Baldé boubacar 55870
Sacewicz Mathieu 54025

RAPPORT PROJET TRANSVERSALE INDUSTRIEL 2023-2024

Objectifs :

Réaliser un robot mobil (voiture) capable de se déplacer et éviter les obstacles pour qu'il puisse arriver au bout de son parcourt.

Matériels :

- Raspberry pie 4
- Capteur ultrason : HC-SR04
- Micro-Servo: 9g SG0
- RGBLED Module
- Passive Buzzer Module
- Jumper
- Battery 7 volt
- Piece de montage
- Bridge i2c

Réalisation :

- Étape 0 : on a lu le guide du robot et installer les librairies nécessaires
- Étape 1 : on a testé les différents composants
- Étape 2 : on a monté le robot
- Étape 3 : on a implémenté l'algorithme pour réaliser le déplacement et la détection d'un obstacle
- Étape 4 : on a ajouté des options au robot
- Etape 5 : on a fait une intelligence artificielle capable de reconnaitre un panneau stop pour se faire nous avons entrainer un model a l'aide d'un outil « cascade gui trainer »
  Ps : nous ne l'avons pas mis au robot car nous avons eu des problèmes d'installation de package car notre Raspberry pie est plus récent à celle de la camera mjepg.
  Mais l'avons testé sur un autre rasberry pie 3 et cela fonctionner.
- Etape 6 : on a fait une démonstration au professeur.

Baldé boubacar 55870
Sacewicz Mathieu 54025

Code Arduino pour le robot:

Tout d'abord, nous avons cloné le répertoire se trouvant ici :

https://github.com/Freenove/Freenove_Three-wheeled_Smart_Car_Kit_for_Raspberry_Pi/archive/master.zip

puis on a essayé de comprendre le fonctionnement du code ensuite on a implémenté le code du robot

