

---

# Travail de Bachelor

## Guidage d'une voiture RC par vision

### RÉMY MACHEREL

#### 25 novembre 2021

---

Etudiant :

Rémy Macherel

## Table des matières

<b>1 Glossaire général</b>	<b>3</b>
<b>2 Introduction</b>	<b>4</b>
2.1 Contexte . . . . .	4
2.2 Structure du document . . . . .	4
2.3 Description du concept de la voiture "intelligente" . . . . .	4
<b>3 Objectifs du travail de Bachelor</b>	<b>5</b>
<b>4 Matériel Initial</b>	<b>6</b>
4.1 Voiture . . . . .	6
4.2 Ordinateur de bord . . . . .	7
4.3 Refroidisseur . . . . .	8
4.4 Caméra . . . . .	9
4.5 Système d'alimentation . . . . .	9
4.6 Communication . . . . .	9
4.7 Route . . . . .	10
<b>5 Modifications apportées au matériel initial</b>	<b>11</b>
5.1 Changement de batterie . . . . .	11
5.2 Modification d'implémentation du PiJuice . . . . .	12
5.2.1 Vérification de température . . . . .	13
5.2.2 Support de PiJuice HAT . . . . .	13
5.3 Changement de couleur des lignes . . . . .	14
<b>6 Analyse de systèmes de suivi de route</b>	<b>15</b>
6.1 Implémentation initiale . . . . .	15
6.2 Autres solutions . . . . .	15
6.2.1 Traitement sur GPU . . . . .	15
6.2.2 Refonte complète et utilisation du CNN pour le suivi . . . . .	15
<b>7 Solution de suivi choisie</b>	<b>16</b>
7.1 Deep Learning . . . . .	16
7.2 Réseau Neuronal Convolutif (CNN) . . . . .	18
7.2.1 Convolution . . . . .	18
7.2.2 Fonction d'activation . . . . .	21
7.2.2.1 Problème de dying ReLu [8] . . . . .	22
7.3 Modèle Nvidia . . . . .	23
<b>8 Langage de programmation</b>	<b>24</b>
<b>9 Création d'une base de données</b>	<b>25</b>
9.1 Pilotage Manuel . . . . .	25
9.2 Enregistrement d'images et de valeurs de direction . . . . .	26
9.3 Augmentation du dataset . . . . .	28
9.3.1 Floutage . . . . .	28

---

9.3.2 Ajustement de luminosité . . . . .	29
9.3.3 Inversion horizontale aléatoire . . . . .	30
9.3.4 Fonction regroupant toutes les modifications . . . . .	30
<b>10 Entrainement du CNN</b>	<b>31</b>
10.1 Pré-traitement de l'image . . . . .	31
10.1.1 Espace de couleurs YUV . . . . .	31
10.2 Google Colaboratory . . . . .	32
10.3 Keras Sequential Model . . . . .	33
10.4 Optimiseur de modèle . . . . .	34
10.5 Fonction de perte . . . . .	34
10.6 Séparation du dataset . . . . .	34
10.7 Phase d'entraînement . . . . .	35
10.8 Résultats d'entraînement . . . . .	35
10.9 Conversion du modèle . . . . .	36
<b>11 Architecture logicielle</b>	<b>37</b>
11.1 Classe Car . . . . .	37
11.2 Classe DataCollector . . . . .	38
11.3 Application NavigatorApp . . . . .	39
11.3.1 Fonctionnement de l'application . . . . .	39
11.4 Fichier de configuration . . . . .	41
<b>12 Algorithme de suivi de route</b>	<b>42</b>
12.1 Capture et pré-traitement de l'image . . . . .	42
12.2 Calcul de l'angle de direction . . . . .	42
12.3 Calcul de la vitesse . . . . .	43
12.4 Fin de route . . . . .	44
12.5 Tests sur images fixes . . . . .	45
12.6 Implémentation finale . . . . .	45
<b>13 Conclusion</b>	<b>46</b>
13.1 Problèmes rencontrés . . . . .	46
13.1.1 Batterie et alimentation du RPi . . . . .	46
13.1.2 PWM . . . . .	46
13.1.3 Récupération valeur de steering . . . . .	46
13.1.4 Fluidité du pilotage Manuel . . . . .	47
13.1.5 Quantité de données d'entraînement . . . . .	47
13.2 Possibilités d'amélioration . . . . .	48
13.2.1 Acquisition d'images à l'aide d'un simulateur ou autre système de suivi de route . . . . .	48
13.2.2 Utilisation d'un autre SBC . . . . .	48
<b>Bibliographie</b>	<b>52</b>
<b>Annexes</b>	<b>53</b>

## 1. Glossaire général

<b>TB :</b>	Travail de Bachelor
<b>RPi :</b>	RaspBerry Pi
<b>RC :</b>	Radio-Commandée (pour la voiture)
<b>PWM :</b>	Pulse Width Modulation
<b>CPU :</b>	Central Processing Unit
<b>CNN :</b> (neurones convolutif)	Convolutionnal Neural Network (Réseau de neurones convolutif)
<b>SBC :</b>	Single Board Computer (Ordinateur de bord)
<b>OS :</b>	Operating System (système d'exploitation)
<b>Dataset :</b>	Set de données

## 2. Introduction

### 2.1 Contexte

Ce TB est une suite d'un précédent travail effectué une année auparavant par un autre étudiant de la HEIG-VD. L'objectif de mon travail est d'améliorer le système de guidage de la voiture RC. D'un commun accord avec le professeur responsable, il a été décidé que le matériel de la voiture ne serait pas modifié dans l'ensemble.

### 2.2 Structure du document

Dans un premier temps, les objectifs de ce travail seront décrits. Ensuite, une description du matériel réceptionné et de ses caractéristiques sera faite. Une analyse du précédent système sera menée, d'autres hypothèses seront explorées et le choix de la solution implémentée découlera de cette étude.

Viendront ensuite l'architecture logicielle ainsi que l'implémentation de la solution choisie pour le suivi de route.

Pour conclure, les problèmes rencontrés seront listés et de potentielles améliorations seront évoquées.

### 2.3 Description du concept de la voiture "intelligente"

Ce qu'on appelle de nos jours une voiture intelligente est un véhicule dont certaines (ou toutes) actions sont réalisées de manière autonome par un ou plusieurs ordinateurs de bord. L'ordinateur de bord a donc pour effet de partiellement ou totalement remplacer les actions de l'être humain. Cette "intelligence" artificielle est déjà présente dans les voitures actuelles avec par exemple le régulateur de vitesse (aussi appelé tempomat) permettant de maintenir une distance toujours égale avec la voiture de devant. C'est dans ce cas une seule des actions de la voiture qui est réalisée par l'ordinateur de bord. Le niveau maximal d'autonomie pour la voiture serait une conduite totale sans aucune participation du conducteur.

La voiture dite autonome se doit de maîtriser toutes sortes d'environnements et de situations de conduite, de la détection de panneaux à la navigation en passant par différents types de routes ou de conditions (neige, pluie, etc.). Ce sont donc tous ces facteurs qui rendent ce sujet extrêmement complexe et vaste. C'est pourquoi dans le cadre de ce travail, nous nous limiterons à rendre autonome le suivi de route sur une route préfabriquée dans des conditions variant uniquement au niveau de la luminosité.

### 3. Objectifs du travail de Bachelor

Les objectifs fixés en accord avec le professeur responsable sont les suivants :

1. Analyse de solutions de guidage.
2. Choix d'une solution et implémentation de celle-ci.
3. Amélioration du système de guidage en termes de rapidité et de flexibilité.

## 4. Matériel Initial

Le contenu de ce chapitre est entièrement tiré du rapport de travail de Bachelor de M. Maxime Charrière.[1]

Les points les plus importants seront donc rapportés dans cette section afin de décrire précisément le matériel initial.

### 4.1 Voiture

La voiture choisie est donc une voiture RC de type : *T2M Mad Pirate Brushless 1/10e*. Ce choix a été effectué par mon prédecesseur après comparaison avec d'autres types de véhicules. Les avantages de ce modèle sont :

- Prix abordable (~300CHF)
- Conception simple à utiliser (Par exemple pour le guidage ou la vitesse, commandés depuis le RPi)
- Batterie présente sur le modèle et donc pas besoin d'alimenter la voiture en plus du RPi.
- Volume disponible pour placer la batterie, le SBC et effectuer le câblage

L'avantage de la voiture RC citée précédemment réside dans la simplicité de son guidage. En effet, l'accélération ainsi que la direction se font grâce à un signal PWM envoyé sur les différents moteurs. Mon collègue ayant effectué dans son travail des mesures permettant de déterminer les valeurs de "Duty Cycle" à appliquer aux moteurs pour leurs différentes fonctions, il en ressort les caractéristiques suivantes :

	Freq	Duty cycle (% de temps de la période que la sortie est à 1)		
		Low	Middle	High
Steering	50Hz	6% (Left) (1.2ms)	7.5% (1.5ms)	9% (Right) (1.8ms)
Speed	50Hz	6% (Back) (1.2ms)	7.5% (1.6ms)	10% (Forward) (2ms)

FIGURE 4.1 – Mesures PWM des actuateurs

La contrôleur moteur de la voiture possède également un système d'arrêt d'urgence fondé sur le principe suivant :

Si l'on applique un duty cycle correspondant à la position *arrêt* de la voiture (1.6 ms), le moteur se retrouve simplement sans alimentation mais l'énergie cinétique acquise par la voiture lors de son déplacement prolongera celui-ci.

Le contrôleur est donc équipé d'un système de freinage qui s'active lorsque l'on applique un duty cycle plus faible que la position neutre (<1.6ms) alors que la voiture était en mouvement d'avance (duty cycle plus élevé que 1.6ms). Ce système permet donc d'arrêter la voiture instantanément sans subir le mouvement dû à l'énergie cinétique.

## 4.2 Ordinateur de bord

L'ordinateur de bord présent sur la voiture est un Raspberry Pi 4B 4GB. Ce modèle dispose de sorties configurables en PWM (pour le contrôle des moteurs de la voiture), d'une communication bluetooth et wifi, de connexions pour une caméra ainsi que de 4 coeurs (utiles pour le multithreading). On peut voir sur la figure suivante un tableau comparatif des différents SBC étudiés pour ce projet :

Table comparative		
	Positif	Négatif
<i>Raspberry Pi 4B 4Gb</i>	 <ul style="list-style-type: none"> <li>63fr</li> <li>2 PWM</li> <li>WiFi / BT</li> <li>Camera</li> <li>Forte communauté</li> </ul>	<ul style="list-style-type: none"> <li>Pas de GPU / NPU pour du ML</li> <li>Pas de refroidisseur</li> </ul>
<i>Arduino Uno</i>	 <ul style="list-style-type: none"> <li>38fr</li> <li>15 PWM</li> <li>Forte communauté</li> <li>Faible consommation</li> </ul>	<ul style="list-style-type: none"> <li>Pas un processeur, mais un microcontrôleur</li> <li>Pas de WiFi / BT</li> </ul>
<i>NVIDIA Jetson Nano</i>	 <ul style="list-style-type: none"> <li>Comporte une GPU</li> <li>Camera</li> </ul>	<ul style="list-style-type: none"> <li>159fr</li> <li>1 PWM</li> <li>WiFi / BT en option</li> </ul>
<i>Google Coral Dev Board</i>	 <ul style="list-style-type: none"> <li>Comporte une NPU</li> <li>3 PWM</li> <li>WiFi / BT</li> <li>Camera</li> </ul>	<ul style="list-style-type: none"> <li>153fr</li> </ul>
<i>Rock Pi N10</i>	 <ul style="list-style-type: none"> <li>Comporte une NPU</li> <li>2 PWM</li> <li>Camera</li> </ul>	<ul style="list-style-type: none"> <li>148fr</li> <li>Pas de WiFi / BT</li> </ul>

FIGURE 4.2 – Comparaison des différents SBCs

Initialement le Raspberry Pi n'est pas conçu pour utiliser des réseaux de neurones. Afin de remédier à ce problème, la "**Google Coral Accelerator**", une NPU (en anglais *Neural Processing Unit*) peut être utilisée. Cet accélérateur comporte une puce "*Edge TPU*" qui est identique à celle présente sur le SBC "*Google Coral DevBoard*". Il est bon de savoir que le terme TPU est uniquement le nom donné par Google à leurs puces NPU.

Le choix de la combinaison "*Raspberry Pi 4B*" + "*Google Coral Accelerator*" plutôt que le "*Google Coral Dev Board*" a été fait grâce à la communauté très présente du RPi. En effet, grand nombre de tutoriels sont faits sur RPi, ainsi que la majorité des problèmes que l'on peut rencontrer sur RPi sont souvent étudiés sur des forums et des correctifs sont disponibles.



FIGURE 4.3 – Raspberry Pi 4B et Google Coral Accelerator

### 4.3 Refroidisseur

Le RPi ne comporte pas de système de refroidissement, de ce fait, un système de refroidissement actif, le *Joy-It Block Active* a été ajouté.



FIGURE 4.4 – Raspberry Pi avec refroidisseur actif type Joy-It Block Active

## 4.4 Caméra

Le module de caméra utilisé est une PiCamera, choisie pour sa compatibilité directe avec le Raspberry ainsi que la bibliothèque développée par Raspberry pour son utilisation.

## 4.5 Système d'alimentation

Il a été décidé de laisser la batterie initiale de la voiture (Batterie Ni-Mh 1800mAh) pour ce qui est du contrôle et de l'alimentation des servomoteurs de direction. Pour ce qui est de l'alimentation nécessaire au RPi, un module **PiJuice HAT** [2] a été plébiscité pour plusieurs raisons :

- RPi indépendant du reste de la voiture
- Utilisation du RPi même pendant la période de charge de la batterie véhicule (qui nécessite d'être débranchée afin d'être rechargée)
- Supervision de l'état de la batterie depuis le RPi
- Fonction d'UPS afin d'arrêter le RPi de manière contrôlée en présence d'une sous-alimentation.



FIGURE 4.5 – PiJuice HAT

## 4.6 Communication

Pour la réalisation de ce travail, la communication avec le RPi s'effectue par protocole TCP/IP. En effet, la voiture se connecte automatiquement sur le wifi correspondant au partage de connexion de l'ordinateur. L'avantage de ce mode de communication est qu'il peut être mis en place dans tout endroit ayant un accès wifi et permet grâce aux outils suivants de gérer les différents aspects et fonctionnalités du RPi :

- Shell à distance avec le protocole SSH
- Bureau à distance grâce au logiciel VNC
- Code à distance par Visual Studio Code

## 4.7 Route

La route a été confectionnée en se basant sur une mise à l'échelle de routes réelles avec les valeurs suivantes :

- Largeur des routes nationales françaises : 3.5 m
- Largeur d'une voiture standard : 1.8 m
- Largeur d'une voiture RC : 30 cm

Donc la largeur de la route est de :

$$L_{RouteRC} = \frac{L_{routeFr}}{L_{voiture}} * L_{voitureRC} = 58cm \quad (4.1)$$

La route est confectionnée sur la base de cartons de 1.5m de long, assemblés en les scotchant ensemble afin de rendre celle-ci plus modulable. La couleur initiale des lignes était le jaune, ce qui sera modifié par la suite (voir section 5.3).



