Projet de Recherche (PRe)

Spécialité : Machine learning

Année scolaire : 2022-2023

Conduite en convoi basé sur la détection d'images

Mention de confidentialité

le document est non confidentie. Il peut donc être consultable en ligne par tous.

Auteur :
Vanessa Lopez Noreña

Promotion :
2024

Tuteur ENSTA Paris :
David Filliat

Tuteur organisme d'accueil :
Hossam Afifi

Stage effectué du 22/05/2023 au 18/08/2024

Nom de l'organisme d'accueil : Télécom SudParis
Adresse : 19 place Marguerite Perey
91120 Palaiseau - France

Vanessa Lopez Noreña/ Télécom SudParis

Remerciements

Je tiens à remercier mon tuteur Hossam AFIFI pour l'opportunité de travailler avec lui et de me soutenir dans le développement du projet en me donnant des conseils et des instructions. A Ghalid ABIBI pour m'avoir fourni le matériel nécessaire à ce développement et pour m'avoir guidé dans la continuité de mon stage. A M. Badii JOUABER pour le temps accordé au suivi et à l'accompagnement du projet. Au laboratoire SAMOVAR pour l'espace optimal et les bonnes conditions de travail.

Enfin, je tiens à remercier ma famille et mes amis pour m'avoir soutenue à toutes les étapes de ma formation.

Résumé

Le présent rapport synthétise le travail réalisé dans le cadre du projet de stage de recherche sur les véhicules autonomes. Il s'agit de développer un algorithme pour la conduire autonome de un modèle réduit de voiture capable de suivre un véhicule "leader" en utilisant la vision par ordinateur et l'apprentissage automatique avec des réseaux neuronaux pour le détecter au sein du laboratoire SAMOVAR.

Dans cette perspective, nous aborderons dans le cadre du projet un algorithme de détection d'objets *YOLO*, le traitement d'images, les protocoles de communication pour la connexion des dispositifs, les mécanismes de contrôle du véhicule et les potentielles améliorations envisageables.

Mots-clés : YOLO, traitement d'images, détection d'objets, apprentissage automatique, vision artificielle.

Abstract

This report describes the work carried out as part of the research internship on autonomous vehicles. which consists of the development an autonomous car capable of following a leader vehicle, using computer vision and machine learning with neural networks to detect it in the SAMOVAR laboratory.

In this context, the project will cover a *YOLO* object detection algorithm, image processing, communication protocols for connecting devices, vehicle control mechanisms and potential improvements.

Keywords : YOLO, image processing, object detection, machine learning, computer vision.

Table des matières

Table des figures

Introduction

Actuellement, les systèmes de transport intelligents (STI) sont cruciaux pour améliorer l'utilisation des véhicules. La conduite autonome a la capacité de révolutionner l'approche des défis liés à la circulation, à la consommation d'énergie et à la sécurité routière.

La conduite en convoi repose principalement sur la conduite autonome. Un convoi est formé par un ensemble de véhicules autonomes interconnectés le long d'un trajet, avec un véhicule de tête appelé "Leader". Un leader dirige des véhicules autonomes en formation, régulant leur vitesse et améliorant la sécurité avec la réduction des distances entre eux.

L'objectif du stage consiste à concevoir un algorithme pour la conduire autonome de un modèle réduit de voiture apte à suivre un véhicule en utilisant des réseaux neuronaux pour la détection. À cette fin, deux véhicules robotique ELEGOO est disponible dans les installations du laboratoire SAMOVAR et nous proposons d'utiliser *YOLOv8*, un outil d'analyse d'image et détection d'objets développé par Joseph Redmon et Santosh Divvala à l'Université de Washington, afin de gérer la détection du vehicule dans le flux vidéo.

La méthodologie mise en œuvre pour le projet consiste à établir d'abord une détection des personnes, avec des réseaux neuronaux déjà entraînés. Par la suite, lorsque ce processus est fonctionnel, ce développement est transféré à l'identification et au suivi d'un véhicule ELEGOO, en caractérisant un réseau neuronal avec d'images de cette voiture et d'autres similaires. Pour ce faire, un flux vidéo capturé par une carte Raspberry est envoyé via une communication wifi à un ordinateur pour exécuter la détection d'objet, qui, lorsqu'il a identifié la cible et sa position, transmet certaines commandes de direction du véhicule à une carte arduino qui contrôle les moteurs pour suivre l'objet détecté.

Ce rapport décrit en détail la progression de l'élaboration du prototype mentionné ci-dessus, il s'articule autour de trois chapitres : le premier chapitre présentera les composants utilisés ainsi que l'architecture et le fonctionnement pour le développement du projet. Le deuxième chapitre abordera la communication entre les systèmes et l'utilisation de l'architecture de détection d'objets *Yolo* pour l'identification et le suivi des personnes. Le troisième chapitre sera consacré à l'entraînement d'un réseau neuronal avec *Yolo* pour la détection des voitures ELEGOO. Enfin, le dernier chapitre concrétisera tout le travail et les possibilités d'amélioration qui peuvent se présenter dans le cadre du projet.

I Composition globale du véhicule

La construction du véhicule a été essentiellement basée sur le kit robotique ELEGOO V3.0 de développement qui est partie du matériel déjà existant dans le laboratoire pour l'élaboration des projets de recherche au sein du SAMOVAR. On a fait des modifications sur les codes et l'ajout de nouveaux composants pour améliorer les fonctionnalités et les performances spécifiques pour ce projet.

L'architecture générale du projet est structurée en plusieurs catégories clés. Un capteur utilisant une caméra Raspberry, des unités de traitement comprenant une Carte Raspberry et une Carte ELEGOO Uno, des actionneurs représentés par des moteurs, un pré-actionneur qui est le driver des moteurs, et enfin une alimentation assurée par des batteries externes.

FIGURE 1 – Smart Robot Car Kit V3.0. Source : ELEGOO

I.1 Technologies utilisées

Microcontrôleur Elegoo uno R3

Le Microcontrôleur Elegoo Uno R3 est une carte de développement basée sur l'Arduino Uno avec des caractéristiques améliorées. Cette carte est équipée d'un microcontrôleur ATmega328P de haute qualité avec un cadencé à 16 MHz. Le Microcontrôleur Elegoo Uno R3 peut être programmé à l'aide du langage Arduino, qui est basé sur le C/C++. Le rôle de la carte ELEGOO UNO est de contrôler la direction et la vitesse des moteurs au moyen de commandes reçues de la carte Raspberry.

FIGURE 2 – ELEGOO UNO R3
Source : ELEGOO

Driver moteur L298N double

Le L298N est un double pont en H qui permet de contrôler la vitesse en utilisant la modulation de largeur d'impulsion (PWM) et la direction des moteurs à courant continu (DC) en utilisant un système de pont en H pour inverser le courant dans les enroulements du moteur, permettant ainsi de faire tourner les moteurs de manière indépendante dans les deux sens.

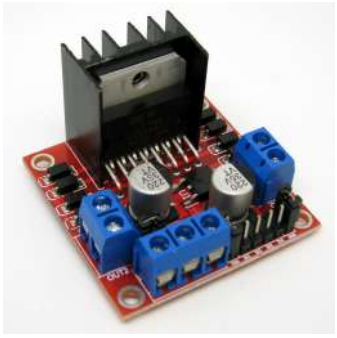


FIGURE 3 – Driver L298N

I.2 Composants ajoutés

Raspberry Pi 3 Model B+

La Raspberry Pi 3 est équipée d'un processeur quad-core 64 bits ARM Cortex-A53 cadencé à 1,4 GHz, soutenu par 1 Go de mémoire RAM. Elle est intégrée de Wi-Fi, Bluetooth, ports USB pour connecter des périphériques externes, HDMI pour la sortie vidéo, Ethernet pour une connexion réseau filaire rapide, un lecteur de carte microSD pour le stockage du système d'exploitation et des données, ainsi que d'un connecteur GPIO. On a utilisé la carte Rapsberry pour mettre en place l'envoi de flux au l'ordinateur pour faire le traitement d'images et recevoir les commandes.

