

IMT LILLE DOUAI

COMPTE RENDU

PROJET



Méthodes appliquées pour la conception d'un modèle décisionnel d'optimisation de la conduite d'un véhicule réduit et autonome

6 décembre 2019

Groupe 1

Pierre MONTROEUL

Antoine FACCHINI

Groupe 2

Lorin LODIGIANI-DECHAMP

Thibault DOUCE

Introduction

Ce rapport est le résultat d'un mois de travail fait par 2 binômes. Il montre l'effort de chacun à produire un travail qui se veut itératif. On l'espère, il permettra à d'autres de se baser dessus pour pouvoir aller plus loin.

A travers ce rapport, nous espérons mettre en avant trois axes à suivre pour reprendre le projet. Le premier est de ne pas reproduire les erreurs que l'on a pu faire. Le deuxième est de présenter tout ce qui fonctionnait à la fin de ce mois.

Le troisième axe est fait tel qu'un groupe puisse bénéficier des idées que nous avions pour la suite du projet afin de proposer des pistes de démarrage et d'amélioration. Ceci se fera grâce à la vision du projet que nous avons suite à cette courte période.

Le rapport est découpé en de nombreuses parties et sous-parties avec des noms complets. Ces parties se veulent les plus indépendantes possibles afin de situer facilement chaque information.

Toujours dans cette optique, la solution finale est mise en avant avant d'exposer le parcours et les difficultés rencontrées.

Un binôme (groupe 1) s'est occupé du paramétrage de la voiture, des programmes permettant sa conduite autonome et de leur intégration dans la voiture. Le deuxième binôme (groupe 2) a étudié les équipements de la voiture.

Nous n'oubliions pas de remercier monsieur Wannous qui nous a confié ce projet et qui nous a guidé pendant ce mois.

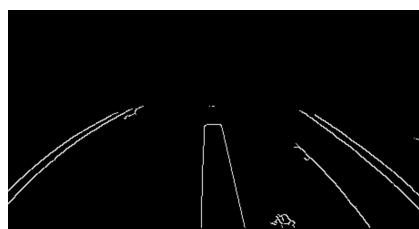


FIGURE 1 – La vision de la voiture RC

Table des matières

I Prise en main de l'existant	6
1 Mise en contexte	7
1.1 Le matériel	7
1.2 Le software	9
2 L'utilisation du matériel	10
II Le circuit	11
1 Lieu de mise en place	12
2 Problématique du scotch	13
3 Design du circuit	14
III Mise en place des environnements	16
1 De développement	17
1.1 Python	17
1.2 Environnement virtuel	17
1.3 Pytorch	18
1.3.1 CUDA	18
1.4 Tensorflow	19
1.5 Jupyter	19
1.6 Jupyter online	19
2 De production, la raspberry	21
2.1 Le hat	21
2.2 La raspberry	21
2.2.1 L'OS	21



2.2.2	La configuration du réseau	22
2.2.3	Swap	22
2.3	Les packages python	23
2.3.1	Les wheels vs les sources	23
2.3.2	Les principales installations	23
IV	Pré-traitement	25
1	L'image en entrée	26
2	Flou Gaussien	27
3	Détection de bords avec CANNY	28
4	ROI	29
5	Lower resolution & crop	30
6	Le résultat	31
7	Digressions	32
7.1	Calibration de la caméra	32
7.1.1	Création de la matrice de calibration	32
7.1.2	Distortion des images	33
7.1.3	Intégration	33
7.2	Bird eye	33
7.3	HSV	34
V	Prise de décision faible	35
1	L'idée	36
2	Hough line	37
3	Point de convergence	38
4	Problème d'optimisation de prise de décision	39

VI Prise de décision forte	40
1 L'idée	41
2 Les données	42
2.1 Capture de vidéos	42
2.2 Le labeling	42
2.2.1 Les labels	42
2.2.2 La méthode	42
2.3 Digressions	43
3 Définition du modèle	45
3.1 Objectifs	45
3.2 CNN	45
3.3 Modèle complet	46
3.4 Fonction de coût	46
4 Entraînement et évaluation	47
4.1 Entraînements	47
VII Interface et Déploiement	48
1 Objectifs	49
2 L'accéléromètre	50
3 Gestion de la direction et de la vitesse	51
3.1 Orientée objet	51
3.1.1 Initialisation	51
3.1.2 L'envoi de commande	51
3.1.3 Côté développeur	51
3.2 Calibration	52
3.3 Smoothing	52
4 Caméra	54
4.1 Optimisation de la récup	54
4.2 La classe PiRGBAnalysis	54
5 Problèmes physiques	55
6 Sauvegarde d'un modèle	56



VIII Des pistes d'amélioration	57
1 Le matériel	58
2 Le réseau de neurones	59
2.1 Le labeling	59
2.2 La structure	59
2.3 Le déploiement	59
Conclusion	61

Première partie

Prise en main de l'existant

1

Mise en contexte

L'année précédente, ce projet a été créé de toutes pièces grâce à des étudiants en UV Projet, comme nous actuellement. Nous avons donc démarré avec une voiture montée pour mener à bien notre objectif.

Il était donc primordial de s'imprégnier de leurs travaux pour être le plus efficace possible et optimiser notre temps de projet, celui-ci étant relativement court (de quatre semaines) alors que les gagnants de l'édition précédente du concours, l'équipe de Polytech Lille avait huit semaines.

La voiture de nos prédécesseurs est donc fonctionnelle et complète pour les fonctionnalités de base de la conduite autonome.