FIGURE 4.6 – Dimensions de la route et couleur initiale. Source : [1]

## 5. Modifications apportées au matériel initial

### 5.1 Changement de batterie

La batterie initiale liée au module **PiJuice HAT** était une batterie de 1820mAh. Le RPi pouvant consommer jusqu'à 2.5 A au maximum mais en moyenne 1.2 A celle-ci ne permettait au RPi que de fonctionner durant environ 1.5h **maximum**.

Il a donc été choisi de remplacer cette batterie par une autre d'une capacité de 12000 mAh. Ce qui offrirait alors une autonomie d'environ 10h !

Calcul de la durée de vie de batterie :

$$\text{Durée} = \frac{\text{Capacité}[mAh]}{\text{Consommation}[mAh]} \quad (5.1)$$

L'unique conséquence de ce changement de batterie est que la batterie de 12000mAh étant nettement plus grande que celle de 1800 mAh, on ne peut plus désormais l'enficher directement la batterie dans le PiJuice HAT (comme à la figure 4.5). Il a donc fallu pallier à ce problème en plaçant la nouvelle batterie en dessous de la voiture et relier celle-ci au module PiJuice via des câbles.

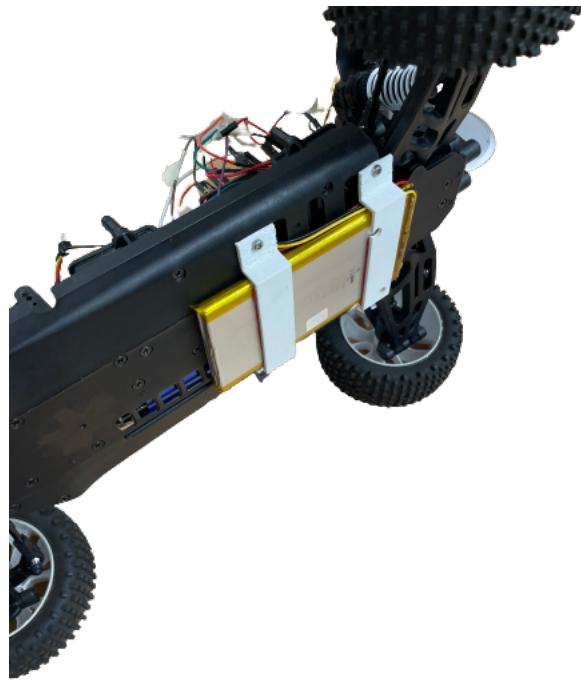


FIGURE 5.1 – Modification du montage pour la nouvelle batterie

## 5.2 Modification d'implémentation du PiJuice

Dans le prototype reçu le module PiJuice se trouvait séparé du RPi, monté sur un support et lié au RPi par des câbles dits "Jumper". Cette implémentation permettait un montage soigné et offrait la possibilité de refermer la carrosserie de la voiture. Cependant, le module PiJuice établissant une connexion via bus I2C avec le RPi et fournissant un courant jusqu'à 2.5 A (consommation maximale du RPi), la connexion par fils Jumper causait passablement de problèmes.

Premièrement, les perturbations présentes naturellement ainsi que celles générées par les servomoteurs de direction mais également les défauts présents dans les câbles (mauvaises connections par exemple) provoquaient des interruptions sur le bus et le RPi assimilait cela à une perte de connexion avec le module et donc potentiellement de son alimentation. A la suite de cela, il avait tendance à rebooter en s'éteignant.

Finalement, la conduction d'un courant pouvant aller jusqu'à 2.5 A dans de tels fils généreraient une forte chute de tension et le Raspberry indiquait par une alerte en permanence une tension d'alimentation trop faible. A cause de cela, lors de l'exécution d'un script (dans mon cas un script de multithreading demandant l'utilisation d'une majorité des ressources du RPi), le RPi avait le même comportement que cité précédemment, c'est-à-dire qu'il s'éteignait car la tension était trop faible pour son bon fonctionnement.

Afin de remédier aux problèmes cités ci-dessus, j'ai placé le module PiJuice en connexion directe sur le connecteur 40 pins du Raspberry. En agissant ainsi, on évitait les pertes de communication ou les chutes de tension dues aux fils.

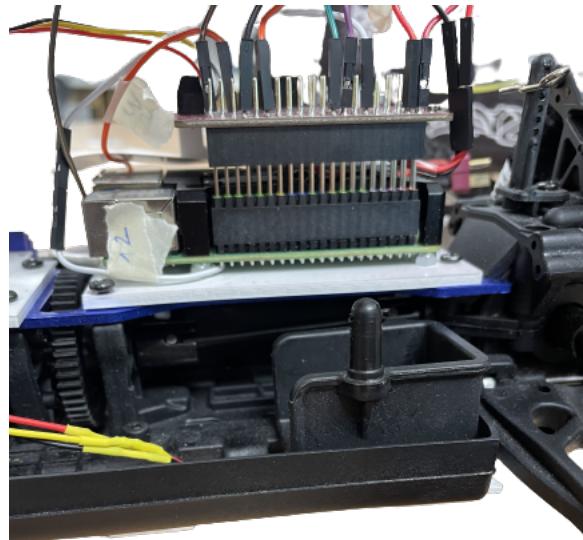


FIGURE 5.2 – Implantation du PiJuice HAT directement sur le RPi

Sur cette figure, on constate que le PiJuice HAT est dorénavant directement monté sur le connecteur 40 pins du RPi. A noter qu'une rallonge de connecteur a dû être rajoutée car, de par la présence du refroidisseur, il était impossible de le connecter directement.

### 5.2.1 Vérification de température

Le module PiJuice étant maintenant positionné au dessus des sorties de ventilation du RPi, des mesures ont été effectuées à l'aide d'un thermomètre infrarouge afin de vérifier que le RPi était toujours correctement refroidi mais également que le module PiJuice ne chauffe pas trop.

Durée fonctionnement [minutes]	Température mesurée [°C]
10	26
20	30
30	37
40	38
50	38

TABLE 5.1 – Table des mesures des températures de fonctionnement sur le PiJuice HAT

On observe donc que la température ne dépasse pas les 40°C même après quasiment une heure de fonctionnement. De plus, les composants du module PiJuice sont fabriqués pour fonctionner jusqu'à une température maximale de 80°C, les températures mesurées sont donc largement en dessous du maximum.

### 5.2.2 Support de PiJuice HAT

Le module étant connecté directement sur le RPi par le biais d'une rallonge, il ne disposait donc d'un seul support de fixation (les pins) et ceci le rendait parfois instable et sa fixation n'était pas optimale. Il a donc fallu créer un support pour le PiJuice Hat venant s'enficher dans une des rainures du refroidisseur et disposant de deux perçages filetés pour permettre la fixation du module. Ce support a été désigné puis imprimé en 3D. Le module étant enfiché dans le refroidisseur une ouverture a également été placée afin de permettre à l'air évacué par les ventilateurs de continuer à s'échapper et ainsi ne pas chauffer le module. Le dessin 3D du support est disponible en annexe mais en voici un aperçu :

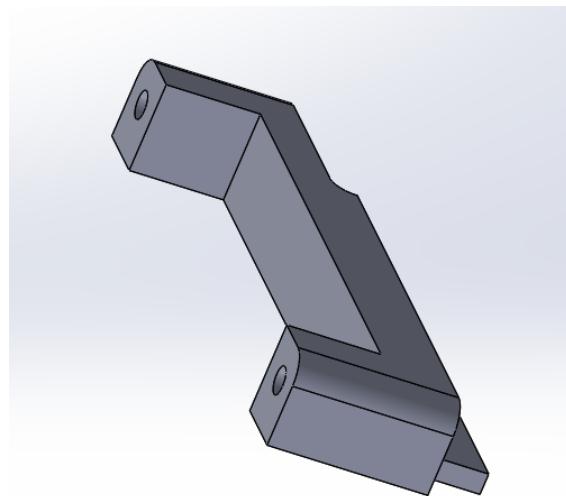


FIGURE 5.3 – Aperçu du support confectionné

### 5.3 Changement de couleur des lignes

Comme cité dans la section 4.7, il a été choisi de modifier la couleur des lignes. En effet, le pourquoi de ce choix sera expliqué plus tard dans le preprocessing mais Nvidia conseillait d'utiliser l'espace de couleurs YUV pour leur modèle. En effet, cet espace de couleurs utilisant principalement les informations de luminance, cela permettait un meilleur contraste des lignes. Cependant avec les routes initiales, la ligne étant jaune sur un carton brun, la différence de luminance n'était pas flagrante. C'est pourquoi les lignes ont été réalisées dans ce modèle, à l'aide d'un ruban adhésif mat de couleur noire (sans reflets) afin que lors de la conversion en YUV, les lignes soient clairement visibles sur l'image.

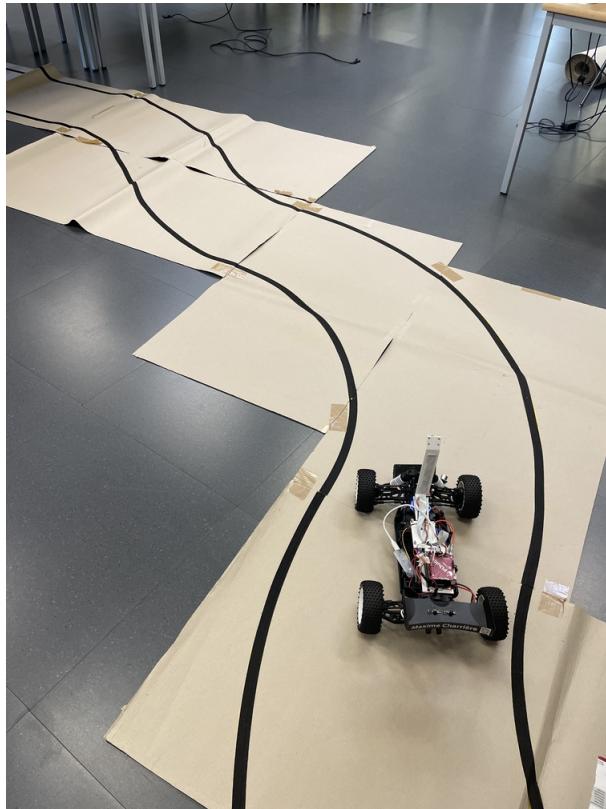


FIGURE 5.4 – Route avec couleur de lignes modifiée

## 6. Analyse de systèmes de suivi de route

### 6.1 Implémentation initiale

La première partie du travail consistait à réaliser une analyse critique du système précédemment implémenté ainsi qu'une recherche de solutions pour améliorer le suivi de route.

Le premier point repéré dans la solution de M. Maxime Charrière est que la vitesse de la voiture est relativement lente. Ce phénomène est dû au temps de processing de l'image. En effet, l'utilisation de *Numpy* et *OpenCV* (deux bibliothèques Python permettant le traitement de données ainsi que le traitement d'image) est performante. Toutefois, elle demande beaucoup de ressources au niveau du processeur car leurs différentes fonctions nécessitent une grande quantité de calculs matriciels et d'application de filtres etc. Le code précédent présentait donc des fonctionnalités qui demandaient un temps jugé trop élevé, telles que les fonctions de "Calibration", "warping" (passage en vue dite *d'oiseau*), ainsi que le polyfit (fonction permettant de déterminer une fonction polynomiale représentant au mieux une courbe donnée). Les valeurs de temps d'exécution de ces différentes fonctions sont disponibles au point 5.4.1 du rapport de travail de M. Charrière [1].

La résolution de l'image utilisée précédemment peut elle aussi influencer la durée d'exécution du thread (et plus particulièrement des fonctions citées ci-dessus). En effet, la résolution utilisée était de 640x480 Pixels.

### 6.2 Autres solutions

Plusieurs solutions étaient donc envisageables afin d'améliorer le suivi de route ainsi que la vitesse d'exécution.

#### 6.2.1 Traitement sur GPU

Etant donné que le traitement d'image était composé d'une multitude d'actions traitées successivement alors qu'elles pourraient l'être en parallèle, il est imaginable de gagner en performances en utilisant une GPU (le RPi 4B disposant d'une puce GPU).

#### 6.2.2 Refonte complète et utilisation du CNN pour le suivi

L'une des possibilités consiste en l'utilisation d'un CNN pour le suivi de route. Cette solution permettrait d'entrainer un réseau et de limiter grandement les traitements à appliquer à l'image. De plus, avec l'aide du *Google Coral Accelerator*, l'exécution du modèle entrainé de CNN est extrêmement rapide.

## 7. Solution de suivi choisie

La solution qui a été choisie est donc celle de l'utilisation d'un CNN pour le suivi de route. En effet, en se basant sur le matériel initial, et en le comparant aux connaissances actuelles et autres options, le choix d'un réseau de neurones et de l'accélérateur matériel se sont imposés car ils permettaient de diminuer drastiquement le temps de traitement. En effet, l'entraînement du CNN prend passablement de temps (voir chapitres 9 et 10), mais une fois celui-ci entraîné, son exécution est très rapide et le Preprocessing à effectuer sur l'image capturée par la caméra devient moindre.

### 7.1 Deep Learning

Cette solution implique donc l'utilisation d'un domaine appelé le *Deep Learning (Apprentissage profond)* [3]. Le deep learning est une branche du domaine de l'intelligence artificielle qui se base sur le fonctionnement du cerveau humain. Le deep learning utilise ce que l'on appelle des réseaux de neurones artificiels qui sont élaborés et interreliés en s'inspirant des observations actuelles du fonctionnement des réseaux de neurones du cerveau humain.