FIGURE 4 – Raspberry Pi 3 model B+
Source : Raspberry

Raspberry Camera v2.1

Afin de permettre la detection des personnes, on a utilisé la camera Raspberry Pi V2.1 pour sa compatibilité et sa facilité d'utilisation. Elle est équipée d'un capteur Sony IMX219 de 8 mégapixels, capable de capturer des images avec une résolution allant jusqu'à 3280 x 2464 pixels. La caméra se connecte facilement à la Raspberry Pi via le port CSI (Camera Serial Interface), et son utilisation est prise en charge par divers logiciels et bibliothèques, ce qui la rend compatible

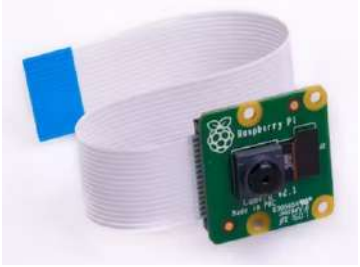


FIGURE 5 – Camera Raspberry V2.1
Source : Raspberry

avec une variété d'applications.

Batterie externe

Une batterie externe, est un appareil portable qui permet de recharger des appareils électroniques. Elle est équipée d'une batterie lithium-polymère, capable de stocker une certaine quantité d'énergie électrique. Elle est rechargeable à l'aide d'un câble USB et d'un adaptateur secteur standard. La batterie Li-Po modèle 1700-0067 avec une sortie de 5v-2.4A (max) est utilisée pour alimenter la carte Raspberry sans qu'il soit nécessaire de la brancher directement sur une prise de courant, ce qui évite au véhicule d'être relié à un câble d'alimentation.

FIGURE 6 – Batterie Ansmann 5.4
Source : Ansmann

Point d'access TL-MR3020

Nous avons utilisé le point d'accès *TL-MR3020* version 1 de la société *TP-LINK* du matériel de laboratoire existant. Il est un routeur portable permet de partager une connexion haut débit 3G/4G avec une vitesse sans fil jusqu'à 150 Mbps.Il suffit de se connecter à un réseau wifi créé par l'appareil et d'entrer le mot de passe défini dans le dispositif. Cet point d'accès est d'une importance cruciale dans le projet pour la communication entre le raspberry et l'ordinateur.

FIGURE 7 – Point d'access
Source : TP-LINK

Ordinateur Linux

Un ordinateur est nécessaire en raison du poids informatique du traitement de l'image. Le programme principal s'exécute sur cet appareil, qui effectue la détection de la cible et décide des actions à effectuer par la voiture. Les spécifications de l'ordinateur *Linux* basé sur le projet *GNU* utilisé se trouvent en annexe A.1.

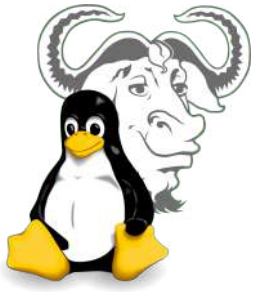


FIGURE 8 – Linux / GNU
Source : lignux

I.3 Architecture finale du véhicule

Après avoir pris connaissance de tous les composants du projet, le point d'accès est connecté à l'ordinateur pour l'alimenter et établir la connexion entre eux. Du côté du véhicule, tous les périphériques de la carte Raspberry, tels que l'appareil photo et la batterie externe, sont connectés. Une connexion est également établie au moyen d'un câble USB 2.0 de type A/B entre les cartes Raspberry et arduino, qui seront par la suite le moyen de communication série pour l'envoi de données permettant de contrôler la direction du véhicule. Le branchement physique de la voiture est illustré le plus clairement sur la figure 9.

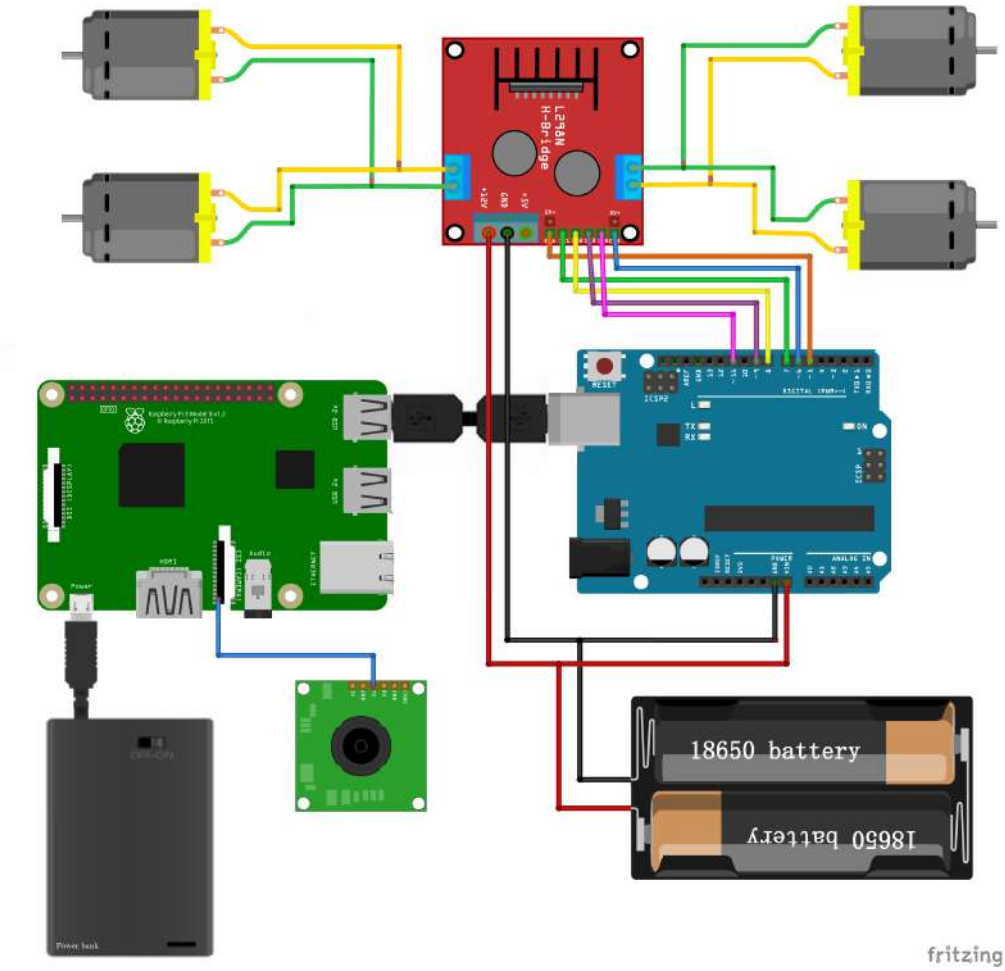


FIGURE 9 – Architecture de la voiture

II Détection des personnes

Ce chapitre a pour objet la description étape par étape du projet, ainsi que les améliorations et les erreurs commises au cours du processus. Une introduction aux réseaux neuronaux convolutifs sera donnée pour comprendre l'identification d'objets avec l'algorithme *Yolov8*.

