



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Hes-SO
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

TRAVAIL DE BACHELOR

VOITURE RC PILOTÉE PAR IA

Par

Maxime CHARRIÈRE

Professeur responsable

Pierre BRESSY

Département : Technologies industrielles (TIN)
Filière : Génie électrique
Orientation : Electronique et Automatisation industrielle (EAI)

PRÉAMBULE

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 24.07.2020

RÉSUMÉ

Find more on



github.com/maximecharriere/AutonomousRcCar

TABLE DES MATIÈRES

1. GLOSSAIRE.....	7
2. INTRODUCTION GÉNÉRALE	8
2.1. PLAN DU DOCUMENT	8
2.2. UNE VOITURE PILOTÉE PAR IA, C'EST QUOI ?.....	8
3. PRÉ-ÉTUDE.....	9
3.1. CHOIX DE LA VOITURE.....	9
3.1.1. <i>Caractéristiques techniques</i>	10
3.2. CHOIX DE L'ORDINATEUR DE BORD	11
3.2.1. <i>Calcul interne ou streaming</i>	11
3.2.2. <i>Choix du SBC</i>	11
3.2.3. <i>Refroidissement</i>	15
3.3. CHOIX DES CAPTEURS	16
3.3.1. <i>Détection de route</i>	16
3.3.2. <i>Détection de panneaux</i>	16
3.3.3. <i>Détection d'obstacles</i>	16
3.4. CHOIX DU SYSTÈME D'ALIMENTATION	18
3.4.1. <i>PiJuice</i>	18
3.4.2. <i>Servo et moteur</i>	19
3.5. CHOIX DU SYSTÈME DE COMMUNICATION.....	20
3.6. CHOIX DU LANGAGE DE PROGRAMMATION	20
3.7. DETECTION D'OBJET PAR ML	21
3.7.1. <i>Réseau neuronal convolutif (CNN)</i>	21
3.7.2. <i>MobilNet</i>	25
3.7.3. <i>Single Shot Detector (SSD)</i>	25
3.7.4. <i>Quantization</i>	26
4. CONCEPTION MÉCANIQUE	27
4.1. VOITURE.....	27
4.2. ROUTE	28
4.3. SIGNALISATION	29
5. CONCEPTION ÉLECTRONIQUE.....	30

5.1.	ELECTRONIQUE DE LA VOITURE	30
5.2.	ELECTRONIQUE DU FEU BICOLOR.....	32
6.	INSTALLATION RPI	33
7.	ARCHITECTURE LOGICIELLE	34
7.1.	LA CLASSE "CAR"	34
7.2.	L'APPLICATION.....	35
7.2.1.	<i>Un ou plusieurs thread?</i>	35
7.2.2.	<i>Gestion de la vitesse et du guidage</i>	36
7.2.3.	<i>Une application simple d'utilisation</i>	36
7.3.	LE SUIVIT DE ROUTE	36
7.4.	DÉTECTION DE SIGNALISATION	37
7.5.	FICHIER DE CONFIGURATION	37
8.	SUIVIT DE ROUTE	38
8.1.	CALIBRATION	39
8.1.1.	<i>Coefficients de distorsion</i>	39
8.1.2.	<i>Apprentissage de la calibration</i>	40
8.1.3.	<i>Rectification d'une image</i>	41
8.2.	WARPING	42
8.3.	BALANCE DE COULEUR	43
8.4.	ISOLEMENT DES LIGNES.....	43
8 . 5 .	REGROUPEMENT DES PIXELS EN LIGNE	45
8 . 6 .	DIRECTION DES LIGNES	46
8.7.	ANGLE DES ROUES	48
9.	DETECTION DE SIGNALISATION	50
9.1.	APPRENTISSAGE DU RESEAU NEURONAL	50
9.1.1.	<i>Création de notre dataset</i>	50
9.1.2.	<i>Ressources utilisées</i>	52
9.1.3.	<i>Apprentissage</i>	53
9.2.	TRAITEMENT DU RÉSULTAT ET ACTION SUR LA VOITURE	53
9.2.1.	<i>Distance de l'objet</i>	54
9.2.2.	<i>Couleur du feu</i>	55
10.	DETECTION D'OBSTACLES.....	56
10.1.	PROBLÈME RENCONTRÉ	56
11.	PROBLÈMES RENCONTRÉS	57
11.1.	PWM PAS CORRECTE.....	57
11.2.	LE RPi S'ARRÊTE.....	57
11.3.	MISE A JOUR DE L'OBJECT DETECTION API	57

12.	AMÉLIORATIONS	58
12.1.	PLAQUE DE SUPPORT DU RPI	58
12.2.	SUIVIT D'OBJET PAR IMAGE PROCESSING.....	58
12.3.	FAIRE TOURNER OPENCV SUR LA GPU	58
12.4.	CONDUITE DE NUIT	58
13.	PLANNING	59
14.	REMERCIEMENTS.....	60
15.	BIBLIOGRAPHIE.....	61
16.	NON-PLAGIAT	63
17.	ANNEXES	64
17.1.	ÉNONCÉ DU TB	64
17.2.	CLAUSE DE CONFIDENTIALITÉ	64
17.3.	FICHIER DE CONFIGURATION	64
17.4.	SSD MOBILENET V2 LEARNING NOTEBOOK	64

1. GLOSSAIRE

TB	Travail de Bachelor
RPi	Raspberry Pi
SBC	Single-board computer (ici l'ordinateur de bord)
RC	Radio commandé
PWM	Pulse Width Modulation
BT	BlueTooth
ML	Machine Learning
CPU	Central Processing Unit
GPU	Graphic Processing Unit
NPU	Network Processing Unit
TPU	Tensor Processing Unit
CNN	Convolutional Neural Networks (Réseau neuronal convolutif)
SSD	Single Shot Detector

2. INTRODUCTION GÉNÉRALE

2.1. Plan du document

Dans un premier temps les **choix de matériel et langages** utilisés seront discuté dans le pré-étude. Une explication théorique sur le **fonctionnement du réseau neuronal** utilisé pour la détection de panneaux de signalisations sera aussi donnée dans cette partie.

Nous passerons ensuite sur la **conception mécanique, électronique** ainsi que l'**architecture logicielle** de ce projet.

Puis le fonctionnement des trois algorithmes principaux consistant au **suivit de route**, la **détection de panneaux** et la **détection d'obstacle** sera développé.

Pour finir, les **problèmes rencontrés** ainsi que les **améliorations à apporter** seront présentés.

2.2. Une voiture pilotée par IA, c'est quoi ?

Doit être indépendant d'aide externe -> pas de GPS

3. PRÉ-ÉTUDE

3.1. Choix de la voiture

Malgré le fait que le choix d'une voiture RC soit spécifié dans l'énoncé du TB (17.1), une étude des choix possibles a été effectué.

Il est possible :

- De construire sa propre voiture miniature
- D'utiliser une vraie voiture
- D'utiliser un karting
- D'acheter une voiture RC déjà fonctionnelle

Compte tenu des raisons évidentes de sécurité et de moyen, l'utilisation d'une vraie voiture ou d'un karting est impossible.

Il serait idéal de confectionner soi-même la voiture afin de maîtriser les caractéristiques de chaque composant, mais le TB ne portant pas sur ce sujet et une telle conception prenant beaucoup de temps, il a été décidé d'acquérir une voiture déjà montée.

Afin de choisir une voiture RC, les points suivants ont été pris en compte :

- Prix (-300fr)
- Conception simple à comprendre et modifier (pour commander le guidage et le moteur par un SBC et non par radiofréquences)
- Batterie (autonomie et pouvoir éventuellement alimenter le SBC) / recharge facile
- Taille (pour y installer batterie, SBC et capteurs)

Après recherche de ce qu'il existait sur le marché, il a été choisi d'acheter la voiture :

- T2M Mad Pirate Brushless 1/10e



Figure 1 T2M Mad Pirate Brushless 1/10e

3.1.1. CARACTÉRISTIQUES TECHNIQUES

Il a été primordial de pouvoir **guider la voiture simplement**, et sur ce type de voiture l'accélération et le guidage se fait grâce à un simple PWM.

De base, la manette envoie à la voiture la position de ses potentiomètre de direction et gaz par onde radio, et un récepteur radio se trouvant dans la voiture convertit ces informations en PWM qui commande un servo-moteur pour la direction et un contrôleur de moteur brushless pour l'accélération.

Après plusieurs mesures effectuées à l'oscilloscope sur les sorties PWM du récepteur radio, les **caractéristiques suivantes ont été montrées** :

	Freq	Duty cycle		
		Low	Middle	High
Steering	50Hz	1.2ms (Left) (6%)	1.5ms (7.5%)	1.8ms (Right) (9%)
Speed	50Hz	1ms (Back) (5%)	1.5ms (7.5%)	2ms (Forward) (10%)

Tableau 1 Mesures PWM des actuateurs

La voiture est alimentée par une batterie :

- NiMh 7.2V 1800mAh

Arrêt d'urgence de la voiture

Le contrôleur du moteur a une fonction spéciale d'arrêt d'urgence de la voiture.

Effectivement, si le duty-cycle correspondant à la position arrêté de la voiture (7.5%) est appliqué, le moteur ne sera simplement plus alimenté et la voiture continuera à avancer grâce à son énergie synectique.

Le contrôleur du moteur comporte donc un système de frein. Ce frein s'active si un duty-cycle plus petit que la position neutre (ici < 7.5%) est directement appliquée alors que la voiture avançait.

Alors que pour aller en marche arrière, une pause (temps non spécifié) doit être fait en position neutre avant d'appliquer le duty-cycle désiré servant à reculer. (1)

3.2. Choix de l'ordinateur de bord

Etant donné que le but du projet est d'agir sur le système de guidage de la voiture en fonction de son environnement, il faut que la voiture comporte des capteurs, des actuateurs ainsi qu'un ordinateur de bord (SBC) afin de prendre les décisions.

3.2.1. CALCUL INTERNE OU STREAMING

Il est possible d'effectuer le traitement des informations et la prise de décision directement avec le SBC se trouvant dans la voiture, mais une autre solution très intéressante doit être prise en compte.

Nous pourrions uniquement acquérir les données des différents capteurs sur la voiture, puis envoyer par onde électromagnétique (WiFi / Radio / Bluetooth) ses informations à une machine externe plus puissante, qui exécutera les différents algorithmes et renverra uniquement l'angle des roues et la vitesse à appliquer à la voiture.

Grace à cette technique dite « de streaming », le SBC à bord de la voiture pourra être très rudimentaire étant donné qu'il ne devra seulement lire et écrire les entrées/sorties.

L'avantage est qu'un tel SBC consommera très peu (autonomie de la voiture accru), et la puissance de calcul de la machine externe peut être très grande et est facilement adaptable.

De plus, la rapidité de transfert par onde est de nos jours bien assez grande pour transmettre les quelques informations et images que nous aurons à traiter.

Cependant, un tel système signifie que la voiture ne peut pas fonctionner sans être connecté à la machine externe, ce qui ne la rend plus intégralement autonome. De plus une perte de connexion peut vite arriver. Cette solution n'a donc pas été retenue.

3.2.2. CHOIX DU SBC

Le SBC doit donc pouvoir gérer la lecture des différents capteurs, commander les actionneurs et exécuter l'intégralité du code de traitement. Il serait préférable que celui-ci ait aussi une carte réseau afin de pouvoir le programmer à distance sans devoir si connecter par câble.

Les points suivants ont donc été pris en compte dans le choix du SBC :

- Puissance de calculs (CPU / GPU / NPU)
- Facilité d'utilisation
- Communauté l'utilisant
- Prix
- Consommation électrique
- Entrée / sorties (PWM hardware, gestion camera)
- Temps de démarrage
- Connection (WiFi / Bluetooth)

CPU, GPU, NPU. Quelle différence ?

Les opérations d'exécution d'un algorithme sont effectuées dans l'unité de traitement de l'ordinateur, qui est généralement une CPU. Cependant, certains types de calculs ne sont pas adaptés à l'architecture d'une CPU.

Effectivement, une CPU ne pourra exécuter en parallèle qu'un nombre de calcul correspondant à son nombre de cœur, allant généralement de 2 à 8.

Ce type d'architecture est excellente pour exécuter des opérations séquentielles, car la fréquence des CPU peut monter très haut, cependant dans le cas de traitement d'images ou d'exécution d'un réseau de neurone, une GPU ou NPU est plus efficace.

Ces domaines font appels à des millions d'opérations simples pouvant s'exécuter en parallèle, et l'architecture des GPU et NPU est justement optimisé afin de faire du parallélisme.

Les NPU ont encore l'avantage d'accélérer l'usage des réseaux de neurones en implémentant directement dans ses circuits certaines fonctions normalement réalisées par du code machine.

Table comparative

- Pour plus d'infos sur les comparatifs des SBC, voir (2) et (3)

	Positif	Négatif
Raspberry Pi 4B 4Gb	 <p>63fr 2 PWM WiFi / BT Camera Forte communauté</p>	<p>Pas de GPU / NPU pour du ML Pas de refroidisseur</p>
Arduino Mega	 <p>38fr 15 PWM Forte communauté Faible consommation</p>	<p>Pas un processeur mais un microcontrôleur Pas de WiFi / BT</p>
NVIDIA Jetson Nano	 <p>Comporte une GPU Camera</p>	<p>159fr 1 PWM WiFi / BT en option</p>
Google Coral Dev Board	 <p>Comporte une NPU 3 PWM WiFi / BT Camera</p>	<p>153fr</p>
Rock Pi N10	 <p>Comporte une NPU 2 PWM Camera</p>	<p>148fr Pas de WiFi / BT</p>

Tableau 2 Compartif des SBC

L'Arduino a tout de suite été retiré car le multithreading y est impossible, la puissance de calcul est très faible et il y est impossible d'y coder en Python (pour le choix du Python, voir. 3.6).

La carte Nvidia et Rock Pi ont été retiré par leur manque de PWM et de WiFi/BT. Il serait totalement possible d'y ajouter des modules externes afin de remédier à ce problème, mais il était préférable d'avoir tout intégré à la carte de base afin d'éviter d'éventuels problèmes. De plus très peu de tutoriels et support concernant ces cartes ont été trouvé sur internet.

Le problème du RPi est qu'il n'est pas fait pour faire tourner des réseaux de neurones.

Cependant une NPU externe, la « **Google Coral Accelerator** » qui est largement utilisé par la communauté peut résoudre ce problème.

La puce présente dans cet accélérateur est la « Edge TPU » et est la même que celle se trouvant dans le « Google Coral Dev Board » vu dans le précédent comparatif.

Note : Une TPU est la même chose qu'une NPU, mais est le nom donné par Google

Le benchmark suivant montre bien le gain qu'apporte cet accélérateur lorsqu'il est connecté au RPi.

Note : Le benchmark a été effectué sur le réseau de neurone MobilNet V2 utilisant des images de 300x300 pixels (voir 3.7.2)

Board	Time (ms)
Raspberry Pi 3B+	654,0
Raspberry Pi 4B	483,5
NVIDIA Jetson Nano	309,3
NVIDIA Jetson Nano (optimisation GPU)	72,3
Google Coral dev board	20,9
Raspberry Pi 4B + Google Coral Accelerator (USB2)	102,3
Raspberry Pi 4B + Google Coral Accelerator (USB3)	18,2

Tableau 3 SBC Benchmarking for Machine Learning

Source : (4)

Le choix entre le « Google Coral dev board » et le duo « Raspberry Pi 4B » + « Google Coral Accelerator » a été fait en **prenant en compte la communauté très présente du coté du RPi**.

Les différents problèmes auxquels nous pouvons faire face avec le RPi ont pour la plupart déjà été discuté sur des forums, et la majorité des tutoriels sont fait sur un RPi.



