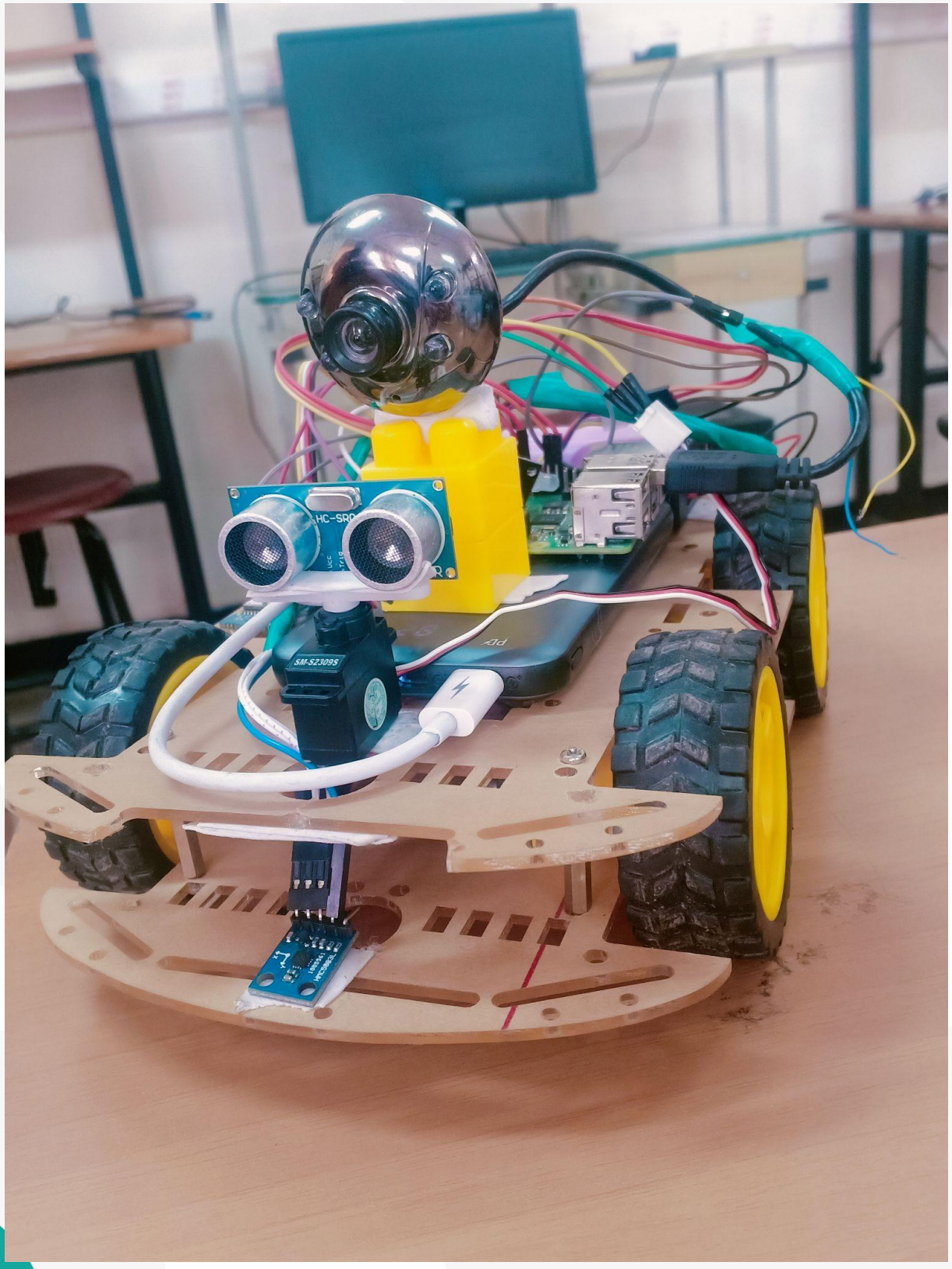


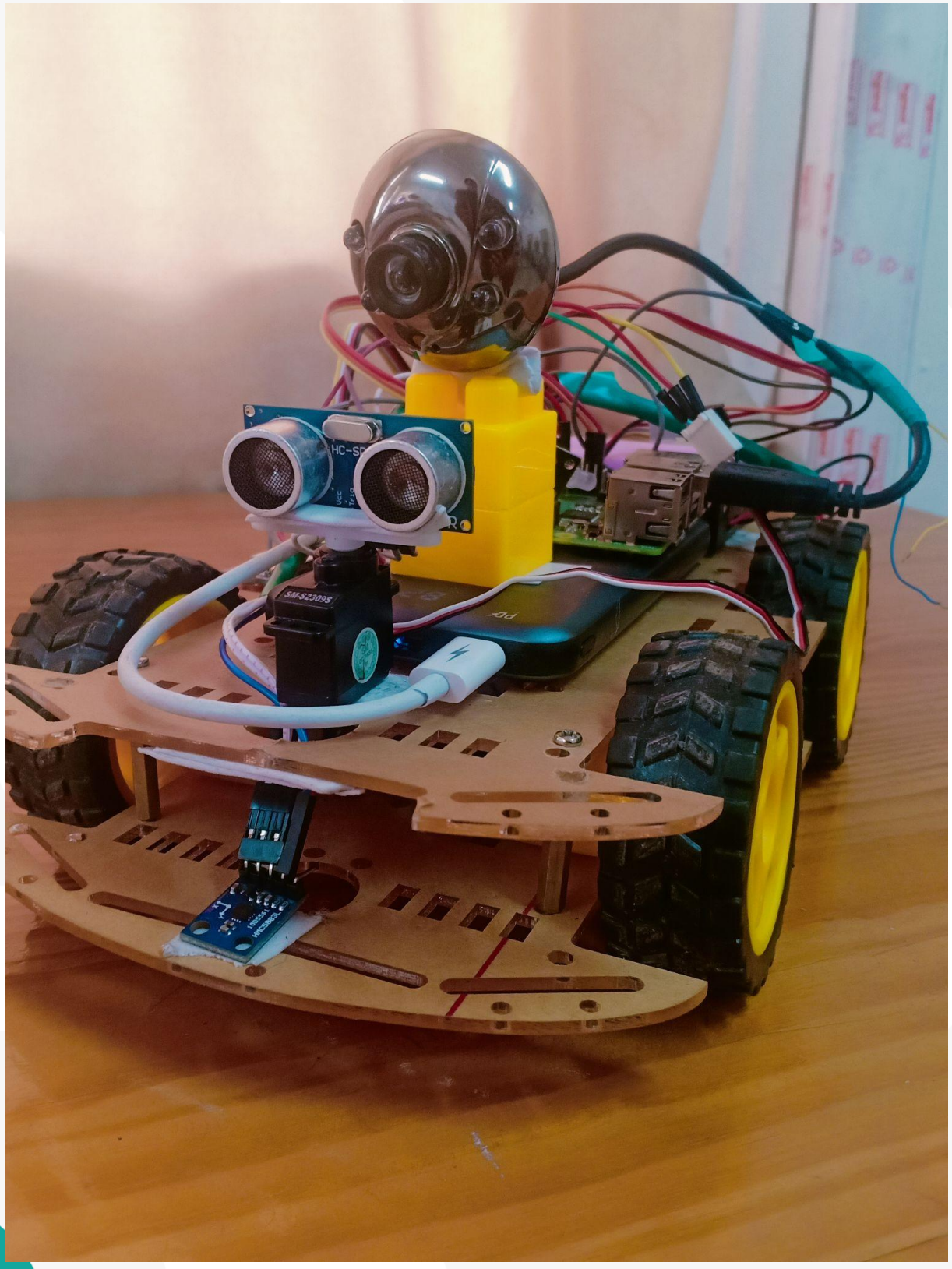