Les réseaux de neurones artificiels (ANN) sont principalement constitués d'un grand nombre de nœuds de calcul interconnectés (appelés neurones), qui travaillent de manière distribuée pour apprendre collectivement à partir des données d'entrée afin d'optimiser la sortie finale. Les données sont introduites dans la couche d'entrée, qui les distribue aux couches cachées comme illustré à la figure 10. Les couches cachées prennent alors les décisions de la couche précédente et évaluent comment un changement stochastique en leur sein diminue ou améliore le résultat final, ce que l'on appelle le processus d'apprentissage. Avoir plusieurs couches cachées empilées est communément appelé apprentissage en profondeur. [1]

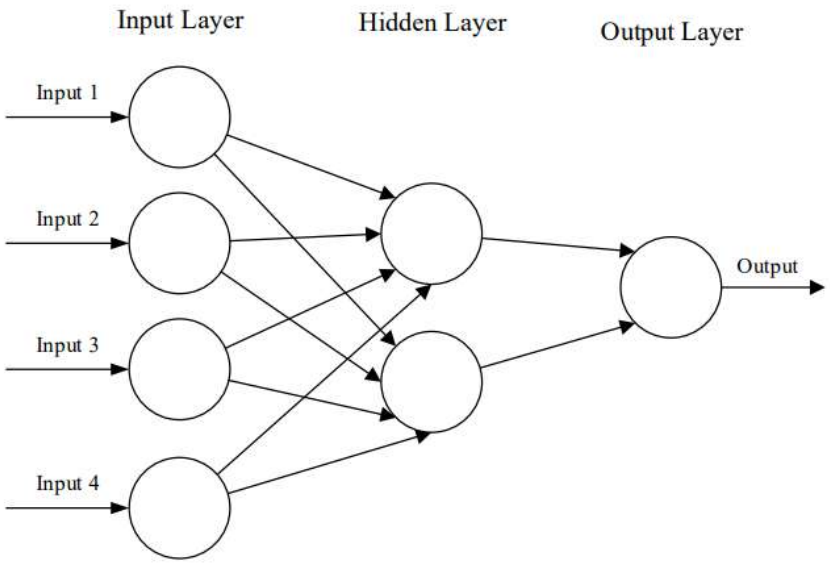


FIGURE 10 – Un réseau neuronal simple à trois couches, composé d'une couche d'entrée, d'une couche cachée et d'une couche de sortie. Source: An Introduction to Convolutional Neural Networks

Les CNN sont analogues aux ANN traditionnels, mais les CNN sont principalement utilisés dans le domaine de la reconnaissance des formes dans les images. Cela nous permet d'encoder des caractéristiques spécifiques à l'image dans l'architecture, ce qui rend le réseau plus adapté aux tâches centrées sur l'image comme la détection de modèles tels que les bords, les lignes verticales ou horizontales, etc.

Les modèles de détection d'objets basés sur des réseaux de neurones profonds gagnent en popularité en raison de leur capacité à apprendre des caractéristiques complexes à partir des données. **YOLO** (You Only Look Once) est une architecture de réseau de neurones convolutifs (CNN) conçue pour détecter rapidement et précisément des objets dans des images en temps réel. YOLO divise l'image en une grille de cellules, prédit plusieurs boîtes de détection, calcule la confiance avec l'objet cible et élimine les boîtes avec une confiance inférieure au seuil fixé. La version 8 de YOLO, également connue sous le nom de YOLOv8, développée par Ultralytics, propose une approche polyvalente avec un vaste ensemble de données étiquetées pour la prédiction, la validation, l'entraînement et l'exportation de modèles.

Nous partons de la base établie en *Yolov8* pour tester l'algorithme avec la détection de personnes et tout le traitement d'image que cela implique, puis nous le transposons à la voiture. L'ensemble du processus de détection et de suivi d'une personne sera réalisé afin de développer un programme fonctionnel et après, il suffit de remplacer la détection des personnes par celle des véhicules.

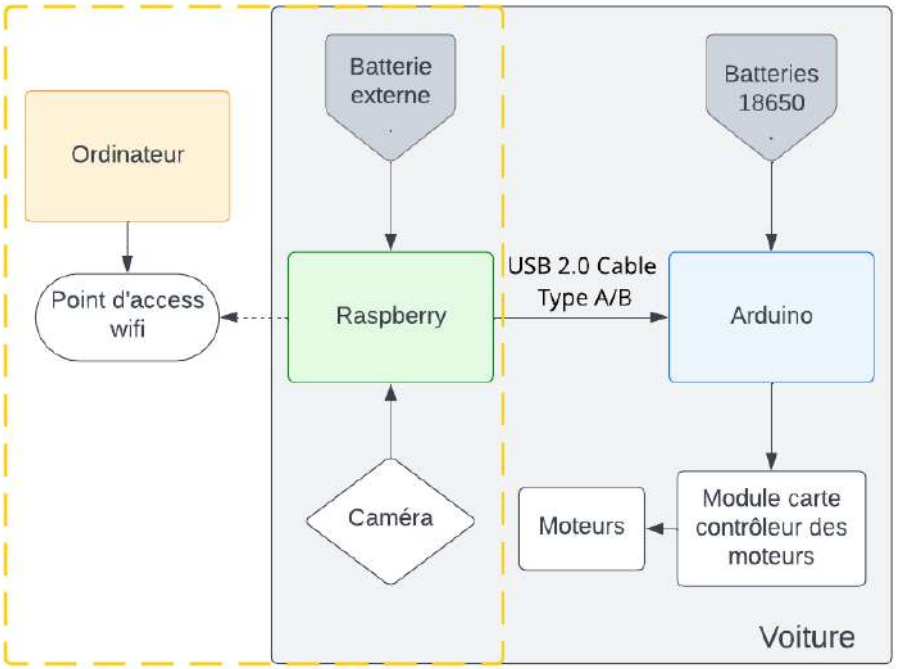


FIGURE 11 – Architecture du projet

Pour mettre en œuvre ce mécanisme, un système d'exploitation est téléchargé sur la carte Rapsberry pour activer ses fonctionnalités de base et permettre au matériel de fonctionner efficacement, puis la caméra est configurée sur ce dispositif pour envoyer le flux vidéo à l'ordinateur qui exécute la détection des personnes et transmet les commandes nécessaires pour modifier la trajectoire du véhicule qui suit la cible. Enfin, les instructions reçues par le raspberry sont adressées à la carte Arduino, qui pilote les mouvements du véhicule en contrôlant les moteurs.

II.1 Système d'exploitation (OS) Raspberry Pi

Le développement du projet commence par le téléchargement d'un système d'exploitation sur la Raspberry pour gérer ses ressources, exécuter des programmes et fournir une interface permettant aux utilisateurs d'interagir avec lui. Pour effectuer cette tâche, nous avons utilisé le logiciel *balenaEtcher*, lequel est un puissant flasher d'image de système d'exploitation (il transfère une copie exacte d'un OS enregistré sous forme d'image sur une unité de stockage) construit avec des techniques web pour établir un le flashage sûr d'une carte SD ou d'une clé USB en protégeant contre l'écriture accidentelle sur les disques durs et garantit que chaque octet de données a été écrit correctement [2].

Ensuite, avec l'aide du *balena Etcher*, le système d'exploitation *Raspberry Pi OS (64-bit) with desktop* a été installé pour faciliter la visualisation du programme et la communication sur une carte micro SD 8GB. Ce système a été choisi pour avoir une visualisation de bureau avec de bonnes performances et occuper peu d'espace dans la mémoire SD, puisque seulement 8 GB micro SD est disponible pour le développement de ce projet.

Le système d'exploitation est de type 64 bits, ce qui signifie qu'il peut tirer parti des avantages tels que des performances améliorées, augmentation des processus multitâches et une meilleure gestion et capacité de la mémoire par rapport à un système 32 bits, Un noyau de 6.1 qui gère de nombreux détails fondamentaux du système d'exploitation, notamment la mémoire, le stockage sur disque et les réseaux de bas niveau [3] et un version Debian 11 (bullseye). Debian est une association afin de créer un système d'exploitation *open source*. Sa distribution est caractérisée par un grand nombre de logiciels pré-compilés de haute qualité et la prise en charge d'un grand nombre d'architectures matérielles. Les éléments des logiciels libres de Debian proviennent de GNU, Linux, GNOME, Mozilla, Python, Java, MariaDB, SQLite et de nombreux autres projets de logiciels libres indépendants. Debian intègre cette diversité de logiciels libres dans un seul système [4].