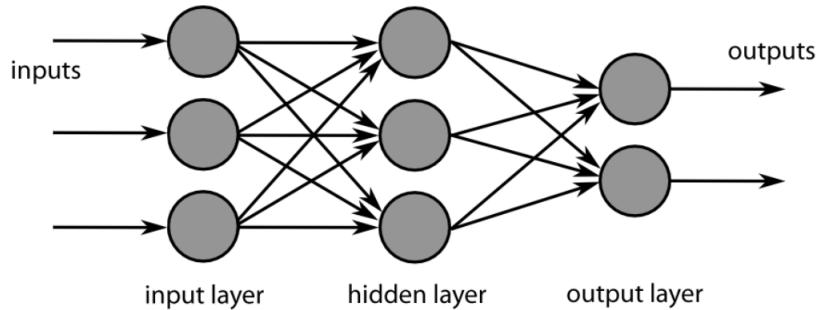


FIGURE 7.1 – Schéma représentatif d'un réseau de neurones

Comme on peut le voir, un réseau de neurones est composé de trois éléments :

- Une couche d'entrée (Input Layer)
- Une couche cachée (Hidden Layer)
- Une couche de sortie (Output Layer)

Le fonctionnement de base d'un réseau de neurones est le suivant :

En partant d'une base de données contenant différentes entrées avec la sortie leur correspondant, on va chercher à apprendre les paramètres (aussi appelés poids) de chaque couche qui permettent, en les multipliant par leurs entrées respectives, de déterminer une sortie prédictive le plus proche possible de la sortie souhaitée. Il est important de préciser qu'à la base les valeurs de paramètres ne sont pas connues. Elles sont donc générées aléatoirement et c'est l'entraînement du réseau à l'aide de la base de donnée préparée qui permettra d'optimiser ces poids afin de rendre le réseau capable de déterminer une sortie correcte en fonction de son entrée.

Plus le nombre de neurones est grand, plus le réseau sera qualifié de "profond", d'où le terme "deep Learning".

Un réseau de neurones peut convenir à plusieurs types d'applications dont les plus connues sont :

1. La classification
2. La régression

La classification, comme son nom l'indique, consiste à déterminer dans quelle classe (liste de classes définies au préalable) se situe l'information fournie en entrée. Prenons par exemple une classification d'espèces animales qui permettrait de définir sur la base d'une image si l'animal présent dans celle-ci est un mammifère, un reptile, etc.

La régression quant à elle consiste à déterminer une fonction qui représentera au mieux celle décrite par les données d'entraînement.

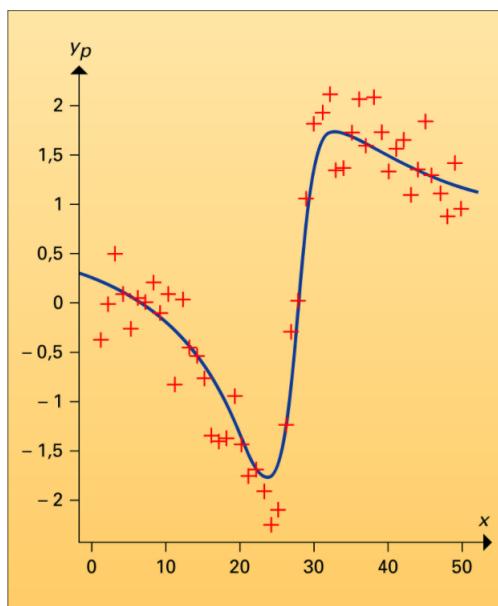


FIGURE 7.2 – Exemple de régression à une variable [4]

La figure 7.2 ci-dessus démontre bien le set d'entraînement par les croix rouges disposées dans le graphique ainsi que la courbe bleue qui représente donc l'approximation de fonction permettant de représenter au mieux et d'approcher les résultats du set d'entraînement.

Dans le cas du pilotage de la voiture, nous aurons donc affaire à un problème de **régression**. En effet, les valeurs d'inclinaison des roues étant des valeurs numériques, il s'agira de déterminer pour chaque image (capture de la route) l'inclinaison correspondante.

## 7.2 Réseau Neuronal Convolutif (CNN)

Le réseau de neurones qui sera utilisé dans notre application sera plus précisément un *Réseau neuronal convolutif*, dont l'abréviation en *CNN*. Les CNN sont une sous-catégorie de réseaux de neurones conçue spécialement pour traiter des images en entrée. Le plus souvent les CNN sont utilisés dans la classification d'images car ils permettent d'extraire des informations pertinentes des images. Cette extraction est ce que l'on appelle un **apprentissage de features**. Cet apprentissage comporte plusieurs étapes que nous allons citer et décrire ci-dessous. Elles sont au nombre de trois, la **convolution**, l'activation **ReLU** ou **elu** et le **pooling** (qui ne sera pas expliqué car non utilisé).

### 7.2.1 Convolution

La convolution réside en l'application d'un filtre mathématique à une image. En d'autres termes, il s'agit de balayer une image d'entrée avec une matrice de taille définie et de remplacer chaque pixel par la valeur de la somme de la multiplication du pixel par cette même matrice. Cette méthode permet d'extraire d'une image d'entrée des informations pertinentes. On appelle la matrice en question **masque ou noyau de convolution**.

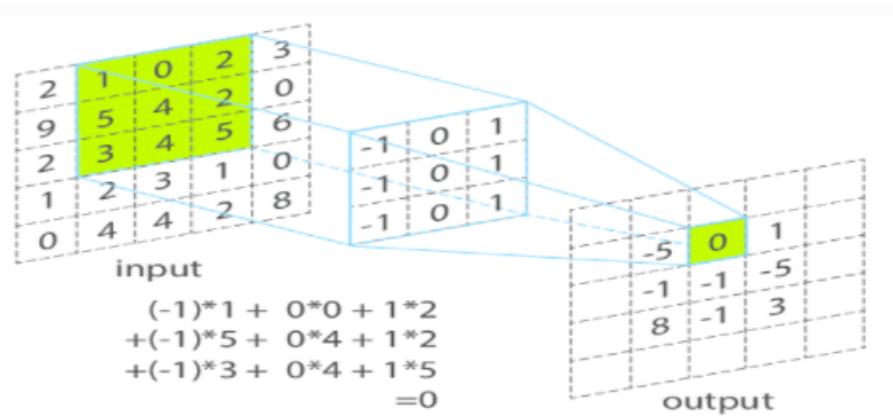


FIGURE 7.3 – Exemple de convolution [5]

On voit donc ici sur la figure 7.3 que ce que l'on appelle le masque (matrice de 3x3 au centre de la figure) est "glissé" au travers de l'image et le résultat de chaque pixel de sortie est la somme des opérations de la partie de l'image ciblée par le masque, par le filtre lui-même.

Une convolution se définit par plusieurs paramètres, lesquels sont :

- Dimensions de l'entrée (Hauteur,largeur, nombre de canaux, et taille du lot)
- Hauteur et largeur du noyau de convolution
- Déplacement en hauteur et largeur du noyau
- Padding en hauteur et en largeur

On remarque aussi qu'une convolution, si on ne lui spécifie pas de valeurs de padding (remplissage des espaces), réduit également la taille de l'image en sortie. En effet les dimensions de sortie sont calculables de la manière suivante pour la hauteur :

$$OutHeight = \frac{inHeight + topPadding + bottomPadding - kernelHeight}{strideHeight} + 1 \quad (7.1)$$

Les variables représentant :

- OutHeight** Hauteur de la sortie
- inHeight** Hauteur de l'entrée
- topPadding** Padding vertical en haut
- bottomPadding** Padding vertical en bas
- kernelHeight** Hauteur du noyau de convolution
- strideHeight** Déplacement vertical du noyau

Et pour la largeur :

$$OutWidth = \frac{inWidth + rightPadding + leftPadding - kernelWidth}{strideWidth} + 1 \quad (7.2)$$

Les variables représentant :

- OutWidth** Largeur de la sortie
- inWidth** Largeur de l'entrée
- rightPadding** Padding horizontal à droite
- leftPadding** Padding horizontal à gauche
- kernelWidth** Largeur du noyau de convolution
- strideWidth** Déplacement horizontal du noyau

Si l'on applique les équations 7.1 ainsi que 7.2 sur les données que l'on observe à la figure 7.3, on remarque que la taille de sortie sera de  $\frac{5+0+0-3}{1} + 1 = 3$  pour la hauteur et  $\frac{5+0+0-3}{1} + 1 = 3$  également pour la largeur.

Dans le cadre des couches convolutionnelles d'un CNN, les dimensions seront définies pour chaque plan de l'image. (Une image peut comporter plusieurs plans, comme par exemple une image dite RGB qui correspond à un plan avec une valeur pour chaque couleur, rouge, vert, bleu).

La convolution permet donc d'extraire des informations pertinentes comme des contours ou des lignes d'une image en lui appliquant les masques adéquats. Voyons par exemple l'effet de l'application d'un filtre sur une image :

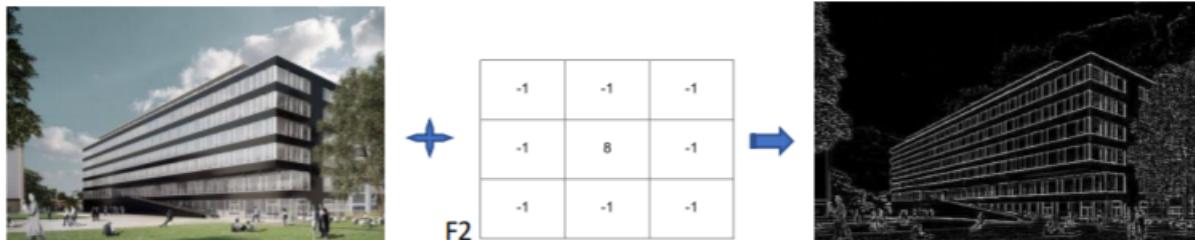


FIGURE 7.4 – Effet de la convolution [6]

On constate donc sur la figure ci-dessus l'effet de l'application du filtre appelé **F2** sur l'image du bâtiment. On observe ainsi que les valeurs de ce filtre permettent d'extraire les contours à l'intérieur d'une image. Dans le cas d'un CNN, les valeurs contenues dans les filtres sont les paramètres des couches de convolution et donc celles-là seront adaptées et optimisées à chaque étape de l'apprentissage du réseau.

### 7.2.2 Fonction d'activation

La fonction d'activation [7] est une fonction mathématique qui permet de reproduire le potentiel d'activation biologique du cerveau humain. Elle va en quelque sorte définir si l'information peut passer ou non en fonction d'un seuil de stimulation. En clair, elle permettra de déterminer si l'on active ou non la réponse d'un neurone. Le neurone en lui-même appliquera donc cette fonction sous la forme suivante :

$$X = \sum(\text{entrée} * \text{poids}) + \text{biais} \quad (7.3)$$

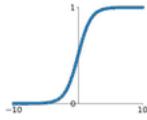
Les fonctions d'activation servent également à lisser ou normaliser la donnée de sortie avant de la transmettre aux neurones suivants.

Il existe plusieurs fonctions d'activation possible dans le domaine des réseaux de neurones lesquelles sont :

## Activation Functions

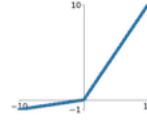
### Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



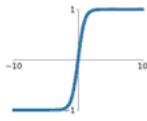
### Leaky ReLU

$$\max(0.1x, x)$$



### tanh

$$\tanh(x)$$

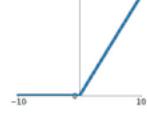


### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

### ReLU

$$\max(0, x)$$



### ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



FIGURE 7.5 – Fonctions d'activation [7]

La fonction la plus utilisée dans ces domaines est intitulée ReLu (Rectified Linear Unit) car c'est celle qui a tendance à converger le plus rapidement et elle se trouve être peu gourmande en termes de calculs processeurs. Cependant, elle présente parfois un problème appelé le **Dying ReLu**.

### 7.2.2.1 Problème de dying ReLu [8]

Ce problème consiste en une saturation pour les nombres négatifs. En effet, on observe que lorsque le nombre est négatif, la fonction ReLu impose une valeur de 0, ceci a pour conséquence d'entraîner la mort de certains neurones. Prenons la figure suivante comme exemple :

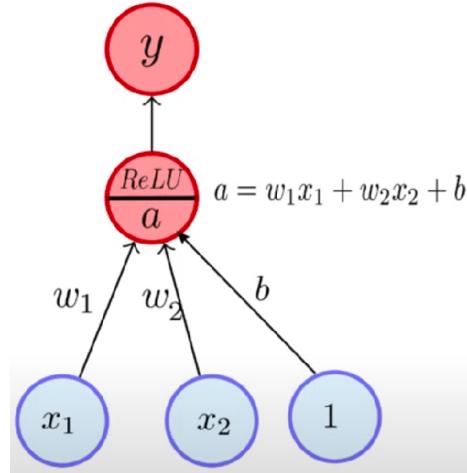


FIGURE 7.6 – Figure exemple dying ReLu

Nous avons comme entrées du neurone les résultats de la couche précédente  $X1, X2$ , le terme de biais  $b$  et les poids du neurone  $W1, W2$ .  $a$  est ce que l'on appelle la pré-activation du neurone, c'est-à-dire l'application des différents poids mais sans la fonction d'activation. Si  $a$  est négatif, lorsque l'on applique la fonction ReLu, la valeur de sortie du neurone deviendra automatiquement 0 et aucun des poids ne sera utile à la sortie et ils ne seront donc ni mis à jour ni optimisés durant l'apprentissage. Le principe est le même si les poids  $W1$  et  $W2$  sont négatifs, le résultat de ce neurone sera toujours négatif et donc il ne participera pas à l'optimisation du gradient (ci-après) et sera écarté. C'est donc ce phénomène que l'on appelle **Dying ReLu**.

Afin de pallier à ce problème, on utilisera la fonction d'activation **elu** (graphique sur la figure 7.5), qui, lorsque la valeur est positive, fonctionne de la même manière que la *ReLU*, mais qui va prendre une valeur faible calculée par  $\alpha * (e^x - 1)$ , ce qui en ne fixant pas la valeur à zéro permet de laisser une valeur (si petite soit-elle) passer par ce neurone et ainsi de ne pas le "tuer" et continuer son optimisation.