C'est le binôme 2 qui s'est chargé de cette prise en main.

1.1 Le matériel

Elle comporte le matériel électronique suivant :

- Raspberry PI 3 B +
- Hat Adafruit contrôleur de servo
- Caméra grand angle
- Batterie Externe
- Carte mémoire de 64 GB de classe 10

Le tout est installé sur une voiture de modèle CONQUEST 10ST XLR 2WD BRUSHLESS HELION-RC modifiée avec une planche de bois pour installer les différents composants, la caméra étant installée grâce à un support imprimé en impression 3D.

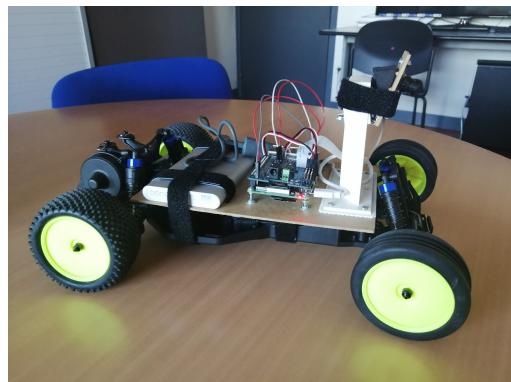


FIGURE 1.1 – Photo de la voiture

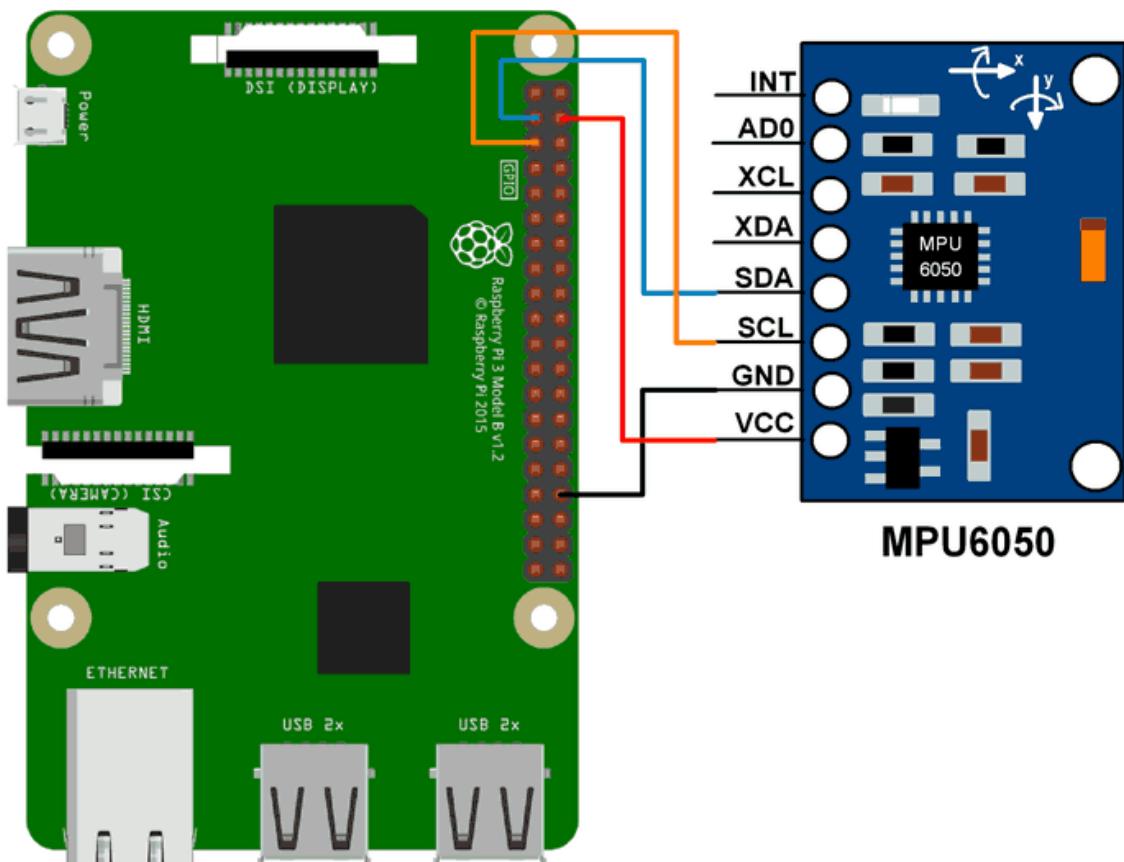


FIGURE 1.2 – Carte Raspberry

1.2 Le software

Dans leur rapport, il est indiqué qu'à la fin de leur projet, ils ont réussi à faire fonctionner la conduite autonome, grâce à leurs modèles.

Dans leur programme installé sur la Raspberry, qui était celui proposé par IronCar, il y avait un serveur lancé sur la Raspberry pour pouvoir accéder à une interface graphique via un ordinateur, dans le but de contrôler la voiture : manuellement, mode direction automatique, mode autonome et le mode Repos.

Les différents modes :

- Le mode manuel : contrôle à distance grâce à l'ordinateur.
- Le mode automatique direction automatique : la direction est déterminée par le modèle, mais la vitesse est contrôlée manuellement par ordinateur, cela permet d'essayer un modèle sans mettre en danger l'intégrité de la voiture.
- Le mode autonome, la voiture gère la direction et la vitesse grâce au modèle, on peut toujours décider de l'échelle de vitesse adoptée par la voiture.
- Le mode Repos permet d'arrêter la voiture.