II.2 Configuration Camera Raspberry

La caméra est connectée à la carte Raspberry via le port spécifique à cette périphérie CSI (Camera Serial Interface) et en utilisant l'outil `libcamera-hello` sur le terminal linux, est possible vérifier la connexion et la configuration correctes de la caméra avec le système. En cas d'erreur, l'outil `vcgencmd` est utilisé pour sortir des informations du GPU VideoCore sur le Raspberry Pi qui, avec la commande `get_camera`, affiche le status activé et la détection de la camera Raspberry Pi, où 1 signifie oui, 0 signifie non. Cette prise en charge doit être activée à l'aide de *raspi-config* qui est un outil permettant de configurer le système tels que options de réseau, d'interface, de démarrage, etc. Pour la configuration de la caméra, il faut aller dans la partie options d'interfaçage pour contrôler la connexion avec les périphéries et activer l'interface de la caméra CSI. [5, Section Configuration]

Une fois cette configuration terminée, à l'aide de la commande pour la détection des caméras : `vcgencmd get_camera`, le système fournit des informations sur l'état de la caméra et sa détection. Si la caméra du système est active et détecte par la carte Raspberry, aucun changement au fichier de configuration n'est requis. Dans le cas contraire, la carte Raspberry utilisée peut ne pas avoir par défaut le bon pilote d'affichage (`Error: *** no cameras available ***`) Alors, il faut aller au fichier *config.txt* et assurer que `dtoverlay=vc4-fkms-v3d` ou `dtoverlay=vc4-kms-v3d` est actuellement actif.

L'annexe A.2 contient la configuration utilisée pour le projet, où un module de caméra Raspberry a été spécifié et l'annulation de la détection automatique de la caméra a été nécessaire avec l'instruction `camera_auto_detect=0`. [5, Section Camera software]. Après avoir obtenu la confirmation de l'état actif et de la détection de la caméra, la commande *libcamera* mentionnée ci-dessus devrait fonctionner.

Il convient de noter qu'initialement, l'application *raspicam* a été utilisée pour vérifier le fonctionnement de la caméra, mais selon les nouvelles publiées en novembre 2021, par Raspberry, les nouvelles versions du Raspberry Pi OS ne prendront plus en charge les applications raspicam et la bibliothèque Picamera Python. Comme alternative, *Picamera2* est proposé, ainsi que le logiciel open source *Libcamera*, qui est basé sur Linux. [6]

La détection de la caméra a posé de nombreuses difficultés, car les composants avaient déjà été utilisés dans le cadre de projets antérieurs par conséquent, l'usure et la fatigue pouvaient interrompre les connexions ou provoquer des défaillances matérielles.

II.3 Communication Raspberry - Ordinateur

La connexion entre la carte Raspberry et l'ordinateur a été réalisée via un point d'accès alimenté par l'ordinateur, en choisissant la fonction de routeur 3G/4G, qui est utilisée pour créer un réseau 3G privé avec des appareils locaux. Ensuite, il suffit de connecter l'ordinateur et la carte Raspberry au point d'accès sans fil avec le nom d'utilisateur et le mot de passe spécifiés sur le router.

Pour tester la connexion entre ces appareils, un appel est effectué vers l'autre appareil en utilisant la commande *ping* et son adresse IP. Si aucune réponse n'est reçue du destinataire, vérifiez la connexion au point d'accès des deux appareils.

II.4 Envoyer le flux vidéo à l'ordinateur

Une fois que la caméra est correctement configurée et opérationnelle avec la carte Raspberry Pi et la communication est établie via le point d'accès avec l'ordinateur comme illustré dans le carré avec les bordures en jaune pointillé de la figure 11, un flux vidéo est envoyé à un ordinateur pour exécuter la détection d'objets sur cet appareil, étant donné que les caractéristiques des dispositifs ont permis d'observer que le traitement et l'identification des images sont extrêmement lents pour une carte Raspberry model 3.

Enfin, les commandes du tableau suivant sont utilisées sur le terminal de chaque appareil pour vérifier la communication et l'envoi de données entre eux.

Sur la carte Rapsberry :	`libcamera-vid -t 0 --width 1920 --height 1080 --codec h264 -inline --listen -o tcp://address-IP-Ordinateur:8888`
Sur l'ordinateur	`vlc tcp/h264://address-IP-Raspberry:8888/`

Pour cette connexion, nous utilisons le protocole TCP/IP qui ne nécessite que de fournir l'adresse IP et le port attribué à la communication. La commande de la carte Raspberry exécute la bibliothèque *Libcamera* mentionné ci-dessus pour prendre la vidéo et les autres paramètres correspondent aux configurations du flux vidéo à envoyer, tels que la taille de la vidéo, le type de compression, la diffusion en ligne, etc. D'autre part, la commande de l'ordinateur, utilise la bibliothèque *VLC media player*, qui est un lecteur vidéo pour recevoir le flux vidéo avec l'adresse IP et le port définis.

Le "Transmission Control Protocol" (TCP) est un protocole de couche de transport, et le "Internet Protocol" (IP) est un protocole de couche de réseau.

La couche de réseau comprend la transmission de l'intégrité des données pour une connexion point à point afin que les données ne soient pas perdues ou dupliquées. La couche accomplit

cette tâche en maintenant l'ordre séquentiel des trames transmises sur une connexion de données et en détectant et en corrigeant les erreurs de transmission par des retransmissions.

La couche de transport fournit une transmission sans erreur à l'utilisateur final. Pour améliorer l'utilisation, elle multiplexe plusieurs connexions de transport sur une connexion réseau. Elle contrôle le flux de données pour éviter de surcharger les ressources du réseau. [7]

La commande effectué ci-dessus de connexion via CLI ("command line interface") transmet avec succès le flux vidéo entre les appareils, cependant il y a un retard de 2 seconds dans la transmission et il n'est pas pratique pour le projet car il faut lancer la commande à chaque fois depuis le terminal pour établir la connexion et, plus tard, il sera nécessaire de l'intégrer à d'autres fonctions requises pour le projet.

Par conséquent, nous avons utilisé "Hypertext Transfer Protocol" (HTTP) qui est un protocole au niveau de l'application pour les systèmes d'information distribués, collaboratifs et hypermédias. La communication HTTP s'effectue via TCP/IP [8] avec le code "A simple webserver to stream MJPEG video to a web page", basé sur la section "Streaming to a network" du chapitre 9 du manuel *The Picamera2 Library*. [5, Section Datasheets/Camera].

Pour envoyer le flux vidéo de la carte Raspberry à l'ordinateur, le code pour créer un serveur web est exécuté sur l'appareil Raspberry. D'autre part, nous allons recevoir la vidèo sur le navigateur web de l'ordinateur avec l'adresse suivante : `http://address-IP-Raspeberry:8000` pour afficher le flux est en cours de transmission.

II.5 Intégration de flux vidéo dans YOLOV8

Nous utilisons cet outil dans un premier temps pour la détection des personnes, par conséquent un premier test CLI a été effectué sur le terminal du système proposée par le guide de *Ultralytics Yolov8 Docs* [9, Section Quickstart] en utilisant la commande suivante :

```
yolo predict model=yolov8n.pt source='https://address-IP-Raspberry:8000/stream.mjpg' imgsz=320
```

Cette commande permet de détecter des personnes en utilisant le flux vidéo envoyé par la caméra Raspberry avec une grande rapidité et une bonne précision. Cependant, comme indiqué précédemment, il faut lancer la commande à chaque fois depuis le terminal pour effectuer la détection et, par la suite, il sera nécessaire de faire un traitement des boîtes de délimitation afin de développer le programme et de prendre des décisions sur la trajectoire à suivre par le véhicule.

Nous avons donc décidé de réaliser un programme en *Python* basé sur le modèle pré-configuré *yolov8n.pt* avec des poids déjà entraînes de l'algorithme *YOLO V8*, obtenus à partir d'une base de données COCO (Common Objects in Context) de 330.000 images, composée de diverses catégories pour l'apprentissage des réseaux neuronaux. Cette pré-configuration du modèle Yolov8 nano est choisie car il s'agit de la version la plus rapide et la plus petite de Yolov8, étant donné qu'un modèle pas trop lourd est nécessaire pour ce type d'identification afin de déterminer une action pour la trajectoire.