Figure 2 RPi et Coral Accelerator

La vitesse de classification avec le Coral Accelerator a été personnellement testé, une fois via le port USB3 puis USB2.

```
pi@raspberrypi:~/Documents/AutonomousRcCar/ExemplesExternes/coral/tflite/python/examples/classification $ python3 classify_image.py --input images/parrot.jpg
---INFERENCE TIME---
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
20.3ms
2.9ms
3.0ms
3.1ms
3.1ms
-----RESULTS-----
Ara macao (Scarlet Macaw): 0.76562
pi@raspberrypi:~/Documents/AutonomousRcCar/ExemplesExternes/coral/tflite/python/examples/classification $ python3 classify_image.py --input images/parrot.jpg
---INFERENCE TIME---
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
118.2ms
9.9ms
10.2ms
9.9ms
9.9ms
-----RESULTS-----
Ara macao (Scarlet Macaw): 0.76562
```

Figure 3 Vitesse de classification USB2 vs USB3

Exactement le même code a été exécuté et nous voyons que la classification prend ~3ms via USB3 contre ~10ms avec le USB2 (la classification est effectué 5x afin de pouvoir voir si ce temps est constant),

Il m'a été conseillé de cependant faire attention au temps de démarrage de certains SBC, et le RPi met environ de 7 à 25s à démarrer, ce qui est acceptable. (5)

3.2.3. REFROIDISSEMENT

Le RPi ne comporte aucun système de refroidissement, ce qui peut poser problème lors d'une utilisation intensive et prolongée du processeur. De plus, comme nous le verrons au point 3.4.1 la batterie est situé juste en dessus du processeur. Par sécurité, un système de refroidissement actif comportant deux ventilateurs a donc été ajouté.



Figure 4 Joy-It Block Active

3.3. Choix des capteurs

La voiture doit répondre à trois problématiques. Chacune a des besoins spécifiques afin d'être résolus. Il sera discuté ici des différents capteurs utilisables pour chaque problématique.

3.3.1. DÉTECTION DE ROUTE

Afin de détecter la route, deux technologies ont été retenus :

- Une caméra
- Un lidar

Le lidar envoie un laser et mesure le temps que le rayon met à revenir afin de mesurer des distances, dans le but par exemple de constituer un nuage de point représentant son environnement 3D.

Dans notre cas où des lignes peintes doivent être détectées, nous pourrions penser qu'un tel capteur n'est pas utilisable.

Cependant, certains lidars envoient des rayons sur plusieurs longueurs d'ondes (plusieurs couleurs) et mesure l'intensité lumineuse du rebond afin de déterminer les propriétés lumineuses de la surface touchée. (6)

Par contre, un tel système est très lourd et onéreux. **La caméra sera donc retenue.**

3.3.2. DÉTECTION DE PANNEAUX

Ici, uniquement l'utilisation d'une **caméra** a été trouvée.

Il serait possible d'intégrer des puces RFID dans les panneaux et de détecter que l'on s'en approche, mais ceci signifierait que l'environnement externe doit s'adapter à la voiture, alors que c'est à la voiture autonome de s'adapter intégralement à son environnement.

3.3.3. DÉTECTION D'OBSTACLES

De nombreuses technologies peuvent être utilisées dans ce but, comme :

- Une **camera**
 - Doit utiliser un algorithme d'analyse d'image, très gourmand en puissance de calcul et en temps
- Un **laser** (modèle considéré : *VL53L0X*)
 - Distance : 3 – 120cm
 - Résolution : 1mm
 - Angle : <3°
- Des **infrarouge** (modèle considéré : *TF Mini LiDAR*)
 - Distance : 30 – 1200cm
 - Résolution : 1cm
 - Angle : 2.3°
- Des **ultrasons** (modèle considéré : *HC-SR04*)
 - Distance : 2 – 400cm
 - Résolution : 3mm
 - Angle : 30°
 - Peut être perturbé par le bruit ambiant

- Un **lidar 360°** (modèle considéré : *RPLIDAR A1*)
 - Permet de détecter des obstacles sur 360°
 - Permet de connaître précisément la position de l'obstacle, afin de par exemple le contourner
 - Déetecte uniquement sur un plan (pas en 3D)
 - Consomme beaucoup de courant
 - Information compliquée à traiter

Etant donné que les obstacles ne sont pas forcément exactement devant le capteur mais doivent être détecté sur toute la largeur de la voiture, l'angle de détection est important.

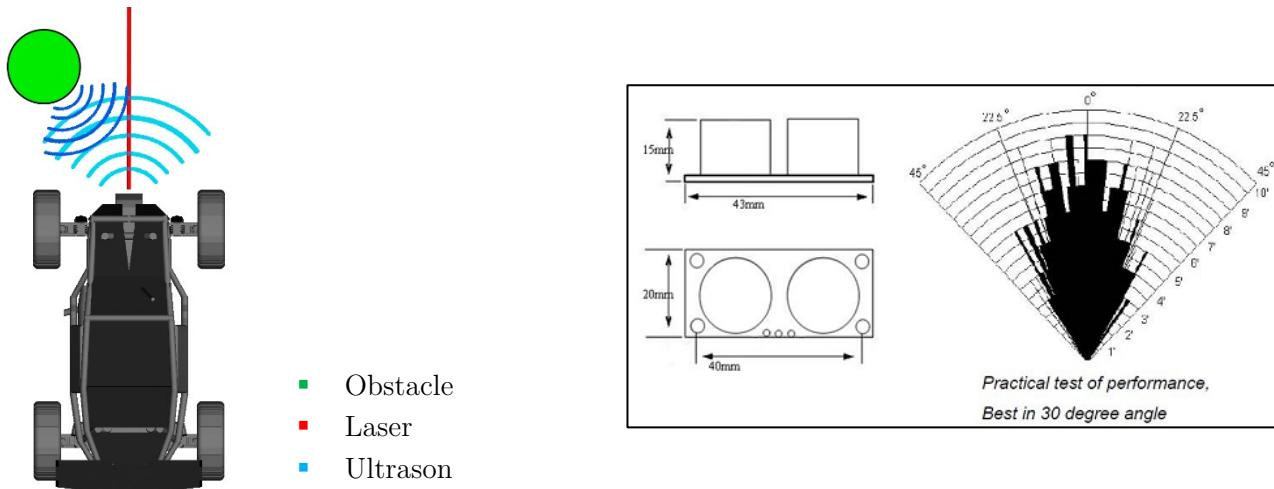


Figure 5 Angle de détection laser / ultrason

C'est pour ceci que le **capteur à ultrason a été choisi**. De plus les objets se trouvant de 5 à 200cm doivent être détecté, ce qui correspond à la distance de détection du capteur.

Choix de la camera

3.4. Choix du système d'alimentation

La voiture ne doit pas être connecté à une quelconque source d'alimentation externe. Il faut donc la faire fonctionner sur batteries.

Les composants devant être alimenté sont les suivants et avec la tension spécifiée :

- Le contrôleur du moteur brushless (1)
 - Batterie NiMh 5 à 9 cellules (6 - 10.8v)
 - Batterie LiPo 2 à 3 cellules (7.2 - 11.1v)
- Le servo-moteur de direction
 - 4 – 6V
- Le RPi (celui-ci alimentera ensuite directement la caméra, le capteur à ultrason, le Coral Accelerator ainsi que les ventilateurs de refroidissement)
 - 5V régulé

Le contrôleur du moteur fourni également une tension régulée à 6V.

3.4.1. PIJUICE

Nous pourrions imaginer utiliser uniquement la batterie de la voiture afin de tout alimenter, en utilisant par exemple un régulateur de tension à 5V pour alimenter le RPi.

Il a cependant été préféré d'utiliser un système d'alimentation dédié et spécialement conçu pour le RPi, le **PiJuice HAT** (7) pour les raisons suivantes :

- Pouvoir rendre le PRi indépendant du reste de la voiture afin de le déplacer plus facilement durant la phase de conception
- Pouvoir utiliser le RPi tout en rechargeant les batteries (la batterie de la voiture doit être déconnecté afin de pouvoir la recharger)
- Pouvoir superviser l'état de la batterie depuis le RPi
- Pouvoir déconnecter le câble d'alimentation et passer sur la batterie sans interruption
- Pouvoir arrêter le RPi de façon contrôler et automatique lorsque le niveau de batterie est faible



Figure 6 PiJuice HAT

3.4.2. SERVO ET MOTEUR

Le servo a tout d'abord été connecté sur le RPi car celui-ci consomme environ 1A et le PiJuice peut théoriquement fournir jusqu'à 2.5A (valeur non atteinte, voir 11.4). Cependant lors d'utilisation intensive du servo, le RPi s'arrêtait, étant sous-alimenté. Le servo a alors été connecté à la sortie VSYS du PiJuice. Cette sortie fournit la puissance restante, garantissant que le minimum nécessaire au bon fonctionnement du RPi soit fourni. Mais dans ce cas le servo fonctionnait moins bien et la batterie se vidait vite.

L'alimentation du servo a alors été fourni par la batterie de la voiture via le contrôleur du moteur fournissant une tension de 6V.

- Schéma électronique détaillé au point 4.

3.5. Choix du système de communication

Etant donné que la voiture doit être autonome, elle ne devrait pas avoir besoin de communiquer avec l'extérieur. Cependant afin de pouvoir programmer la voiture, la débuguer, visualiser ce qu'elle fait et voit, remonter des informations sur cette dernière et la commander manuellement à distance, il est important d'avoir un système de communication sans fil à bord de la voiture.

Comme nous pourrons le voir dans le planning (chapitre 13), il a tout d'abord été prévu de mettre en place un système de communication par radiofréquence entre le SBC de la voiture et l'ordinateur (le choix du RPi n'était à ce moment pas encore fait).

Un tel système demande une installation électronique conséquente sur la voiture et sur l'ordinateur, ainsi que la mise en place d'un protocole de communication afin de transférer des informations.

Après avoir fait le choix du RPi, le transfert d'information par protocole TCP/IP (internet) entre la voiture et l'ordinateur est devenu évidente. La connexion peut se faire par câble ou par WiFi, partout dans le monde et est rapide. Toute l'électronique nécessaire à son fonctionnement est déjà intégrée dans le RPi et nos ordinateurs. De plus, tous les outils nécessaires afin de gérer le RPi à distance existent déjà :

- Shell à distance avec le protocole SSH
- Bureau à distance avec VNC
- Gestionnaire de fichier distant avec Samba
- Codage / déboggage à distance avec Visual Studio Code

Note : Différentes installations détaillées au point 6

Il a été décidé de pouvoir gérer la voiture manuellement si besoin. Une connexion à une manette de jeu a donc aussi été mis en place par Bluetooth.

3.6. Choix du langage de programmation

Le **Python** a été choisi pour ce projet malgré le fait que mes connaissances en C / C++ / C# soient beaucoup plus grandes (33 crédits dédiés à ces langages, contre 0 pour le Python).

Ce choix a été pris pour les raisons suivantes :

- La quasi-totalité de la communauté RPi code en Python, donc les tutoriels, forums et librairies sont en Python
- Le Python permet une gestion simple des matrices (notamment utile pour les images)
- Les principales librairies de deep learning sont en Python
- Il m'intéressait d'apprendre ce langage

Il a donc fallu apprendre tout le fonctionnement de ce langage. Entre les packages, la gestions des classes et ce qui en découle (héritage, propriété, variable publique & privé, etc.), les threads, les allocations, les évènements, ou encore les fonctions et opérateurs de base du langage.

Bien que beaucoup de principes ressemblent au C++, le langage reste bien différent et avec ses particularités.

3.7. Détection d'objet par ML

La détection de panneaux se fera grâce à du deep learning et plus précisément avec un réseau neuronal convolutif.

Ce chapitre expliquera le principe d'un CNN, puis l'architecture du classifieur MobilNet utilisé dans ce projet et pour finir le fonctionnement d'un SSD, réseau trouvant des objets dans une image.

3.7.1. RÉSEAU NEURONAL CONVOLUTIF (CNN)

Le réseau neuronal convolutif est un type de réseau de neurone utilisé pour le traitement d'image et qui est indispensable dans ce domaine.

Si nous effectuons par exemple la classification d'objets se basant uniquement sur la valeur de chaque pixel de l'image (sans pré-traitement que nous verrons plus loin), nous serons confrontés à des problèmes si l'objet n'est pas au centre de l'image, si l'image comporte un arrière-plan ou si la couleur change. Un tel procédé fonctionne pour une base de données comme Fashion-MNIST (8) (à gauche) où tout est en niveau de gris, centré et dans la même position, par contre ne fonctionnera jamais sur l'image de droite, plus complexe.

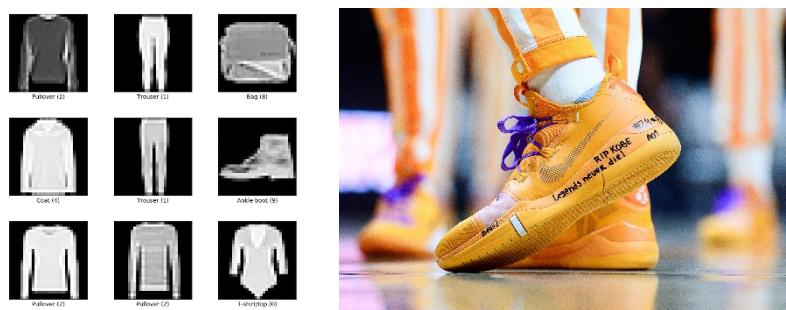


Figure 7 Image simplifiée VS cas réel

Il faut donc extraire des informations de l'image ayant plus de sens avant de pouvoir effectuer une classification. **C'est l'apprentissage de features.**

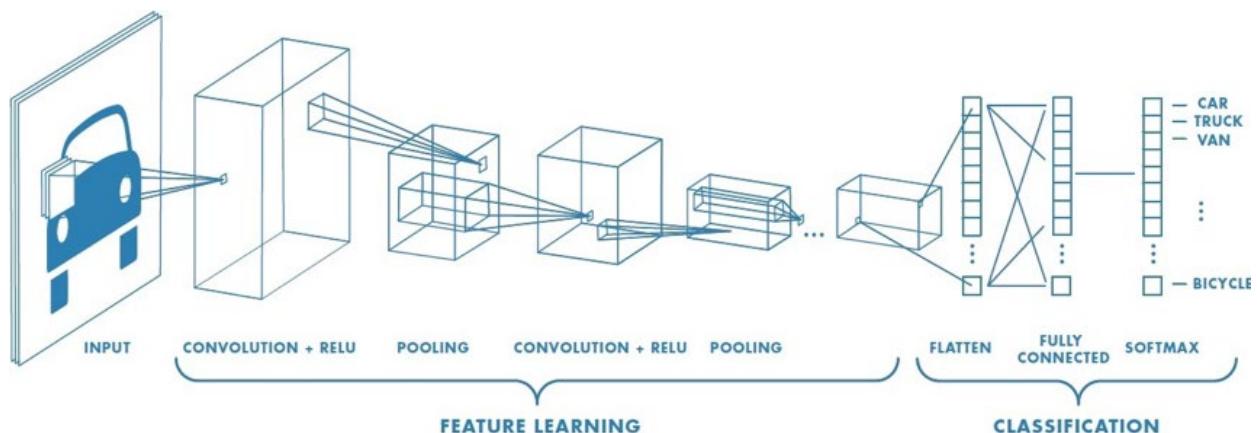


Figure 8 Etapes d'un CNN Source et explications : (9)

Nous voyons que l'apprentissage de features peut contenir plusieurs parties. La **convolution**, le **ReLU**, le **pooling** et aussi le **batch normalization** qui n'est pas monté ici.

3.7.1.1. La convolution standard

Le principe de la convolution est de faire un balayage sur chaque pixel de l'image avec un masque, en effectuant un produit scalaire entre le masque et les pixels avoisinant le pixel analysé, afin d'obtenir une nouvelle valeur pour ce pixel qui servira à créer une nouvelle image (voir (10)).

Ainsi par exemple pour trouver les fortes différences de valeurs de pixels sur l'axe vertical (dérivée numérique), et ainsi trouver les contours d'un objet, le masque suivant peut être utilisé :

Note : Le pixel vert est le pixel analysé

$$\text{pixels} \bullet \text{masque} = \begin{bmatrix} 0 & 64 & 128 \\ 48 & 192 & 144 \\ 142 & 226 & 168 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \\ 0 * 1 + 2 * 64 + 1 * 128 + 0 * 48 + 0 * 192 + 0 * 144 - 1 * 142 - 2 * 226 - 1 * 168 = \boxed{-506}$$

(Nous obtenons ici une valeur négative car le résultat est négatif si nous passons du foncé au clair et positif inversement. Généralement la valeur absolue est prise afin d'afficher les deux bordures.)