Le code présent sur la Raspberry était le code proposé par Ironcar pour faire fonctionner la voiture autonome grâce à un serveur web.

Le code existant avait besoin d'un modèle pour le traitement de l'image, il a fallu donc regarder comment il était construit pour savoir ce que le modèle devait retourner.

Après recherche dans le code, le modèle doit fournir une direction et un indice de confiance. La variable direction correspond à la direction optimale que la voiture doit prendre pour rester sur la route. Cet indice correspond à la confiance accordée sur le traitement de l'image, dans le cas d'un indice de confiance bas, par exemple si les lignes sont mal détectées, il pourra être décidé de ralentir la voiture, dans le but d'éviter les mauvaises prises de décision.

Toutes ces informations sont explicitées dans la partie du rapport du binôme 1.

2

L'utilisation du matériel

Lors du projet, nous avons réutilisé le matériel déjà présent et installé par nos prédécesseurs car nous l'avons jugé largement suffisant pour pouvoir réaliser ce projet.

Il y a trois points de réglage auxquels il faut prêter attention :

- Le réglage de l'orientation des roues : par le passé, ces derniers étaient implémentés en dur dans le code. Désormais, un programme existe pour pouvoir le régler à tout moment. Il y a une seule et unique chose à laquelle il prêter attention : les valeurs choisies doivent correspondre au moment où les roues arrêtent de tourner (des valeurs au-delà risqueraient d'abîmer le moteur)
- Le réglage du moteur : la puissance de dernier est toujours codée en dur dans le programme. Pour le réglage de ce dernier, il faut faire attention à d'une part, ne pas le régler trop fort afin que la voiture n'aille pas trop vite (cela pose des problèmes de direction dans les virages) ou trop faible (le couple risque d'être par moment trop bas et donc que la voiture ne puisse pas redémarrer après avoir ralenti dans un virage)
- Le réglage de la caméra : il faut prendre garde à ce que l'angle de la caméra ne change pas. De plus, l'objectif de la caméra tourne de manière à changer la focal de l'objectif, il faut donc prendre garde à ce que la focal de l'objectif soit correctement réglé, sans quoi l'image sera floue.

Outre les réglages physiques de la caméra (focal et inclinaison), la caméra possède de nombreux réglages possibles au niveau du software (colorimétrie, saturation, résolution, ...). Ils sont tous recensés sur ce site : <https://picamera.readthedocs.io/en/release-1.10>. Pour notre part, nous avons choisi de mettre le mode d'exposition en automatique et de corriger la luminosité de l'image de manière à améliorer le contraste des bandes vis-à-vis du sol.

Deuxième partie

Le circuit

1

Lieu de mise en place

La salle E 104 N étant trop petite pour pouvoir accueillir un circuit complet et représentatif de l'ensemble des conditions auxquelles la voiture puisse être exposées, nous avons dû l'installer ailleurs.

Cette année encore, le personnel de la bibliothèque a bien voulu nous prêter le sous-sol de la bibliothèque à deux conditions :

- Respecter les règles de vie de la bibliothèque ainsi que de ne pas gêner les éventuels utilisateurs du sous-sol,
- Qu'à la suite de notre passage, la salle soit dans l'état exact où nous l'avions trouvée (notamment au niveau des traces de colle).

2

Problématique du scotch

Le binôme 2 a su que les scotchs de l'an dernier avaient laissé des traces de colle au sol, nous en avons recherché un nouveau (tout en faisant en parallèle un test avec les scotchs déjà existant).

D'après les recherches, que nous avons faites dans plusieurs magasins, le scotch le plus adapté au contrainte lié au lieu est du scotch de masquage (un scotch utilisé en peinture pour pouvoir peindre des surfaces de plusieurs couleurs différentes).

Malheureusement, ce scotch n'est pas vraiment adapté à nos besoins dans la mesure où il est généralement de couleur crème et mate. Il ne permet donc pas d'avoir un fort contraste avec le sol de la bibliothèque qui est en lino imitation bois claire.

Finalement les tests et les références des scotch existant ont montré que tant que le scotch est retiré dans les 15 jours, il ne laisse aucune marque.

Une commande sur internet (là où le choix est plus large) nous a été inenvisageable dans la mesure où ne devions commencer rapidement nos tests. Il s'agit d'une voie à exploiter lors de prochain PST si tenté que l'étude du circuit soit faite plus en amont que la nôtre.

Enfin, lors de la réalisation du circuit, il nous a manqué une dizaine de mètre de scotch jaune pour pouvoir le compléter. Un rouleau de remplacement a été trouvé chez Auchan (du scotch jaune de masquage, qui était ce qui ressemblait le plus à l'ancien scotch et qui n'était pas du scotch de fixation ou gaffer).

Après avoir trouvé le scotch qui était le plus adapté à notre situation, nous avons donc créé notre circuit selon quelques critères qui correspondent à ceux du concours Ironcar :

- Deux bandes extérieures et une bande centrale
- Largeur de la voie d'environ 1 mètre
- Longue ligne droite
- Virage large et serré

3

Design du circuit

Dans un but perfectionniste, nous avons volontairement complexifié les trajectoires. Les virages serrés permettant de confirmer le bon fonctionnement de la voiture. Une détection trop lente ou trop aléatoire n'aurait pas pu permettre de passer ces virages.