### 7.3 Modèle Nvidia

Le modèle de réseau de neurones est un modèle développé par l'entreprise Nvidia dans le cadre d'un de leurs projets de voitures autonomes [9]. Il s'agit donc d'un modèle de réseau de neurones convolutif dont la structure est la suivante :

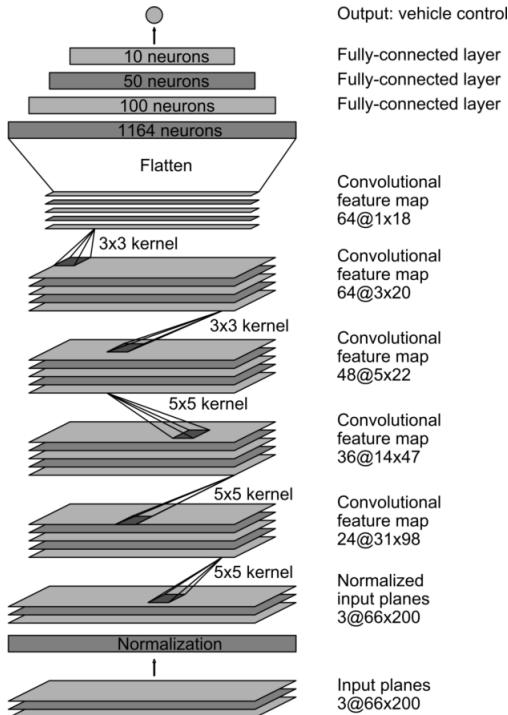


FIGURE 7.7 – Modèle CNN Nvidia

On peut donc découvrir ici le format de l'image d'entrée qui est de 66x200 pixels. Nvidia conseille également l'utilisation du format de couleur YUV (voir plus loin) afin de faciliter la détection de lignes. Sur la figure 7.7 ci-dessus on peut donc voir les 4 couches de convolution permettant, sur le principe expliqué à la section 7.2, d'extraire les informations importantes relatives aux lignes de l'image. Ces couches de convolution sont suivies de quatre couches pleinement connectées (fully connected layers en anglais), qui elles sont les couches "de base" d'un réseau de neurones qui vont ici se charger, en fonction des informations extraits par convolution, de calculer et définir la valeur de l'angle de rotation.

## 8. Langage de programmation

Pour ce projet, le langage **Python** a été choisi malgré une relativement faible connaissance de celui-ci à la base pour plusieurs raisons :

- Grande communauté contenant aides, tutoriels, librairies
- Calcul matriciel optimisé en python (utilisé pour les images)
- Librairies très complètes sur le machine learning (création de modèles, entrainement, etc.)

Un désavantage de ce langage est que sa rapidité d'exécution est relativement faible, ce qui dans notre cas est handicapant, cependant le fait d'utiliser un CNN ainsi que l'accélérateur matériel *Coral USB Accelerator* permet de compenser cette lenteur. De plus, malgré une bien plus grande expérience avec des langages comme le C (et ses dérivées C++, C#), les points cités ci-dessus ainsi que la possibilité de réutiliser des fonctions programmées par mon prédécesseur M. Maxime Charrière m'ont poussé à utiliser ce langage. Il a donc naturellement fallu acquérir les compétences nécessaires en Python telles que le multithreading, la gestion de classes ainsi que les bases de ce langage.

## 9. Création d'une base de données

Du fait de l'utilisation d'un CNN pour la détection de lignes, il a fallu créer une base de données sur laquelle entraîner le réseau. Pour cette réalisation, plusieurs étapes ont été nécessaires, celles-ci seront développées dans ce chapitre.

### 9.1 Pilotage Manuel

La première étape était de créer un script permettant le pilotage manuel de la voiture. En effet, je me suis tourné vers cette option afin de simuler au mieux le comportement d'un être humain conduisant la voiture.

Pour ce faire, une manette de type *Dualshock* (manette de Playstation 4) a été utilisée. La manette communique avec le RPI via le protocole *Bluetooth* et chacun de ses joysticks commande la vitesse et la direction alors que le bouton "rond" sert à stopper le script (voir figure 9.1 ci-dessous).



FIGURE 9.1 – Pilotage manuel avec manette DualShock

L'implémentation du contrôle par manette dans le script passe tout d'abord par la librairie Python *evdev* [10]. Cette librairie permet de décoder les signaux transmis par des périphériques d'entrée (comme un clavier, une souris, une manette,etc.) dans le but de les exploiter ensuite dans n'importe quel projet. Cette librairie se charge également de faire le lien entre des événements générés dans le noyau de l'OS et l'espace utilisateur (par exemple ici un script Python).

Lors de la connection via bluetooth, un fichier d'événement est créé dans le répertoire */dev/input/*, ce fichier représente un périphérique connecté au RPI, il suffit donc de repérer lequel correspond à la manette, dans notre cas, le fichier *event2*. La librairie *evdev* permet ensuite de décoder ces fichiers d'événements et dans notre cas par exemple d'observer l'appui sur un bouton ou le mouvement d'un joystick.

Maintenant que nous avons, grâce à *evdev* un objet de type *InputDevice* et donc un moyen d'accéder aux interactions de la manette avec le noyau, il faut utiliser la librairie *asyncio* [11]. Cette librairie permet de réaliser une programmation asynchrone. La programmation asynchrone est un principe de

programmation qui octroie l'accès à l'exécution de plusieurs processus de manière quasi-indépendante. Dans notre cas, si le processus principal devait attendre une modification d'une des variables de la manette avant de continuer, cela le rendrait très lent et le pilotage de la voiture deviendrait impossible. Il a donc été nécessaire de créer une fonction nommée *event\_manager* qui est une fonction asynchrone permettant de décoder les valeurs des différents boutons/joysticks de la manette et écrire sur les sorties du RPI les valeurs à envoyer pour la direction et la vitesse. On a ensuite créé avec evdev ce qu'on appelle une boucle d'évènements (event loop dans le code) qui permettra d'appeler la fonction *event\_manager* jusqu'à ce que celle-ci soit terminée (c'est-à-dire que le bouton "rond" soit appuyé). De cette manière, il est possible de piloter manuellement la voiture grâce à la manette DualShock.

## 9.2 Enregistrement d'images et de valeurs de direction

Une fois le script de pilotage manuel fonctionnel, il s'est agi d'enregistrer des images de la route ainsi que l'angle de direction correspondant. Pour ce faire, la classe *DataCollector* a été créée. Cette classe fonctionnera sur un thread séparé et se chargera de capturer une image grâce à la caméra de la voiture, ainsi que de récupérer la valeur d'angle de direction de la voiture correspondant à cette image. La capture d'une image se fait en moyenne tous les 30ms. La classe DataCollector accède à la caméra ainsi qu'aux valeurs de direction par l'intermédiaire de son attribut *car* qui est une référence à l'objet représentant la voiture (voir schémas UML plus loin).

Les angles de direction sont initialement définis par la valeur du PWM, cependant, à des fins de compréhension ainsi que d'affichage, lors de l'enregistrement, ils seront définis de la manière suivante (cf figure 9.2) :

Le virage maximal à gauche est représenté par un angle de 45° alors que le virage maximal sur la droite est représenté par un angle de 135°, c'est alors naturellement qu'une ligne droite est représentée par un angle de 90°.

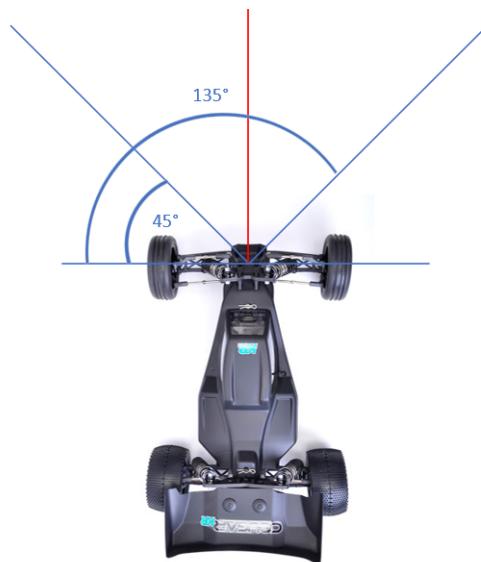


FIGURE 9.2 – Schéma des valeurs d'angles de direction

Le format utilisé pour l'enregistrement est le suivant :  
 Le script recherche dans son chemin d'accès le dossier nommé *DataCollected*, puis dans ce dossier la présence de sous-dossiers intitulés *SETx* (x étant un nombre entier indiquant l'indice du set de données). Une fois ceci fait, on crée un nouveau sous dossier *SETx* avec l'indice correspondant à la suite des dossier déjà présents. Prenons l'exemple suivant :

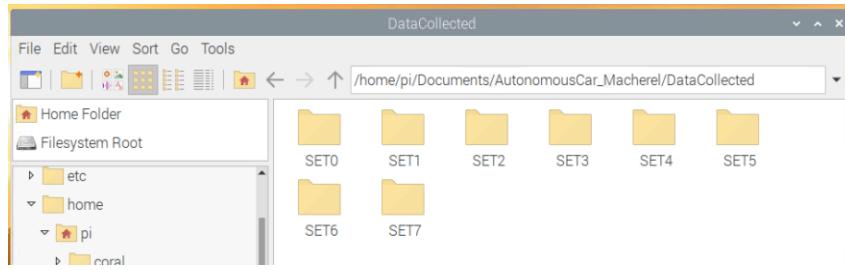
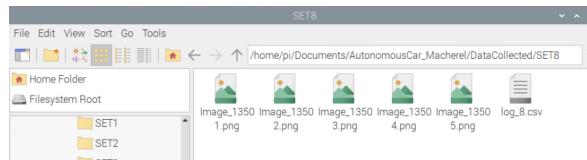


FIGURE 9.3 – Contenu du dossier DataCollected

On voit donc ici qu'au lancement du script le dossier *DataCollected* contient déjà sept sets de données, il va donc automatiquement créer un dossier *SET8* dans lequel il stockera les informations capturées.

Ces données seront mémorisées de la manière suivante :

Chaque image aura un nom sous la forme *Image\_Indice.png*, où l'indice est un nombre stocké dans un fichier de configuration s'incrémentant à chaque capture. Les valeurs des angles de direction seront, quant à elles, stockées dans un fichier au format *.csv* appelé *log\_IndiceDuDossierSETx*, où chaque ligne contiendra le nom de l'image et sa valeur d'angle correspondante séparés par une virgule.



(a) Contenu du dossier créé SET8

	A
1	Image_13501.png,90.0
2	Image_13502.png,90.0
3	Image_13503.png,92.64705882352942
4	Image_13504.png,92.64705882352942
5	Image_13505.png,92.29411764705883
6	

(b) Contenu du fichier log\_8.csv

FIGURE 9.4 – Dossier SET8 et contenu du fichier Log

## 9.3 Augmentation du dataset

Afin de limiter le nombre d'heures de conduite manuelle ainsi que le nombre d'images nécessaires à l'entraînement, plusieurs fonctions ont été créées dans le but d'augmenter la diversité du dataset. Différentes possibilités d'augmentation des images ont donc été mises en place :

### 9.3.1 Floutage

Une première fonction utilisée est celle du floutage gaussien. Le flou gaussien est un **filtre d'atténuation**, il permet d'uniformiser une image en floutant ses parties et donc d'harmoniser les détails de celle-ci. Ce filtre est appliqué en réglant la valeur de chaque pixel sur la moyenne de tous les pixels présents dans un rayon défini (d'où le terme gaussien représenté par la fameuse courbe de Gauss). Plus ce rayon est grand, plus l'image sera floutée. [12]

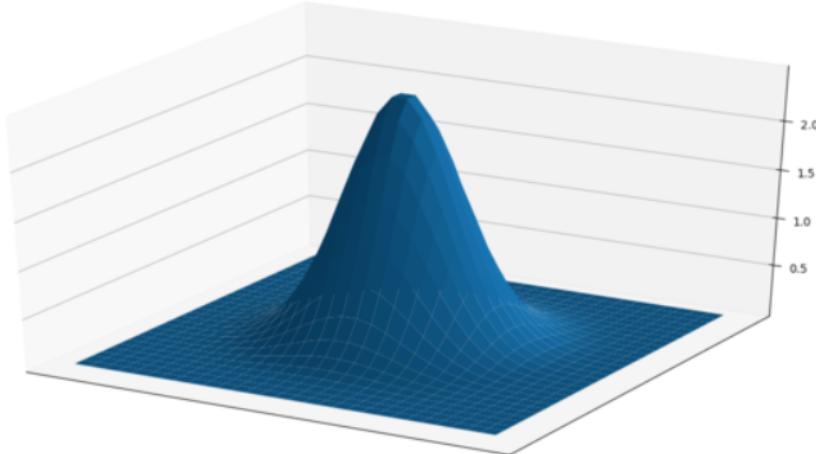


FIGURE 9.5 – Courbe gaussienne

Pour obtenir une idée plus claire de ce fonctionnement, il faut imaginer la figure 9.5 superposée sur le masque de filtrage. La valeur de chaque point du graphique correspond au poids donné au pixel correspondant dans le masque. Ainsi les points proches du centre du masque se retrouvent "plus importants" (plus grand poids) que ceux situés à la périphérie de celui-ci. [13]

On utilisera donc cette méthode afin de flouter l'image initiale et de simuler par exemple une présence de poussières ou même de brouillard dans une situation réelle. Dans le code, nous utiliserons la fonction *blur* de la librairie *OpenCV* [14]. Afin de rendre la modification par floutage aléatoire, la taille du masque de floutage sera choisie aléatoirement entre 1 (pas de changement) et 5 (floutage maximal).

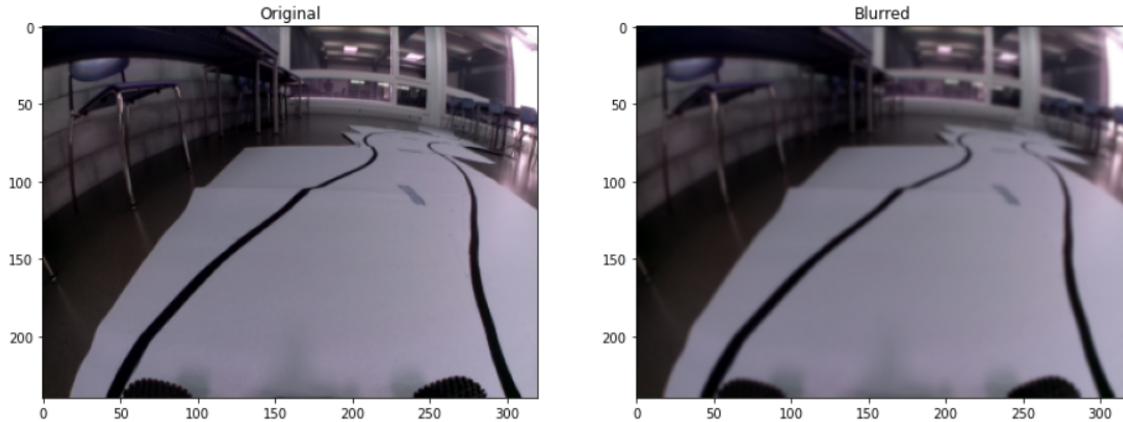


FIGURE 9.6 – Exemple d'image floutée avec un masque de 5x5

### 9.3.2 Ajustement de luminosité

La deuxième fonction utilisée pour l'augmentation du dataset est une fonction permettant de modifier le taux de luminosité de l'image. Cette fonction permet de générer des images plus claires ou plus foncées dans le but de simuler des environnements à luminosité variable.

