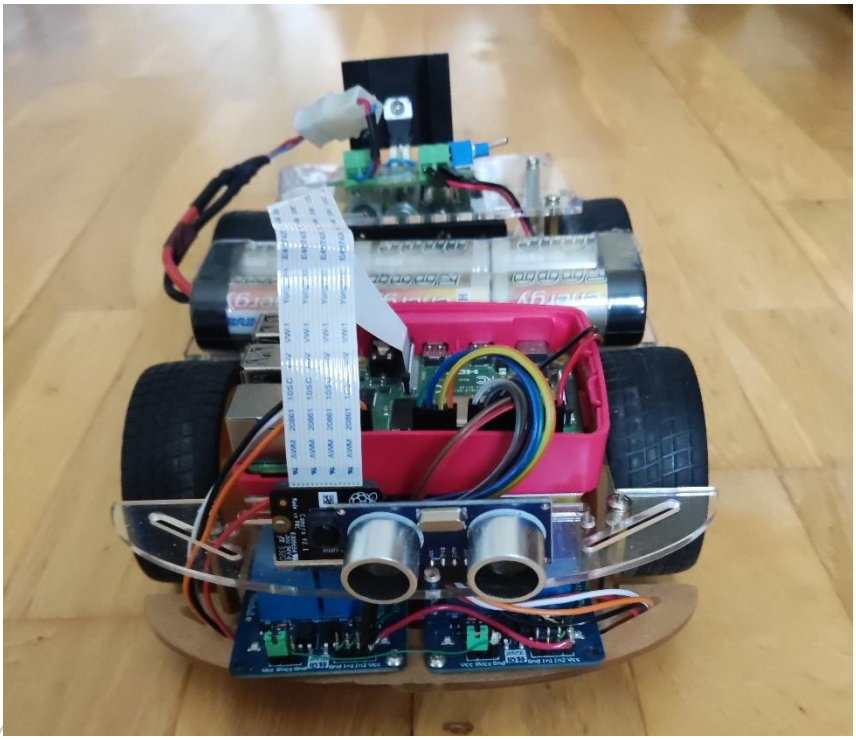
UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Compte rendu du projet

09/12/2021

Pilotage d'un Robot Car éviteur d'obstacle et reconnaissance d'objet



Raspberry Pi®

python™

TRAN Vincent
Master IIA

Sommaire

Remerciement..... 3

Présentation du projet 3

Spécification pour le besoin fonctionnel du robot 3

Capteur d'ultrason : HC-SR04 4

1) Présentation du capteur d'ultrason : HC-SR04..... 4
2) Branchement et fonctionnement du capteur 4
3) Calcul de distance de l'objet rencontré..... 5
4) Principe du test de mesure sur l'approche de l'obstacle 5

Pilotage du robot 6

1) Automatisation du robot lors du rencontre de l'obstacle 6
2) Fonctionnement de la caméra en temps réel 6

2.1. Présentation du module de la caméra 6

2.2. Librairie OpenCV..... 7

Reconnaissance objet : Réseaux neurones artificiels par MobileNet SSD..... 7

1) Classification de l'image : MobileNet..... 7
2) Détection de l'objet : Single Shot Multibox Detector 8
3) Implémentation de l'apprentissage du modèle MobileNet SSD..... 9
4) Application à la reconnaissance d'objet du Pi Caméra 10

Conclusion 11

Annexes 12

Annexe 1 : Branchement des connecteurs de la Raspberry Pi 4..... 12

Annexe 2 : Test fonctionnel de mesure de distance 12

Annexe 3 : Algorithme du programme du robot lors du rencontre d'un obstacle 13

Annexe 4 : Programme entier du projet robotique 14

Annexe 5 : Programme sur la reconnaissance d'objet MobileNetSSD 20

Remerciement

Je tiens à remercier à Mme Seddiki d'avoir confirmé la continuation du projet que j'avais travaillé en projet innovant avec M Ali Cherif et d'avoir fourni tous les matériels nécessaires pour mettre au point des nouvelles fonctionnalités du Robot.

Présentation du projet

A l'occasion de ce projet, j'ai continué ce projet que j'ai effectué l'année dernière pour améliorer les différentes fonctionnalités du robot.
En effet, l'objectif est de piloter le robot sans besoin d'interaction humaine grâce à un capteur d'ultrason et d'utiliser une caméra pour effectuer de la reconnaissance d'objet. Ce robot va permet de détecter les différents objets s'il est réellement dangereux pour intervenir dans la surveillance ou dans la sécurité routière par exemple.

Le robot est composé finalement :
-4 moteurs à courant continu fixé à 4 roues
-4 modules à relais sur 5V
-Deux plaques pour fixer les composants électroniques
-Un Raspberry Pi 4 modèle B
-Une batterie 5V/4A + une pile d'alimentation
-Un régulateur de tension contenant à un condensateur de découplage et d'un interrupteur électronique.
-Un module d'une caméra du type NOIR v2
-Un Micro SD
-Un capteur d'ultrason du type HC-SR04

Dans notre Raspberry Pi, j'ai utilisé dans une machine Linux (système d'exploitation Raspbian) qui va connecter à distance en réseau local l'environnement du bureau grâce au logiciel VNC Server.

Spécification pour le besoin fonctionnel du robot

Nous avons décidé d'enlever l'interface graphique et de remplacer par un nouveau script en Python pour rendre très autonome le robot. Plus précisément, le robot va automatiquement détecter les différents obstacles qui se trouve à la face avant du robot et d'identifier les objets en utilisant de la reconnaissance d'objet.