En combinant la dérivée verticale et horizontale, nous pouvons obtenir un tel résultat



Figure 9 Détection de contours Source : (10)

Ceci est un seul exemple de filtre. Mais il est possible de rechercher des formes, des lignes directrices et bien d'autres caractéristiques qui aideront à la classification.

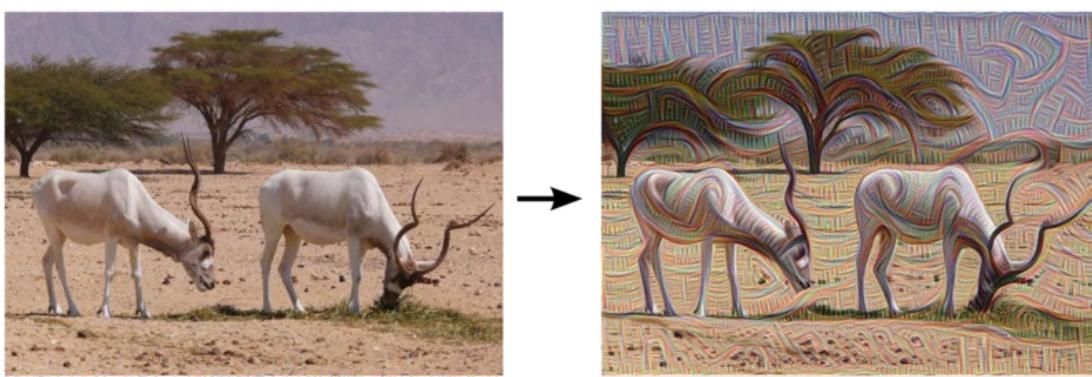


Figure 10 Traitement d'une image par le filtre DeepDream

Maintenant, étant donné que nous sommes dans le domaine du deep learning, les valeurs numériques des filtres n'ont pas été choisi manuellement. Elles ont été initialisées aléatoirement puis entraînées afin d'effectuer un traitement sur l'image qui permettra de faire la meilleure prédiction. Il est donc impossible de dire que tel couche du réseau de neurone sert par exemple à trouver les contours.

La convolution, traitement trop gourmand en ressources ?

Les multiplications sont des opérations demandant passablement de ressources afin d'être calculées, et le fait d'effectuer des convolutions demande le calculs d'énormément de multiplications.

Un rapide calcul nous montre le nombre de multiplications à réaliser afin de faire des convolutions :

$$\begin{aligned} D_K^2 &: \text{taille du kernel carré} \\ M &: \text{profondeur de l'image (3 si image RGB)} \\ D_F^2 &: \text{taille de l'image de } D_G * D_G \text{ pixels} \\ N &: \text{nbr de convolution voulu} \end{aligned}$$

- Multiplications par pixel : $D_K^2 * M$
- Multiplications pour toute l'images : $D_K^2 * M * D_F^2$
- Multiplications pour N convolutions : $D_K^2 * M * D_F^2 * N$

Dans le cas du CNN MobilNet qui sera choisi dans notre cas (voir 0), l'utilisation de la convolution standard demanderait **462 Million de multiplications** (11) pour traiter une image.

Ce nombre a pu être réduit en utilisant la Depthwise Separable Convolution.

3.7.1.2. Depthwise Separate Convolution

Note : Pour plus d'explications avec animations 3D, voir (11) et (9)

Avec une convolution normale l'application du filtre se fait en une fois sur toutes les couches de l'image avec un kernel spécifique à ce filtre. Pour chaque nouveau filtre toute la convolution est à refaire.

Lors d'une Depthwise Separable Convolutions, l'opération est séparée en 2 étapes :

- Depthwise Convolution
- Pointwise convolution

Depthwise Convolution

La première étape consiste à appliquer un kernel différents à chaque couche de l'image. Nous obtenons donc une nouvelle image prétraitée de même profondeur que l'image initiale (3 pour une image RGB). Pour rappel lors d'une convolution normale toutes les couches sont traitées en même temps affin d'en ressortir un scalaire.

Pour l'instant le nombre de multiplication est le même que lors d'une convolution normale :

- Multiplications pour toute l'images : $D_K^2 * D_F^2 * M$

Pointwise convolution

L'optimisation de calcul vient de cette étape. Ici, un kernel de 1x1xM vient regrouper les M couches précédentes.

Nous pourrions penser que de ce fait le nombre de multiplication totale est plus grande lors d'une depthwise separate convolution étant donné qu'elle vaut

- Multiplications pour toute l'images : $D_K^2 * D_F^2 * M + D_F^2 * M$

Cependant pour chaque nouveau filtre, uniquement la seconde partie est répétée. Ce qui nous fait :

- Multiplications pour N convolutions : $D_K^2 * D_F^2 * M + D_F^2 * M * N$

L'efficacité d'optimisation de cette convolution augmente donc proportionnellement au nombre de filtre utilisé.

Le rapport entre une convolution normal et optimisé vaut : $\frac{1}{N} + \frac{1}{D_K^2}$

Le réseau MobilNet utilise ainsi plus que **52.3 Million de multiplication** (contre 462M avant).

3.7.1.3. Batch Normalization

Toutes données en entrée d'un réseau neuronal doivent être normalisé afin d'être sur la même échelle. Par exemple si un réseau prend en entrée l'âge d'une personne et sa taille, une valeur ira environ de 0 à 100 ans alors que l'autre de 0.4 à 2 mètres. Ces différences d'échelle vont diminuer les performances du réseau.

Un tel procédé peut aussi être effectué entre deux couches internes du réseau car certains neurones peuvent produire un résultat disproportionné par rapport aux autres neurones, et il faut normaliser ces résultats avant de les passer aux neurones suivants. Ce principe s'appelle justement batch normalization.

3.7.1.4. ReLU

Un ReLU (Rectified Linear Unit) est une fonction d'activation pour un neurone qui suit la règle suivante :

$$f(x) = \max(0, x)$$

Ce qui signifie que le neurone prend en entrée la valeur qui lui est passé, à condition que celle-ci soit positive. Sinon le neurone n'est pas activé.

3.7.1.5. Pooling

Le pooling, ou « mise en commun » est un principe servant à réduire la taille d'une image et peu très bien être effectué au milieu du CNN, par exemple après l'application d'un filtre.

L'image est découpée en carré de $n*n$ pixels puis une fonction de mise en commun est appliquée à chaque carré. Par exemple la fonction « Max-Pool 2x2 » va prendre des carrés de 2x2 et en garder la valeur max. Ainsi l'image est réduite d'un facteur 4.

3.7.2. MOBILNET

➔ Article complet sur MobilNet: voir (11)

MobilNet est une architecture de CNN spécialement conçu pour demander peu de ressources afin de pouvoir tourner en temps réel sur des appareils mobiles.

Pour comparaison, le réseau VGG16 comporte environ 150M de paramètres à entraîner, contre 5M pour le MobilNet (12).

Son architecture est la suivante :

1. 32 convolutions normales sont effectuées sur les trois canaux de l'image d'entrée avec un kernel de 3*3.

Donc un filtre de taille 3*3*3*32

2. Un enchaînement de depthwise separate convolution. Les deux étapes vues précédemment sont affiché ici de façon distincte.

La depthwise convolution est noté « Conv dw » et utilise un kernel de 3*3

La pointwise convolution est noté « Conv » et utilise un kernel de 1*1

3. Un pooling moyenneur sur des carrés de 7*7
4. La couche full connected et le softmax servant à la classification

Il faut noter qu'après chaque couche convulsive, une batch normalization (3.7.1.3) et un ReLU (3.7.1.4) est effectué avant de passer à la prochaine convolution.

3.7.3. SINGLE SHOT DETECTOR (SSD)

Cependant, le réseau MobilNet sert à classifier une image entière, mais dans notre cas nous pouvons avoir plusieurs objets dans un même image. C'est ici que le Single Shot Detector vient résoudre ce problème.

Le SSD est un réseau neuronal servant à déterminer les régions où des objets peuvent se trouver. Il vient se placer avant le réseau MobilNet afin de fournir à ce dernier pas l'entièreté de l'image, mais la partie contenant un objet.

Nous pourrions très bien balayer l'entièreté de l'image avec des rectangles de différentes tailles et formes, et pour chaque rectangle chercher si un objet si trouve avec le réseau MobiINet. Par contre un tel procédé demanderait d'analyser énormément de régions, et notamment des régions où il n'y a rien.

Le réseau SSD va découper l'image en carré, et va prédire pour chaque carré un certain nombre de box possible en se basant sur le contenu des pixels de ce carré. Il aura préalablement appris le lien entre les pixels et les box.

Dans l'exemple suivant, le carré rouge est analysé. La forme de l'anse a été repérée et le réseau peut donc proposer ce qui devrait être la position de la tasse.

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Tableau 4 Architecture MobilNet

Source: (14)

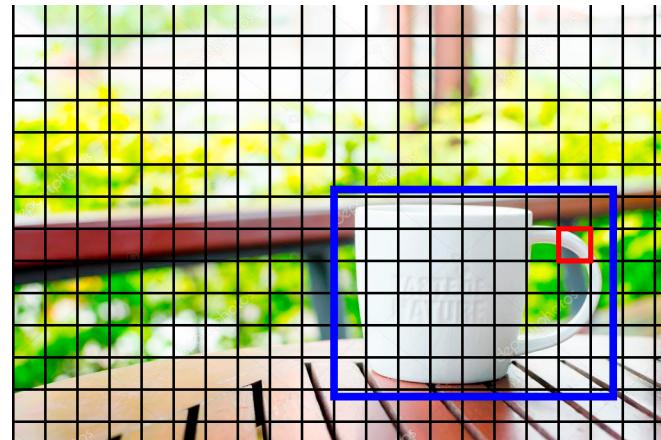


Figure 11 Principe du SSD

En combinant le résultat de tous les carrés et en prenant les résultats qui ressortent le plus souvent, il est possible de précisément prédire la position d'un objet.

Note : Il existe d'autre réseau de détection comme RCNN, Fast RCNN, Faster RCNN ou YOLO qui sont expliqué ici : (13)

3.7.4. QUANTIZATION

Les réseaux neuronaux utilisent généralement des nombres à virgule flottante codé sur 32 bits pour représenter les poids de chaque neurone.

Cependant, le Coral Accelerator que nous utilisons demande uniquement des entiers sur 8bits afin de pouvoir effectuer une accélération matérielle.

La quantization consiste donc à passer tous les paramètres du réseau neuronal de leur valeur réel à une valeur entière correspondante.

4. CONCEPTION MÉCANIQUE

4.1. Voiture

Deux pièces ont dû être confectionnées. Le support pour venir fixer le RPi et celui pour la caméra.

Ces supports ont tout d'abord été fait avec ce que j'avais chez moi, ne pouvant pas avoir accès aux locaux de l'école, et ont été fait en bois et en tôle.

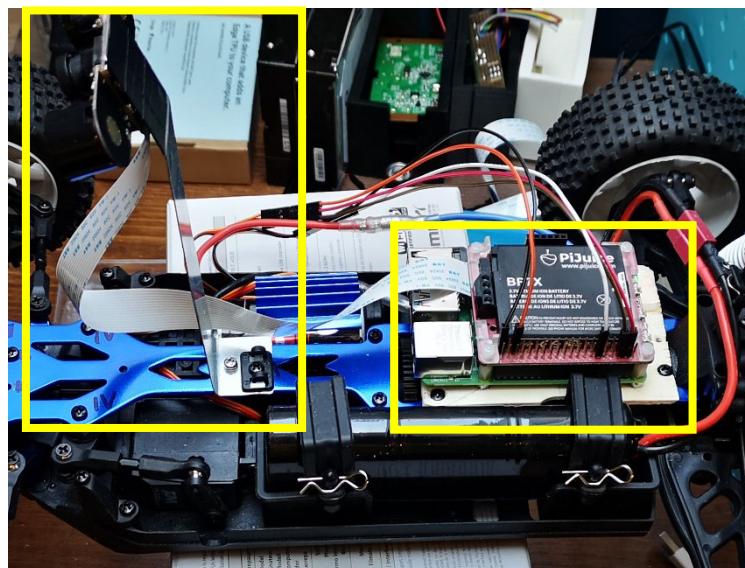


Figure 12 Support RPi et camera, V1

Le problème de ce support de caméra est que la tôle étant fine, peu rigide et longue, le moindre mouvement de la voiture faisait beaucoup vibrer la caméra, ce qui rendait son utilisation impossible.

Après avoir eux accès à une imprimante 3D, un nouveau support bien plus rigide, avec une bonne hauteur et inclinaison a été conçu.

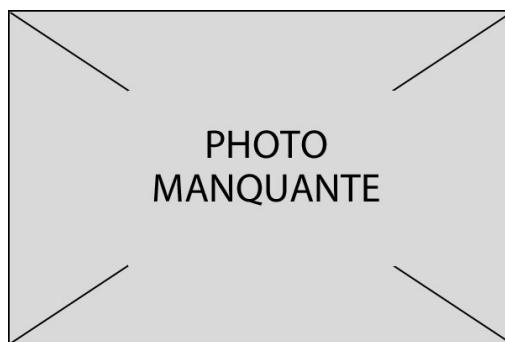


Figure 13 Support camera, V2

4.2. Route

La largeur de la route a été choisi en se basant sur les vraies largeurs de route.

- Largeur des routes nationales française (14): 3.5m
 - Largeur d'une voiture standard (15): 1.8m
 - Largeur voiture RC : 30cm
- Largeur route miniature : $L_{routeRC} = \frac{L_{route}}{L_{voiture}} * L_{voitureRC} = 58cm$

Ensuite pour les virages, un rayon de courbure moyen de 90cm a été choisi, afin que la route passe chez moi.



Figure 14 Rayon de courbure de la route et parties modulables

Pour finir, une route modulable a été conçu avec des parties de route en cartons de 1m50 se fixant les uns aux autres avec des scratchs afin de pouvoir modifier le parcours et la déplacer plus facilement.

Le choix des couleurs des bordure sera discuté au point 8.4.

4.3. Signalisation

La voiture étant un modèle à 1/10^{eme}, il a été choisi de garder la même proportion pour les panneaux et feux.

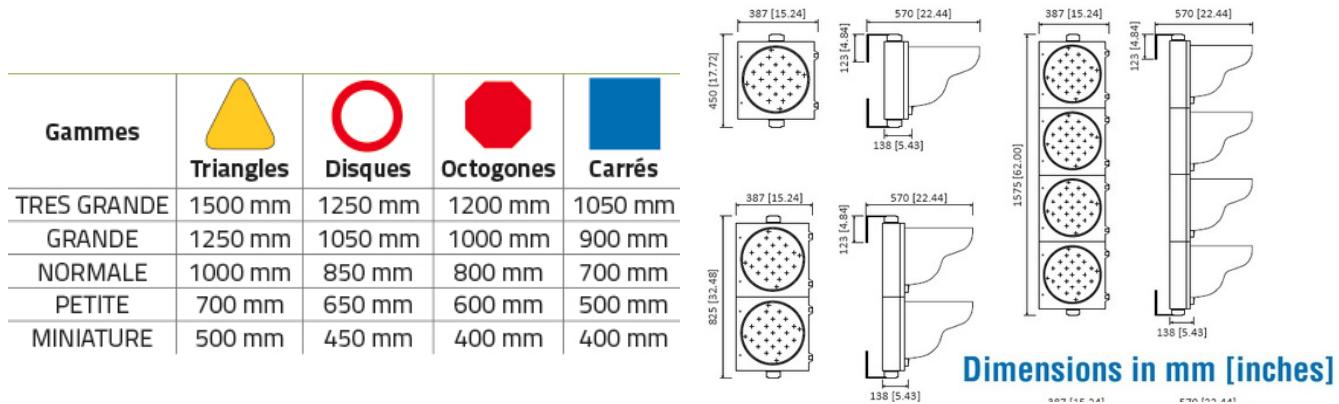


Figure 15 Taille des panneaux et feux Source : (11) et (12)

Le tout a été imprimé en 3D. Une taille « NORMALE » de panneau a été choisi et comme nous le verrons au point 5.2, de la place a été laissé dans le feu afin d'y mettre un Arduino Nano et une antenne BT.