mouv2.py

```
# -*- coding: utf-8 -*-
#############################################################################
# Filename    : mouvement.py
# authors     : boubacar - mathieu
#############################################################################
from Command import COMMAND as cmd
from mDev import *
import time
import numpy as np

mdev = mDEV()
speed = 400
val = 'true'
val2 = 'false'
def get_distance():
    SonicEchoTime = mdev.readReg(mdev.CMD_SONIC)
    distance = SonicEchoTime * 17.0 / 1000.0
    return distance

def move_forward():
    mdev.writeReg(mdev.CMD_DIR1, 0)
    mdev.writeReg(mdev.CMD_DIR2, 1)
    mdev.writeReg(mdev.CMD_SERVO1, numMap(90 + 0, 0, 180, 500, 2500))
    mdev.writeReg(mdev.CMD_PWM1, speed)
    mdev.writeReg(mdev.CMD_PWM2, speed)

def move_backward():
    mdev.writeReg(mdev.CMD_DIR1, 1)
    mdev.writeReg(mdev.CMD_DIR2, 0)
    mdev.writeReg(mdev.CMD_SERVO1, numMap(90 + 0, 0, 180, 500, 2500))
    mdev.writeReg(mdev.CMD_PWM1, speed)
    mdev.writeReg(mdev.CMD_PWM2, speed)

def lights_blue():
    mdev.writeReg(mdev.CMD_IO1, 1)
    mdev.writeReg(mdev.CMD_IO2, 1)
    mdev.writeReg(mdev.CMD_IO3, 0)

def lights_red():
    mdev.writeReg(mdev.CMD_IO1, 0)
    mdev.writeReg(mdev.CMD_IO2, 0)
    mdev.writeReg(mdev.CMD_IO3, 1)

def police_lights():
    lights_red()
    time.sleep(0.1)
```

```
    lights_blue()
    time.sleep(0.1)


def turn_left():
    mdev.writeReg(mdev.CMD_SERVO1, numMap(90 + 45, 0, 180, 500, 2500))
    mdev.writeReg(mdev.CMD_DIR1, 0)
    mdev.writeReg(mdev.CMD_DIR2, 1)
    mdev.writeReg(mdev.CMD_PWM1, speed)
    mdev.writeReg(mdev.CMD_PWM2, speed)

def turn_right():
    mdev.writeReg(mdev.CMD_SERVO1, numMap(90 - 45, 0, 180, 500, 2500))
    mdev.writeReg(mdev.CMD_DIR1, 0)
    mdev.writeReg(mdev.CMD_DIR2, 1)
    mdev.writeReg(mdev.CMD_PWM1, speed)
    mdev.writeReg(mdev.CMD_PWM2, speed)

def buzz(detect):
    if(detect == val):
        mdev.writeReg(mdev.CMD_BUZZER,140)
    elif(detect==val2):
        mdev.writeReg(mdev.CMD_BUZZER,0)

def bip():
    buzz(val)
    time.sleep(0.25)
    buzz(val2)
    time.sleep(0.25)

def scan_area():
    try:
        # Set initial position of SERVO2
        initial_servo2_angle = 90
        mdev.writeReg(mdev.CMD_SERVO2, numMap(initial_servo2_angle, 0, 180,
500, 2500))
        time.sleep(1)  # Wait for SERVO2 to reach the initial position

        distances = {'right': None, 'straight': None, 'left': None}
        buzz(val2)

        # Move SERVO2 to the left
        for angle in range(0, 181, 10):
            mdev.writeReg(mdev.CMD_SERVO2, numMap(angle, 0, 180, 500,
2500))
            time.sleep(0.15)
            distance = get_distance()
            if(angle%20==0):
                lights_blue()
            else:
                mdev.setLed(1,0,0)
            #print(f"Angle: {angle} - Distance: {distance} cm")

            if angle >= 0 and angle < 60:
                distances['right'] = distance
            elif angle >= 60 and angle < 120:
                if angle == 90 :
                    distances['backward'] = distance
                distances['straight'] = distance
            elif angle >= 120 and angle <= 180:
                distances['left'] = distance
```

```
        print("Distances:", distances)

        # Set SERVO2 back to the initial position
        mdev.writeReg(mdev.CMD_SERVO2, numMap(initial_servo2_angle, 0, 180,
500, 2500))
        time.sleep(1)  # Wait for SERVO2 to reach the initial position

        # Determine the direction based on the highest distance
        max_distance_angle = max(distances, key=distances.get)

        if distances['backward'] < 11:
            print("I'm going BACKWARDS")
            move_backward()
            for x in np.arange(0, 1.0, 0.25):
                buzz(val)
                time.sleep(0.25)
                buzz(val2)
                time.sleep(0.25)

            time.sleep(0.75)
            stop()

        elif max_distance_angle == 'right':
            buzz(val2)
            print("I'm going RIGHT")
            turn_right()
            time.sleep(0.75)
            stop()
        elif max_distance_angle == 'left':
            print("I'm going LEFT")
            buzz(val2)
            turn_left()
            time.sleep(0.75)
            stop()
        elif max_distance_angle == 'straight':
            print("I'm going FORWARD")
            buzz(val2)
            move_forward()
            time.sleep(0.75)
            stop()


    except Exception as e:
        print(f"Error: {e}")

def stop():
    mdev.writeReg(mdev.CMD_SERVO1, numMap(90 + 0, 0, 180, 500, 2500))
    mdev.writeReg(mdev.CMD_PWM1, 0)
    mdev.writeReg(mdev.CMD_PWM2, 0)

if __name__ == "__main__":
    while(True):
        scan_area()
```

Baldé boubacar 55870
Sacewicz Mathieu 54025

Explication du code

° Nous avons créé des méthodes de déplacement du robot :

- move_forward : va avancer le robot en avant
- move_backward : va faire un déplacement en arrière
- turn_left : tourner les roues du robot à gauche
- turn_right: tourner les roues du robot à droite

° Nous avons créé des méthodes pour la lumière du rgbled module :

- lights_blue : mettre la lumière de la led en bleu
- lights_red : mettre la lumière de la led en rouge

° Nous avons créé une méthode pour le passive buzzer module :

- buzz(detected) : si detected = True il va activer le buzzer avec une fréquence de 140 sinon il va la désactiver.