Pour que le robot soit autonome, il faudrait que le robot soit capable d'exécuter en boucle infini à plusieurs temps d'arrêt pour qui arrive à se retrouver sa vraie position du robot. Une fois qui retrouve sa position, le robot doit être capable de chercher le chemin où il souhaite d'aller.

Capteur d'ultrason : HC-SR04

1) Présentation du capteur d'ultrason : HC-SR04

Le capteur à ultrasons HC-SR04 utilise le principe du sonar pour déterminer la distance à un objet. Il offre une mesure sans contact avec une bonne précision et des lectures stables. Sur le module, nous trouvons un émetteur à ultrasons et un récepteur. Le capteur a une portée de 2 à 400 cm. La précision de la mesure est de +/- 0,5 cm.

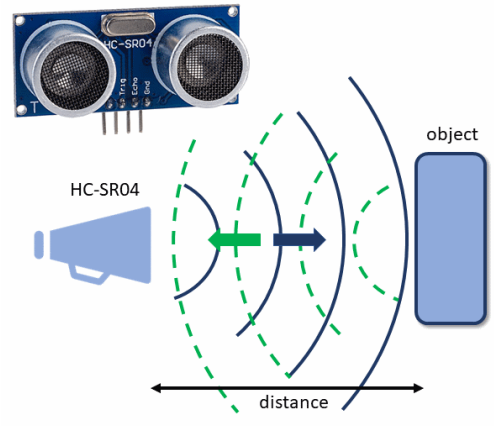
Caractéristiques :

- Dimensions : 45 mm x 20 mm x 15 mm
- Plage de mesure : 2 cm à 400 cm
- Résolution de la mesure : 0.3 cm
- Angle de mesure efficace : 15 °
- Largeur d'impulsion sur l'entrée de déclenchement : 10 µs (Trigger Input Pulse Width)

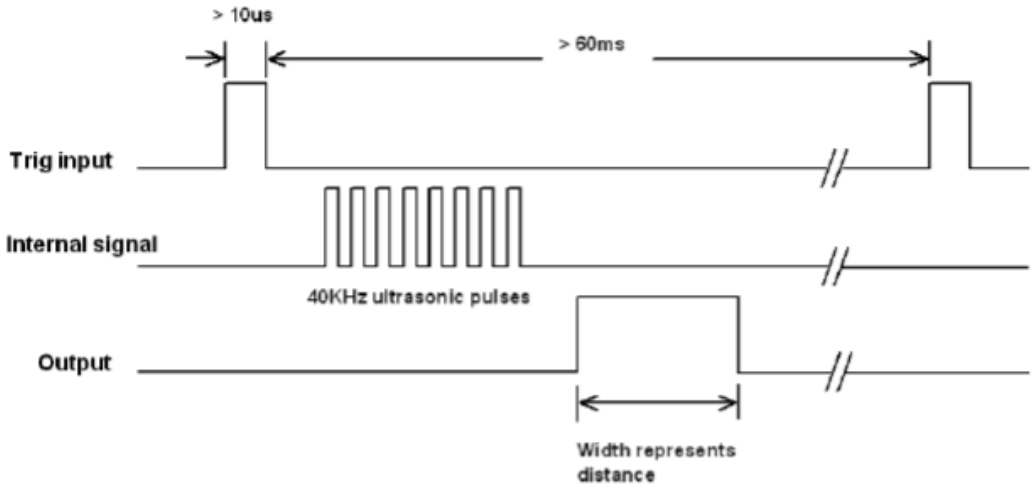
2) Branchement et fonctionnement du capteur

Il existe 4 broches de connexion :
• Vcc = Alimentation +5 V DC
• Trig = Entrée de déclenchement de la mesure (Trigger input)
• Echo = Sortie de mesure donnée en écho (Echo output)
• GND = Masse de l'alimentation

Le capteur utilise des signaux sonores ultrasoniques pour définir la distance:
– L'émetteur envoie un signal ultrason dans la direction de l'objet.
– Lorsque le signal atteint l'objet, il est renvoyé au capteur.
– Le récepteur peut détecter le signal réfléchi.



Pour déclencher une mesure, il faut présenter une impulsion "high" (5 V) d'au moins 10 µs sur l'entrée "**TRIG**". Le capteur émet alors une série de 8 impulsions ultrasoniques à 40 kHz, puis il attend le signal réfléchi. Lorsque celui-ci est détecté, il envoie un signal "high" sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.



3) Calcul de distance de l'objet rencontré

La distance parcourue par un son se calcule en multipliant la vitesse du son, environ 340 m/s par le temps de propagation, soit : ***d = v · t*** (***distance = vitesse · temps***).
Le HC-SR04 donne une durée d'impulsion en dizaines de µs. Il faut donc multiplier la valeur obtenue par 10 µs pour obtenir le temps t. On sait aussi que le son fait un aller-retour. La distance vaut donc la moitié.
On a donc utilisé finalement: ***d = (nb_impulsion*vitesse du son · temps)/2***
C'est-à-dire : ***distance = nb_impulsion*343 *100/2***

4) Principe du test de mesure sur l'approche de l'obstacle

Sur le script dans l'annexe 1, on a défini deux pins qui sont trigger = 21 et echo = 20. Et on constate d'après les résultats ci-dessous que si la distance est grande on a moins de risque de rencontrer un obstacle à l'inverse pour l'autre. La distance est mesurée en cm.

```
>>> %Run capteur.py
  Distance : 61.75433397293091
>>>
```