Initialement, nous avons suivi la méthode "Streaming Source for-loop" proposée par Ultralytics [9, Section modes/predict] pour la détection d'objets dans une vidéo. Cependant, le traitement de l'image avec la bibliothèque Python *Opencv* a généré un delai si important dans l'image qu'elle a dû être écartée. Par contre, il a été décidé d'afficher l'image à l'aide de la fonction *plot* de Yolo [9, Section modes/predict], qui permet un affichage simple et rapide de l'image avec en utilisant Opencv seulement pour afficher l'image et éviter les retards présentés ci-dessus avec tout le traitement de cette bibliothèque.

Les paramètres permettant d'obtenir le comportement souhaité pour la prédiction, ont été configurés comme indiqué dans le tableau 1 pour détecter uniquement la classe des personnes, avec un maximum d'une personne par image et un seuil de confiance de l'objet supérieur à 0,5, c'est-à-dire que le programme doit parvenir à une détection supérieure à 50% de sécurité pour être considéré comme une détection valide.

Argument	Valeur	Description
stream	True	Traitement des données à l'aide du protocole d'adresse IP
classes	0	Filtrer les résultats par classe, 0 est la catégorie personnes
max_det	1	Nombre maximal de détections par image
conf	0.5	Seuil de confiance de l'objet pour la détection

TABLE 1 – Paramètres extraits de Ultralytics

Après avoir effectué toutes les configurations pour la détection, la prédiction est faite en suivant les 2 premières lignes de l'algorithme 1 et le résultat est un cadre de limitation autour de la personne, indiquant le seuil de confiance et la catégorie "personne".

II.6 Instructions pour déterminer les actions du véhicule

Dans cette section, nous procédons à l'identification de la position des objets dans l'espace de l'image afin de localiser le centre du boîte de délimitation qui détecte la personne et l'are dans laquelle elle se trouve afin de déterminer le degré de déplacement latéral par rapport au centre et l'éloignement de la cible par rapport au véhicule, respectivement.

En identifiant le déplacement latéral par rapport au centre, il est possible de déterminer la rotation que la voiture doit effectuer pour ne pas perdre la cible et la garder au centre de son champ de vision. D'autre part, l'éloignement de la cible permet de déterminer la vitesse du véhicule : si la cible est trop éloignée, la voiture doit augmenter sa vitesse pour s'approcher et ne pas la perdre de vue, et si la cible est trop proche, il doit ralentir pour ne pas la dépasser. Ce traitement des données est effectué à l'aide de calculs et de fonctions spécifiques de *Yolov8*, qui permettent d'obtenir les coordonnées des bords des boîtes de délimitation de la cible. Enfin, une série de conditions sont définies pour déterminer les commandes de rotation et de vitesse que le véhicule doit suivre, comme indiqué dans l'algorithme 1, de la ligne 12 à la ligne 25.

Algorithm 1: Algorithme de l'ordinateur

```
1  Require :
2      M    Poids de pré-entraînement YOLO
3      Cam Stream de la carte Raspberry
4  results ← M.predict(Cam, settings)
5  for r in results do
6      carres = r.boxes
7      for c in carres do
8          classif = int(box.cls[0])            ▷ Prendre la catégorie de chaque carré
9          if classif = 0 then
10             Calcule la largeur et longueur de l'objet en fonction des coordonnées de la
                boîte.
11             Calcule l'are et le centre en X en fonction des largeur et longueur de l'objet.
12             if aire trop petit then
13                 Vitesse : rapide
14             else if aire de bonne taille then
15                 Vitesse : normal
16             else
17                 Vitesse : Lent
18             end
19             if le centre est plus à droite then
20                 Rotation : Droit
21             else if le centre est plus à gauche then
22                 Rotation : Gauche
23             else
24                 Rotation : None
25             end
26             Envoie des commandes de vitesse et rotation à la carte Raspberry
27         end
28         Imprime l'image pour visualiser le processus
29     end
30 end
```

II.7 Envoi de commandes au Raspberry

Pour envoyer des commandes de l'ordinateur au raspberry, nous avons utilisé la bibliothèque Python *Socket*, qui permet de créer une connexion entre un connecteur «client» qui est le point final d'une conversation et un connecteur «serveur», qui est l'expéditeur du message pour la communication avec des connecteurs.

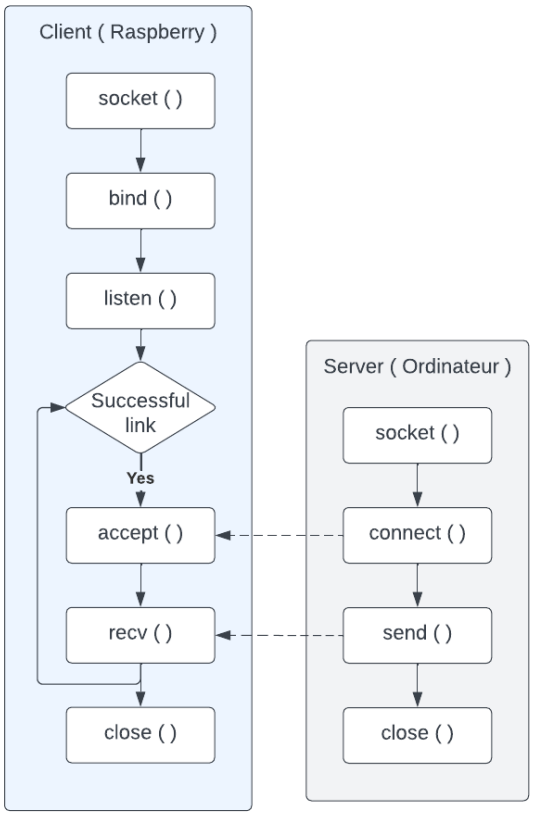


FIGURE 12 – Architecture socket de l'ordinateur - Carte Raspberry

Grâce à la documentation Python concernant cette bibliothèque [10, Section Socket], nous avons pu établir un canal de communication pour envoyer les instructions à exécuter par le véhicule. Les fonctions utilisées pour cette communication sont décrites dans l'organigramme de la figure 12, qui consiste à créer un connecteur (socket), établir la communication en spécifiant l'adresse IP et le port de connexion, envoyer ou recevoir le message et fermer le connecteur. Pour la fonction d'envoi, *socket.sendall( )* a été utilisé car il s'agit d'une méthode qui garantit que toutes les données mises en mémoire tampon sont envoyées sur le connecteur avant la fin de l'appel de la fonction.

Ce code a été fusionné avec le code principal pour effectuer la lecture du flux vidéo et la détection des objets en parallèle avec l'envoi de commandes concernant le changement de trajectoire de la voiture. Pour cette intégration, nous utilisons un fragment de serveur

de Streaming de la section "Envoi du flux vidéo à l'ordinateur" qui définit les classes, leurs méthodes et initialise l'enregistrement vidéo avec la caméra comme le montrent les lignes 7 à 12 de l'algorithme 2. En plus, un port parallèle est créé pour maintenir un canal de streaming et transmettre la vidéo en permanence en utilisant la bibliothèque Python *Threading*, qui permet de créer et de gérer des threads dans un programme où le thread est initialisé en appelant l'action à exécuter et en démarrant la parallélisation [10, Section Threading]. Cet ajout est illustré dans les lignes 1 - 9 de l'algorithme 2.

II.8 Envoi de commandes au Arduino

Enfin, lorsque les instructions sont reçues de la carte Raspberry, il faut envoyer ces instructions à l'arduino, qui contrôle et gère les connexions avec le driver pour actionner les moteurs. Pour cette tâche, nous avons implémenté la bibliothèque *Pyserial*, qui encapsule l'accès au port série [11].