Pour réaliser cela, on utilisera la librairie *imgaug* et plus particulièrement son package *augmenters* [15]. La fonction *Multiply* de cette librairie permet de définir un masque correspondant à la taille d'une image rempli de valeurs aléatoires définies entre un minimum et un maximum transmis lors de l'appel de la fonction. La méthode *augment\_image* permet alors de multiplier l'image initiale par ces poids afin d'agir sur l'obscurité de l'image.

Dans notre cas, on fera varier la luminosité aléatoirement entre 70 et 130%.

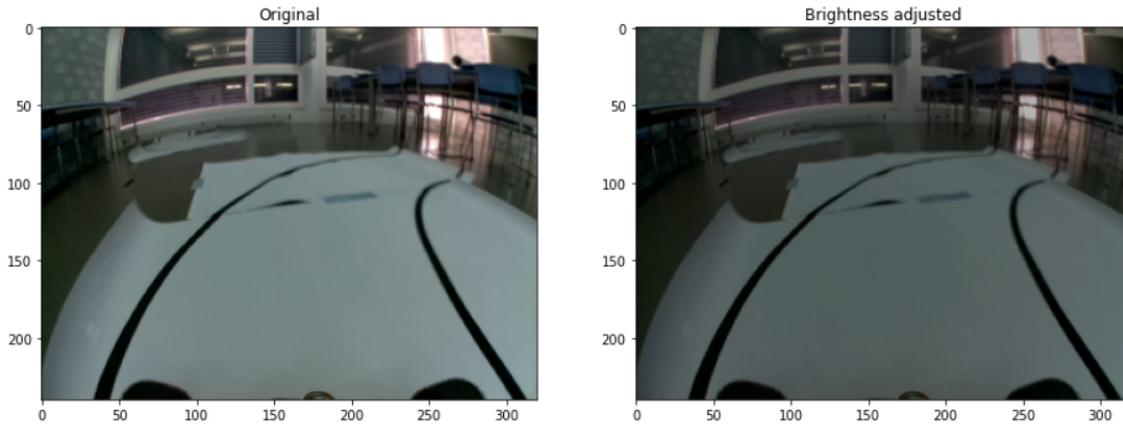


FIGURE 9.7 – Image avec luminosité diminuée de 30%

### 9.3.3 Inversion horizontale aléatoire

Enfin, la dernière fonction permet l'inversion horizontale de l'image. La fonction prend donc en entrée l'image et sa valeur de direction. Si nous prenons par exemple un virage à gauche, lors de son retournement, celui-ci devient un virage à droite, la valeur de direction doit donc également être modifiée. Pour ce faire on utilisera la fonction *flip* de la librairie *OpenCV* [16]. Cette fonction sert à retourner un tableau 2D (ici une image) selon l'axe choisi.

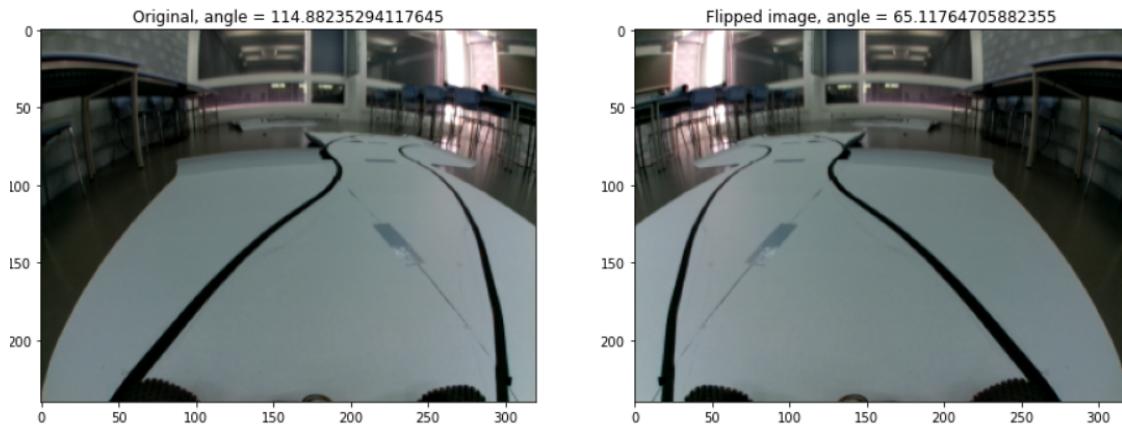


FIGURE 9.8 – Image inversée

### 9.3.4 Fonction regroupant toutes les modifications

Toutes les fonctions de modification citées ci-dessus sont regroupées ensuite dans une fonction nommée *RandomModify*, qui appliquera aléatoirement l'une ou l'autre de ces modifications d'images. Cette fonction sera appelée lors de la génération dynamique de données pour l'entraînement (voir section 10.7) afin qu'à chaque génération de données d'entraînement sur la base du dataset initial, certaines images soient aléatoirement modifiées.

## 10. Entrainement du CNN

Dès lors que les images sont stockées, il est temps d'entraîner le CNN. Pour réaliser ceci, les librairies et ressources suivantes ont été utilisées :

### 10.1 Pré-traitement de l'image

Une fois les données collectées, un pré-traitement de l'image est nécessaire avant l'entraînement du réseau de neurones. En effet, comme présenté dans la section 5.3, Nvidia conseille l'utilisation de l'espace de couleurs YUV (qui sera expliqué ci-dessous). De plus, l'image sera tout d'abord tronquée environ aux 3/4 (185 pixels sur les 240 initiaux en hauteur seront retirés). Il a été décidé de procéder ainsi car toute la zone supérieure s'avère obsolète lors de l'entraînement du réseau. Pire même, elle pourrait générer des erreurs et des mauvaises interprétations de la part du CNN car il pourrait par exemple focaliser son entraînement sur un élément du fond de l'image et non pas sur les lignes. En réduisant de cette manière l'image, on s'assure que l'entraînement se fera bien uniquement sur les lignes détectées. Une fois la région d'intérêt isolée, il s'agit de redimensionner l'image à la taille requise par le CNN, soit 200x66 pixels.

#### 10.1.1 Espace de couleurs YUV

Le principe de l'espace colorimétrique YUV est de représenter une couleur sur la base de trois composantes, une de **luminance (Y)**, et deux de **chrominance (U & V)**. La composante de luminance est une moyenne pondérée des trois composantes primaires, R (rouge), V (vert) et B (bleu), les coefficients de pondération sont issus directement de la sensibilité de l'œil humain aux couleurs primaires [17]. Ainsi la composante Y (luminance) se calcule ainsi :

$$Y = 0.2989 * R + 0.5866 * G + 0.1145 * B \quad (10.1)$$

Ou R (red), G (green), B (blue) sont les composantes primaires de l'image de base. Les composantes de chrominance, quant à elles, correspondent à une différence normalisée entre la composante primaire et la luminance. Ainsi :

$$U = 0.5647 * (B - Y) \quad (10.2)$$

$$V = 0.7132 * (R - Y) \quad (10.3)$$



FIGURE 10.1 – Lien entre la luminance et l'espace RGB [18]

On observe donc sur la figure 10.1 le gain réalisé à utiliser des bandes noires pour les lignes de la route. Dans les faits, la couleur noire ne change pas en termes de luminance ce qui permet aux lignes de rester très visibles dans l'image.

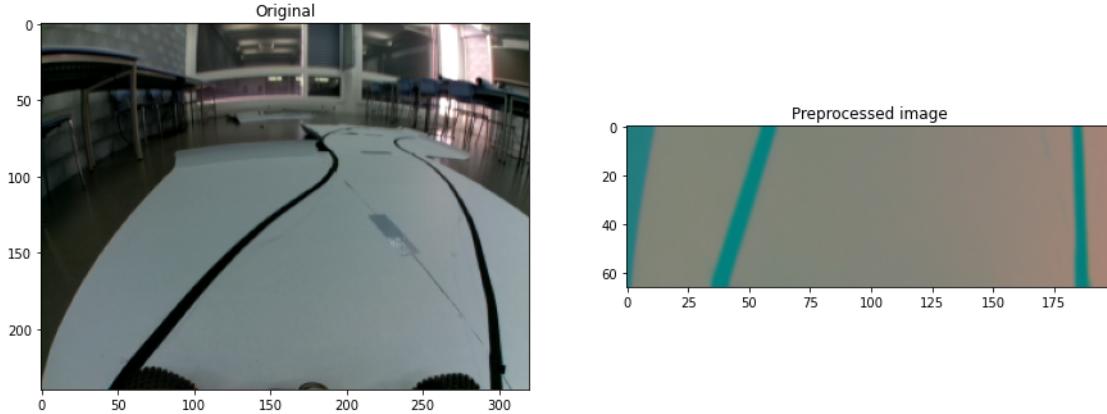


FIGURE 10.2 – Image après le pré-traitement

On voit donc ici sur la figure 10.2 que les lignes sont bien distinctes et que le CNN n'a pas de perturbations sur lesquelles son entraînement pourrait dévier.  
Initialement, un second processing était appliqué à l'image afin d'enlever également les bords de route parfois apparents. Cependant, la méthode consistait en un balayage et repérage des lignes pour supprimer toute luminance différente à l'extérieur de celles-ci et il arrivait parfois que la ligne se trouve sur le bord de l'image selon le pilotage manuel effectué si la voiture se trouvait en bord de route. Cela causait des problèmes à l'effacement. C'est pourquoi il a été décidé de ne maintenir que le premier traitement visible à la figure 10.2.

## 10.2 Google Colaboratory

Google Colaboratory ou Google Colab est un service de Google permettant d'exécuter du code Python sur une machine virtuelle. Le site permet également d'éditionner le code et fonctionne entièrement comme un IDE, c'est à dire qu'il a à sa disposition des outils tels que l'assistance de complétion de code. L'avantage de ce service est qu'il est hébergé par Google et qu'il permet d'accéder gratuitement à différentes ressources informatiques telles que des GPU ou NPU. En d'autres mots, ce service rend disponibles des ressources de Google pour effectuer les énormes quantités de calculs nécessaires à l'entraînement d'un CNN sur des serveurs distants. Ainsi il accélère le processus et limite l'utilisation de son propre PC dans cette manœuvre. Un autre avantage de ce service est la connexion à son espace Google Drive personnel afin d'automatiquement sauvegarder des fichiers dans celui-ci.

### 10.3 Keras Sequential Model

La bibliothèque Keras offre l'opportunité d'intéragir avec différents algorithmes de réseaux de neurones, dont Tensorflow que nous verrons plus tard. Cette librairie est utilisée ici afin de construire le modèle (cf section 7.3). En effet, keras rend possible la création de son propre modèle de réseau de neurones en y ajoutant par code des couches, voici donc la création du modèle via Keras :

```

def nvidia_model():
    model = Sequential(name='Nvidia_Model')
    # Convolution Layers
    model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Dropout(0.2)) # not in original model. added for more robustness
    model.add(Conv2D(64, (3, 3), activation='elu'))

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dropout(0.2)) # not in original model. added for more robustness
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))

    # output layer: turn angle (from 45-135, 90 is straight, <90 turn left, >90 turn right)
    model.add(Dense(1))

    # since this is a regression problem not classification problem,
    # we use MSE (Mean Squared Error) as loss function
    optimizer = Adam(learning_rate=0.5*1e-3) # lr is learning rate
    model.compile(loss='mse', optimizer=optimizer)

    return model

```

FIGURE 10.3 – Déclaration d'un objet avec les paramètres du réseau de Nvidia

Sur la figure 10.3, on remarque une différence entre l'image 7.7 et la déclaration ici faite. En effet la première couche cachée de normalisation n'est pas présente car la normalisation de l'image a été effectuée dans le pré-traitement. De plus, deux couches dites "dropout" ont été ajoutées pour augmenter la robustesse du modèle. Le dropout permet de "désactiver" aléatoirement des sorties de neurones (la probabilité étant le paramètre entre parenthèses, ici 0.2 donc 20%) durant l'apprentissage. Cette méthode permet de simuler des modèles différents et de les apprendre conjointement. En forçant certains neurones à s'inhiber durant une itération d'apprentissage, on oblige les autres neurones actifs à compenser ceci et cela améliore leur apprentissage. Cette technique empêche également ce que l'on appelle l'interdépendance entre les neurones [19].

## 10.4 Optimiseur de modèle

L'optimiseur est la fonction qui permet de mettre à jour à chaque itération de l'apprentissage les poids des différents neurones afin de réduire la fonction de perte. Pour ce modèle, l'optimiseur **Adam** est utilisé. L'algorithme Adam est une extension à la descente de gradient et est un des algorithmes les plus utilisés dans le domaine du deep Learning, il a été présenté par Diederik Kingma et Jimmy Ba [20]. Dans les grandes lignes, le principe d'Adam est le même que celui de la descente de gradient (pour plus d'informations sur la descente de gradient -> [21]) à quelques différences près. Ces différences sont : Tant que le gradient est dans la même direction que les précédents, la vitesse d'apprentissage (Learning Rate en anglais) sera augmentée, c'est à dire que les paramètres seront mis à jour plus rapidement. Cependant, si le gradient prend une autre direction, la vitesse d'apprentissage reviendra à sa valeur initiale spécifiée dans la déclaration de l'optimiseur (ici, 0.005).

## 10.5 Fonction de perte

La fonction de perte (ou loss function en anglais) sert à évaluer l'écart entre les prédictions du modèle et les valeurs d'apprentissage, l'objectif de la phase d'apprentissage est donc de minimiser cette fonction et ainsi améliorer les qualités de prédiction du modèle. La fonction de perte dans notre cas est la fonction MSE (Mean Squared Error), elle se présente de la manière suivante :

$$MSE = \frac{1}{N} * \sum_{j=1}^N (y_i - \hat{y}_j)^2 \quad (10.4)$$

La fonction calcule donc la moyenne des carrés de la différence entre la valeur prédite et la valeur réelle.

## 10.6 Séparation du dataset

Lors de l'entraînement d'un réseau de neurones, il est important de séparer le set de données en deux parties que l'on appelle le *training Set* et le *validation Set*. Pour ce faire on utilisera la fonction *train\_test\_split* de la librairie *sklearn.model\_selection* [22]. Il est important de séparer le dataset afin d'avoir des données utilisées uniquement pour l'entraînement mais également des données dites de validation qui permettront de "tester" le modèle et d'observer ses performances à chaque étape de l'apprentissage.