Loin de l'obstacle

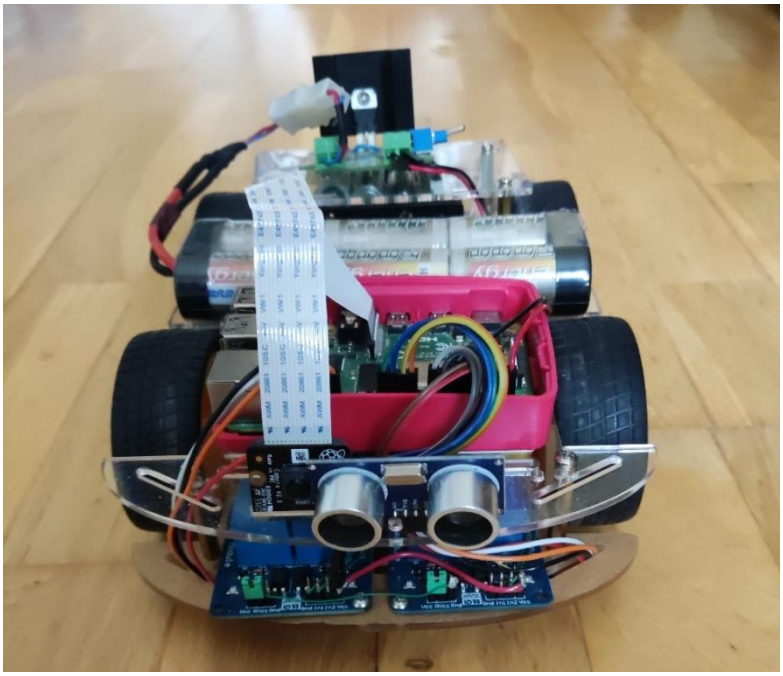
```
Python 3.7.3 (/usr/bin/python3)
>>> %cd /home/pi/Documents
>>> %Run capteur.py
  Distance : 21.989989280700684
>>>
```

Proche de l'obstacle

Pilotage du robot

1) Automatisation du robot lors du rencontre de l'obstacle

Une fois qu'on récupère la valeur de distance, il faudrait que notre robot soit capable de rencontrer les obstacles à l'infini. Pour cela on doit définir une boucle « while True » ce qui permet de récupérer toutes les valeurs de distances et puis on compare toutes les valeurs de distance si le robot est capable de rencontrer plus ou moins proche un obstacle.



Dans l'algorithme défini en annexe 3, pour que le robot soit capable de rencontrer un obstacle, on fixe une distance minimum à 30 et si sa distance est inférieure à 30, le compteur s'incrémente et le robot forcera à l'arrêt et au recul et au bout d'une seconde le robot va soit tourner à droite ou à gauche. Si non le robot continuera à avancer jusqu'à qui rencontre un obstacle.

2) Fonctionnement de la caméra en temps réel

2.1. Présentation du module de la caméra

Notre caméra est utilisée au module spécifique du Pi Camera. Voici les caractéristiques utilisées pour l'utilisation de la caméra. Le module de caméra de carte Raspberry Pi est à canal unique, 8 mégapixels Il est doté d'une capture de fréquence d'image maximale de 30 images/seconde. Cette carte caméra haute définition (HD) se connecte à n'importe quel Raspberry Pi ou module de calcul, ce qui vous permet de créer des vidéos HD et des photos fixes.

Caractéristiques :

- **Capteur Omnivision 5647** avec objectif à focale fixe
- Capteur 5 Mégapixels
- Résolution photo : 2592 x 1944
- Résolution vidéo maximum : 1080p
- Images par seconde maximum : 30fps
- Taille du module : 20 x 25 x 10mm
- Connexion par câble plat à l'interface 15-pin MIPI Camera Serial Interface (CSI) (Connecteur S5 du Raspberry Pi)

2.2. Librairie OpenCV

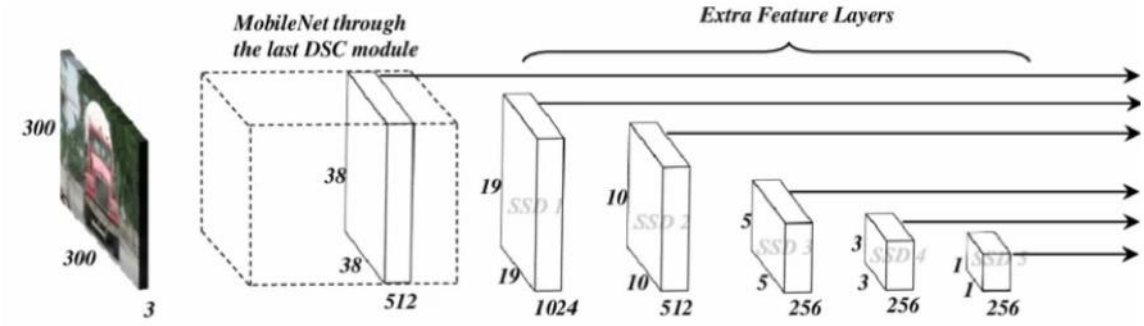
OpenCV qui signifie (Open Source Computer Vision Library) est une bibliothèque de logiciels open source de vision par ordinateur et d'apprentissage automatique. Cette bibliothèque contient au moins plus de 2500 algorithmes optimisés dans le Machine Learning qui peuvent être utilisés pour faire de la reconnaissance des visages, identifier des objets, suivre des objets en mouvement par exemple.

Pour notre Raspberry Pi, on pourra utiliser par la vision d'ordinateur et de l'apprentissage automatique pour transmettre des images et des vidéos en temps réel. La Raspberry PI est une excellente plateforme pour utiliser Open CV.