5. CONCEPTION ÉLECTRONIQUE

5.1. Electronique de la voiture

Le montage électronique de la voiture est relativement simple. La majorité des composants peuvent simplement être connecté les uns avec les autres.

Une carte des composants et de leurs connections suffit donc à reproduire le montage électronique.

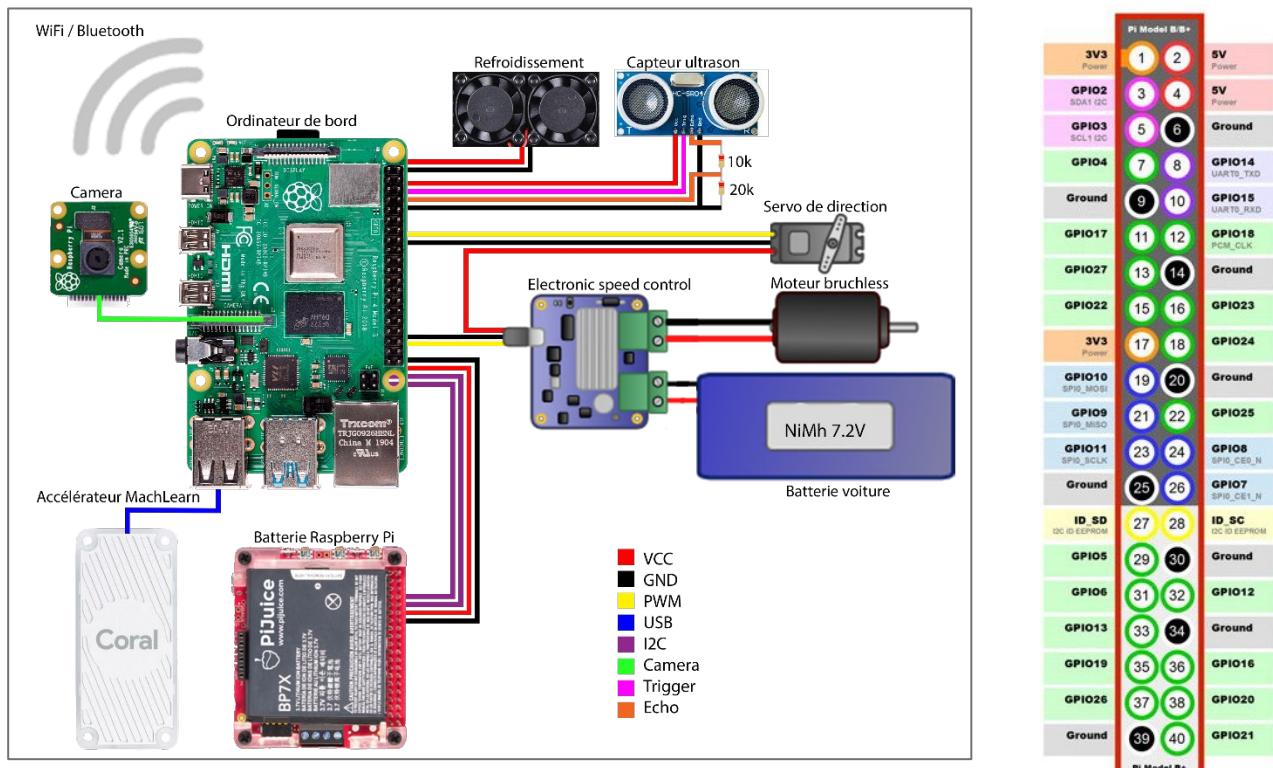


Figure 16 Montage électronique

- 2 (5V): HC-SR04 VCC
- 4 (5V): Fan VCC
- 6 (GND): Fan GND
- 9 (GND): HC-SR04 GND
- 12 (GPIO18): Speed controller PWM
- 14 (GND): Speed controller GND
- 16 (GPIO23): HC-SR04 Trigger
- 18 (GPIO24): HC-SR04 Echo
- 35 (GPIO19): Servo PWM
- 39 (GND): Servo GND

Le PiJuice vient simplement se plug via le connecteur 40pin et ne condamne aucune pin. Il faut juste noter que l'**adresse 14 du bus I²C (valeur modifiable)** est utilisé par le PiJuice.

Un diviseur de tension 5V → 3.3V a tout de même dû être soudé à la sortie du capteur de distance.

Les GND des deux batteries, du RPi et du servo sont bien tous connectés.

Les PWM ont par ailleurs poser passablement de problèmes, qui seront discutés au point 11.3.

Le RPi et la batterie ont tout d'abord été placé séparément afin de prendre moins de hauteur et ainsi permettre de remettre la carrosserie de la voiture. Cependant une telle connexion entre les deux éléments avec des jumpers (tout ce que j'avais chez moi en période de confinement) menaient à des problèmes de communication sur le bus I²C dû à des perturbations. Les fonctionnalités de la batterie étaient inutilisables. Il a donc fallu tout de même connecté la batterie au RPi directement via le connecteur 40 pins.

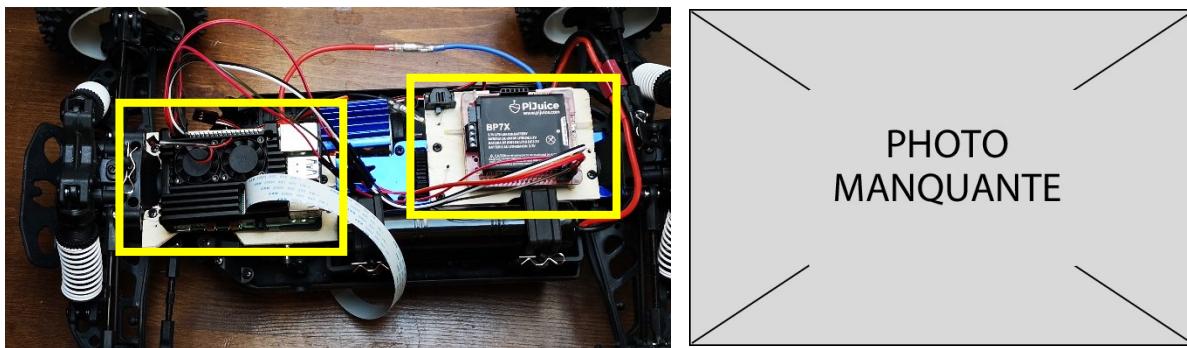


Figure 17 Montage batterie séparée et ensemble

Par contre, étant donné que le système de refroidissement du RPi (voir 3.2.3) prend trop de place, il n'a pas été possible de fixer la batterie comme prévu.

Il aurait été possible de le fixer sous le RPi, mais ce dernier n'a pas un connecteur 40pin prévu pour fixer des modules par-dessous. Il aurait fallu remplacer celui d'origine, au risque d'abimer le RPi.

Il a donc été choisi de monter une rallonge sur le connecteur du RPi.

5.2. Electronique du feu bicolor

Le feu devait à la base contenir un Arduino Nano ainsi qu'une antenne BT afin de pouvoir le guider à distance depuis un téléphone ou la voiture. Cependant la décision a été prise de simplifier sa conception et de le commander manuellement via un switch afin d'y consacrer moins de temps et pouvoir l'investir sur d'autres parties du travail.

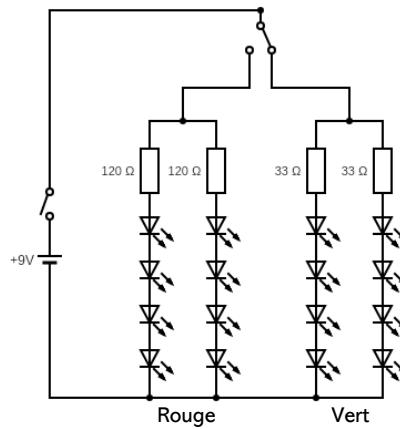


Figure 18 Circuit électrique d'un feu

Le feu est alimenté par une pile 9V, contient un interrupteur principal et un switch permettant de passer du rouge au vert.

Afin d'avoir une source lumineuse bien répartie derrière la vitre du feu, au minimum 8 LED sont nécessaires.

Par soucis de simplicité de conception, les LED ont été mis en série. Cependant les LED rouges ayant une chute de tension (V_f) de 1.7V et les verte de 2.1V, uniquement 4 LED pouvaient être mis en série ($4 * 1.7 = 6.8V$ et $4 * 2.1 = 8.4V$). Deux blocs de 4 LED ont donc été faits.

Le calcul des résistances découle du courant voulu dans les LED. Ce dernier a été choisi afin d'avoir un compromis entre intensité lumineuse et consommation du feu et a été choisi à 20mA.

Les résistances se calculent de cette façon :

$$R = \frac{V_{source} - (Nbr_{LED} * V_f)}{I_{LED}}$$

Et valent donc 110Ω (120Ω de la série e12) pour le rouge et 30Ω (33Ω de la série e12) pour le vert.

6. INSTALLATION RPI

Voir si je le fais ou pas
Script Pi Juice

7. ARCHITECTURE LOGICIELLE

Une première architecture logicielle a été pensé au début du projet, mais a beaucoup évolué lors du développement et l'acquisition de nouvelles connaissances. La version finale est présentée ici.

7.1. La classe “Car”

Cette classe est une représentation de la voiture dans le monde réel et est à la base de toute communication entre le soft et les différentes capteurs et actuateurs.

Nous pouvons voir que la voiture est constituée de :

- Un capteur de distance (UltrasonicSensor)
- Un moteur brushless (SpeedController)
- Un servo de guidage (SteeringController)
- Une caméra (PicameraController)

Le contrôleur du moteur et le servo étant les deux piloté grâce à un PWM, leurs classes correspondantes héritent toutes les deux de la classe _PwmActuator.

Cette dernière va permettre de généraliser l'initialisation et le démarrage des sorties PWM.

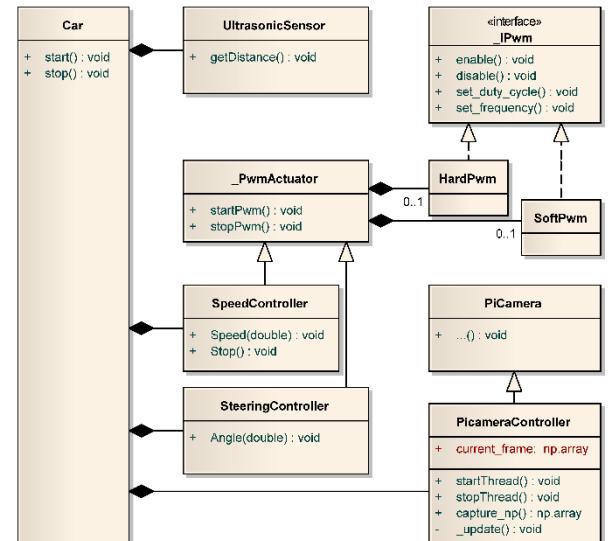


Figure 19 UML de la voiture

Après avoir rencontré un problème de précision des PWM (voir point 11.3), il a dû être nécessaire d'implémenter un PWM hardware et de ne plus utiliser la classe de base « RPi.GPIO.PWM » qui utilise un PWM software.

Afin de permettre de passer facilement d'un PWM software à hardware, les classes SoftPwm et HardPwm réalisent le même interface (_IPwm). Ainsi leurs utilisations est parfaitement identique et les différences d'implémentation sont gérées en interne.

Lors de la création d'un objet _PwmActuator, son paramètre « hardware :bool » définira quelle sorte de PWM sera créé.

La classe PicameraController hérite de la classe PiCamera (provenant du package portant le même nom) et lui ajoute deux fonctionnalités non présentes de base.

- L'implémentation d'une fonction de capture d'image directement au format Numpy
- La gestion d'un thread dédié à la capture d'image, capturant à la vitesse spécifiée dans le paramètre « framerate » de la caméra et plaçant le résultat dans la variable current_frame afin que celui-ci puisse être utilisé par d'autres threads.

7.2. L'application

L'application contient :

- La représentation de la voiture
- L'algorithme de détection de panneaux
- L'algorithme de suivi de route
- L'algorithme de détection d'obstacles
- Et un gestionnaire d'input qui est dans notre cas la manette de jeux.

Les différents algorithmes utilisent les capteurs de la voiture afin d'acquérir l'image courante (mis à jour par le thread de la caméra) ou la distance fournie par le capteur à ultrason.

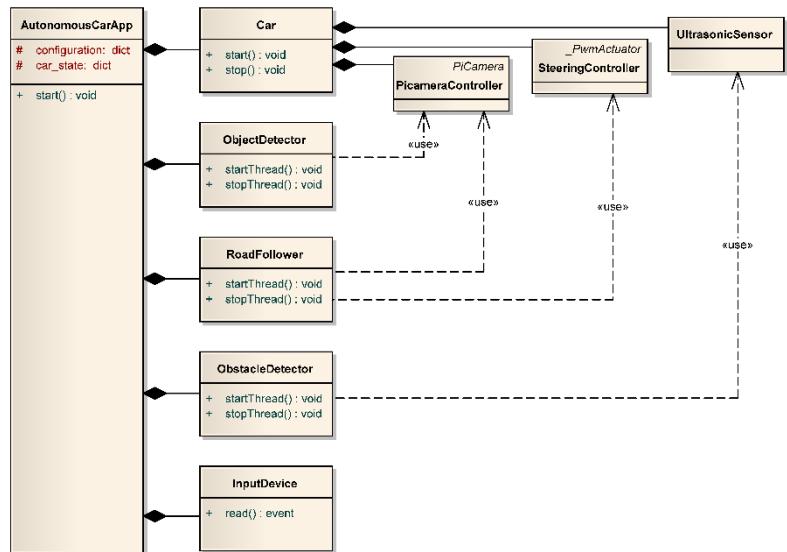


Figure 20 UML de l'application

7.2.1. UN OU PLUSIEURS THREAD?

L'application a tout d'abord été conçue de façon séquentielle. L'image était d'abord capturée, traitée, puis chaque algorithme analysait ce qu'il fallait faire un après l'autre, puis pour finir la vitesse et la direction étaient modifiés.

Le problème est que chaque partie possède des phases d'attente du matériel externe.

- La capture d'image n'est pas immédiate et attend sur la caméra
- La détection d'obstacle doit attendre que l'écho revienne
- La détection de panneaux attend que le Coral Accelerator ai trouvé les panneaux

De plus en faisant de façon séquentielle, si une partie du programme est plus lente (dans notre cas nous verrons que le suivi de route est plus lente) tout le reste du programme sera ralenti et ne pourra pas traiter plus d'image par seconde que la partie la plus lente. En utilisant des threads, la vitesse de chacun peut être géré indépendamment.

Ainsi le programme contient les 5 thread suivant :

(Les threads ont les mêmes noms que les classes les instanciant)

- Main
- PicameraController
- RoadFollower
- ObstacleDetection
- ObjectDetection

7.2.2. GESTION DE LA VITESSE ET DU GUIDAGE

La direction des roues est directement modifiée par le thread de suivi de route car seul ce dernier peut modifier cette valeur.

Cependant la vitesse peut être modifier par tous les algorithmes. Il faut donc coordonner ces décisions.

L'application contient pour ceci un dictionnaire « car_state » partagé entre les threads et contenant les décisions de chacun. Le thread principal vient ensuite lire ce dictionnaire et prendre la décision finale pour modifier la vitesse de la voiture.

```
car_state = {
    'stop_flags': {
        'no_road'      : False,
        'stop_sign'    : False,
        'red_light'   : False,
        'obstacle'     : False,
        'manual_stop' : False
    },
    'speed_limit' : 25
}
```

7.2.3. UNE APPLICATION SIMPLE D'UTILISATION

L'application principale se lance simplement en appelant son unique fonction « start ». Le voiture et les différents algorithmes contiennent eux aussi une fonction start et stop afin de pouvoir simplement les lancer et les arrêter. Ces classes prennent aussi en charge l'utilisation de l'attribut « with » afin de pouvoir gérer proprement et de manière sécurisé le démarrage et l'arrêt des différentes parties même lorsqu'une exception est levé.