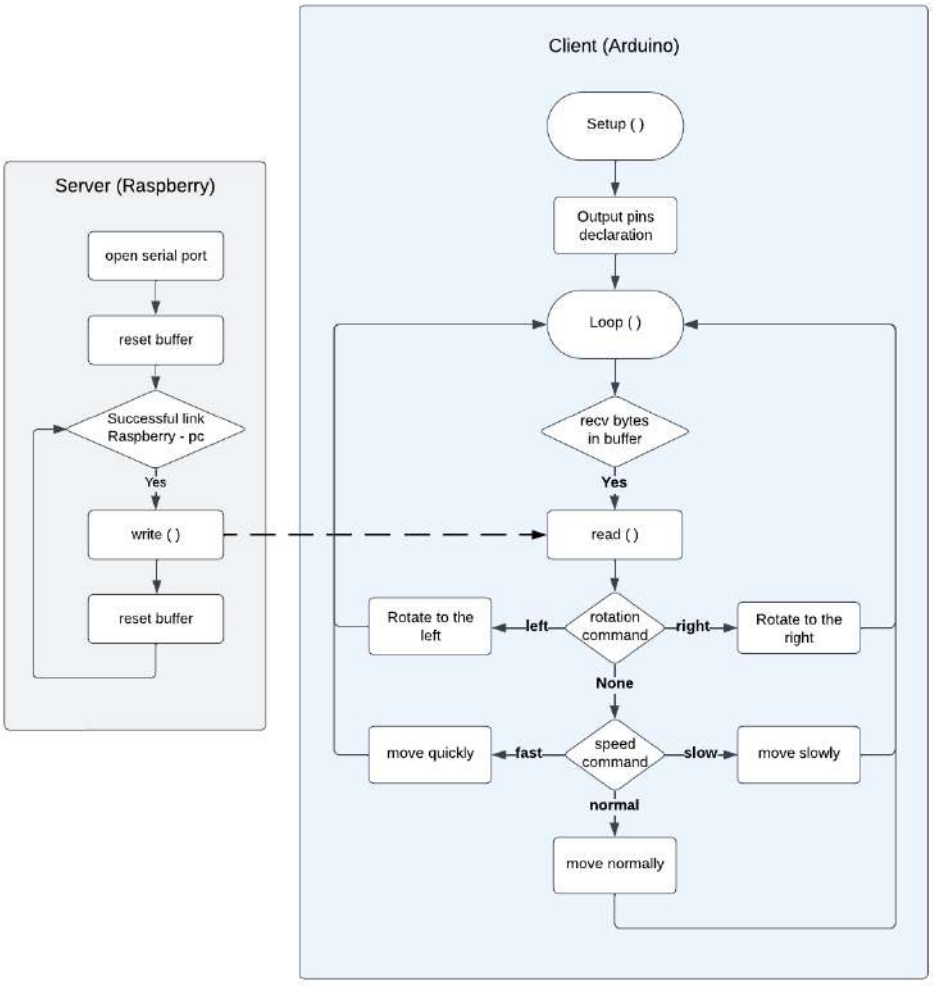


FIGURE 13 – Architecture en série des cartes Raspberry - Arduino

L'algorithme de la figure 13 décrit les instructions exécutées pour établir la communication entre la carte Raspberry et la carte Arduino, où le serveur initialise le port série, réinitialise la mémoire tampon de toutes les données qu'il a pu stocker, envoie la commande et réinitialise à nouveau la mémoire tampon pour répéter ce processus. Cette dernière étape est cruciale, car sans réinitialisation constante du tampon, toutes les données de communication sont stockées et, par la suite, l'envoi de commandes à l'arduino devient très lent, ce qui provoque des retards dans toutes les processus. D'autre part, le client,initialise les ports de sortie et, à l'intérieur de la boucle, reçoit les commandes de rotation et de vitesse pour effectuer différentes actions sur le moteur.

.

Algorithm 2: Algorithme de la carte Raspberry

```
1  Definition :
2      - Page HTML
3      - Serveur web
4      - Gestion des requêtes HTML entrantes sur le serveur
5  Configuration de la caméra
6  Démarrer l'enregistrement vidéo
7  try :
8      Création d'un Thread avec un port de communication
9      Démarrage de Thread
10     Création et initialisation de sockets. Figure 12
11     Création et initialisation de communication sérielle. Figure 13
12     while Communication réussie entre Raspberry et PC :
13         Recevoir des messages de l'ordinateur
14         Envoi de messages à la carte Arduino
15     end
16 end
17 except Interruption du clavier :
18     Fermer la communication série
19 finally :
20     Fermer la communication série
21     Arrêter l'enregistrement vidéo
```

III Détection de la voiture

Dans ce chapitre, nous utilisons le même mécanisme, la même architecture et le même code pour envoyer des données entre les différents dispositifs, à la différence que nous chargeons maintenant les poids du réseau neuronal entraîné pour la détection des voitures.

Le réseau neuronal implémenté fonctionne avec apprentissage supervisé qui consiste à apprendre à l'aide d'entrées pré-étiquetées, qui servent de cibles. L'objectif de cette forme d'apprentissage est de réduire l'erreur de classification globale des modèles en calculant correctement la valeur de sortie de l'exemple d'entraînement à mesure de l'entraînement. [1]

Pour entraîner un réseau neuronal, une base de données robuste d'images représentatives est nécessaire pour que le modèle prenne en compte les caractéristiques permettant de détecter l'objet. Si la base de données est trop petite, le réseau neuronal risque d'avoir du mal à généraliser l'identification de l'objet aux différentes situations et variations qui peuvent survenir dans les images, en plus, il peut s'adapter de façon excessive aux détails particuliers des images fournies, au lieu de capturer des modèles généraux et, lors de la détection des objets, il ne pourra identifier que les objets qui se trouvaient dans les mêmes conditions que celles de la base de données.

L'apprentissage par transfert a été effectué comme suggéré dans le guide Ultralytics [9, Modes/Train section] pour entraîner un réseau neuronal, en chargeant les poids définis par *yolov8* avec l'ensemble de données *COCO128* et en ajoutant les nouvelles images de véhicules pour modifier les poids spécifiquement pour la détection des voitures.

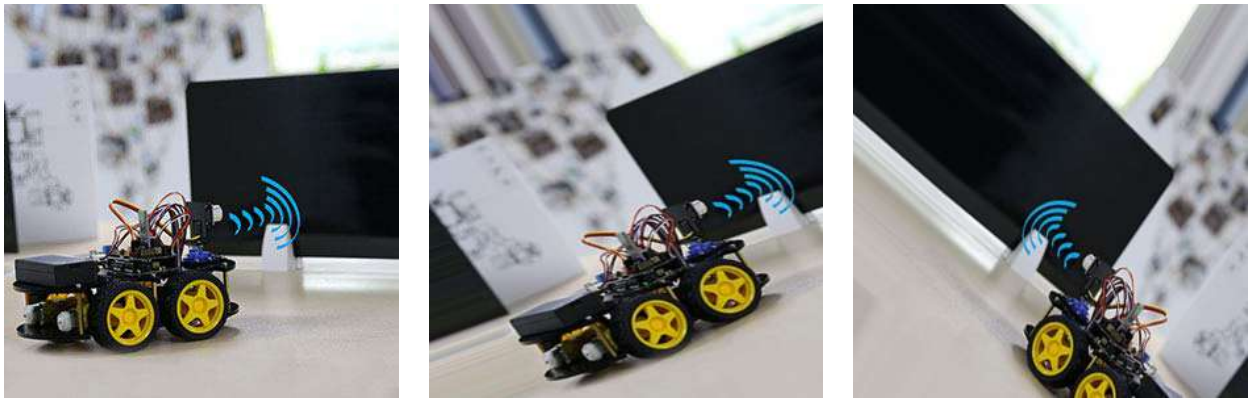
La présente section explique comment l'entraînement du réseau a été réalisé en suivant le guide de Piotr Skalski qui indique les étapes à suivre pour l'identification des objets avec *yolov8* en créant un jeu de données personnalisées [12] avec la plataforme *Roboflow*, qui fournit tout ce qui est nécessaire pour transformer les images en informations, et décrire l'intégration de ce modèle dans le code pour la détection des véhicules.