Reconnaissance objet : Réseaux neurones artificiels par MobileNet SSD

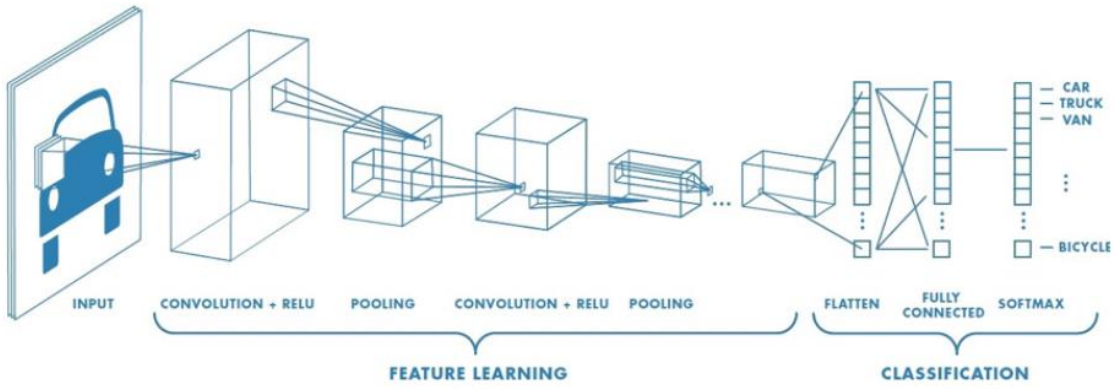
1) Classification de l'image : MobileNet

Pour commencer, MobileNet est une architecture de réseau de neurones utilisé spécifiquement le CNN (Convolution Neural Network) profonds légère conçue pour les mobiles et les applications de vision embarquée.



Architecture MobileNet

Il est utilisé dans de nombreuses applications du monde réel, telles qu'une voiture autonome, les tâches de reconnaissance doivent être effectuées sur un appareil limité en termes de calcul. C'est un algorithme utilisé pour **faire de la classification d'image et de la détection d'objet**. Cependant, ces réseaux neurones effectuent des convolutions qui sont des opérations très coûteuses en calcul et en mémoire.

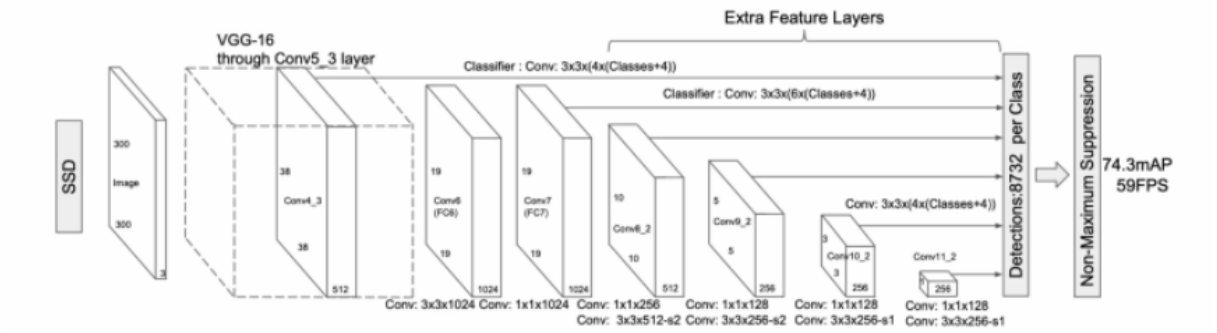


L'image ci-dessus représente la convolution standard dans le CNN. Pour déterminer le contenu de l'image, il existe deux phases :

- La phase Feature Learning : composée de plusieurs convolutions successives pour faire ressortir les caractéristiques de l'image
- La phase de Classification : permet de prédire si l'image en entrée soit par exemple une voiture, un piéton, un vélo,etc…

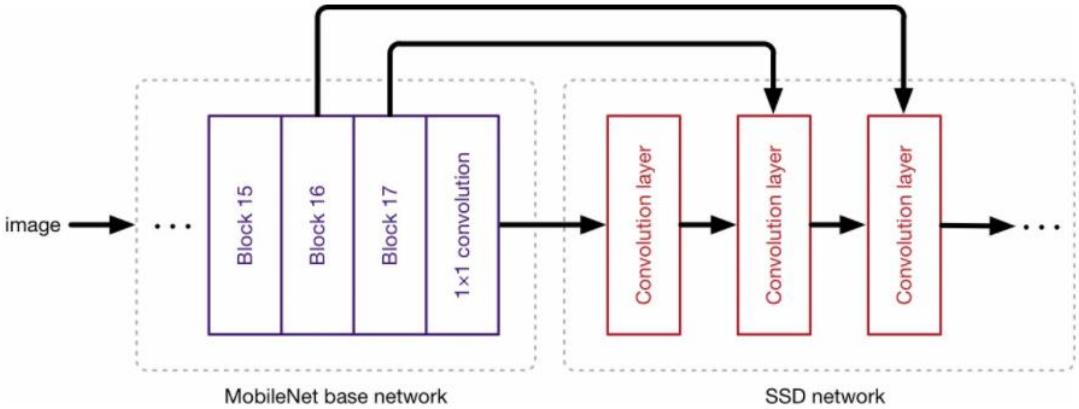
2) Détection de l'objet : Single Shot Multibox Detector

Le Single Shot Multibox detector est un Framework qui va permettre d'effectuer la détection d'objet en détectant plusieurs objets dans un seul image. Il est basé par un réseau convolutif **pour produire une collection de taille fixe de cadres de délimitions et de scores pour la présence d'instances de classe d'objets dans ces cadres**.