° Nous avons créer une méthode pour le capteur d'ultrason :

- scan_area : celle-ci va nous permettre de rechercher un obstacle à l'aide de la méthode get_distance() qui va calculer la distance d'un mur.

```
for angle in range(0, 181, 10):
    mdev.writeReg(mdev.CMD_SERVO2, numMap(angle, 0, 180, 500,2500))
    time.sleep(0.15)
    distance = get_distance()
    if(angle%20==0):
        lights_blue()
    else:
        mdev.setLed(1,0,0)
    #print(f"Angle: {angle} - Distance: {distance} cm")

    if angle >= 0 and angle < 60:
        distances['right'] = distance
    elif angle >= 60 and angle < 120:
        if angle == 90 :
            distances['backward'] = distance
        distances['straight'] = distance
    elif angle >= 120 and angle <= 180:
        distances['left'] = distance
```

partie du code qui va allumer la led, détecter un obstacle et va créer une plage d'angle pour que le rebot choisisse le bon mouvement.

- si l'angle est entre 0 et 60, le robot va tourner à droite
- si l'angle est entre 60 et 120, le robot va avancer tout droit
- si l'angle est entre 120 et 180, le robot va tourner à gauche

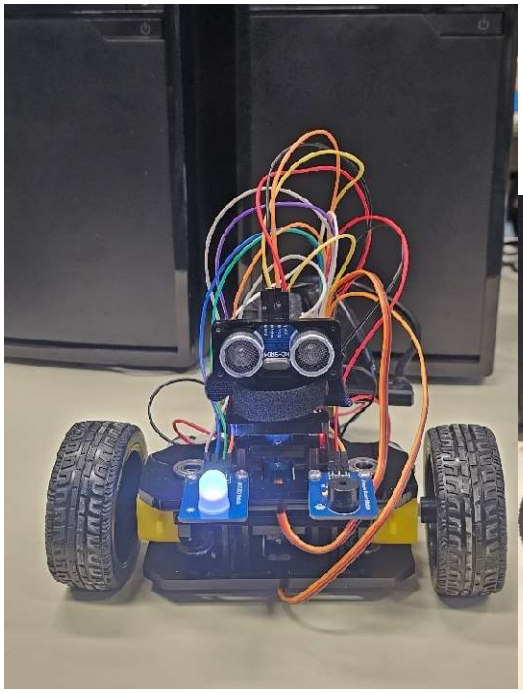
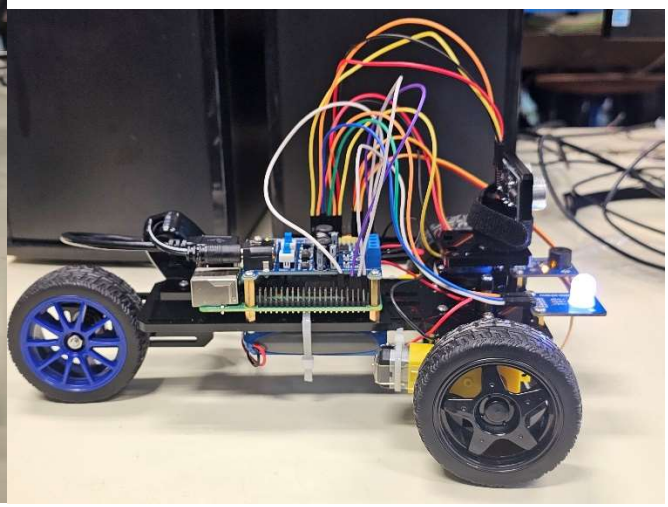
Baldé boubacar 55870
Sacewicz Mathieu 54025

```
        mdev.writeReg(mdev.CMD_SERVO2, numMap(initial_servo2_angle, 0, 180,
500, 2500))
        time.sleep(1)  # Wait for SERVO2 to reach the initial position

        # Determine the direction based on the highest distance
        max_distance_angle = max(distances, key=distances.get)

        if distances['backward'] < 11:
            print("I'm going BACKWARDS")
            move_backward()
            for x in np.arange(0, 1.0, 0.25):
                buzz(val)
                time.sleep(0.25)
                buzz(val2)
                time.sleep(0.25)

            time.sleep(0.75)
            stop()

        elif max_distance_angle == 'right':
            buzz(val2)
            print("I'm going RIGHT")
            turn_right()
            time.sleep(0.75)
            stop()
        elif max_distance_angle == 'left':
            print("I'm going LEFT")
            buzz(val2)
            turn_left()
            time.sleep(0.75)
            stop()
        elif max_distance_angle == 'straight':
            print("I'm going FORWARD")
            buzz(val2)
            move_forward()
            time.sleep(0.75)
            stop()
```