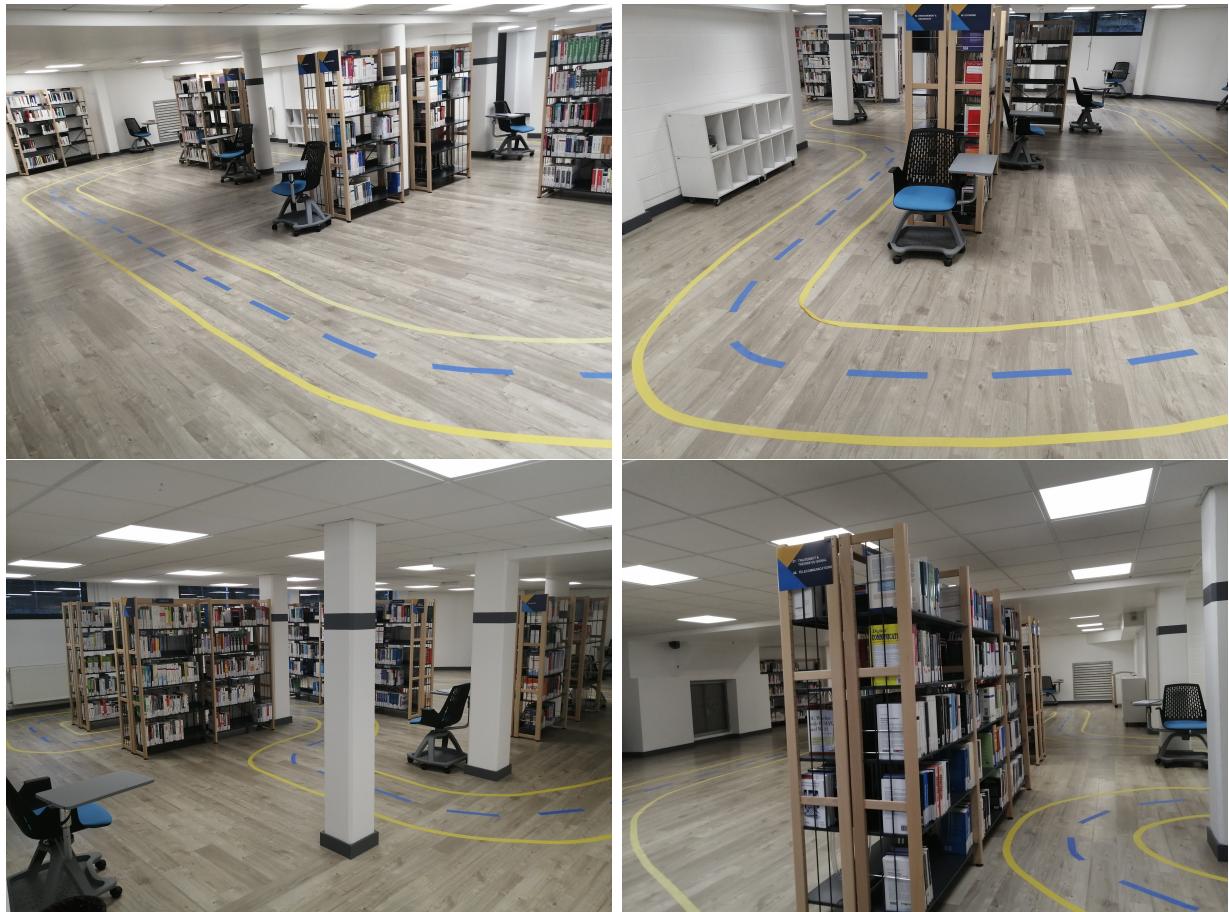


FIGURE 3.1 – Circuit de la bibliothèque

Troisième partie

Mise en place des environnements

1

De développement

La première étape du projet était de définir l'environnement de travail où le projet allait évoluer.

1.1 Python

Langage devenu incontournable pour l'analyse des données. Le choix de sa version s'est naturellement tourné vers la 3.7. La sous-version a peu d'importance. Dans certain cas, il est installé en 32 bits limitant la quantité de RAM possiblement allouée. C'est vite problématique lors d'un chargement complet de grands jeux de données.

L'installation Pour installer Python sous Windows, il faut se rendre [ici](#). Pour Linux, il est déjà installé.

1.2 Environnement virtuel

Pour conserver un environnement propre, durable et partageable, il est nécessaire de créer un environnement virtuel. Les packages installés seront propres à cet environnement. Il ne seront pas disponible pour le reste de l'OS.

On peut donc installer (ou désinstaller) une version précise d'un package sans se soucier des conséquences sur le reste de la machine. Mais aussi, on peut récupérer la liste des packages installés uniquement dans cet environnement permettant lors d'une mise en production par exemple, d'installer directement la liste.

La gestion des environnements virtuels est directement disponible à partir de la version 3.3 de Python.

```
1      $ python3 -m venv myvenv
2      $ source myvenv/bin/activate # Pour Linux
3      $ myvenv\Scripts\activate # Pour Windows
4
5      $ which python # vérifier l'activation
```

FIGURE 1.1 – Créer, activer un environnement virtuel

Fraîchement créé, il est utile de mettre à jour les composantes d’installation de packages. Si *pip* et *setuptools* ne sont pas à jour, il y aura souvent des erreurs lors de l’installation de packages.

```
1      $ pip install --upgrade pip setuptools
```

FIGURE 1.2 – Mettre à jour des packages avec pip

1.3 Pytorch

Pytorch a été choisi initialement comme framework de deep learning pour son aspect orienté objet. Son installation est très largement guidée sur les machines traditionnelles¹.