Architecture SSD

Le SSD est conçu pour être indépendant du réseau de base et peut donc fonctionner sur tous les réseaux de base tels que MobileNet ou YOLO. MobileNet a été intégré à la base dans le réseau SSD qui deviendra MobileNet SSD. En implémentant du SSD dans MobileNet on limitera l'exécution de réseaux de neurones à hautes ressources et à forte consommation d'énergie sur des appareils bas de gamme dans des applications en temps réel.



3) Implémentation de l'apprentissage du modèle MobileNet SSD

Globalement, le MobileNet SSD va permettre de diviser l'image à l'aide d'une grille et chaque cellule de la grille est responsable de la détection des objets dans cette région de l'image. Donc lorsqu'on effectue la détection d'objets, **cela signifie de prédire la classe et l'emplacement d'un objet dans cette région.** Si aucun objet n'est présent, nous le considérons comme la classe d'arrière-plan et l'emplacement est ignoré. On peut prendre par exemple d'utiliser une grille 4x4 et chaque cellule de la grille est capable d'afficher la position et la forme de l'objet qu'elle contient. Et chaque grille sont définies en fonction de sa taille et d'une forme

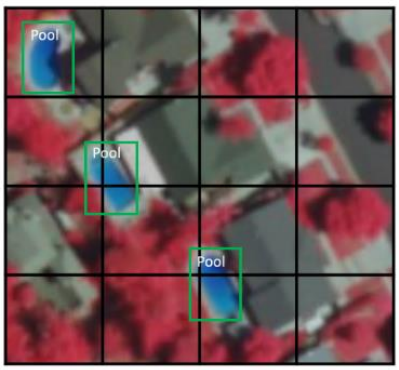
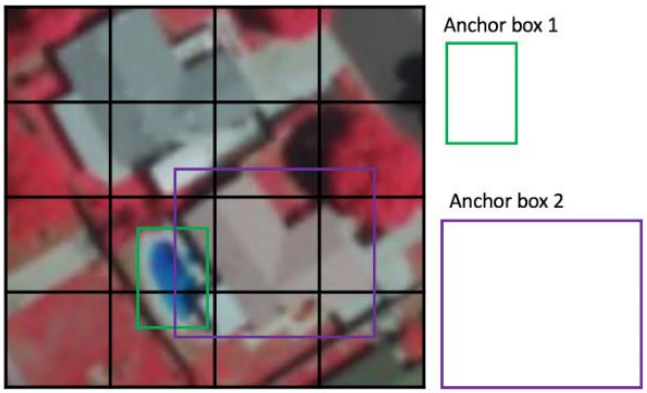


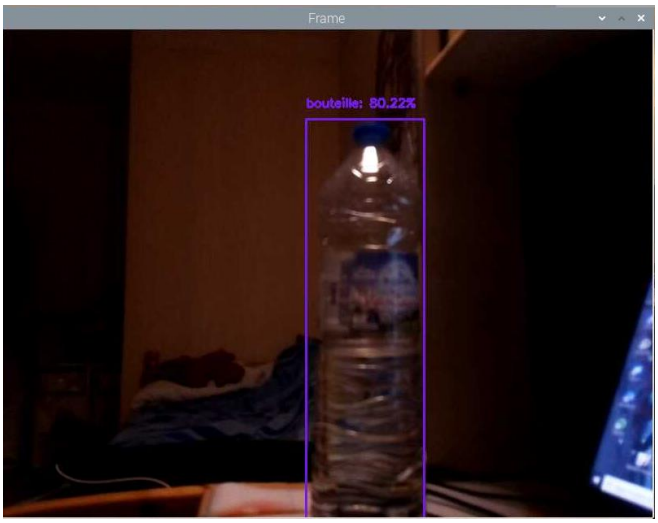
Figure 4. Example of a 4x4 grid



4) Application à la reconnaissance d'objet du Pi Caméra

Nous avons utilisé notre Raspberry pour implémenter notre module de caméra pour faire la reconnaissance d'objet. Il faudrait utiliser la librairie de OpenCV pour gérer des calculs de traitement d'images.
Notre projet de reconnaissance d'objet contient trois fichiers :
-Le programme python appelé : ***reconnaissance_objets.py*** représente notre algorithme qui effectuera la classification des images et la détection des objets.
-Le fichier ***MobileNetSSD_deploy.caffemodel*** contenant la liste des 21 types d'objets
-Le fichier de configuration : ***MobileNetSSD_deploy.prototxt*** qui servira à déployer les données dans le MobileNetSSD

Dans le code source reconnaissance_objets.py, nous avons modifié le code pour enlever la partie sur les envois des captures par email vu que ce n'est pas utile de partager et stocker les données.
Voici en image ci-dessous la prédiction des objets qui a pris en compte les objets obtenus.
On remarque dans **l'image qui a réussi à prédire la bouteille et la personne ne situent pas dans la même région et de plus la taille de la grille et les couleurs sont différents.**



Cependant, lors de l'exécution de la Raspberry Pi malgré les différents réglages (résolution, mémoire RAM,…) la Raspberry Pi n'est pas capable d'obtenir des meilleurs FPS (Frame per second) pour rendre le traitement d'image plus fluide lorsqu'on effectue le mouvement de notre caméra vu que les ressources de mémoire RAM sont trop lourds.

```
[INFO] elapsed time: 49.44
[INFO] approx. FPS: 0.63
```