Dans notre cas, le set de base étant de 3292 images, il a été décidé de laisser un taux de 80% des images en images d'entraînement et d'utiliser les 20% restants comme images de test. On obtient donc ainsi un training set de 2633 images ainsi qu'un validation set de 659 images.

## 10.7 Phase d'entraînement

Maintenant que le réseau est créé et les données prêtes, il est temps de lancer la phase d'entraînement. Pour cet exercice, on utilise la fonction `fit()` du modèle, qui est une fonction de Keras permettant d'entrainer les CNN. La particularité utilisée dans notre entrainement est qu'on ne va pas utiliser un set de données d'entraînement statique en envoyant directement toutes les images mais plutôt une génération dynamique de données à partir des images de base. Pour cela une fonction nommée `data-Generator` a été créée. Elle prend en arguments la liste des chemins d'accès aux images, la liste des valeurs d'angle de direction, la taille du lot souhaité en retour ainsi qu'un paramètre booléen indiquant si ce sont des images d'entraînement ou non. La différence faite par le booléen est que lorsque celui-ci est à `True`, on appliquera la fonction de modification aléatoire de la section 9.3.4.

```
def dataGenerator(imagePaths, steeringAngles,batchSize,isTraining):
    while True:
        batchImages = []
        batchSteering = []
        for i in range(batchSize):
            randomIdx = random.randint(0,len(imagePaths)-1)
            imagePath = imagePaths[randomIdx]
            image = imreadModif(imagePath)
            steeringAngle = steeringAngles[randomIdx]
            if isTraining:
                # Image modification for training
                image,steeringAngle = randomModify(image,steeringAngle)
            #
            # print(randomIdx)
            # print(imagePath)
            image = ImagePreprocess(image)
            batchImages.append(image)
            batchSteering.append(steeringAngle)
        yield (np.asarray(batchImages),np.asarray(batchSteering))
```

FIGURE 10.4 – Aperçu de la fonction de génération dynamique de données

L'entraînement est ensuite réalisé grâce à la fonction `fit()` qui est paramétrée pour effectuer 18 étapes de 400 pas avec pour chaque étape, 150 pas de validation. On définit également la taille du lot d'images utilisées pour l'entraînement, ici nous utilisons un lot de 400 images, qui sera testé puis entraîné 400 fois par étape d'apprentissage. Les pas de validation fonctionnent de la même manière que ceux d'entraînement à la seule différence qu'ils seront effectués sur des images non utilisées dans l'entraînement afin de vérifier la progression d'apprentissage du CNN.

## 10.8 Résultats d'entraînement

A la fin de l'entraînement d'une durée d'environ 2h (jusqu'à 3h selon les paramètres choisis), on peut évaluer ses performances à l'aide de la statistique *R-squared*. Cette valeur se trouve entre 0 et 100% et représente une image de la proximité entre les valeurs prédictes et les valeurs d'entraînement. Si l'on prend par exemple un set de 100 images tirées aléatoirement dans le dataset, on obtiendra un taux de prédictions correctes de 80%, ce qui est un bon résultat pour ce genre de réseaux de neurones, sachant que si le réseau prédit un angle de 92° alors que l'angle d'entraînement est de 90°, la fonction de vérification considérera cela comme un résultat faux. Alors que dans certains cas, l'angle sera plus adapté que lors de l'entraînement.

## 10.9 Conversion du modèle

Une fois le modèle entraîné, celui-ci est de type Keras et ses différents paramètres sont des valeurs en virgule flottante. Cependant, le Google Coral USB Accelerator spécifie qu'il est obligatoire d'utiliser des modèles de type **TensorFlow Lite**, qui sont des modèles avec des paramètres en entiers 8 bits, ce qui leur permet d'être plus rapides lors de leurs calculs. Il n'est pas possible d'entraîner un modèle TensorFlow Lite, cependant il est possible de convertir un modèle Keras ou TensorFlow en TFlite. Dans notre cas, on utilisera une quantification de la plage dynamique [23] afin de convertir le modèle Keras en TensorFlow Lite et pouvoir l'utiliser avec l'accélérateur Coral USB. Cette méthode consiste à quantifier post-entraînement les poids de la virgule flottante à l'entier 8 bits. Lors de l'appel au CNN, les poids seront reconvertis en virgule flottante et calculés par des noyaux à virgule flottante. Cette conversion n'est effectuée qu'une seule fois et mis en cache ensuite. La quantification en plage dynamique permet de réduire entre 3 et 4 fois le volume du CNN, et d'améliorer 2-3 fois sa vitesse.

```
Float model in Mb: 0.9679412841796875
Quantized model in Mb: 0.254150390625
Compression ratio: 3.808537463976945
```

FIGURE 10.5 – Aperçu de la réduction de volume après quantification

Il existe également d'autres possibilités de quantification comme la quantification entière-entières qui consiste à réellement quantifier toutes les mathématiques du modèle en nombres entiers. Cependant cette méthode implique de calibrer la plage de tous les tenseurs à virgules flottantes (tous les poids des couches,biais,activations). Toutefois, certains tenseurs ne sont pas calibrables à moins d'effectuer quelques cycles d'inférence, c'est pourquoi il est nécessaire de fournir lors de la quantification un ensemble de données représentatif pour étalonner ces derniers.

La différence de taille entre la quantification dynamique ou entière-entières ne change pas, cependant la deuxième peut s'avérer plus rapide avec l'utilisation d'un edge TPU.

## 11. Architecture logicielle

Ayant défini la conservation du même matériel que lors du précédent TB, il a été possible de récupérer certaines classes et scripts dans ce projet.

### 11.1 Classe Car

La classe **Car** a principalement été récupérée de l'ancien projet, et adaptée aux besoins de celui-ci. En effet, cette classe représente la voiture, elle contient donc les éléments suivants :

- Le contrôle du moteur de vitesse (SpeedController)
- Le contrôle du moteur de direction (SteeringController)
- Une caméra (PicameraController)

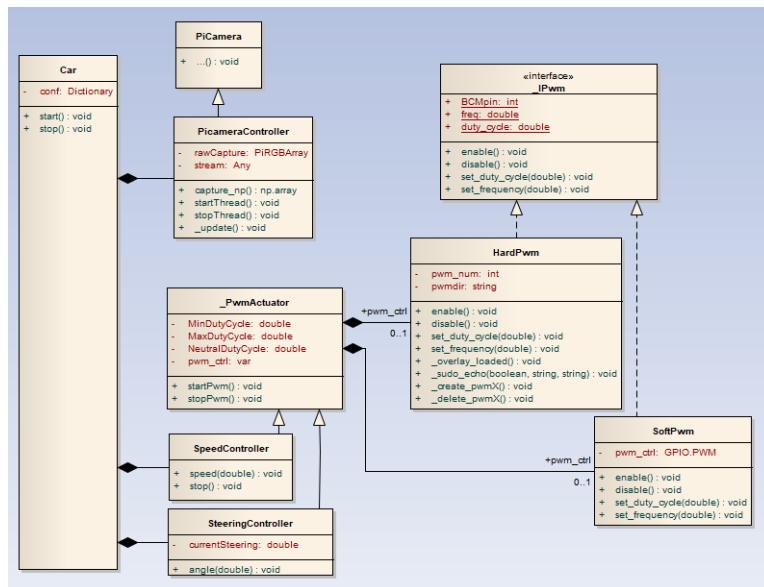


FIGURE 11.1 – Diagramme UML de la classe Car

Les éléments de cette classe ont pu être récupérés de l'ancien travail et adaptés. En effet il a fallu retirer certaines propriétés de la classe Car qui n'étaient plus utiles et y ajouter des propriétés telles que *currentSteering* qui permet d'accéder à la valeur de direction actuelle du contrôleur (son duty cycle).

On observe également qu'il existe dans ce projet deux types de PWM. Le premier est un signal software implémenté par la librairie RPi.GPIO.PWM qui génère de manière logicielle un signal de modulation. Pour des raisons de précision, un générateur PWM hardware a du être créé en se basant sur une puce prévue à cet effet dans le raspberry. Du fait de ces deux possibilités, la classe interface **\_IPwm** a été prévue par M. Charrière afin que leurs utilisations soient parfaitement identiques et que les différences soient gérées à l'intérieur de chaque objet. Il suffit donc uniquement de spécifier lors de la création d'un objet **\_PwmActuator**, à l'aide du paramètre "hardware :bool", si l'on souhaite créer un PWM hardware ou software.

La classe PicameraController hérite de la classe PiCamera (provenant du package du même nom [24]) et y ajoute deux fonctionnalités :

- Fonction de capture d'image au format numpy directement
- Gestion d'un thread permettant de capturer à vitesse spécifiée des images en continu.

Ces explications sont en parties tirées du rapport de M. Charrière [1].

## 11.2 Classe DataCollector

La classe DataCollector permet la sauvegarde des données d'entraînement (cf. section 9.2), elle est donc composée d'une référence à la classe voiture citée ci-dessus lui permettant d'accéder à ses divers composants tels que la caméra pour l'accès aux images et le contrôleur de direction pour ce qui est de la valeur transmise aux roues. Elle dispose également de plusieurs méthodes permettant de récupérer l'image, de la sauvegarder dans le dossier et d'en garder le nom dans une liste pour créer à la fin du processus le fichier de logs contenant également l'angle correspondant à celle-ci.

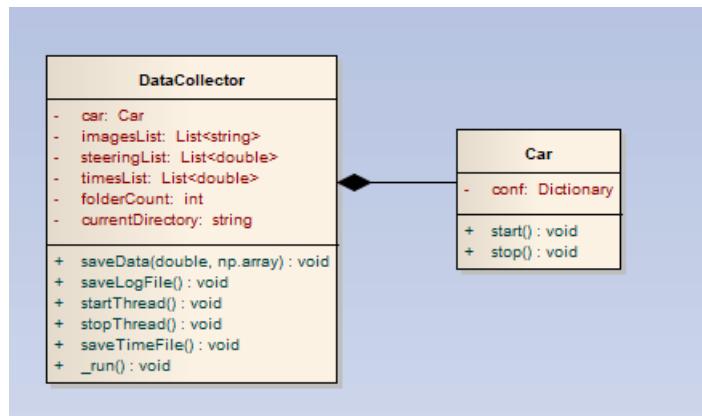


FIGURE 11.2 – Diagramme UML de la classe DataCollector

Cette classe contient également la gestion d'un thread automatisant tout le processus de capture et sauvegarde afin de pouvoir simplement déclarer un objet de type DataCollector dans le script de pilotage manuel et ainsi permettre la sauvegarde en parallèle du pilotage.

## 11.3 Application NavigatorApp

L'application principale permettant le suivi de la route contient les éléments suivants :

- Un objet *car* représentant la voiture.
- L'algorithme de suivi de route.
- Une possibilité d'affichage ou non de l'image capturée ainsi que la direction calculée.

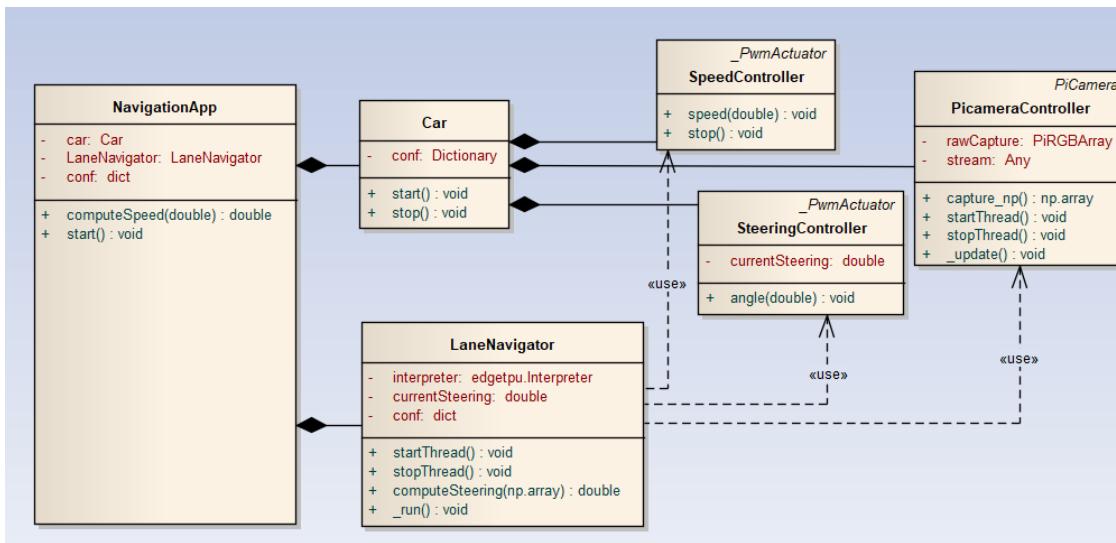


FIGURE 11.3 – Diagramme UML de l'application

### 11.3.1 Fonctionnement de l'application

L'application fonctionne sous la forme de plusieurs threads lesquels sont :

- Le thread de la caméra.
- Le thread principal.
- Le thread de calcul de la direction.

Il a été choisi d'implémenter ces différents threads afin que l'on ne retarde pas un processus par rapport à un autre. En d'autres mots, dans cette configuration chaque thread se charge d'une tâche particulière et n'attend pas sur les autres pour continuer. Ainsi la caméra capture sans arrêt des images, l'application principale se charge de l'affichage des informations et le thread du *LaneNavigator* s'occupe de récupérer l'image, de calculer l'angle de direction et de définir la vitesse sur la base de celui-ci.

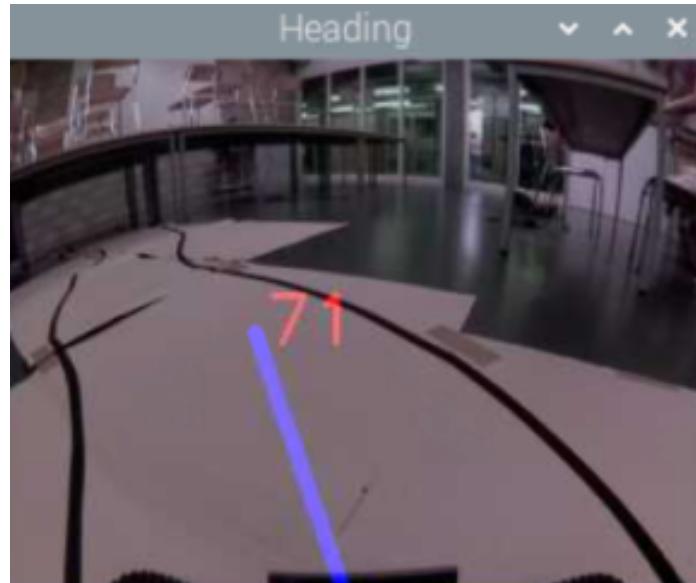


FIGURE 11.4 – Affichage généré par l'app principale

En tout temps de fonctionnement de l'application, celle-ci affiche comme sur la figure 11.4 l'image capturée ainsi que la direction calculée par le LaneNavigator.