Après un peu d’utilisation, nous sommes rendus compte que notre choix aurait dû plutôt se tourner vers Tensorflow. Pytorch est de plus en plus utilisé dans le monde de la recherche, à tel point qu’il est plus utilisé que Tensorflow². Cependant Tensorflow est plus utilisé dans l’industrie. Bien que le projet ne se place pas dans un cadre industriel, on peut tirer un avantage de choisir une solution pensée pour une grande variété d’environnements et de contraintes.

1.3.1 CUDA

Dès qu’une machine possède une carte graphique Nvidia, il est indispensable de rendre compatible n’importe quel framework de deep learning avec CUDA. Ceci pour des questions de performances et de rapidité du traitement de l’information. Une nouvelle fois, l’installation est bien documentée³.

-
1. <https://pytorch.org/get-started/locally/>
 2. <http://penseeartificielle.fr/meilleur-framework-machine-learning-2019/>
 3. <https://docs.nvidia.com/cuda/index.html>

1.4 Tensorflow

Tensorflow est un autre framework de deep learning. À la date de ce projet, la version 2.0 est sortie. Cependant sur Raspbian, il n'y a pas de wheels officielles⁴.

La dernière version n'était pas nécessaire, nous avons opté pour la 1.14.

1.5 Jupyter

Jupyter est un outil apprécié dans le domaine de la data science. Il peut être vu comme un IDE dans le navigateur principalement orienté utilisateur. Les outils présents dans des IDE pour programmer dans de gros projets comme le *refactoring* sont assez peu développés. En revanche, les sorties (texte, graphique ou barre de progression) sont optimisées avec des widgets.

Il permet de sectionner le code en cellules. Ces cellules peuvent être exécutées dans le désordre et plusieurs fois. L'état d'une cellule est partagé avec toutes les autres, ce qui permet d'utiliser les variables dans tout le notebook. Si l'on prend l'exemple d'un dataset, il ne sera à charger qu'une seule fois sur toute la période d'expérimentation et plusieurs cellules pourront se servir de son contenu pour réaliser des calculs différents (à condition de ne pas modifier le dataset lors d'un calcul, évidemment).

Jupyter est un package python à installer avec *pip*. Pour lancer jupyter, il faut dans un terminal executer la commande :

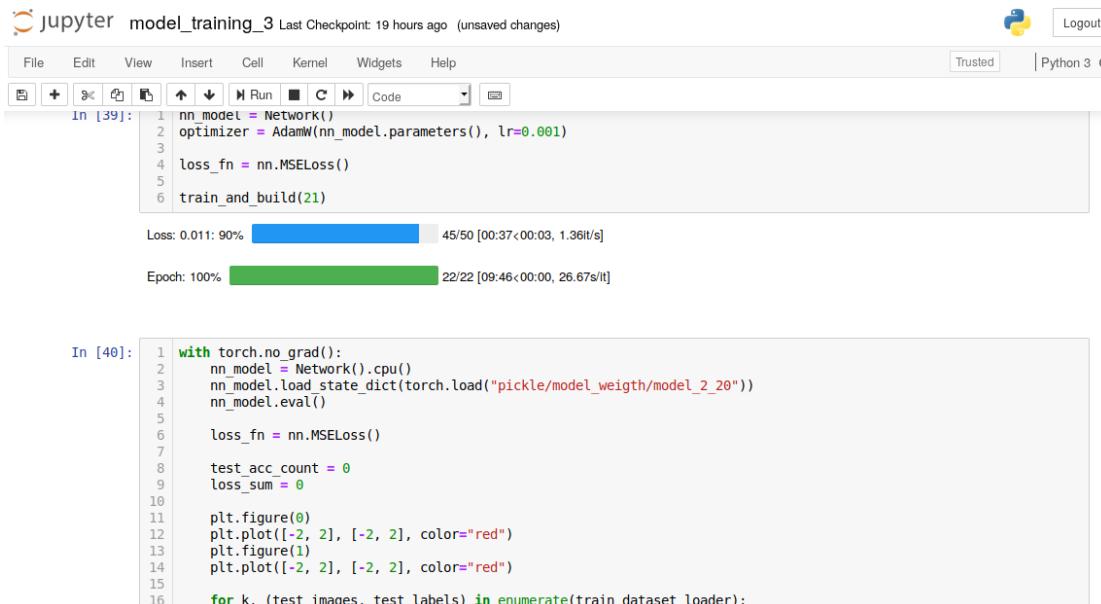
`$ jupyter notebook` s dgcfhvtlèlilmh,fk ;tjyoxcdgl ; et pour windows ?

1.6 Jupyter online

Jupyter a été pensé pour être utilisé de manière distante. Pour le groupe 1, il a été décidé de l'utiliser sur un serveur dédié.

Nous (le binôme 1), avons travaillé ensemble sur la voiture et sur son développement. Pour travailler ensemble, il fallait avoir le même environnement. Une personne avait son ordinateur sous Windows et l'autre sous Ubuntu et cela aurait pu compliquer la tâche. En utilisant un serveur dédié nous avons pu faire face aux problèmes de compatibilité, de version ou encore d'installation concernant les packages ou encore Python. Toutes nos versions étaient identiques.