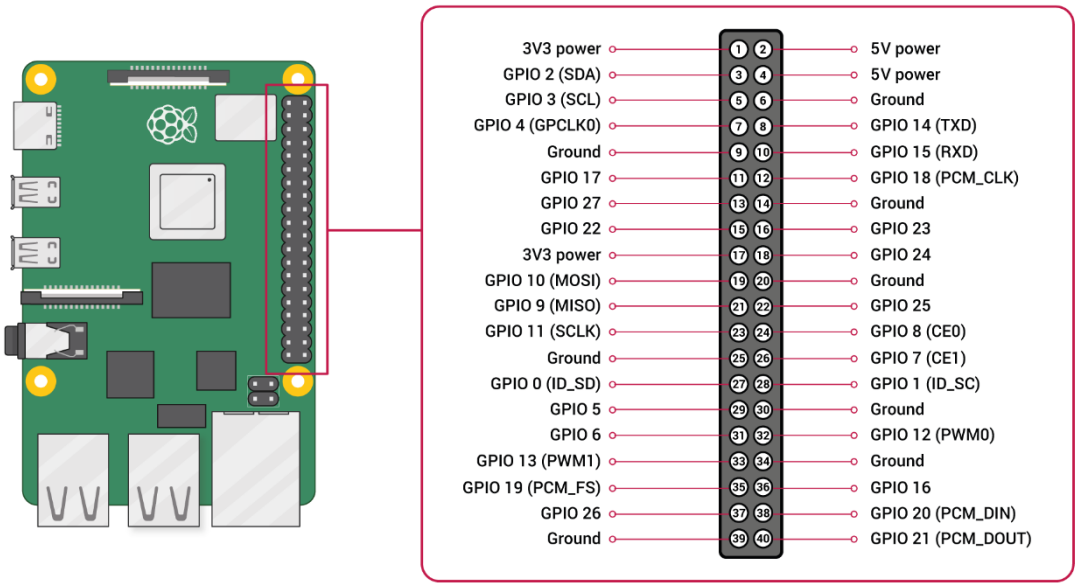
Conclusion

Le robot est plutôt bien opérationnel à l'intelligence du robot mais dispose beaucoup de contraintes qui nécessite des besoins spécifiques au niveau matériel et logiciel pour améliorer la performance du robot. En effet pour le contrôle du robot, la vitesse du moteur tourne en mode continu et il faudrait optimiser la vitesse du moteur tout en réglant l'asservissement du moteur si on avait plus de temps de travail en présentiel. Et à ce qui concerne à la reconnaissance d'objet, la Raspberry Pi n'est pas compatible pour gérer la fluidité du traitement d'image vu que lors de l'exécution du code demande trop de ressources en termes de mémoire ce qui engendre un ralentissement du système.
La durée de l'autonomie du robot n'est pas suffisante pour mettre en marche du robot Possiblement, on pourrait simplement d'ajouter une puce de mémoire pour augmenter la vitesse de l'exécution du système.

Finalement, d'après l'annexe 4, ce qu'on a principalement réussi à mettre en place, on a réussi à exécuter un script en python qui permet de commander le robot automatiquement sans besoin d'interagir et en parallèle de l'exécution, le robot prendra en charge l'enregistrement en capture vidéo dans un temps limité.