## 11.4 Fichier de configuration

Un fichier de configuration a été mis en place afin de définir et réunir les principales constantes du programme. Ainsi, on évite de devoir modifier ces constantes dans le code et on limite le risque d'en oublier une. De cette manière, il suffit de donner au lancement de l'application le chemin d'accès au fichier de configuration qui sera ensuite lu pour que ses informations puissent être transmises aux différents objets qui en ont besoin.

Ce fichier a été écrit dans le langage *YAML* pour des raisons de forte compatibilité avec le Python mais également du fait de la possibilité d'utiliser la librairie PyYaml [25] qui est très efficace pour lire et éditer ce genre de fichiers.

```
CONTROLLER:  
  btn_stop: BTN_B  
  event_filename: /dev/input/event2
```

FIGURE 11.5 – Exemple d'une déclaration Yaml

On voit sur la figure 11.5 un exemple de déclaration en Yaml, celui-ci correspondant aux informations importantes pour l'utilisation de la manette lors du pilotage manuel (nom du fichier d'évènement et bouton permettant l'arrêt). Pour lire ce fichier il suffit d'importer le package Yaml puis d'appeler yaml.load(nom du fichier de config) et cette fonction retournera un dictionnaire python où chaque clé d'accès au champ est définie par la balise correspondante dans le fichier Yaml. Par exemple ici, si l'on veut accéder au champ correspondant au bouton d'arrêt il suffit d'exécuter le code suivant :

```
[ ] import yaml  
fileName = 'conf_MAR.yaml'  
config = yaml.load(fileName)  
stopButton = config['CONTROLLER']['btn_stop']
```

FIGURE 11.6 – Exemple de lecture de fichier de configuration en Yaml

## 12. Algorithme de suivi de route

### 12.1 Capture et pré-traitement de l'image

Afin de déterminer la direction à prendre, on récupère l'image actuelle de la caméra (la caméra possédant un thread qui capture en continu le flux vidéo, il est possible de récupérer en tout temps l'image actuelle) à laquelle on applique le pré-traitement (section 10.1).

### 12.2 Calcul de l'angle de direction

Une fois l'image pré-traitée, on utilise le modèle entraîné chargé lors de l'initialisation du programme pour déterminer l'angle de direction à suivre. L'utilisation d'un CNN et du Coral Accelerator permet un calcul très rapide de la direction. Une fois cet angle déterminé, on transmet celui-ci aux actuateurs de la voiture afin de tourner les roues dans la direction souhaitée. Le pré-traitement étant faible en ressources et l'utilisation du Coral Accelerator très efficiente, on arrive à un temps moyen de 24 ms pour ce qui est du calcul de l'angle de direction (pré-traitement et CNN), contrairement à environ 250 ms pour le précédent suivi de ligne (cf. [1], section 5.4.1).

```
----- ACTIVE THREAD -----  
<_MainThread(MainThread, started 3069631184)>  
<Thread(PicameraController, started 2451477600)>  
<Thread(LaneNavigator, started 2438984800)>  
Mean treatment time : 0.024666278230182692
```

FIGURE 12.1 – Affichage du temps de traitement moyen à la fin du script

## 12.3 Calcul de la vitesse

Afin de simuler au mieux un comportement humain, la voiture se déplacera avec une vitesse variant en fonction de l'angle de rotation. Ainsi, lorsque la direction sera droite celle-ci ira plus vite que lorsqu'elle doit tourner. La voiture étant initialement prévue pour des courses modèle-réduit, celle-ci peut atteindre des vitesses relativement élevées, cependant dans le cas de notre application atteindre les vitesses maximum est quelque chose de compliqué du au temps de traitement. Pour palier à ceci, deux vitesses (minimum et maximum) ont été définies dans le fichier de configuration. Pour l'exemple on affichera la fonction calculant le duty cycle à fournir au contrôleur en fonction de la valeur d'angle.

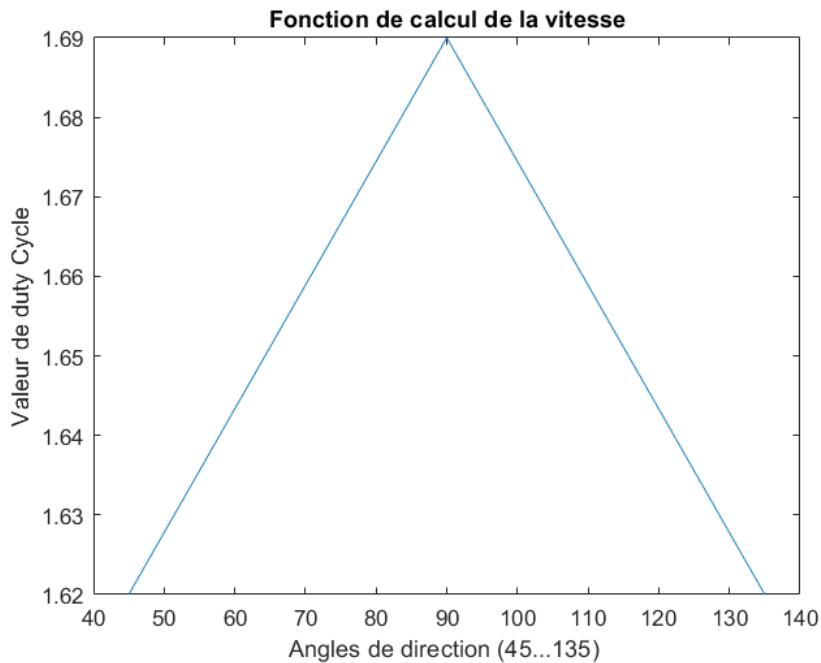


FIGURE 12.2 – Fonction de calcul du duty cycle (vitesse) en fonction de la direction

La fonction présente sur la figure 12.2 est la suivante :

$$a1 = \frac{MaxDC - MinDC}{45}, b1 = MaxDC - (a1 * 90) \quad (12.1)$$

$$a2 = -\frac{MaxDC - MinDC}{45}, b2 = MaxDC - (a2 * 90) \quad (12.2)$$

$$DC = a1 * steering + b1, si 45 \leq steering \geq 90 \quad (12.3)$$

$$DC = a2 * steering + b2, si 90 \leq steering \geq 135 \quad (12.4)$$

Avec :

**MaxDC** Valeur maximale du duty cycle (ici 1.69)

**MinDC** Valeur minimale du duty cycle (ici 1.62)

**DC** Valeur du duty cycle à appliquer au contrôleur

**steering** Valeur de l'angle de direction déterminé par le CNN

## 12.4 Fin de route

La détection de la fin de la route n'étant pas prise en charge par le CNN (ceci nécessiterait une refonte de l'architecture du modèle), une fonction a été implémentée pour ceci. Pour ce faire, l'image est convertie en niveau de gris, puis une fonction de la librairie opencv nommée *Canny* permettant la détection de contours est appliquée afin de faire ressortir les lignes. On calcule ensuite une moyenne sur les deux moitiés de l'image. Si cette moyenne est supérieure à 3, cela indique qu'une ligne est présente, dans le cas contraire la ligne n'est pas présente et on arrêtera la voiture.

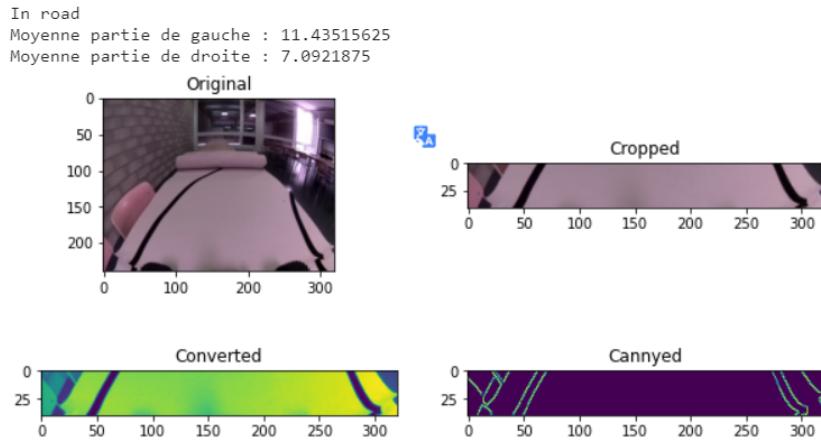


FIGURE 12.3 – Exemple d'image correcte sur la route

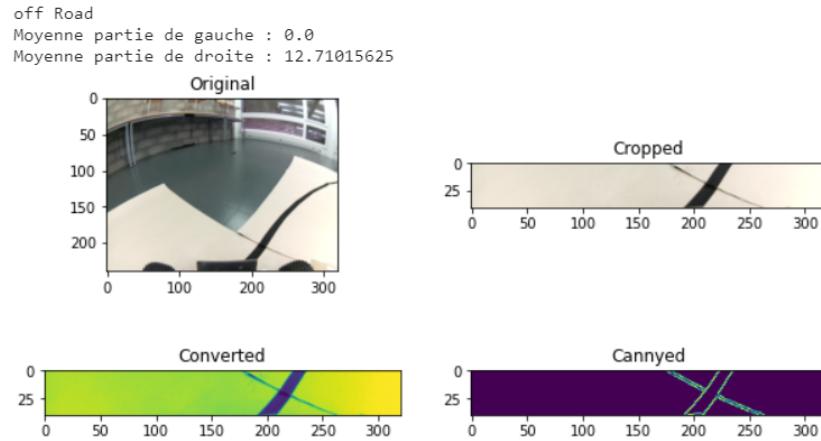


FIGURE 12.4 – Exemple d'image hors route

Cette fonction n'est pas extrêmement optimisée mais a dû être implémentée lors des tests afin que la voiture s'arrête lorsqu'elle sort de la route. Cependant lors de l'implémentation finale, la route sera un circuit fermé, ce qui permettra à la voiture de rouler en continu sans s'arrêter.

## 12.5 Tests sur images fixes

Une fois le modèle entraîné, des tests ont tout d'abord été effectués sur des images fixes afin de vérifier les informations calculées par le CNN. Pour ce faire, le modèle a été chargé puis appelé afin de calculer le résultats sur différentes images, et le résultat sera symbolisé par une ligne affichant la direction à prendre.

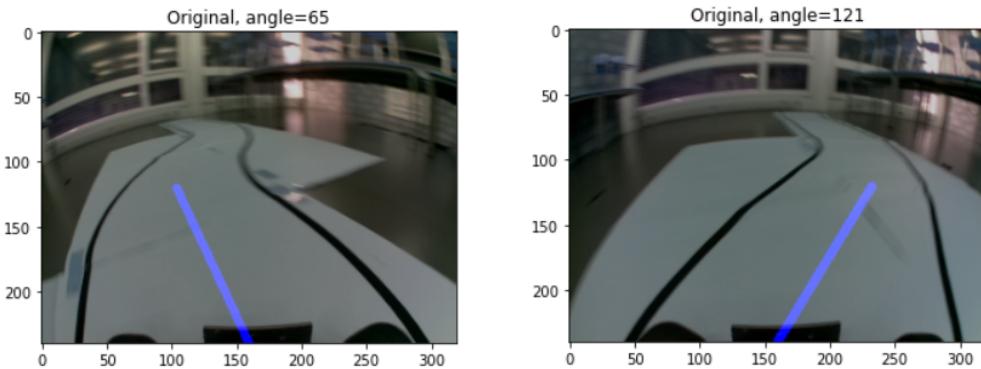


FIGURE 12.5 – Exemples de tests sur images fixes

## 12.6 Implémentation finale

L'implémentation finale est celle décrite dans la section 11.3, sous forme d'une classe simple implémentant plusieurs threads et affichant une prévisualisation de la direction prise par la voiture comme sur la figure 11.4. Les classes furent créées dans le but de pouvoir utiliser l'instruction *with* du python qui elle offre la possibilité de gérer un thread de manière automatique (création, lancement, exécution, fin). Pour réaliser ceci, les méthodes `__enter__` et `__exit__` ont été créées. Ces méthodes permettent l'implémentation d'objets utilisables avec l'instruction *with*. Ces méthodes sont automatiquement appelées par cette instruction au début et à la fin de son exécution.

## 13. Conclusion

### 13.1 Problèmes rencontrés

#### 13.1.1 Batterie et alimentation du RPi

Le premier problème rencontré lors de ce projet est celui de la batterie et de l'alimentation du RPi. Comme expliqué aux sections 5.1 et 5.2, la batterie était tout d'abord d'une autonomie trop faible, puis ensuite les pertes dues aux fils causaient des interruptions du RPi et rendaient impossible le lancement de scripts. Ces deux problèmes, principalement le deuxième se sont avérés très chronophages et ont freiné l'avancée initiale du projet. Cependant après avoir apportés les modifications, le fonctionnement était redevenu normal.

#### 13.1.2 PWM

L'implémentation la plus simple d'un signal PWM sur le RPi consistait à utiliser la librairie RPi.GPIO, cependant ceci créait un PWM software. C'est à dire que le raspberry se chargeait de manière logicielle de commuter une sortie en respectant la fréquence et le duty cycle précisé à l'aide de temps de pause par la fonction *time.sleep()*. Cependant, lorsque d'autres processus fonctionnaient sur le RPi, ces temps n'étaient pas parfaitement respectés, et la plage d'utilisation du signal pour les contrôleurs étant faible (cf. figure 4.1), ces variations de précisions causaient passablement de problèmes, dont des sauts de vitesse par exemple. Un PWM dit *Hardware* a donc été implémenté, utilisant une puce prévue à cet effet sur le raspberry à l'aide du tutoriel suivant : [26]. Cette méthode implique l'écriture de fichiers système du RPi en ligne de commande. Un script python permettant ceci fut implémenté lors du précédent travail.

Cependant, pour des raisons inconnues à l'heure actuelle, il arrive que le signal PWM provenant de la puce ne s'active pas ou même que le contrôleur moteur ne démarre pas. Le créateur du tutoriel [26] a été contacté mais aucune réponse n'a, à l'heure actuelle, été reçue.