4. <https://www.piwheels.org/project/tensorflow/>



The screenshot shows the Jupyter Notebook interface. At the top, there's a toolbar with File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, Logout, and a Trusted badge. Below the toolbar, there are two code cells labeled In [39] and In [40].

```
In [39]:
```

```
1 nn_model = Network()
2 optimizer = AdamW(nn_model.parameters(), lr=0.001)
3
4 loss_fn = nn.MSELoss()
5
6 train_and_build(21)
```

Output: Loss: 0.011: 90% 45/50 [00:37<00:03, 1.36it/s]

```
In [40]:
```

```
1 with torch.no_grad():
2     nn_model = Network().cpu()
3     nn_model.load_state_dict(torch.load("pickle/model_weight/model_2_20"))
4     nn_model.eval()
5
6     loss_fn = nn.MSELoss()
7
8     test_acc_count = 0
9     loss_sum = 0
10
11    plt.figure(0)
12    plt.plot([-2, 2], [-2, 2], color="red")
13    plt.figure(1)
14    plt.plot([-2, 2], [-2, 2], color="red")
15
16    for k, (test_images, test_labels) in enumerate(train_dataset_loader):
```

FIGURE 1.3 – L'interface de Jupyter

Le serveur s'est également révélé utile car il fallait avoir recours à une assez grande puissance de calcul. Quand on pouvait utiliser nos cartes graphiques pour les calculs, on utilisait nos ordinateurs mais dans tous les autres cas, ils étaient effectués sur ce serveur. Contrairement à nos ordinateurs, le serveur fonctionnait jour et nuit sans interruption. Nous pouvions également lancer des calculs longs sans nous soucier d'avoir notre ordinateur allumé ou éteint.

Pour déployer rapidement Jupyter, nous avons utilisé Docker⁵. Le docker-compose⁶ est fourni dans le dépôt Github.

5. <https://docs.docker.com/v17.12/install/#server>
6. <https://docs.docker.com/compose/install/>

2

De production, la raspberry

2.1 Le hat

Les servos moteurs sont pilotables avec un signal PWM. C'est de la programmation par interruption : tant que l'on ne change pas les PWM, ils resteront les mêmes et donc la vitesse et la direction prises ne changeront pas non plus.

Pour créer les 2 signaux PWM, un hat¹ est connecté. Un package Python permet de simplifier les interactions. C'est *Adafruit_PCA9685*. Au début, on choisit une fréquence (dans notre cas 60 Hz) et ensuite on appelle la méthode² :

```
1   from Adafruit_PCA9685 import PCA9685
2
3   pwm = PCA9685()
4   pwm.set_pwm_freq(60)
5   pwm.set_pwm(pin , 0 , pwm)
```

2.2 La raspberry

2.2.1 L'OS

Pour Raspberry, le dernier OS Raspbian a été utilisé, plus particulièrement la version Lite qui est sans environnement de bureau. L'utilisation se fera directement en lignes de commande via SSH. Pour installer Raspbian, il faut une carte MicroSD. Avec un créateur de clé USB

-
1. Équivalent au mot shield pour les Arduinos
 2. Code source : https://github.com/adafruit/Adafruit_Python_PCA9685

bootable, il faut ensuite mettre l'image ISO téléchargée³ dessus. Dans certains cas, le créateur de clés bootables ne formate pas la carte pour mettre le bon système de fichier. Il faut choisir etx4 dans ce cas.

2.2.2 La configuration du réseau

Connexion filaire

Pour que la Raspberry puisse accéder à l'Internet, la solution était de la connecter en filaire à un ordinateur et sur celui ci de faire un partage de connexion entre l'interface wifi et celle filaire.

Mettre une IP fixe était la solution la plus simple pour faire une première connexion. Pour activer l'interface, il suffit de ne modifier qu'un seul fichier⁴. Ensuite lors du démarrage, l'IP sera directement allouée.

Connexion WIFI

Pour les tests, la connexion filaire n'allait plus pouvoir se faire puisque la voiture allait se déplacer. Donc sur la Raspberry, un hotspot a été mis en place⁵. C'était nos ordinateurs qui devaient se connecter à la voiture. Il y a un DHCP pour un adressage dynamique. Seul l'IP de la voiture était fixe pour se connecter en SSH sans problème. Comme cela les tests pouvaient se faire très facilement sur n'importe quelle plate-forme.

Le SSH au premier démarrage

Une fois l'OS mis sur la carte, il faut indiquer que l'on souhaite activer le *deamon* du SSH. Raspbian offre une solution simple pour l'activer lors du premier démarrage. Il suffit de mettre un fichier vide avec comme nom *ssh* dans la partition *boot*. Les logins sont par défaut *pi* avec comme mot de passe *raspberry*. Par sécurité, ce mot de passe peut être changé.

2.2.3 Swap

Le modèle de la Raspberry, la 3B+ possède qu'un seul Go de mémoire vive. Elle peut vite être pleine et amener à des plantages ou autres problèmes. Pour éviter cela, une augmentation de la taille du SWAP sera bénéfique. Mais attention, si le swap est trop grand et trop chargé, les processus consommant cette mémoire vive seront très lents.

3. <https://www.raspberrypi.org/downloads/raspbian/>

4. Voir github

5. Voir github

```

1      $ sudo nano /etc/dphys-swapfile
2
3          # cf ligne :
4          CONF_SWAPSIZE=1024 (taille en Mo)

```

FIGURE 2.1 – Modification de la taille du Swap

2.3 Les packages python

Pour l'installation des packages de Python, nous avons choisi d'utiliser exclusivement *pip*.

2.3.1 Les wheels vs les sources

Lors d'une installation d'un nouveau package, *pip* télécharge une *wheel* puis l'installe. Dans une *wheel*, il peut avoir du code compilé.