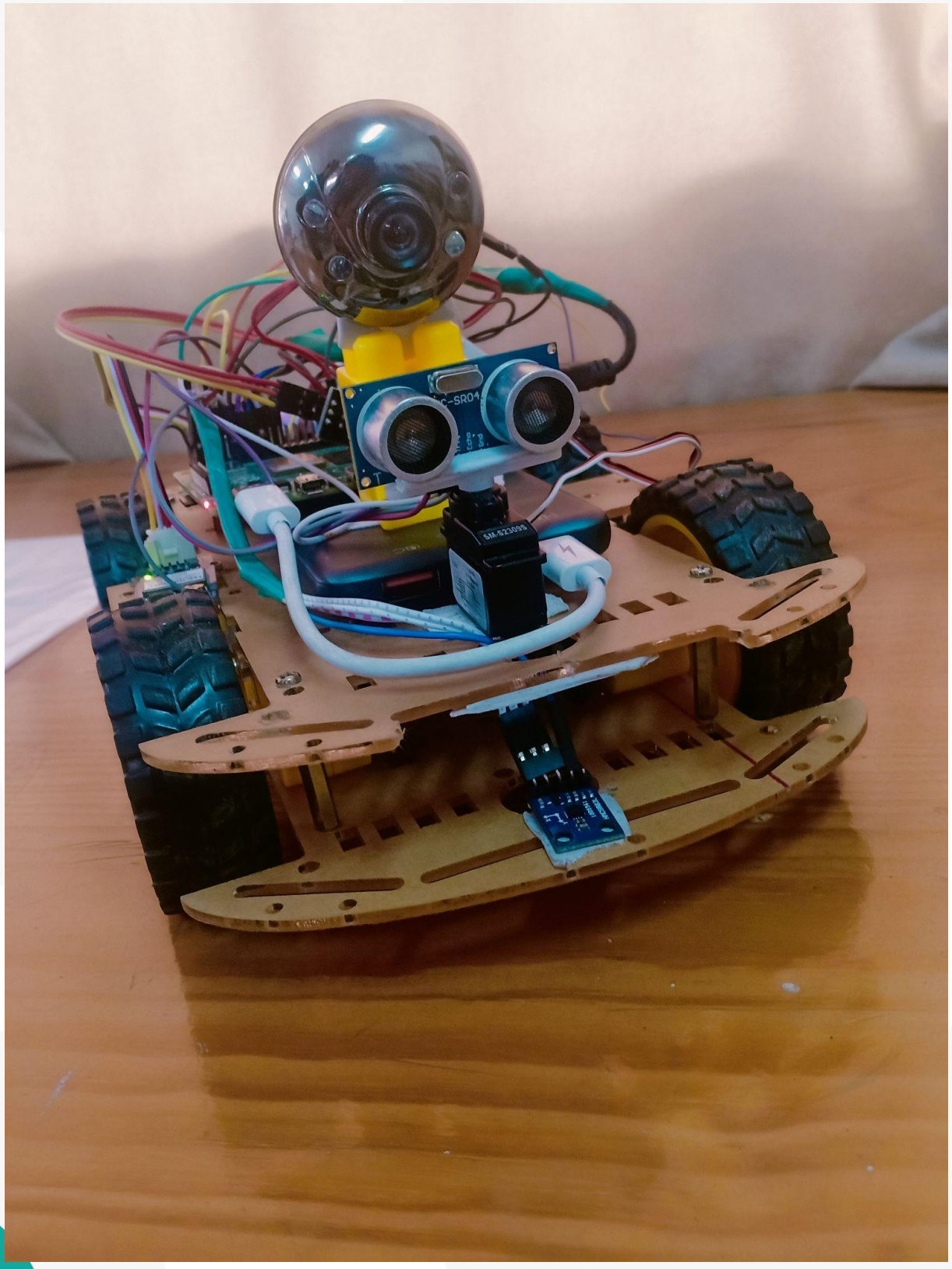



LED SUPER
HDMI


HC-SR04
SM-S2309S


HC-SR0
SM-S2309S


HC-SR
SM-S2309S


SM-S2309S