Afin d'exclure une certaine partie du programme, il suffit de commenter une des lignes suivantes et le thread correspondant ainsi que ses fonctionnalités sera exclu du programme sans impacter le reste du fonctionnement.

```
with self.car: #start motor, steering commande and camera
    with self.roadFollower: #start the road following algorithm
        with self.objectDetector: #start the sign detector algorithm
            with self.obstacleDetector: #start the obstacle detector algorithm
                (...)
```

7.3. Le suivit de route

L'algorithme de suivit de route utilise l'image courante mise à jour par le thread de la caméra, la détord et prend une vue d'oiseau de la route grâce aux fonctions des classes ImgRectifier et ImgWarper (voir point 8), puis la traite afin de savoir la direction de la route.

L'angle des roues est alors transmis au SteeringController de la voiture.

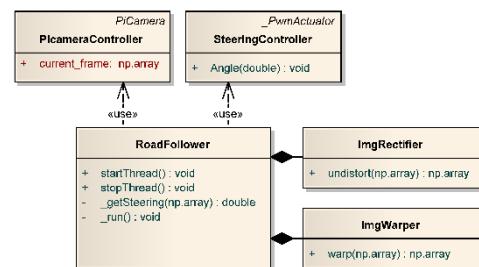


Figure 21 UML du suivit de route

7.4. Détection de signalisation

La détection de panneaux et de feux se fait grâce à un réseau neuronal convolutif (voir 3.7). Ce réseau prend en entrée une image de 300x300 pixel et fournit en sortie une liste des signalisations trouvés et leur position.

Ce réseau a tout d'abord été entraînée séparément sur une machine plus puissante puis son modèle a été exporté et installé sur le RPi.

La classe DetectionEngine sert justement à traiter les images par le réseau neuronal et exécute cette détection sur le Coral Accelerator afin d'accélérer le calcul.

Les différentes classes correspondantes au type de signalisation et réalisant toutes l'interface `_ITrafficSignProcessor` vont servir à déterminer si le panneau est assez proche (fonction `is_nearby`), et à modifier la vitesse de la voiture via le dictionnaire partagé `car_state`.

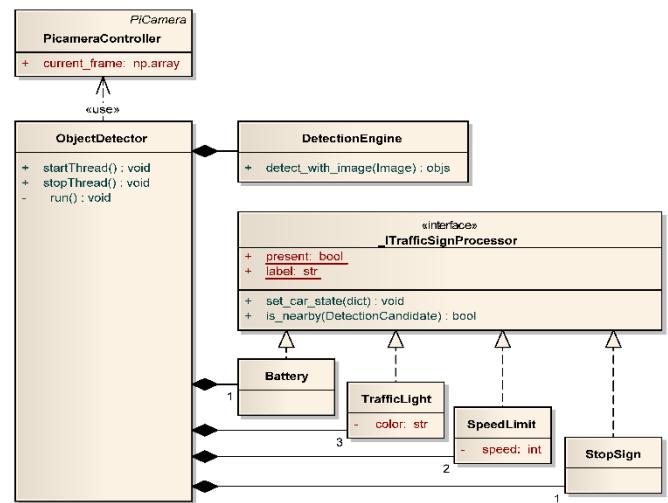


Figure 22 UML détection de signalisation

7.5. Fichier de configuration

➔ Exemple de fichier de configuration en annexe 17.3

Afin de ne pas devoir se rendre dans le code pour modifier certains paramètres et constantes, tout a été réuni dans un fichier de configuration. Ainsi l'application principale (AutonomousCarApp) ne demande pas pleins de paramètre pour être lancé, mais juste le chemin d'accès de ce fichier de configuration.

Le fichier utilise les normes du langage de balisage YAML (plus d'infos, voir (16)).

Une fois le fichier importé en Python, nous obtenons un dictionnaire où les clés sont les balises du fichier de configuration.

YAML file	Python
<pre>CAMERA: parameters: resolution: [640, 480]</pre>	<pre>import yaml conf = yaml.load(CONF_FNAME) conf["CAMERA"]["parameters"]["resolution"] => (640, 480)</pre>

8. SUIVIT DE ROUTE

Le suivit de route peut être fait de deux façons :

- Par deep learning
- Par traitement d'image

Afin d'apprendre à un réseau de neurone de conduire la voiture, il faut au préalable enregistrer une base de données que le réseau pourra utiliser.

Pour enregistrer cette base de données, il faut conduire manuellement la voiture et enregistrer dans un fichier ce que voit la caméra. Pour chaque image capturée, il faut aussi enregistrer l'angle des roues et la vitesse de la voiture.

Ensuite il faut entraîner un CNN (Réseau neuronal convolutif (CNN)3.7.1) qui déterminera un angle et une vitesse en fonction d'une image.

Un tel procédé se fait énormément dans le domaine des voitures autonomes, mais généralement l'entraînement se fait avec des 100^{eme} d'heures de conduit manuelle.

Etant donné que je ne voulais pas consacrer trop de temps à la création de la base de données et que le résultat n'était pas assuré, j'ai préféré me tourner vers la deuxième solution par traitement d'image expliquée ici.

Globalement nous devons repérer les lignes sur l'image et déterminer en fonction de leurs positions ce que la voiture doit faire.

8.1. Calibration

La caméra comportant une forte distorsion étant donné que son objectif est un grand angle. Cette distorsion a pour effet de courber certaines parties de l'image.

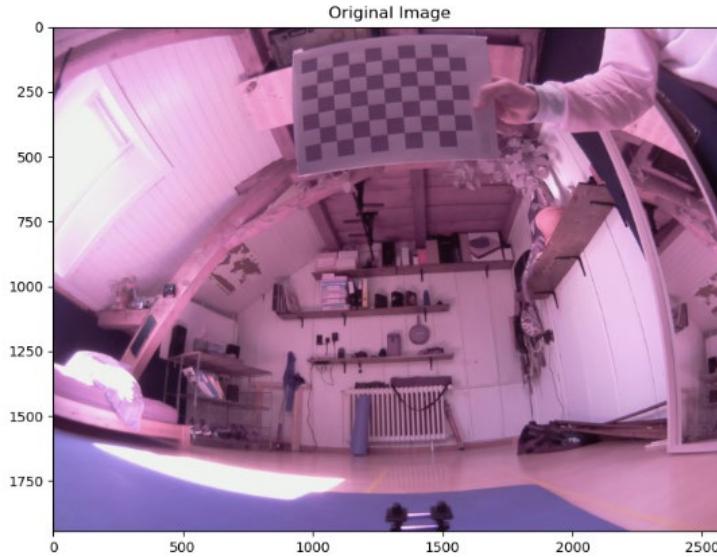


Figure 23 Image avec distorsion

Etant donné que nous voulons analyser la forme des lignes au sol, il est important que l'image ne comporte pas de distorsion.

Afin de remédier à ce problème nous devons trouver certains paramètres caractérisant la caméra.

- La matrice de la caméra, sous la forme $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ avec f_x / f_y les distances focales en x et y (deux distances focales sont ici données, car si l'image est compressé dans un sens, les focales ne seront pas les mêmes) et c_x / c_y un point caractérisant le centre optique de l'image, qui est généralement proche du centre de l'image.
- Les coefficients de distorsion, décrits en 8.1.1

8.1.1. COEFFICIENTS DE DISTORSION

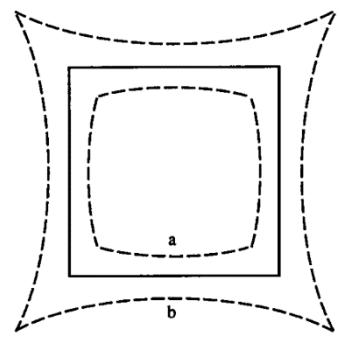
➔ Article sur la distorsion: (17)

Les coefficients de distorsions servent à caractériser la distorsion d'un objectif et se nomment k_1, k_2, k_3, k_4, k_5 et k_6 pour la distorsion radiale de l'objectif, et p_1 et p_2 pour la distorsion tangentielle.

Dans la librairie OpenCV que nous allons utiliser, uniquement les paramètres k_1, k_2, p_1, p_2 sont fournis car les autres n'ont que peu d'effet.

8.1.1.1. Distorsion radiale

La distorsion radiale est dû à la courbure de la lentille et provoque un déplacement des points de l'image centré sur le centre de l'image. La distorsion peut être en forme de bâillet (a) ou de coussinet (b).



8.1.1.2. Distorsion tangentielle

La distorsion tangentielle est dû à un mauvais alignement de la lentille. Elle provoque un déplacement des points de l'image se trouvant loin de l'axe comportant le moins de distorsion (voir Figure 24).

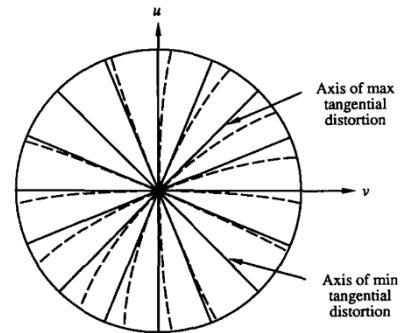


Figure 24 Effet de la distorsion radiale et tangentielle

Source : (17)

8.1.2. APPRENTISSAGE DE LA CALIBRATION

L'apprentissage de ces paramètres se fait grâce à la bibliothèque OpenCV.

Il faut tout d'abord capturer un set de 10 – 20 images comportant un échiquier.

Ensuite la fonction `cv2.findChessboardCorners` va permettre, par traitement d'image, de retrouver les coins des cases de l'échiquier.

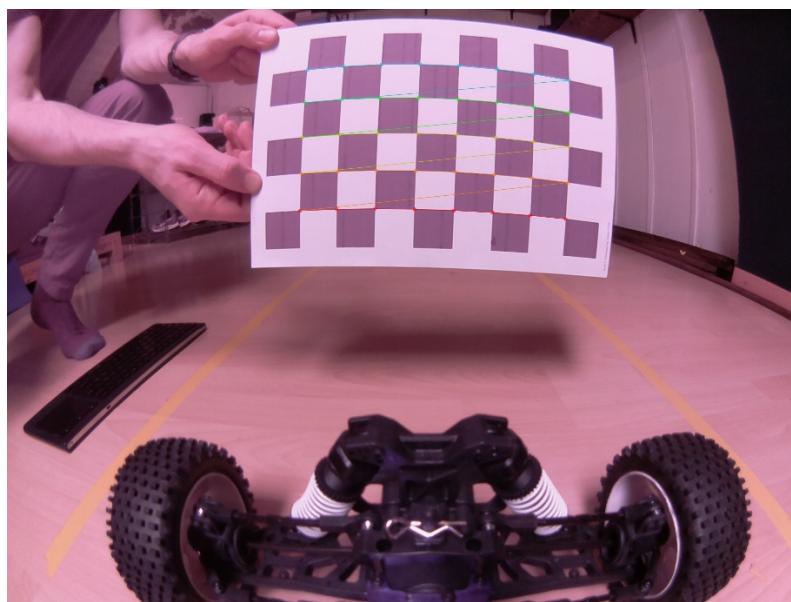


Figure 25 Coins de l'échiquier trouvés

Optionnellement, il est possible d'utiliser ensuite la fonction `cv2.cornerSubPix` afin de raffiner la position des coins.

Les coins de toutes les images vont alors être transmis à la fonction `cv2.calibrateCamera` qui va en déduire la matrice de la caméra et les coefficients de distorsion.

Ces paramètres ont été dans mon cas enregistré dans un fichier `.pickle` afin de pouvoir les ressortir plus tard.

8.1.3. RECTIFICATION D'UNE IMAGE

En utilisant la fonction `cv2.undistort` qui prend en paramètre la matrice de la caméra, les coefficients de distorsion ainsi que l'image à traité, nous pouvons rectifier cette image.

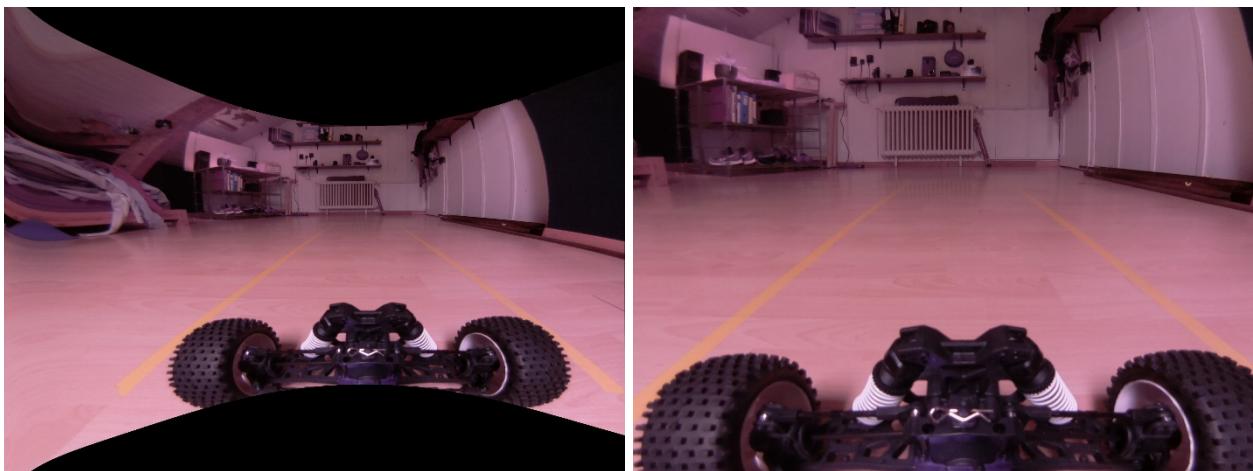


Figure 26 Image après rectification, sans/ avec crop

Attention cependant si l'image à traiter n'a pas la même résolution que les images utilisées lors de l'apprentissage de la calibration.

Les coefficients de distorsion ne sont pas affectés par la résolution, par contre les valeurs de la matrice de la caméra doivent être modifiées. Il faut appliquer le même radio sur ces valeurs que le ratio entre la résolution de l'image à traiter et les images d'apprentissage.

$$\Rightarrow \text{Par exemple : } f'_y = f_y * \frac{h_{img}}{h_{img_{apprentissage}}}$$

8.2. Warping

Maintenant nous voulons déterminer où va la route afin de savoir si nous devons tourner.

Une première approche serait de repérer les lignes et de les prolonger afin de trouver leur point de fuit. Il faut ensuite faire en sorte que ce point reste au milieu de l'image. Cette technique fonctionne pour une route droite, mais pas lorsque celle-ci tourne.

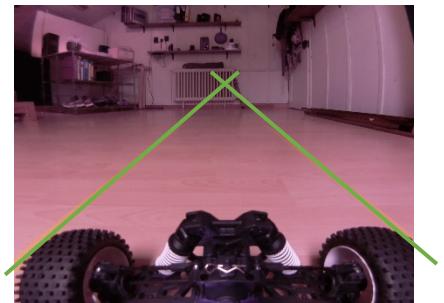


Figure 27 Suivit de route par point de fuite

La deuxième approche consiste à déformer l'image afin d'annuler la perspective et donner l'impression de voir la route en vue d'oiseau. La vraie forme de la route peut ainsi être déterminé.