il va mettre le servo du capteur à ultrason à sa position initiale
puis il va déterminer la direction en fonction de la distance la plus grande suivant l'angle cité avant.
Si la distance est inferieur a 11 il va aller en arrière et va activer le bipeur pendant 0.75 seconde puis il va s'arrêter ensuite il va encore faire un autre scan et rechoisir le mouvement.

Baldé boubacar 55870
Sacewicz Mathieu 54025

Photo du robot:

Lien de la vidéo :

https://youtu.be/5Z57A8BvnRI

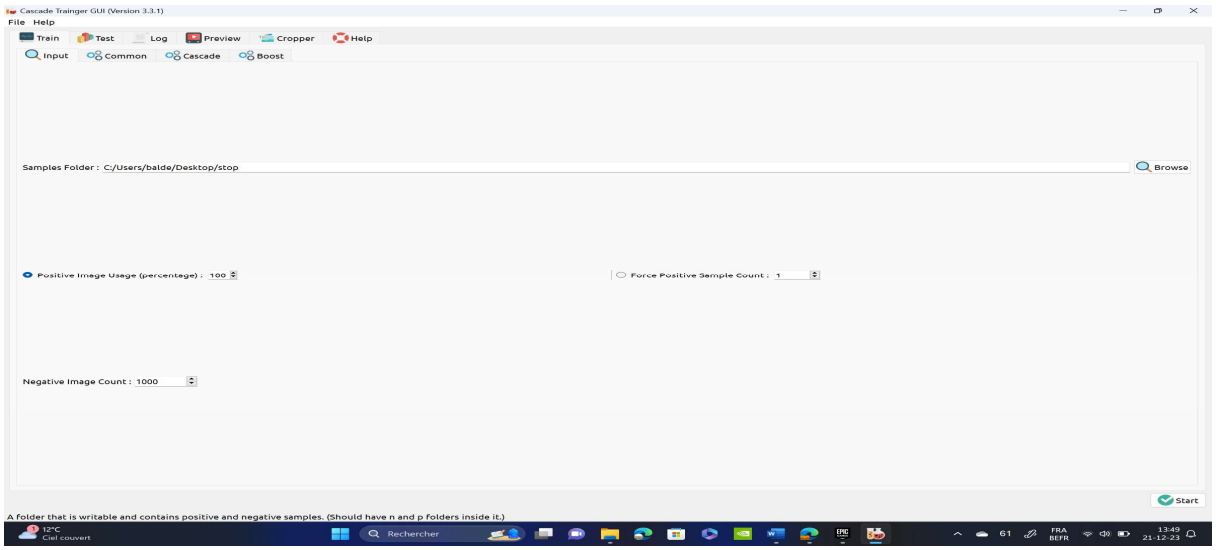
Partie intelligence artificielle:

Pour commencer, nous avons créé un dossier stop puis on a créé des sous-dossiers n et p

- dans le dossier p, nous avons mis des images de panneaux stop
- dans le dossier n, nous avons mis des images qui ne sont pas des panneaux stop

ensuite nous avons mis le dossier dans le logiciel cascade gui trainer pour qu'il entraine le modèle

Baldé boubacar 55870
Sacewicz Mathieu 54025



Une fois réaliser nous avons un classifier contenant un .xml que nous avons renomme stopSign.xml

Code Arduino pour la camera:

camera.py

```
import cv2

stop_cascade = cv2.CascadeClassifier('stopSign.xml')

cap = cv2.VideoCapture(0)

while True:
ret, frame = cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

stop_signs = stop_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=1, minSize=(30, 30))

for(x, y, w, h) in stop_signs:
cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

cv2.imshow('Stop Sign Detection', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
break

cap.release()
cv2.destroyAllWindows()
```

Baldé boubacar 55870
Sacewicz Mathieu 54025

Photo de la détection d'un panneau stop :