La Raspberry a une architecture de processeur différente que nos PCs. C'est un ARMv7l. Le code doit donc être compilé pour ce type de processeur. La plupart des packages ont des wheels compilées pour ce processeur. Elles sont fiables et stables. Cependant, pour de gros packages comme Pytorch, il n'y a pas de wheels officielles.

Face à ce problème, 2 options sont possibles. La première est de chercher une *wheel* non officielle en espérant qu'elle sera fonctionnelle pour notre machine et ses paramètres. La seconde est de compiler les sources.

La compilation peut se faire directement sur la Raspberry ou en cross-compilation, sur un ordinateur plus puissant. Les 2 cas possèdent des points positifs et négatifs. Pour la compilation sur la plateforme où elle sera installée, il n'y aura pas de problème de compatibilité avec les dépendances. Mais la compilation sera très lente. Pour Pytorch, cela s'est compté en plusieurs heures (10h environ). Pour ce qui est de la cross-compilation, cette possibilité n'a pas vraiment été envisagée pour son aspect complexe de mise en place. Si cette méthode est déjà maîtrisée par le binôme, nous ne pouvons que conseiller ce moyen.

2.3.2 Les principales installations

Installer OpenCV2

Ce package est utilisé pour le traitement des images dans la phase de pré-traitement. Il peut être directement installé via *pip*

Installer Pytorch

L'installation de Pytorch a été rendue longue par l'absence de Wheel officielle (avec Pip) et non officielle. Nous avons donc téléchargé le fichier source de Pytorch dans la version 1.3 et nous avons compilé ces sources sur la carte.

Installer Tensorflow

Tensorflow est disponible sur *pip* et peut être installé avec ses dépendances directement sur la voiture. Pour installer la version 1.14, nous avons utilisé la commande suivante :

```
1 $ pip install tensorflow==1.14
```

FIGURE 2.2 – Installation de Tensorflow

Les installations restantes

Les autres packages nécessaires sont principalement Numpy et OpenCv. Des dépendances sont nécessaires pour la plupart des packages, elles sont très souvent installées en même temps que le package principal. La liste entière des packages installés sur la voiture est disponible dans le fichiers *requirements.txt*. Ce fichier est obtenu grâce à l'export de la commande *pip freeze* et permet d'installer la liste entière également grâce à *pip* en une ligne de commande :

```
$ pip install -r requirements.txt
```

Quatrième partie

Pré-traitement

1

L'image en entrée

Le principal but est de récupérer uniquement les lignes, qu'importe le type du sol et la couleur des bandes. Il doit y avoir la même sortie dans tous les cas.



FIGURE 1.1 – L'image à la sortie de la caméra

Le programme donne une image sous forme d'une matrice de pixels qui n'a pas été compressée. La compression nécessite non seulement des ressources mais est dans ce cas complètement inutile car l'image doit être décompressée pour le traitement sans jamais être enregistrée.

Nous avons testé plusieurs résolutions d'image avec différents ratios. Le compromis entre taille et qualité a été de choisir du 456 x 228 pixels.

Par la suite, les chapitres sont dans le même ordre que la chaîne du pré-traitement effectué.

2

Flou Gaussien

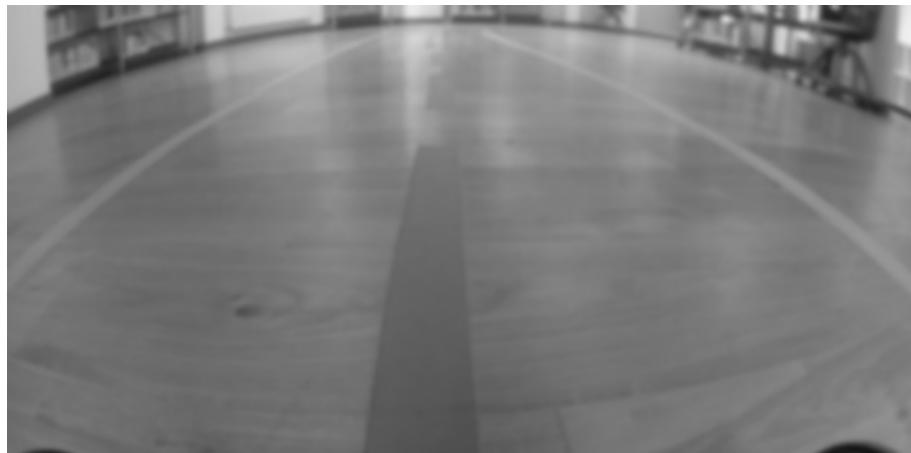


FIGURE 2.1 – L'image après le flou gaussien

Il fallait lisser les images récupérées afin de supprimer les imperfections qui pourraient se trouver dessus. Une jointure entre deux lattes de parquet devait disparaître pour ne pas être détectée comme une bordure. Ce qui permet de faire abstraction théoriquement du type du sol et de sa texture.