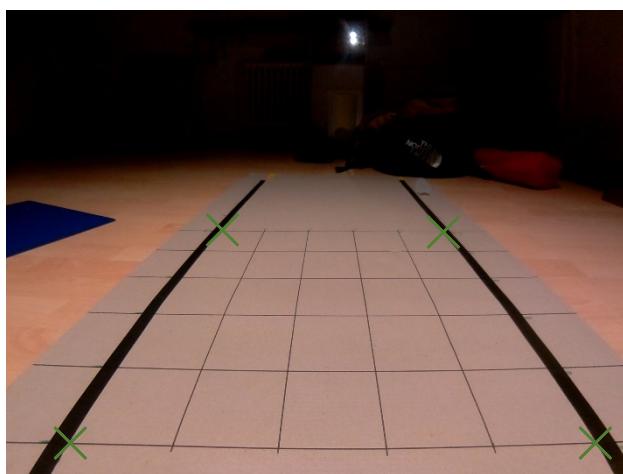


Figure 28 Warping

La technique est de choisir 4 points dans l'image d'origine qui seront ensuite mis en rectangle. Tous les autres pixels doivent être ajusté en conséquence. La fonction `cv2.warpPerspective` peut être utilisée.

Etant donné que sur l'image d'origine le rectangle comporte moins de pixel en haut qu'en bas, lorsque que la déformation est effectuée, des pixels doivent être générés sur le haut, ce qui donne ce flou.

La configuration du warping a été effectué sur une grille afin de pouvoir obtenir un résultat avec un ratio H/L de 1 et que la route ai ainsi la bonne dimension. Initialement le warping avait été fait sans grille, et l'image était aplati ce qui faussait des calculs que nous verrons plus loin.

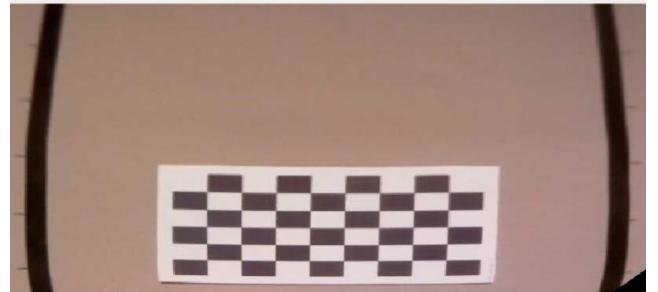


Figure 29 Warping effectué sans grille → Image aplati

8.3. Balance de couleur

Les images capturées par la caméra ne reflète parfois pas les vrais couleurs. Pour y remédier il est possible de modifier des paramètres de la caméra afin d'effectuer une balance de blanc.

Un petit programme (*development_script/awb_gain_selector.py*) a été mis en place dans le but de simplifier la paramétrisation manuel de la balance de blanc. Il fallait effectuer la balance de blanc avec ce script, puis rentrer les paramètres dans le fichier de configuration du programme principal. Le problème était que la balance de blanc devait être refaite à chaque fois que les conditions lumineuses changeaient.

Il a alors été imaginé d'effectuer la balance de blanc de façons automatique dans la phase d'initialisation du programme. Une feuille blanche devrait être positionnée devant l'objectif, et l'algorithme modifierait les paramètres de la caméra dans le but que la feuille soit bien blanche à l'image.

Par contre, il a été remarqué que la teinte de l'image se modifiait parfois simplement en faisant pointer la voiture dans une autre direction.

Il s'est avéré que, étant donné que la caméra n'a pas de filtre à infra-rouge, l'image prend une teinte rosée lorsque la caméra pointe vers une source produisant beaucoup d'infrarouge. Une balance de blanc devient donc inutile dans un tel cas.

Une attention particulière a donc dû être donnée aux sources de lumières éclairant la route.

8.4. Isolement des lignes

Une fois que nous voyons la route depuis le dessus, il faut réussir à isoler les lignes du reste de l'image et ne pas prendre en compte tous objets parasites.

Afin de trouver le meilleur traitement à faire sur l'image dans le but d'isoler les lignes, une image de test comportant beaucoup d'élément parasites a été utilisée.

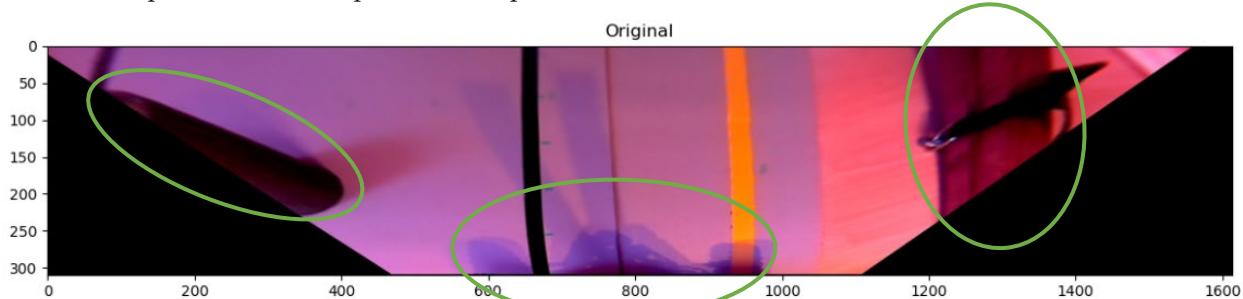


Figure 30 Image de test pour l'isolement des lignes

L'image comporte l'ombre de la voiture, des objets non voulus et une ligne noire et une jaune afin de voir quelle couleur se détecte le plus facilement.

Dans le but de faire ressortir le plus possible notre ligne du reste de l'image, nous allons analyser la représentation de cette image de test dans différents espaces colorimétriques.

L'espace RGB est le plus connu, mais de nombreux autres comme le HLS, HSV, LAB ou LUV existent et ont été analysés ici.

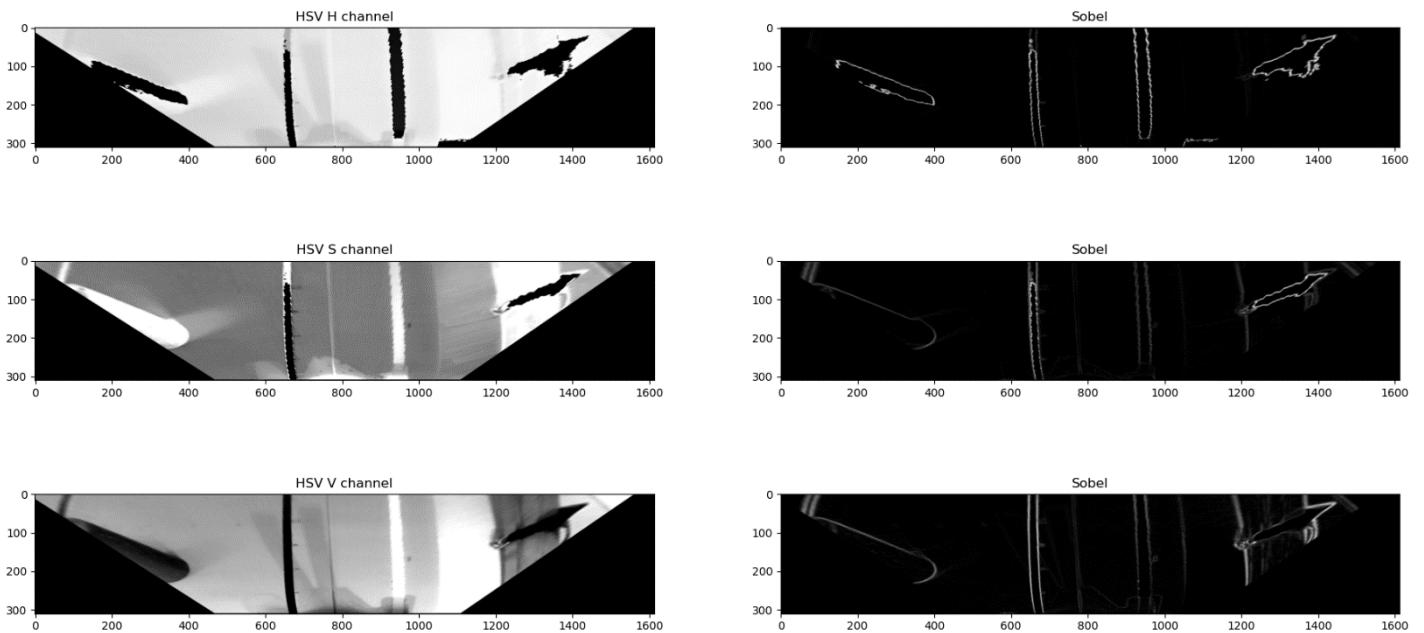


Figure 31 Les 3 couches de l'espace HSV avec leur filtre de Sobel correspondant

Pour chaque espace colorimétrique, les 3 couches ont été montrée séparément ainsi que leur filtre dérivateur correspondant (filtre de Sobel, faisant ressortir les contours).

Note : Le filtre de Sobel n'a finalement pas dû être utilisé, mais il pourrait aider à détecter les lignes étant donné que ces dernières sont très contrastées avec le fond de la route, ce qui veut dire que leur contour est très marqué.

Après analyse de tous les espaces, le HSV était le plus efficace à faire ressortir la ligne, à condition que celle-ci soit jaune.

L'espace HSV représente :

- Hue (teinte) : La couleur du pixel, comme nous la connaissons naturellement. Cette valeur est plus simple à se représenter qu'un mélange de RGB.
- Saturation : Si le pixel comporte beaucoup de cette couleur, ou est plus fade
- Valeur : Si le pixel est foncé ou clair

A partir de la couche Hue, il est simple de dire que nous voulons sélectionner uniquement les pixels jaunes.

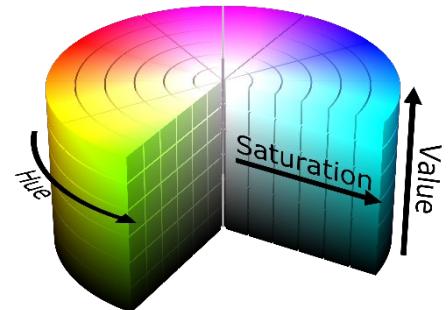


Figure 32 Espace HSV

L'utilisation de la ligne noire est une mauvaise idée, car beaucoup d'élément perturbateur dans l'image peuvent être noir, et ils seront considéré comme des lignes.

Afin d'isoler les lignes, un threshold est effectué sur les trois couches de l'espace HSV.

Pour aider à choisir les valeurs du threshold, un script, "development_script/threshold_slider.py" a été mis en place. Des sliders peuvent être bougés afin de modifier les valeur min et max des pixels à garder pour chaque couche. Les pixels gardés sont alors affichés.



Figure 33 threshold_sliderbar.py

8.5. Regroupement des pixels en ligne

Nous avons maintenant des pixels à 1 et d'autres à 0, mais ces pixels ne représentent toujours pas des lignes.

Pour se faire nous allons grouper tous les pixels se touchant dans un même ensemble. Cette opération est réalisée par la fonction `cv2.connectedComponentsWithStats`.

Cette fonction va notamment retourner plusieurs statistiques sur chaque ensemble de pixel, comme leur aire. L'aire est utilisée afin d'accepter uniquement des regroupements de pixel d'une certaine taille, et ainsi supprimer les parasites visibles en Figure 33.

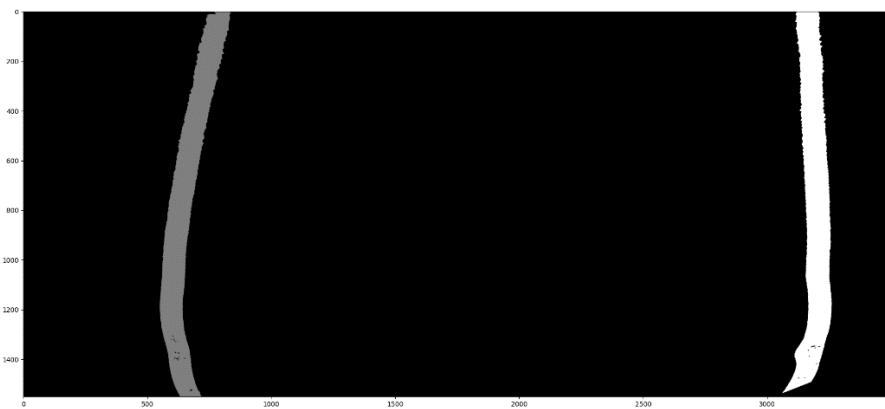


Figure 34 Regroupement des pixels connectés

8.6. Direction des lignes

Maintenant que nous avons une liste de pixel faisant partie de la même ligne, il faut trouver l'équation représentant le mieux cette ligne.

Tout d'abord une transformée de Hough (technique permettant de déterminer un alignement de point) était utilisée afin de définir l'équation représentant la ligne. Cette technique ne fonctionne cependant plus lorsque la route tourne car les points ne sont plus alignés et elle retournait des équations ne représentant plus du tout la route.

Une régression a alors été effectuée sur les pixels afin de trouver cette équation.

→ La régression peut être effectuée grâce à la fonction `np.polyfit`.

La régression était tout d'abord effectuée avec les axes dans le bon sens, et les résultats étaient surprenants. Les axes ont dû être inversé pour que les résultats soient corrects car les lignes sont la plupart du temps vertical sur l'image, et qu'une équation du 2^{em} ordre n'est pas faite pour être vertical (une valeur de x ne peut pas correspondre à plusieurs valeurs sur y).

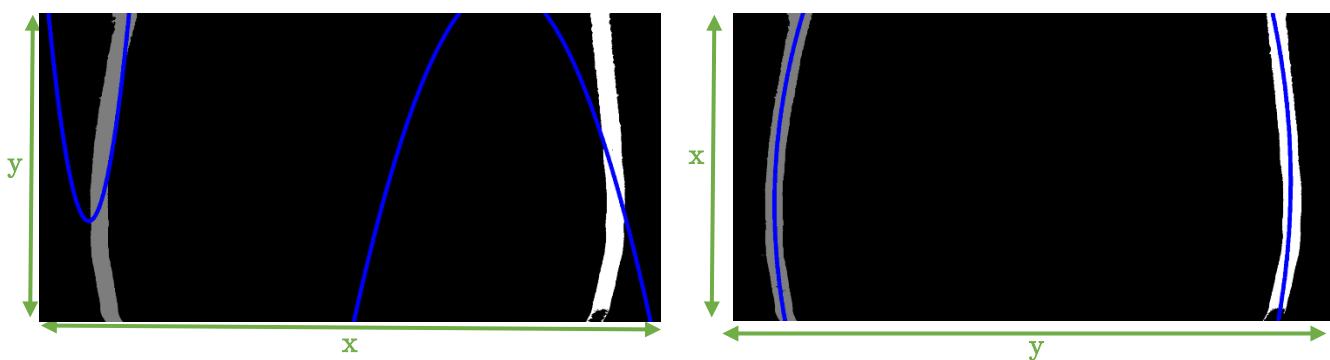


Figure 35 Régression avec axes normaux vs inversés

Finalement une régression linéaire a été utilisée et non du 2^{em} ordre car il suffit de savoir la pente de la ligne pour diriger la voiture, même dans un virage. La courbure de la route (qui serait fourni par le second ordre) n'est pas forcément nécessaire.

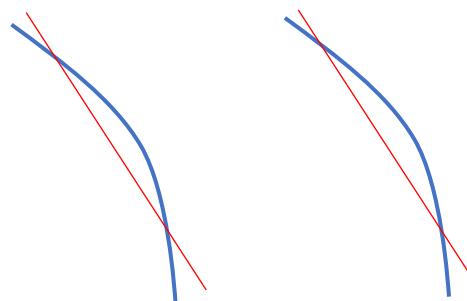


Figure 36 Régression linéaire dans un virage

A CHANGER PAR UNE VRAI IMAGE !