Annexes

Annexe 1 : Branchement des connecteurs de la Raspberry Pi 4

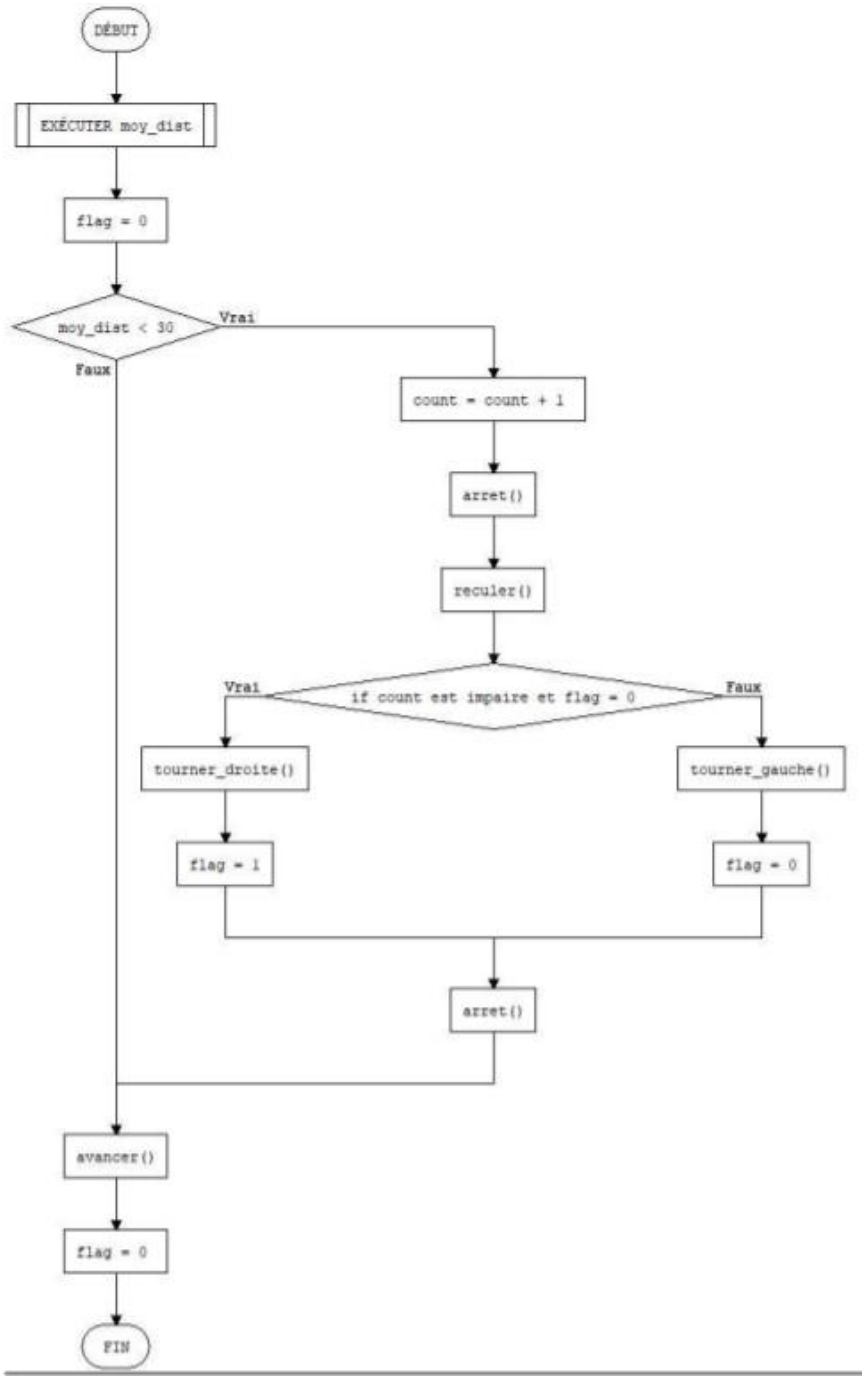


Annexe 2 : Test fonctionnel de mesure de distance

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
trigger = 21
echo = 20
GPIO.setup(trigger,GPIO.OUT)
GPIO.setup(echo,GPIO.IN)
GPIO.output(trigger,False) # Initialiation du Trigger à 0
time.sleep(0.1)
#Vérifie si le robot détecte un obstacle
GPIO.output(trigger,True)
time.sleep(0.00001) #On attend 100 ms d'impulsion
GPIO.output(trigger,False)
while GPIO.input(echo)==0:
    debutImpulsion = time.time()
```

```
while GPIO.input(echo)==1:
    finImpulsion = time.time()
distance = (finImpulsion - debutImpulsion)*343*100/2
#340 correspond à 340 m/s de vitesse du son et on multiplie par 100 pour convertir en cm/s
print("Distance en cm :",distance)
```

Annexe 3 : Algorithme du programme du robot lors du rencontre d'un obstacle



Annexe 4 : Programme entier du projet robotique

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
PIN1_M1 = 17
PIN2_M1 = 27
PIN1_M2 = 23
PIN2_M2 = 24
PIN1_M3 = 25
PIN2_M3 = 8
PIN1_M4 = 7
PIN2_M4 = 1
trigger = 21
echo = 20

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(PIN1_M1,GPIO.OUT)
GPIO.setup(PIN2_M1,GPIO.OUT)

GPIO.setup(PIN1_M2,GPIO.OUT)
GPIO.setup(PIN2_M2,GPIO.OUT)

GPIO.setup(PIN1_M3,GPIO.OUT)
GPIO.setup(PIN2_M3,GPIO.OUT)

GPIO.setup(PIN1_M4,GPIO.OUT)
GPIO.setup(PIN2_M4,GPIO.OUT)
```

```
GPIO.setup(trigger,GPIO.OUT)
GPIO.setup(echo,GPIO.IN)

def avancer():

   GPIO.output(PIN1_M1,GPIO.HIGH)
   GPIO.output(PIN2_M1,GPIO.LOW)

   GPIO.output(PIN1_M2,GPIO.LOW)
   GPIO.output(PIN2_M2,GPIO.HIGH)

   GPIO.output(PIN1_M3,GPIO.LOW)
   GPIO.output(PIN2_M3,GPIO.HIGH)

   GPIO.output(PIN1_M4,GPIO.HIGH)
   GPIO.output(PIN2_M4,GPIO.LOW)

   print('Avancer')

def reculer():

   GPIO.output(PIN1_M1,GPIO.LOW)
   GPIO.output(PIN2_M1,GPIO.HIGH)

   GPIO.output(PIN1_M2,GPIO.HIGH)
   GPIO.output(PIN2_M2,GPIO.LOW)

   GPIO.output(PIN1_M3,GPIO.HIGH)
   GPIO.output(PIN2_M3,GPIO.LOW)

   GPIO.output(PIN1_M4,GPIO.LOW)
```

```
    GPIO.output(PIN2_M4,GPIO.HIGH)
    print('Reculer')
def arret():

    GPIO.output(PIN1_M1,GPIO.LOW)
    GPIO.output(PIN2_M1,GPIO.LOW)

    GPIO.output(PIN1_M2,GPIO.LOW)
    GPIO.output(PIN2_M2,GPIO.LOW)

    GPIO.output(PIN1_M3,GPIO.LOW)
    GPIO.output(PIN2_M3,GPIO.LOW)

    GPIO.output(PIN1_M4,GPIO.LOW)
    GPIO.output(PIN2_M4,GPIO.LOW)

    print('Arret')

def tourner_droite():

    GPIO.output(PIN1_M1,GPIO.HIGH)
    GPIO.output(PIN2_M1,GPIO.LOW)

    GPIO.output(PIN1_M2,GPIO.LOW)
    GPIO.output(PIN2_M2,GPIO.LOW)

    GPIO.output(PIN1_M3,GPIO.LOW)
    GPIO.output(PIN2_M3,GPIO.LOW)

    GPIO.output(PIN1_M4,GPIO.HIGH)
    GPIO.output(PIN2_M4,GPIO.LOW)
```

```
    print('Reculer a droite')

def tourner_gauche():

    GPIO.output(PIN1_M1,GPIO.LOW)
    GPIO.output(PIN2_M1,GPIO.LOW)

    GPIO.output(PIN1_M2,GPIO.LOW)
    GPIO.output(PIN2_M2,GPIO.HIGH)

    GPIO.output(PIN1_M3,GPIO.LOW)
    GPIO.output(PIN2_M3,GPIO.HIGH)

    GPIO.output(PIN1_M4,GPIO.LOW)
    GPIO.output(PIN2_M4,GPIO.LOW)

    print('Reculer a gauche')

def reculer_droite():

    GPIO.output(PIN1_M1,GPIO.LOW)
    GPIO.output(PIN2_M1,GPIO.HIGH)

    GPIO.output(PIN1_M2,GPIO.LOW)
    GPIO.output(PIN2_M2,GPIO.LOW)

    GPIO.output(PIN1_M3,GPIO.LOW)
    GPIO.output(PIN2_M3,GPIO.LOW)

    GPIO.output(PIN1_M4,GPIO.LOW)
```

```
    GPIO.output(PIN2_M4,GPIO.HIGH)
    print('Reculer a droite')

def reculer_gauche():

    GPIO.output(PIN1_M1,GPIO.LOW)
    GPIO.output(PIN2_M1,GPIO.LOW)

    GPIO.output(PIN1_M2,GPIO.HIGH)
    GPIO.output(PIN2_M2,GPIO.LOW)

    GPIO.output(PIN1_M3,GPIO.HIGH)
    GPIO.output(PIN2_M3,GPIO.LOW)

    GPIO.output(PIN1_M4,GPIO.LOW)
    GPIO.output(PIN2_M4,GPIO.LOW)

    print('Reculer a gauche')

    time.sleep()

arret()
count = 0
while True:
    i= 0
    moy_dist=0
    for i in range(5):
        GPIO.output(trigger,False)
        time.sleep(0.1)
        GPIO.output(trigger,True)
        time.sleep(0.00001)
```

```
    GPIO.output(trigger,False)

  while GPIO.input(echo)== 0:
    debutImpulsion = time.time()

  while GPIO.input(echo)== 1:
    finImpulsion = time.time()

  Impulsion = finImpulsion - debutImpulsion
  distance = Impulsion*343*100/2
  moy_dist = moy_dist + distance
  print("Distance moyenne:",moy_dist)

  flag = 0
  if moy_dist < 30 :
    count = count + 1
    arret()
    time.sleep(1)
    reculer()
    time.sleep(0.5)
    if (count%3==1)& (flag==0):
      tourner_droite()
      flag = 1
    else:
      tourner_gauche()
      flag = 0
    time.sleep(1.5)
    arret()
    time.sleep(1)
  else:
    avancer()
```

```
    flag = 0


def Camera():
   camera = PiCamera()
   camera.rotation = 180
   camera.start_preview()
 camera.start_recording('/home/pi/Documents/PROJET_Robot_2021_2022/test.h264')
   time.sleep(5)
   camera.stop_recording()
   camera.stop_preview()
   print('Camera activé')


thread_1 = Thread(target=measure)
thread_2 = Thread(target=Camera)


thread_1.start()
time.sleep(1)
thread_1.join()
thread_2.start()
time.sleep(1)
thread_2.join()
```