#### 13.1.3 Récupération valeur de steering

La récupération de la valeur de steering entre le processus de collection des données et le processus de pilotage manuel a également posé quelques problèmes. En effet, il fallait être sur de récupérer au bon moment la bonne valeur et non avec un décalage entre l'image et la commande. Pour ce faire, un essai a été effectué utilisant des pipes afin qu'à chaque modification de la direction, celle-ci soit transmise au processus de sauvegarde. Cependant cette méthode fut rendue compliquée par le fait qu'un pipe fonctionne sur le principe d'une queue et donc il arrivait que plusieurs valeurs se retrouvent lues simultanément. De plus, il était compliqué de communiquer depuis le script de pilotage le signal de départ (différent d'une valeur de direction) ainsi que d'arrêt. Cela impliquait passablement de recherches sur le fonctionnement des pipes et des formats de variables en python (différents du C par exemple). Pour des raisons de temps, cette solution a donc été abandonnée.

Il a finalement été décidé de passer au thread de sauvegarde des données une référence de l'objet voiture piloté par le thread de pilotage manuel. Ainsi, les deux threads partageaient une ressource et chacun pouvait y accéder et ainsi le DataCollector pouvait récupérer à l'aide d'une variable dans le SteeringController en tous temps la valeur actuelle.

### 13.1.4 Fluidité du pilotage Manuel

Le pilotage manuel s'est également avéré plus compliqué que prévu pour différentes raisons : Tout d'abord, comme expliqué précédemment, la voiture est initialement conçue pour atteindre des vitesses élevées, le contrôle de celle-ci était relativement complexe via une manette connectée en bluetooth au RPi. Il a donc fallu adapter les limites de vitesse afin de rendre la vitesse plus basse et avoir un meilleur contrôle.

Ensuite, il fallait acquérir une certaine habitude pour le pilotage car il était mauvais pour l'entraînement de piloter par à-coup car cela créait des images avec, par exemple, une première partie de virage ou la direction enregistrée indiquait une ligne droite puis la suivante un virage très fort. Ces informations diminuaient également la performance du CNN car pour des images très similaires, la valeur de sortie était très différente.

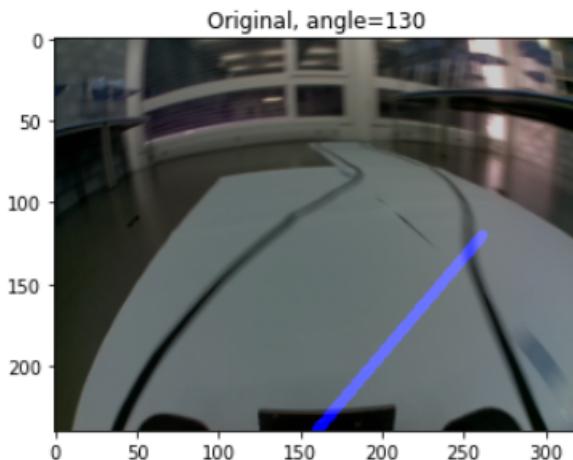


FIGURE 13.1 – Exemple d'image enregistrée avec un à-coup

La figure 13.1 ci-dessus est un exemple de pilotage avec des à-coups, et ce genre d'images perturbaient le bon entraînement du CNN.

### 13.1.5 Quantité de données d'entraînement

Cette section souligne le défaut de l'utilisation du CNN. C'est-à-dire la quantité de données d'entraînement. En effet, comme cité précédemment, les modèles de voitures autonomes sur le marché ou en phase d'élaboration combinent pour leur apprentissage des centaines d'heures de conduites, ce qui n'était pas possible dans notre cas. De ce fait, il arrivait parfois que le réseau soit très bien entraîné sur un type de virage (p.ex. Un virage large sur la gauche) mais éprouvait plus de difficultés à calculer la bonne direction dans le cas d'un autre virage potentiellement inconnu. A cause de ce problème, il a fallu enregistrer parfois plus de données concernant certains endroits de la route afin de permettre au CNN de les "reconnaître".

## 13.2 Possibilités d'amélioration

### 13.2.1 Acquisition d'images à l'aide d'un simulateur ou autre système de suivi de route

Une possibilité afin d'acquérir plus d'images en évitant par exemple le problème cité au point 13.1.4, serait d'utiliser un simulateur permettant de piloter et enregistrer des images. Le fait d'utiliser un environnement virtuel permettrait de réduire les défauts liés au matériel lors de la conduite. De tels simulateurs existent, c'est le cas de par exemple de **CARLA** [27] ou **NVIDIA DRIVE Sim** [28]. Ces solutions n'ont malheureusement pas pu être testées mais il serait intéressant de voir si elles peuvent contribuer à une amélioration de la qualité du réseau neuronal.

Une autre possibilité serait d'utiliser un système de suivi de lignes (comme par exemple un de ceux faisable à l'aide d'opencv), afin de parcourir toutes les images du dataset et ainsi définir à l'aide de ce système les valeurs d'angles en analysant les lignes.

### 13.2.2 Utilisation d'un autre SBC

Dans le cadre de ce projet il a été convenu de conserver le matériel initial, cependant il pourrait être intéressant de tester un SBC plus performant pour l'usage de CNN (comme par exemple le SBC de Nvidia, le Jetson Nano) qui comprend une GPU intégrée et est directement optimisé pour l'utilisation des CNN. Cela permettrait également de pouvoir utiliser le modèle entraîné sans devoir le quantifier et pourrait également probablement améliorer ses performances.

## Table des figures

4.1	Mesures PWM des actuateurs . . . . .	6
4.2	Comparaison des différents SBCs . . . . .	7
4.3	RaspBerry Pi 4B et Google Coral Accelerator . . . . .	8
4.4	RaspBerry Pi avec refroidisseur actif type Joy-It Block Active . . . . .	8
4.5	PiJuice HAT . . . . .	9
4.6	Dimensions de la route et couleur initiale. Source : [1] . . . . .	10
5.1	Modification du montage pour la nouvelle batterie . . . . .	11
5.2	Implantation du PiJuice HAT directement sur le RPi . . . . .	12
5.3	Aperçu du support confectionné . . . . .	13
5.4	Route avec couleur de lignes modifiée . . . . .	14
7.1	Schéma représentatif d'un réseau de neurones . . . . .	16
7.2	Exemple de régression à une variable [4] . . . . .	17
7.3	Exemple de convolution [5] . . . . .	18
7.4	Effet de la convolution [6] . . . . .	20
7.5	Fonctions d'activation [7] . . . . .	21
7.6	Figure exemple dying ReLu . . . . .	22
7.7	Modèle CNN Nvidia . . . . .	23
9.1	Pilotage manuel avec manette DualShock . . . . .	25
9.2	Schéma des valeurs d'angles de direction . . . . .	26
9.3	Contenu du dossier DataCollected . . . . .	27
9.4	Dossier SET8 et contenu du fichier Log . . . . .	27
9.5	Courbe gaussienne . . . . .	28
9.6	Exemple d'image floutée avec un masque de 5x5 . . . . .	29
9.7	Image avec luminosité diminuée de 30% . . . . .	29
9.8	Image inversée . . . . .	30
10.1	Lien entre la luminance et l'espace RGB [18] . . . . .	31
10.2	Image après le pré-traitement . . . . .	32
10.3	Déclaration d'un objet avec les paramètres du réseau de Nvidia . . . . .	33
10.4	Aperçu de la fonction de génération dynamique de données . . . . .	35
10.5	Aperçu de la réduction de volume après quantification . . . . .	36
11.1	Diagramme UML de la classe Car . . . . .	37
11.2	Diagramme UML de la classe DataCollector . . . . .	38
11.3	Diagramme UML de l'application . . . . .	39
11.4	Affichage généré par l'app principale . . . . .	40
11.5	Exemple d'une déclaration Yaml . . . . .	41
11.6	Exemple de lecture de fichier de configuration en Yaml . . . . .	41
12.1	Affichage du temps de traitement moyen à la fin du script . . . . .	42
12.2	Fonction de calcul du duty cycle (vitesse) en fonction de la direction . . . . .	43
12.3	Exemple d'image correcte sur la route . . . . .	44
12.4	Exemple d'image hors route . . . . .	44

---

12.5 Exemples de tests sur images fixes . . . . .	45
13.1 Exemple d'image enregistrée avec un à-coup . . . . .	47

## Bibliographie

- [1] Maxime CHARRIÈRE. *Travail de Bachelor, Voiture RC pilotée par IA.* 2020.
- [2] Pi SUPPLY. *PiJuice.* URL : <https://github.com/PiSupply/PiJuice>.
- [3] Aditya SHARMA. *Differences between Machine Learning and Deep Learning.* 2018. URL : <https://bit.ly/3ht9K4X>.
- [4] Gérard DREYFUS. *Réseaux de neurones et régression non linéaire.* 2021. URL : <https://www.universalis.fr/encyclopedie/reseaux-de-neurones-formels/4-reseaux-de-neurones-et-regression-non-lineaire/#V19N0008>.
- [5] Benjamin PERRET. *Convolution.* 2017. URL : <https://perso.esiee.fr/~perretb/I5FM/TAI/convolution/index.html>.
- [6] Dr. MERZOUGUI. *Chapitre 3 : Réseau de neurones convolutionel CNN.* Date inconnue. URL : [http://staff.univ-batna2.dz/sites/default/files/merzougui\\_ghalia/files/support\\_de\\_cours\\_-deep\\_learning-chapitre3-cnn.pdf](http://staff.univ-batna2.dz/sites/default/files/merzougui_ghalia/files/support_de_cours_-deep_learning-chapitre3-cnn.pdf).
- [7] Bastien MAURICE. *Fonction d'activation.* 2018. URL : <https://deeplearning.fr/cours-theoriques-deep-learning/fonction-dactivation/>.
- [8] KERAS. *What is the Dying ReLU problem in Neural Networks ?* 2020. URL : <https://androidkt.com/what-is-the-dying-relu-problem-in-neural-networks/>.
- [9] Mariusz Bojarski/Ben Firner/Beat Flepp/Larry Jackel/Urs Muller/Karol ZIEBA et Davide Del TESTA. *End-to-End Deep Learning for Self-Driving Cars.* 2016. URL : <https://developer.nvidia.com/blog/deep-learning-self-driving-cars/>.
- [10] Georgi VALKOV. *Python-evdev.* 2016. URL : <https://python-evdev.readthedocs.io/en/latest/>.
- [11] Python Software FOUNDATION. *asyncio — Asynchronous I/O.* 2021. URL : <https://docs.python.org/3/library/asyncio.html>.
- [12] GEO Magazine- LA REDACTION. *Qu'est-ce que le flou gaussien ?* 2017. URL : <https://www.geo.fr/voyage/photographie-conseils-pratiques-flou-gaussien-169306>.
- [13] The CARPENTRIES. *Blurring images.* 2007. URL : <https://datacarpentry.org/image-processing/06-blurring/>.
- [14] GEEKSFORGEEKS. *Python OpenCV / cv2.blur() method.* 2019. URL : <https://www.geeksforgeeks.org/python-opencv-cv2-blur-method/>.
- [15] Alexander JUNG. *imgaug.augmenters.* 2020. URL : [https://imgaug.readthedocs.io/en/latest/source/api\\_augmenters\\_arithmetics.html#imgaug.augmenters\\_arithmetics.Multiply](https://imgaug.readthedocs.io/en/latest/source/api_augmenters_arithmetics.html#imgaug.augmenters_arithmetics.Multiply).
- [16] GEEKSFORGEEKS. *Python OpenCV – cv2.flip() method.* 2020. URL : <https://www.geeksforgeeks.org/python-opencv-cv2-flip-method/>.
- [17] Claude GABRIEL. *Chapitre 4 : modèles et espaces colorimétriques matériels.* Date inconnue. URL : <https://bit.ly/3kHCXet>.
- [18] Alain Le DUFF. *Notions de colorimétrie.* 2004. URL : [http://popovmegamix.free.fr/Z/PROJET\\_SYNTHÈSE\\_TST\\_I2/noticeHTML/Documentation/CoursI2/1%20-%20Colorim%20trie.pdf](http://popovmegamix.free.fr/Z/PROJET_SYNTHÈSE_TST_I2/noticeHTML/Documentation/CoursI2/1%20-%20Colorim%20trie.pdf).

- 
- [19] Nitish Srivastava/Geoffrey Hinton/Alex Krizhevsky/Ilya Sutskever/Ruslan SALAKHUTDINOV. *Dropout : A Simple Way to Prevent Neural Networks from Overfitting.* 2014. URL : <https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf>.
  - [20] Diederik P. Kingma / Jimmy BA. *Adam : A Method for Stochastic Optimization.* 2014. URL : <https://arxiv.org/abs/1412.6980>.
  - [21] Ricco RAKOTOMALALA. *Descente de gradient.* Date inconnue. URL : [https://eric.univ-lyon2.fr/~ricco/cours/slides/gradient\\_descent.pdf](https://eric.univ-lyon2.fr/~ricco/cours/slides/gradient_descent.pdf).
  - [22] F. PEDREGOSA et al. « Scikit-learn : Machine Learning in Python ». In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830.
  - [23] TENSORFLOW. *Quantification post-formation.* 2021. URL : [https://www.tensorflow.org/lite/performance/post\\_training\\_quantization#full\\_integer\\_quantization\\_of\\_weights\\_and\\_activations](https://www.tensorflow.org/lite/performance/post_training_quantization#full_integer_quantization_of_weights_and_activations).
  - [24] Dave JONES. *8. API - The PiCamera Class.* 2013-2014. URL : [https://picamera.readthedocs.io/en/release-1.12/api\\_camera.html](https://picamera.readthedocs.io/en/release-1.12/api_camera.html).
  - [25] Kirill SIMONOV. *PyYAML 5.4.1.* Date inconnue. URL : <https://pypi.org/project/PyYAML/>.
  - [26] LLC JUMPNOW TECHNOLOGIES. *Using the Raspberry Pi hardware PWM timers.* 2017. URL : <https://jumpnowtek.com/rpi/Using-the-Raspberry-Pi-Hardware-PWM-timers.html>.
  - [27] Alexey DOSOVITSKIY et al. « CARLA : An Open Urban Driving Simulator ». In : *Proceedings of the 1st Annual Conference on Robot Learning.* 2017, p. 1-16.
  - [28] NVIDIA. *NVIDIA DRIVE Sim - Powered by Omniverse.* Date Inconnue. URL : <https://www.nvidia.com/en-us/self-driving-cars/simulation/>.

## Annexes

### Planning

### Fichier de configuration

### Codes

Pour ce qui est des codes, les scripts python se trouvent dans le dossier zip en annexe. S'y trouvent également les Jupyter Notebooks créés pour l'entraînement sur Google Colab. Ils sont également accessibles directement sur mon Github au lien suivant : <https://github.com/MacherelR/AutonomousRcCar>

### Dessin 3D du support