III.1 Création de la base de données

La création de l'ensemble de données se fait avec des images du véhicule ELEGOO original trouvées sur internet par les acheteurs de ce produit et des sources officielles, en plus des images de véhicules similaires sont prises pour avoir une base de données robuste et permettre le meilleur entraînement du réseau neuronal dans la détection des voitures. Finalement, 170 images de véhicules sont collectées pour entraîner le réseau (Nous appellerons ces images initiales l'ensemble de données original), mais cette quantité d'images est encore trop faible

pour faire un bon modèle, par conséquent, nous avons procédé à une augmentation des données en ajoutant 100 images avec les figures officielles du véhicule et des outils Python de traitement d'image, qui génèrent des variations d'échelle, de rotation, de recadrage, de retournement et d'agrandissement d'une même image.

L'ensemble de données est augmenté avec un algorithme basé sur l'environnement de prétraitement d'images *keras* qui est l'API de haut niveau de TensorFlow permettant de créer et d'entraîner des modèles de deep learning [13]. Où nous importons les fonctions nécessaires du module. Ensuite, nous définissons le nombre d'images destinées à augmenter l'ensemble de données. Une fois cette étape accomplie, un générateur d'images est mis en place, intégrant les modifications souhaitées. Finalement, les images sont générées et sauvegardées en utilisant la fonction *flow* propre au générateur d'images.



(a) (b) (c)

FIGURE 14 – Création de l'ensemble de données [Source image originale : Amazon]

Nous allons obtenir des variations de 10 images par figure avec différents angles par rapport au voiture, générant 100 images modifiées à partir de 10 images de base, comme illustré dans la figure 14 où l'image originale 14a génère 10 variations, parmi lesquelles 14b et 14c.

III.2 Étiquetage des images

La plateforme *Roboflow* est utilisée pour cette tâche, qui est un outil de développement permettant de construire des modèles de vision par ordinateur [14] où l'intégration de l'ensemble de données et la caractérisation des figures ont été effectuées pour identifier le véhicule dans les images d'entraînement et ensuite exporter le modèle.

Un premier étiquetage a été effectué avec les 270 images de l'augmentation créée avec la base de données originale mentionnée dans la section précédente. Cet étiquetage est d'abord réalisé en déposant les images dans la plateforme *Roboflow* et en délimitant la figure, c'est-à-dire en localisant l'objet souhaité dans l'image avec une boîte et une étiquette. Par la suite, la

plateforme dispose d'options permettant d'augmenter l'ensemble de données par des rotation horizontale, d'ajout de bruit et de quelques carrés noirs pour déformer légèrement la continuité de l'image, créant ainsi 375 nouvelles images d'entraînement afin que le réseau n'apprenne pas les images par cœur, mais qu'il parvienne à généraliser les caractéristiques en identifiant un modèle. Ce générateur de données segmente l'ensemble de données en 87 % d'images de formation, 8 % d'images de validation et 4 % d'images de test.

Enfin, cette base de données est prête à être utilisée par la bibliothèque Python *Roboflow* en exportant l'espace de travail et la version du modèle.

III.3 Entraînement réseau pour détecter le voiture

Le code Python utilisé pour entraîner le réseau avec la base de données personnalisée est un algorithme exécuté sur *Google Colab* libre d'accès et se trouve dans le "Repository Roboflow Notebooks" cité dans le guide de Skalski.

Dans le code, *Yolov8* est téléchargé avec la bibliothèque Ultralytics, et la base de données est importée en établissant une connexion via l'API Roboflow, en sélectionnant un projet et une version spécifique dans l'espace de travail et en téléchargeant l'ensemble de données du modèle.

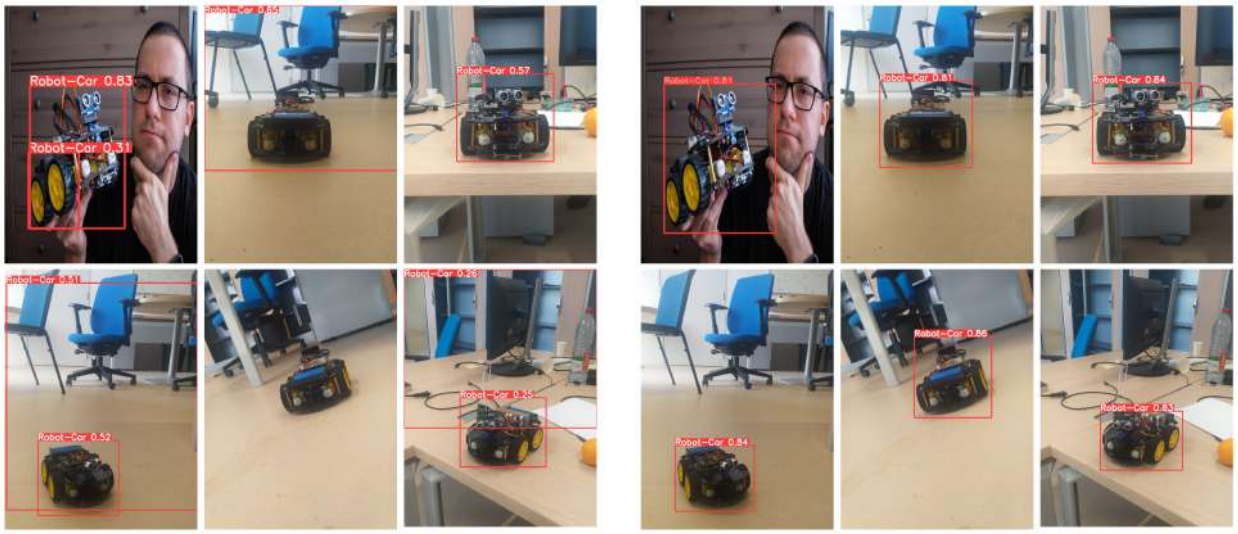
L'apprentissage du réseau neuronal commence en prenant comme base les poids pré-entraînés de *Yolov8*[1], la validation est effectuée sur 54 images, sur lesquelles le modèle évalue ses performances en prenant une image pour faire une prédiction de la cible et en comparant cette prédiction avec l'image étiquetée. Finalement,un test final est effectué pour vérifier le fonctionnement du modèle.

Enfin, il faut extraire le fichier .pt qui contient les valeurs numériques des poids entraînés du CNN qui compose le modèle et intégré dans le même dossier que le code de base afin de pouvoir appeler ces poids facilement dans le programme.

III.4 Analyse des résultats

Cette première version a donné le résultat illustré dans la figure 15a. Cette figure montre que le réseau neuronal ne différencie pas très bien l'environnement et le véhicule dans chaque image. En outre, il est difficile de détecter correctement les voitures et n'effectue pas l'identification dans les images où l'arrière du véhicule est situé. Enfin, le seuil de détection est trop bas pour l'identification des voitures.

1. Une époque est un passage complet de l'ensemble des données d'entraînement, où le modèle ajuste ses paramètres pour apprendre des erreurs passées, améliorant ainsi sa capacité à reconnaître les objets



(a) (b)

FIGURE 15 – Réseau neurone entraîné : Version 1 (a) et version 2 (b)

Pour cette raison, un nouveau réseau neuronal de 1200 images a été réalisée avec plus de références et en mettant en œuvre l'algorithme qui modifie légèrement les images pour obtenir une base de données robuste, En outre, les échantillons d'images prises en laboratoire dans les conditions d'éclairage dans lesquelles le véhicule fonctionnera ont été augmentés, de même que les échantillons de l'arrière de la voiture, puisque cet angle serait l'image la plus fréquemment capturée par un véhicule qui suit une voiture.

Cette deuxième version du réseau neuronal a été entraîné avec la même proportion d'images d'apprentissage, de validation et de test que la version précédente, en ajoutant 25 époques pour améliorer les performances et ajuster les poids, mais en veillant à ne pas attribuer trop d'époques, car cela pourrait entraîner un surajustement et une mémorisation des données. Ces configurations ont permis d'obtenir de meilleurs résultats dans l'identification d'une voiture, avec un meilleur seuil et une détection plus précise du véhicule dans l'image, comme le montre la figure 15b.

Planning du stage

Le projet s'est déroulé sur une période de 13 semaines, allant du 22/05/2023 au 18/08/2023. Le tableau ci-dessous présente l'organisation adoptée pendant le stage, mettant en évidence les principales étapes de la mise en œuvre du projet.