Annexe 5 : Programme sur la reconnaissance d'objet MobileNetSSD

```
#import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
```

```
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
	help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
	help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
	help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initialiser la liste des objets entrainés par MobileNet SSD
# création du contour de détection avec une couleur attribuée au hasard pour chaque objet
CLASSES = ["arriere-plan", "avion", "velo", "oiseau", "bateau",
	"bouteille", "autobus", "voiture", "chat", "chaise", "vache", "table",
	"chien", "cheval", "moto", "personne", "plante en pot", "mouton",
	"sofa", "train", "moniteur"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# chargement des fichiers depuis le répertoire de stockage
print(" ...chargement du modèle...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialiser la caméra du pi, attendre 2s pour la mise au point ,
# initialiser le compteur FPS
print("...démarrage de la Picamera...")
vs = VideoStream(usePiCamera=True, resolution=(1600, 1200)).start()
time.sleep(2.0)
fps = FPS().start()
```

```
# boucle principale du flux vidéo
while True:
	# récupération du flux vidéo, redimension
	# afin d'afficher au maximum 800 pixels
	frame = vs.read()
	frame = imutils.resize(frame, width=800)
	# récupération des dimensions et transformation en collection d'images
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
		0.007843, (300, 300), 127.5)
# determiner la détection et la prédiction
	net.setInput(blob)
	detections = net.forward()
# boucle de détection
	for i in np.arange(0, detections.shape[2]):
# calcul de la probabilité de l'objet détecté
# en fonction de la prédiction
		confidence = detections[0, 0, i, 2]

# supprimer les détections faibles
# inférieures à la probabilité minimale
		if confidence > args["confidence"]:
			# extraire l'index du type d'objet détecté
			# calcul des coordonnées de la fenêtre de détection
			idx = int(detections[0, 0, i, 1])
			box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
			(startX, startY, endX, endY) = box.astype("int")

			# creation du contour autour de l'objet détecté
			# insertion de la prédiction de l'objet détecté
			label = "{}: {:.2f}%".format(CLASSES[idx],
```

```
                        confidence * 100)
                    cv2.rectangle(frame, (startX, startY), (endX, endY),
                        COLORS[idx], 2)
                    y = startY - 15 if startY - 15 > 15 else startY + 15
                    cv2.putText(frame, label, (startX, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
                    # enregistrement de l'image détectée
                    cv2.imwrite("detection.png", frame)
# affichage du flux vidéo dans une fenètre

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # la touche q permet d'interrompre la boucle principale
        if key == ord("q"):
            break

        # mise à jour du FPS
        fps.update()

# arret du compteur et affichage des informations dans la console
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

cv2.destroyAllWindows()
vs.stop()
```