Un dernier filtrage des ensembles de pixel est effectué après la régression, en ne considérant pas les groupes ayant un écart-type trop haut lors de la régression. De ce fait uniquement les groupes de pixels ayant été le mieux adapté donc ressemblant le plus à une ligne seront gardé

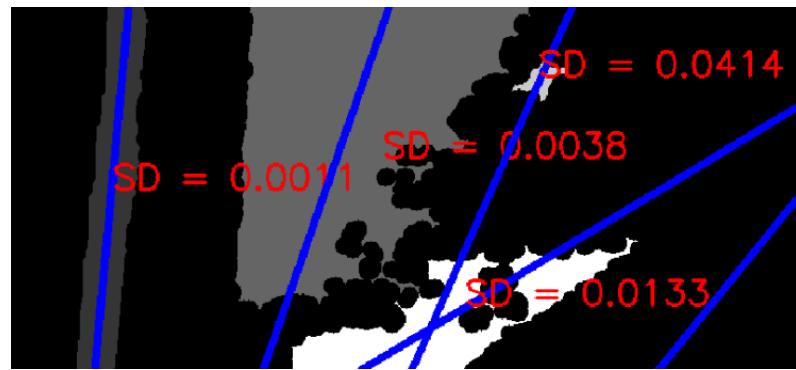


Figure 37 Utilisation de l'écart-type
À gauche la ligne voulu, à droite des ensembles indésirables

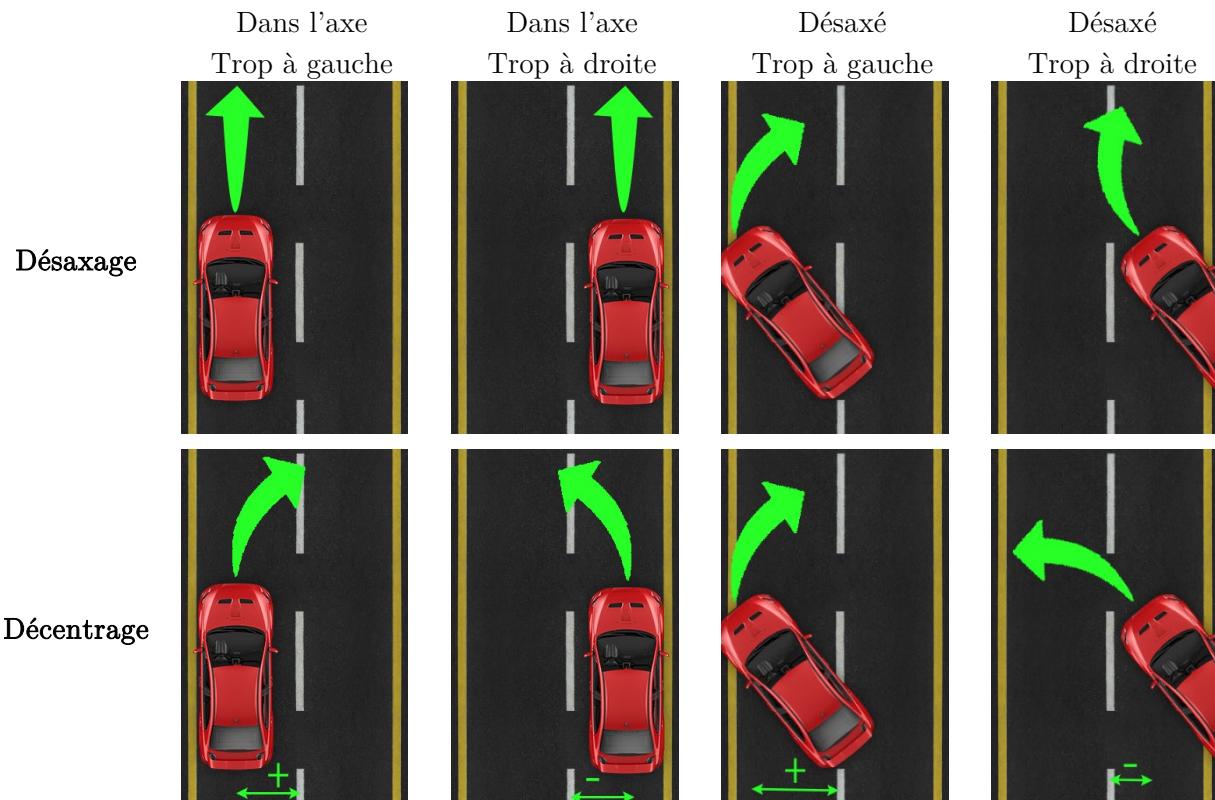
8.7. Angle des roues

Une fois la pente des lignes trouvée, la position de la voiture doit être corrigée en conséquence.

La correction prend en compte deux facteurs :

- Le désaxage de la voiture par rapport à la route, donc si celle-ci forme un angle avec l'axe de la route
- Le décentrage de la voiture, donc si la voiture ne se trouve pas au centre de la route

Le tableau suivant montre 4 cas de figure, et la correction à effectuer (en vert) pour corriger chaque facteur.



Au niveau du désaxage, si la voiture va tout droit, aucune correction n'est faite, par contre si la pente de la droite retournée par la régression linéaire est négative, la voiture doit tourner à droite, et à gauche si elle est positive.

Pour le décentrage, la différence entre le milieu des deux lignes et le centre de l'image (qui correspond au milieu de la voiture) va déterminer comment corriger la direction.

Il faut ensuite combiner les deux facteurs judicieusement afin de rester le plus au centre tout en étant bien axé.

Prenons par exemple le 4^e exemple. La voiture est désaxée donc devrait tourner à droite pour se mettre droite, mais elle se trouve à droite de la route donc va tourner à gauche pour se centrer. Les deux commandes rentrent donc en contradiction. Cependant nous remarquons bien qu'il est plus important de tourner à droite afin de se remettre dans l'axe de la route. La partie gérant le désaxage

doit donc avoir plus de poids. Mais comment décider quel poids donner à la correction de désaxage et de décentrage ?

Ce poids n'a pas été choisi avec une valeur fix, mais dépend du désaxage de la voiture.

- Si la voiture est parfaitement droite (0°), 100% de la correction est attribué au décentrage
- Si la voiture forme un angle de 45° avec la route, 100% de la correction est attribué au désaxage
- Entre deux, la correction dépend des deux facteurs

Il a aussi été choisi d'accentuer la correction plus la voiture s'éloigne de sa position correcte.

Ainsi par exemple si la voiture vacille au milieu de la route, peu de correction est effectué pour la ramener au centre, par contre plus elle se rapproche du bord, plus la correction doit augmentera fortement. De même pour l'angle de la voiture avec la route.

Justement pour le désaxage, étant donné que l'angle de la voiture par rapport à la route nous est fourni par la pente de la ligne, et que la pente vaut $\tan(\text{angle}_\text{voiture})$, cette valeur augmente déjà fortement plus l'angle de la voiture augmente. Il ne faut donc pas la modifier.

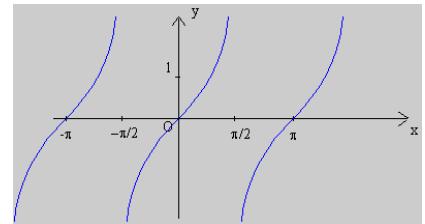


Figure 38 Relation angle voiture à la route (x) et pente de la ligne (y)

Par contre pour le décentrage, ce n'est pas la même chose. Etant donné que le décalage de la voiture avec le centre de la route est donné en pixel et est une valeur linéaire, il faut lui appliquer une fonction tangentielle afin que la correction augmente plus fortement si on se rapproche du bord.

Maintenant que nous savons la correction à appliquer, l'angle des roues peut être modifié.

9. DETECTION DE SIGNALISATION

La détection de signalisation se fait, comme expliqué en 3.7, grâce au réseau neuronal MobilNet SSD. Il est conseillé de lire le point 3.7 pour comprendre ce chapitre.

La programmation de cette détection se sépare en 2 parties :

1. L'apprentissage du réseau
2. La réaction de la voiture à chaque signalisation

9.1. Apprentissage du reseau neuronal

De base, le réseau MobilNet SSD est simplement un modèle, un squelette, où ses 5 millions de paramètres sont vides. Il faut passer par une phase d'apprentissage afin que les filtres convolutifs, la détection d'objet (SSD) et la classification aient les bons paramètres.

Un tel apprentissage demande énormément d'image d'apprentissage et des jours d'entraînement. Heureusement, des modèles pré-entraînées avec différentes bases de données existent.

Si la librairie de machine learning TensorFlow est utilisé, toute sortes de modèles pré-entraînés sont disponible sur leur GitHub. (18)

Dans mon cas, étant donné que le réseau devra tourner sur le Coral Accelerator utilisant le processeur « Edge TPU », un modèle spécialement adapté pour ce dernier a été utilisé.

Pour télécharger les modèles compatibles avec le Edge TPU → (19)

Le nom du modèle utilisé dans mon cas est :

« `ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03` »

Ce modèle a été entraîné par la base de données COCO (20), contenant 1.5 million d'objets labélisé répartis dans 91 catégories.

Etant donné que nous ne voulons pas détecter ces 91 catégories, ce modèle devra être légèrement modifié afin de détecter nos panneaux de signalisations et feux. Ce procédé se nomme « **Transfer learning** ».

Le transfert learning consiste à transférer les connaissances du modèle pré-entraîné à notre nouveau modèle. Dans notre cas toutes les couches d'extraction de feauturs restent inchangées, et uniquement les deux dernières couches du réseau servant à la classification des objets sont réentraînées.

9.1.1. CRÉATION DE NOTRE DATASET

Afin de pouvoir apprendre au réseau à quoi ressemble nos objets, nous devons acquérir un set d'image sur lesquels ils apparaissent ainsi que la position de chaque objet.

Un nombre compris entre 50 à 100 images suffit étant donné que le reste du réseau a déjà été entraîné. Ce nombre peut tout de même varier en fonction de la complexité des objets à détecter et du nombre de catégories.

Dans notre cas, **102 images** ont été capturées.

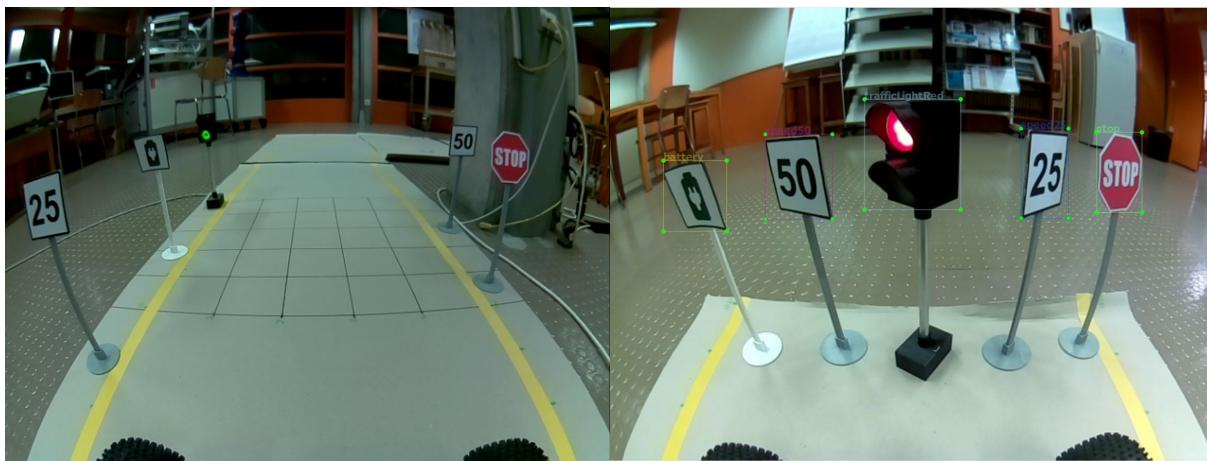


Figure 39 Image acquise pour l'entraînement du réseau avant/après labélisation

Note : Des images sans distorsions ont d'abord été capturé afin d'aider le réseau à détecter les panneaux, mais il s'est avéré que le réseau arrivait aussi très bien détecter les objets sur une image distordu. Il a donc été décidé d'utilisé des images distordu afin d'avoir un plus grand champs de vision (utiliser pleinement le fish eye de la caméra)

Il faut ensuite repérer et annoter la position de chaque objet. L'utilisation du programme **LabelImg** (21) simplifie grandement cette tâche. Un fichier .xml par image est ensuite généré par le programme contenant la classe et la position de chaque rectangle dessiné.

Après labélisation, nous arrivons au nombre suivant d'objets présent dans toutes nos images :

Objet	Nbr d'occurrences
stop	98
speed25	93
battery	92
speed50	90
trafficLightGreen	39
trafficLightRed	38
trafficLightOff	24

Tableau 5 Nbr d'objets dans les images

Nous arrivons à plus de 90 occurrences pour la plupart des objets, ce qui est bien suffisant, sauf pour le feu. Ceci est dû au fait que lors de la prise des images, toutes les sortes de feux devaient représenter la même catégorie (donc 101 occurrences pour le feu) car il était prévu de faire la différence entre les types de feux par traitement d'image, mais il a finalement été préféré de faire cette différence directement par le réseau de neurone (voir 9.2.2).

9.1.2. RESSOURCES UTILISÉES

L'apprentissage du réseau utilise plusieurs librairies et ressources :

Google Colaboratory

Google Colab est un service fourni par Google permettant d'exécuter du code Python depuis le navigateur. Lors de l'accès au Colab l'utilisateur est connecté à une machine virtuelle et il peut directement coder en Python. Le site est un vrai IDE avec entre autres l'aide à la complétion de code. Aucune installation n'est nécessaire et il est possible de faire tourner son code sur un CPU, GPU ou NPU notamment afin d'accélérer l'apprentissage d'un réseau. Il est pour finir possible de connecter son Google Drive à la machine virtuelle afin de pouvoir sauvegarder les fichiers de façon permanente.

Tensorflow, Tensorflow Lite et Tensorboard

Tensorflow est une librairie développée par Google comprenant de nombreuses classes et fonctions simplifiant la création et l'apprentissage d'un réseau de neurone. Tous les algorithmes principaux de ML développés à ce jour se trouvent dans Tensorflow.

Note : Le mot « tensor » signifie un tableau à n dimension, ici utilisé pour illustrer les tensor qu'un réseau neuronal doit traiter

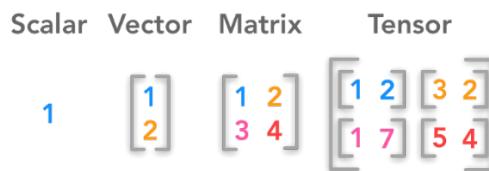


Figure 40 Différence entre un scalaire, un vecteur, une matrice et un tenseur

Tensorflow Lite est un dérivé de cette librairie. Il n'est pas possible d'entrainer un réseau avec, mais juste de l'exécuter. Cette librairie prend moins de place à installer et augmente aussi la vitesse de traitement du réseau.

Sur le RPi, le réseau MobileNet V2 prend par exemple **654ms** à traiter une image avec Tensorflow contre **380ms** avec Tensorflow Lite. (22)

TensorBoard est un affichage graphique monitorant l'avancée de l'apprentissage du réseau neuronal afin de repérer une erreur et voir la qualité de l'apprentissage.

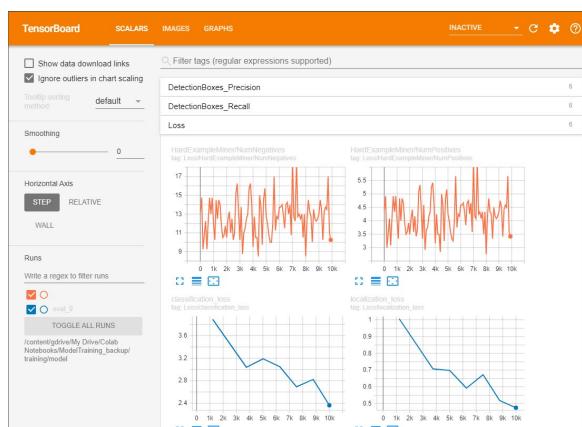


Figure 41 Tableau TensorBoard lors de l'apprentissage des panneaux

Tensorflow Object Detection API

Même avec la librairie Tensorflow et un modèle pré-entraîner, beaucoup de code devrait être écrit afin de réentraîner un modèle de détection d'objet.

C'est pour ceci qu'une API a été mis en place afin d'effectuer ces opérations. (18)

Edge TPU Compiller

Etant donné que le réseau devra tourner sur la puce Edge TPU du Google Coral Accelerator, il doit être compilé à un format compatible à la fin de son entraînement, action qui se fait grâce à ce compilateur.