Semaine	**Task**
Semaine 1	Installation dans le laboratoire, tâches administratives, définition des objectifs du projet
Semaines 2 - 4	État de l'art, recherche de projets connexes antérieurs, étude des réseaux neuronaux convolutifs et réception du matériel
Semaines 3	Mise en œuvre de l'algorithme *Yolo* et détection des personnes
Semaine 4 - 5	Configuration de la carte Raspberry avec le système d'exploitation Debian 11 (Linux), configuration de la caméra et envoi du flux vidéo vers l'ordinateur via le protocole TCP/IP.
Semaine 6 - 8	Création du code principal sur l'oridnateur, traitement des images, définition des commandes de direction du véhicule et établissement d'une communication parallèle pour le retour des commandes.
Semaine 9 - 10	Panne d'appareils électroniques due à une décharge électrostatique (ESD) par contact avec une vis de la voiture, réception d'un nouveau matériel, protection des éléments électroniques, creation du code sur la carte Raspberry et établissement de communication avec la carte Arduino pour contrôler le mouvement du véhicule.
Semaine 10-11	Entraînement de réseaux neuronaux pour la détection de véhicules
Semaine 12 - 13	Rédiger le rapport, planifier la présentation, nettoyer et peaufiner le code, organiser le dépôt github.

Conclusion

L'objectif principal de ce projet est de suivre un véhicule "leader" à l'aide de l'algorithme de détection d'objets *Yolo*. Cette implémentation a été écrite en langage de programmation Python en utilisant 3 dispositifs principaux qui sont un ordinateur, une carte Raspberry et une carte Arduino.

Malgré le long processus d'étiquetage des images pour l'entraînement du réseau neuronal avec *Yolov8*, une bonne reconnaissance des véhicules a été obtenue avec une base de données robuste d'environ 1000 images. La prise d'images dans le scénario réel dans lequel le véhicule fonctionnera a été d'une grande aide pour l'apprentissage du réseau, car elle fournit le contexte et les conditions exacts dans lesquels les prédictions seront faites.

La communication entre les appareils a été réussie et la réception rapide du flux vidéo ainsi que le retour des commandes de direction adressées par IP ont permis un bon suivi du véhicule, cependant, il existe de nombreuses possibilités d'optimisation et d'amélioration des conditions de codage car dans certains scénarios, la voiture perd de vue le véhicule "leader" et recommence l'identification en tournant lentement à sa recherche, ce qui n'est pas la décision optimale pour un suivi constant du véhicule. Il est également possible d'améliorer la base de données, car le modèle prend parfois certains objets de l'environnement comme véhicule "leader"

Nous pouvons également conclure que la protection des dispositifs est indispensable au bon fonctionnement des mécanismes lorsqu'ils sont en contact direct les uns avec les autres.

Un autre domaine d'opportunités

Remplacement ou retrait de dispositifs

- Supprimer la carte Arduino et sa communication pour gérer directement le contrôle du moteur à partir de la carte Raspberry via ses ports d'entrée et de sortie.
- Mettre sur la voiture une carte GPU (telle que la Nvidia Jetson Nano) afin d'exécuter le réseau Yolo en remplaçant la carte Raspberry pour assurer la communication et l'ordinateur pour mettre en œuvre le programme. La voiture serait autonome par rapport au PC et il n'y aurait pas de problème de portée du réseau wifi.

Asservissement du vehicule

- Optimisation du code existant avec une architecture plus robuste dans les conditions de traitement.
- Intégrer un PID pour réguler et maintenir la direction du véhicule sur la cible.

Bibliografia

[1] Keiron O'Shea et Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv : 1511.08458 [cs.NE].

[2] Balena. *balenaEtcher*. Version 1.18.11. 2023. URL : https://etcher.balena.io/.

[3] Mario Santana. "Chapter 6 - Eliminating the Security Weakness of Linux and Unix Operating Systems". In : *Network and System Security (Second Edition)*. Syngress, 2014. URL : https://www.sciencedirect.com/science/article/pii/B978012416689900006X.

[4] Osamu Aoki. *Référence Debian*. Français. Version Version 2.102. Disponible à https://www.debian.org/doc/manuals/debian-reference/debian-reference.fr.pdf. 2023.

[5] Raspberry Pi Ltd. *Raspberry pi Documentation*. Consulté le 1 août 2023. URL : https://www.raspberrypi.com/documentation/computers/.

[6] David Plowman. "Bullseye camera system". In : *The Guardian* (17 nov. 2021). URL : https://www.raspberrypi.com/news/bullseye-camera-system/ (visité le 10/08/2023).

[7] Vijay K. Garg et Yih-Chen Wang. "6 - Communication Network Architecture". In : *The Electrical Engineering Handbook*. Sous la dir. de WAI-KAI CHEN. Academic Press, 2005. URL : https://www.sciencedirect.com/science/article/pii/B9780121709600500748.

[8] Henrik Nielsen et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC Editor, juin 1999.

[9] Glenn Jocher, Ayush Chaurasia et Jing Qiu. *YoloV8 by Ultralytics*. Version 8.0.0. Jan. 2023. URL : https://docs.ultralytics.com/.

[10] Python Software Foundation. *The Python Standard Library*. Consulté le 1 août 2023. URL : https://docs.python.org/3/library.

[11] Chris Liechti. *pySerial*. Consulté le 1 août 2023. URL : https://pythonhosted.org/pyserial/.

[12] Piotr Skalski. *How to Train YOLOv8 Object Detection on a Custom Dataset*. Consulté le 1 août 2023. 2023. URL : https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/.

[13] François Chollet et al. *Keras*. https://keras.io. 2015.

[14] Inc Roboflow. *Roboflow docs*. Consulté le 1 août 2023. URL : https://docs.roboflow.com/.

Glossaire

— **ANN :** Artificial Neural Network
— **Carte SD :** Carte Secure Digital
— **CLI :** Command Line Interface
— **CNN :** Convolutional Neural Network
— **CSI :** Camera Serial Interface
— **ESD :** Electrostatic discharge
— **GPU :** Graphics Processing Unit
— **HTTP :** Hypertext Transfer Protocol
— **IP :** Internet Protocol
— **OS :** Operating system
— **TCP :** Transmission Control Protocol
— **USB :** Universal Serial Bus
— **YOLO :** You Only Look Once

A Annexe

A.1 Spécifications de l'ordinateur utilisé

```
Architecture:                    x86_64
Mode(s) opératoire(s) des processeurs: 32-bit, 64-bit
Boutisme:                        Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
Processeur(s):                   4
Liste de processeur(s) en ligne: 0-3
Thread(s) par cœur:              1
Cœur(s) par socket:              4
Socket(s):                       1
Nœud(s) NUMA:                    1
Identifiant constructeur:        GenuineIntel
Famille de processeur:           6
Modèle:                          45
Nom de modèle: Intel(R) Xeon(R) CPU E5-1603 0 @ 2.80GHz
Révision:                        7
Vitesse du processeur en MHz:    2793.051
Vitesse maximale du processeur en MHz: 2800,0000
Vitesse minimale du processeur en MHz: 1200,0000
BogoMIPS:                        5586.10
Virtualisation:                  VT-x
Cache L1d:                       128 KiB
Cache L1i:                       128 KiB
Cache L2:                        1 MiB
Cache L3:                        10 MiB
Nœud NUMA0 de processeur(s):     0-3
```

A.2 Fichier *config.txt*

Configuration du fichier hébergé dans le chemin */boot/config.txt* avec l'instruction spécifique : *dtoverlay=imx219* pour une caméra raspberry pi V2.1.

```
dtparam=audio=on
start_x=1
display_auto_detect=1

dtoverlay=vc4-kms-v3d
dtoverlay=imx219
max_framebuffers=2
disable_overscan=1

[cm4]
otg_mode=1

[all]

[pi4]
dtoverlay=vc4-fkms-v3d
arm_boost=1

[all]
gpu_mem=128
```

NOTE : Après avoir apporté des modifications à ce fichier, il est nécessaire de redémarrer le Raspberry.