9.1.3. APPRENTISSAGE

Nous pouvons voir sur ce graphique les étapes à effectuer dans le but d'entraîner un modèle qui devra être déployer sur une Edge TPU

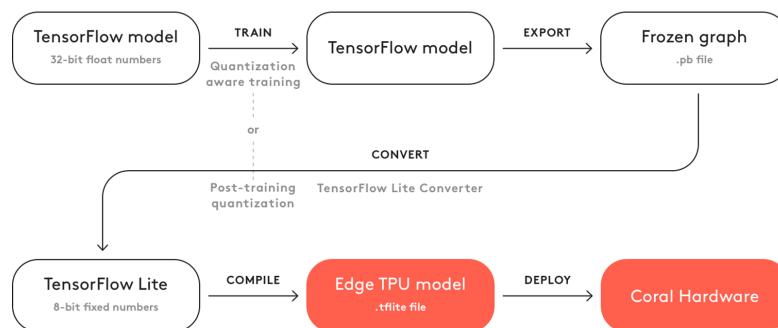


Figure 42 Etape d'apprentissage et de déploiement sur une Edge TPU Source : (23)

Etant donné que toutes les étapes de l'apprentissage sont documentées dans le Colab Notebook, je vous invite à aller le voir en annexe 17.4.

Nous pouvons cependant noter les points suivants :

- Il a été choisi d'exécuter 10'000 pas d'apprentissage, bien qu'après vérification de la progression de l'apprentissage, 1'000 pas auraient suffit car la modèle n'augmentait plus forcément sa précision.
- Les panneaux lointains n'étaient tout d'abord pas repérés car trop petits.
Les paramètres du SSD (partie du réseau qui localise les objets) ont dû être changés afin de ne plus détecter des objets faisant uniquement de 20 à 95% de l'image, mais plutôt de 4 à 40%.
- Le temps d'apprentissage à pris 1h21

9.2. Traitement du résultat et action sur la voiture

La détection des panneaux ne se fait plus dans le Coral Notebook, mais dans un code Python tournant sur le RPi de la voiture.

En une ligne, via le package « edgetpu », nous demandons au Coral Accelerator de traiter une image qu'on lui transmet.

Il nous retourne une liste d'objets trouvé ainsi que leur position.

La distance de chaque objet est ensuite calculée afin de savoir s'il faut le prendre en compte (voir 9.2.1).

Si plusieurs signalisations du même type sont trouvé sur une image, certains ont la priorité :

- Le panneau de limitation de vitesse 25 est prioritaire sur le 50.
- Le feu rouge est prioritaire sur le vert et sur un feu éteint.

Pour chaque type de signalisation trouvé, le dictionnaire « car_sate » (7.2.2) sera modifier afin de mettre à jour la vitesse limite de la voiture ou lui dire se s'arrêter.

Un système de « tampon » a été mis en place, servant à ne pas tout de suite considérer qu'un panneau n'est plus devant la voiture s'il n'a pas été trouvé, mais attendant n frame avant de faire cette conclusion. Ce tampon est nécessaire car il arrive qu'un objet ne soit tout d'un coup plus détecté par le CNN durant un court instant, et il ne faudrait pas redémarrer la voiture pour la stopper juste après.

9.2.1. DISTANCE DE L'OBJET

Nous ne voulons pas que les panneaux détectés trop loin de la voiture soient pris en compte.

Grâce à la perspective, nous pourrons estimer la distance d'un objet en calculant sa taille sur l'image.

Une version simplifiée aurait été de simplement prendre en compte les objets qui ont une hauteur faisant par exemple au moins 30% de l'image. Cependant il est plus pratique de définir que l'on veut par exemple prendre en compte tous les panneaux qui sont à moins de 50cm de la voiture.

Ce procédé demande juste plus de réflexion car il faut calculer la distance d'un objet en fonction de sa taille sur l'image.

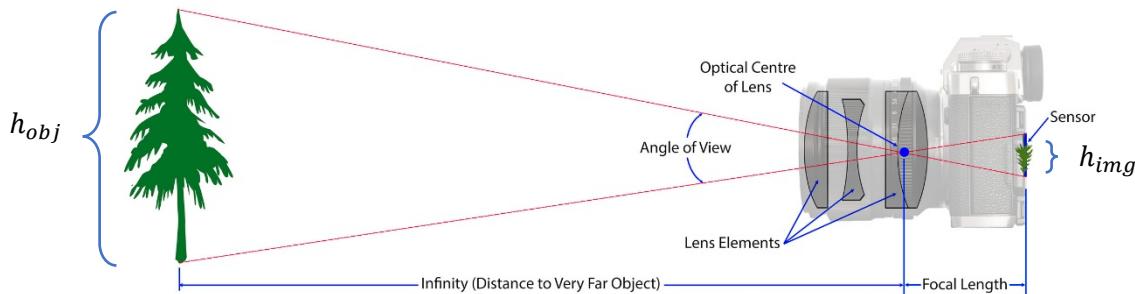


Figure 43 Rapport taille / distance d'un objet

En optique, lorsqu'un objet est loin de l'objectif, nous pouvons établir le rapport suivant :

$$\frac{h_{img}}{h_{obj}} = \frac{\text{focal}}{d_{obj}}$$

Ainsi pour savoir la distance de l'objet nous pouvons établir :

$$d_{obj} = \frac{\text{focal} * h_{obj}}{h_{img}}$$

Dans notre cas, le SSD nous fournit la taille de la box entourant l'objet en pixel. Afin de connaître sa taille en mm sur le capteur nous faisons :

$$h_{img}(mm) = \frac{h_{img}(px) * h_{capteur}(mm)}{h_{capteur}(px)}$$

Nous pouvons donc trouver la distance de l'objet avec :

$$d_{obj}(mm) = \frac{focal(mm) * h_{obj}(mm) * h_{capteur}(px)}{h_{img}(px) * h_{capteur}(mm)}$$

Le problème est que la focal de l'objectif de la caméra n'a pas été trouvé. Elle a donc été mesuré en plaçant le feu (de 80mm de haut) à des distances connues de la caméra, et en notant la taille de la box retournée par le SSD. Les autres paramètres sont connus.

Distance Capteur → Feu (mm)	Hauteur de box (px)	Focale calculée (mm)
150	140	1,49
300	85	1,82
450	57	1,83
600	45	1,93

Tableau 6 Mesure de la focale

Ces différences dans la focale calculée viennent du fait que la caméra a une forte distorsion, donc l'objet aura une taille et forme différente sur le capteur en fonction d'où il est placé devant la caméra, même en restant à même distance de celle-ci.

Le deuxième facteur est que le SSD ne retourne pas forcément des box qui épouse parfaitement la forme de l'objet. Parfois les box sont sur dimensionné.

Finalement, une focal de 1.8mm a été choisi.

9.2.2. COULEUR DU FEU

A la base, tous les feux (rouge, vert, éteint) faisaient partie de la même catégorie dans le CNN, puis un traitement d'image par OpenCV déterminait la couleur du feu.

L'intensité lumineuse de la couche rouge de la partie supérieur du feu était analysée afin de savoir s'il était au rouge, et de même pour la couche verte du bas du feu.

Cette technique fonctionnait bien, mais le traitement par OpenCV prenant beaucoup plus de temps de calcul que le réseau neuronal, ce qui ralentissait toute la détection de signalisation.

Il a donc été choisi d'ajouter 2 classes au CNN afin de directement détecter les différents feux.

10. DETECTION D'OBSTACLES

La détection d'obstacle se fait grâce au capteur de distance à ultrason HC-SR04.

Afin de le faire fonctionner, une pulls de 10us doit lui être fournis sur son entrée Trigger. Le capteur va alors envoyer une rafale de 8 impulsions d'ultrasons et attendre l'écho. Le but d'envoyer une rafale est de générer un pattern particulier afin de pouvoir différencier un écho parasite de l'écho voulu.

Le HC-SR04 va alors mettre à 1 sa sortie Echo durant le nombre de temps que l'écho a pris à revenir.

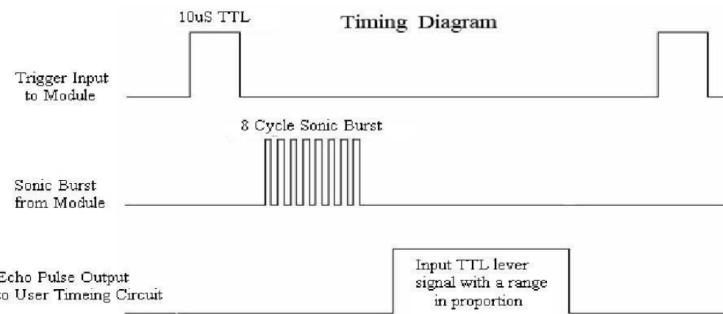


Figure 44HC-SR04 Timig Diagram Source: (24)

Pour connaître la distance, il faut alors effectuer le calcul :

$$d = \frac{t_{echo} * v_{son}}{2} = \frac{t_{echo} * 343}{2} [m]$$

Le thread gérant le capteur à ultrason va alors mettre à jour le dictionnaire « car_state » si un objet est trop près.

10.1. Problème rencontré

Les capteurs officiels ont un timeout si aucun echo n'est repéré et la sortie Echo est remis à 0 automatiquement. Cependant les copies du capteur n'ont pas toujours ce timeout et lorsqu'aucun echo n'est repéré, ils se bloquent et le seul moyen de recommencer une mesure est de couper l'alimentation du capteur.

Ce problème a été remédié en ...

11. PROBLÈMES RENCONTRÉS

RÉUNIR TOUS LES PROBLÈME ICI

- 11.1. Servo de guidage alimenté par le RPi -> Il s'arrêtait
- 11.2. Camera IR pose problème au niveau des couleurs
- 11.3. PWM pas correcte
- 11.4. Le RPi s'arrête
- 11.5. Mise a jour de l'Object detection API

12. AMÉLIORATIONS

12.1. Plaque de support du RPi

12.2. Suivit d'objet par image processing

12.3. Faire tourner OpenCV sur la GPU

12.4. Conduite de nuit

13. PLANNING

14. REMERCIEMENTS

Je tiens à exprimer mes sentiments de gratitude et de reconnaissance à Monsieur Pierre BRESSY, d'abord pour ses conseils et ses encouragements et ensuite pour m'avoir suivi durant toute la durée de ce travail de diplôme en mettant à ma disposition ses connaissances dans la gestion d'un tel projet, sa disponibilité, sa patience ainsi que son expertise technique. Un remerciement particulier aussi à Madame Claire MEYER et Monsieur Guillaume BANGALA, Monsieur Matthias HUSER pour leurs disponibilités, leurs conseils, et encouragements. Ensuite, j'exprime aussi mes remerciements à mes collègues Vincent BERGER, Rémy PAPAUTX, Thomas JOLIAT, Michael HUBER, pour leurs coopérations pendant la validation et les tests de l'application. Enfin, je remercie toute ma famille en particulier mon frère Monsieur Martial MBEFO et son épouse Tatiana MBEFO, pour m'avoir leurs encouragements et soutiens tout au long de ce projet.

15. BIBLIOGRAPHIE

1. Brushed Speed controller WP40 - Instructions. [Online] [Cited: 07 16, 2020.]
<https://asset.conrad.com/media10/add/160267/c1/-/gl/001399921ML02/mode-demploi-1399921-regulateur-de-vitesse-brushed-de-voiture-reely-wp-1040-brushed-charge-admissible-max-180-a-limite-moteur-nombre-de.pdf>.
2. Best Single Board Computers. [Online] [Cited: 07 16, 2020.]
<https://www.seeedstudio.com/blog/2019/11/20/best-single-board-computers-of-2019/>.
3. Rock Pi N10 vs Raspberry Pi 4 vs Jetson Nano. [Online] [Cited: 07 16, 2020.]
<https://www.seeedstudio.com/blog/2019/12/05/rk3399pro-vs-raspberry-pi-4-vs-jetson-nano-ai-and-deep-learning-capabilities/>.
4. Benchmarking Machine Learning. [Online] [Cited: 07 17, 2020.]
<https://www.hackster.io/news/benchmarking-machine-learning-on-the-new-raspberry-pi-4-model-b-88db9304ce4>.
5. Debian Boot-time on Raspberry PI. [Online] [Cited: 07 16, 2020.]
<https://raspberrypi.stackexchange.com/questions/320/what-is-a-typical-boot-time-for-the-standard-debian-distribution-on-a-typical-sd>.
6. Bringing Colour to Point Clouds. [Online] [Cited: 07 17, 2020.] <https://www.gim-international.com/content/news/bringing-colour-to-point-clouds-2>.
7. PiJuice HAT. [Online] <https://uk.pi-supply.com/products/pijuice-standard>.
8. Fashion MNIST Database. [Online] [Cited: 07 19, 2020.] <https://www.kaggle.com/zalando-research/fashionmnist>.
9. MobileNet, optimisation de la convolution pour les réseaux de neurones embarqués. [Online] [Cited: 07 19, 2020.] <https://www.quantmetry.com/mobilenet-optimisation-de-la-convolution-pour-les-reseaux-de-neurones-embarques/>.
10. BRESSY, Pierre. *ISP - LAB04: Convolution in 2 dimensions.* [PDF] 2019.
11. Howard, Andrew, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision. [Online] 04 17, 2017. [Cited: 07 19, 2020.] <https://arxiv.org/pdf/1704.04861.pdf>.
12. BIANCO, SIMONE. Benchmark Analysis of Representative Deep Neural Network Architectures. [Online] [Cited: 07 20, 2020.] <https://arxiv.org/pdf/1810.00736.pdf>.
13. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, YOLO, SSD. [Online] [Cited: 07 20, 2020.] <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>.
14. Surfaces occupées par les infrastructures routières. [Online] [Cited: 07 19, 2020.] http://piece-jointe-carto.developpement-durable.gouv.fr/REG074B/FONCIER_SOL/N_OCCUPATION_SOL/L_EMPRISE_ROUTE_R74/Fiche1-7-1.pdf.

15. Cas de croisements et largeur de chaussée. [Online] [Cited: 07 19, 2020.]
https://mobilitepietonne.ch/wordpress/wp-content/uploads/2017/07/06_2017_Fiche-info_Cas_de_croisement.pdf.
16. PyYAML Documentation. [Online] [Cited: 07 19, 2020.]
<https://pyyaml.org/wiki/PyYAMLDocumentation>.
17. Weng, Juyang, Cohen, Paul and Herniou, Marc. Camera Calibration with Distortion Models and Accuracy Evaluation. [Online] 10 10, 1992. [Cited: 07 21, 2020.]
<https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/camera%20distortion.pdf>.
18. TensorFlow Object Detection API. [Online] [Cited: 07 20, 2020.]
https://github.com/tensorflow/models/tree/master/research/object_detection/.
19. Models Built for the Edge TPU. coral.ai. [Online] [Cited: 07 20, 2020.]
<https://coral.ai/models/>.
20. COCO Dataset. [Online] [Cited: 07 20, 2020.] <https://cocodataset.org/>.
21. GitHub LabelImg. [Online] [Cited: 07 20, 2020.] <https://github.com/tzutalin/labelImg>.
22. TensorFlow and TensorFlow Lite on the Raspberry Pi. [Online] [Cited: 07 17, 2020.]
<https://www.hackster.io/news/benchmarking-tensorflow-and-tensorflow-lite-on-the-raspberry-pi-43f51b796796>.
23. TensorFlow models on the Edge TPU. [Online] [Cited: 07 21, 2020.]
<https://coral.ai/docs/edgetpu/models-intro/>.
24. HC-SR04 Datasheet. [Online]
<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>.
25. Réglementation signalisation routière. [Online] [Cited: 07 19, 2020.]
<https://www.signals.fr/cms/reglementation-signalisation-routiere.html>.
26. How big is a traffic light ? [Online] [Cited: 07 19, 2020.] <https://www.quora.com/How-big-is-a-traffic-light>.

16. NON-PLAGIAT

Je soussigné, Maxime CHARRIÈRE, atteste par la présente avoir réalisé ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Yverdon-les-Bains, le 24.07.2020

Maxime CHARRIÈRE

17. ANNEXES

17.1. Énoncé du TB

17.2. Clause de confidentialité

17.3. Fichier de configuration

17.4. SSD MobileNet V2 learning Notebook