La méthode du flou Gaussien (Gaussian Blur) de la bibliothèque OpenCV permet ce lissage et donne avec une taille de kernel de 3 par 3 (lissée par groupe de 9 pixels), une image suffisamment lisse pour la suite. Le 0 indique que nous ne souhaitons pas de décalage.

```
1     blur = cv2.GaussianBlur (image , (3,3) , 0)
```

FIGURE 2.2 – La fonction du flou Gaussien

3

Détection de bords avec CANNY

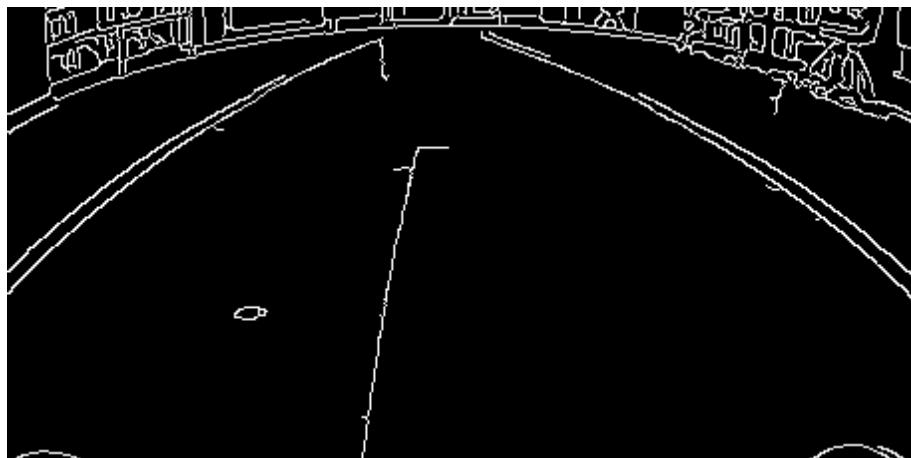


FIGURE 3.1 – La détection de bord

C'est avec la détection de bords que nous allons faire abstraction des couleurs. Il ne détectera que le passage d'une nuance à une autre, du sol à une bande et inversement. Nous avons utilisé la fonction *canny()*.

```
1 canny = cv2.Canny(image, 20, 100)
```

FIGURE 3.2 – La fonction Canny

L'étape de flou Gaussien a son importance pour cette fonction pour ne pas détecter comme "bord" tous les changements mineurs. Les nombres 20 et 100 sont les seuils minimaux et maximaux pour la détection d'un bord.

4

ROI

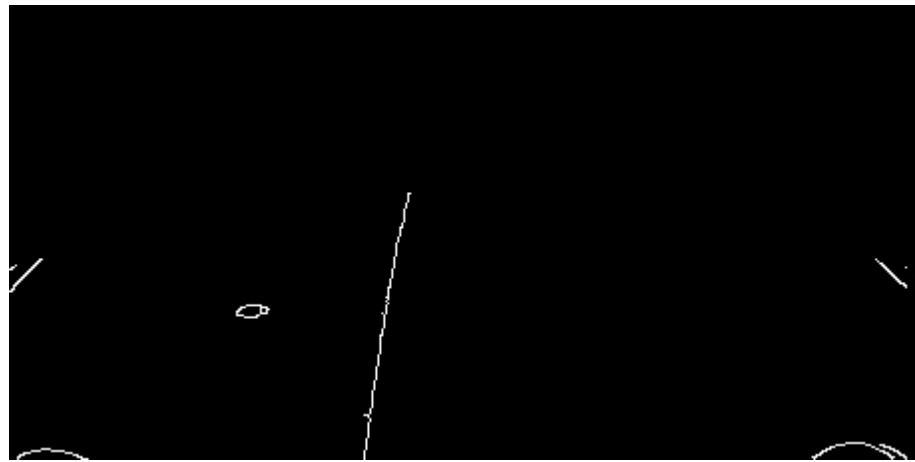


FIGURE 4.1 – La partie de l'image restante après la sélection ROI

La totalité de l'image n'est pas forcément bonne à prendre en compte et notre voiture peut vite se retrouver à détecter des éléments n'ayant pas lieu d'être pris en compte. Pour ceci, nous avons mis en place une zone d'intérêt ou ROI, *Region Of Interest* en anglais.

Le choix de la zone devait éviter l'analyse de bords éloignés de la voiture, sur toute la périphérie de la caméra. La zone conservée était représentée par un polygone.

```

1     mask = np.zeros_like(image)
2     cv2.fillPoly(mask, (points_poly), 255)
3     masked_image = cv2.bitwise_and(image, mask)

```

FIGURE 4.2 – Sélection de la région d'intérêt (ROI)

5

Lower resolution & crop

À ce moment, l'image est une matrice binaire montrant juste les bordures détectées. La résolution de cette image est largement trop grande pour les besoins. En réduisant la largeur et la hauteur par 2, l'information conservée semble très suffisante. Pour ne pas perdre certains pixels blanc dans la réduction, on a choisi une méthode qui calcule la couleur moyenne sur la surface du nouveau pixel.

```

1     image = cv2.resize(image, (0,0), fx=0.5, fy=0.5,
2                           interpolation=cv2.INTER_AREA)
```

FIGURE 5.1 – La fonction de redimensionnement

Il reste une étape pour réduire la taille du réseau de neurones. Avec le masque de la ROI, il n'y a que des pixels noirs dans la partie haute de l'image, ils ne contiennent aucune information. Il y a aussi un problème entre la caméra et certaines étapes de pré-traitement résultant une petite bande blanche à droite de l'image. Comme elle apparaît et disparaît de manière aléatoire, elle peut fausser le réseau, il faut donc l'enlever.

Une image est ici une matrice. Pour recadrer une image, il faut faire un *slicing*. Il est à noter que le pixel (0, 0) est tout en haut à gauche et le pixel (height, width) est en bas à droite.

```

1     width = image.shape[1]
2     # enlève les 45 pixels du haut et les 5 pixels à droite
3     image = image[45:, :width-5]
```

FIGURE 5.2 – Recadre l'image

6

Le résultat

L'image d'exemple est réelle mais l'on se rend compte que la qualité est mauvaise. La qualité de détection peut se révéler assez capricieuse.

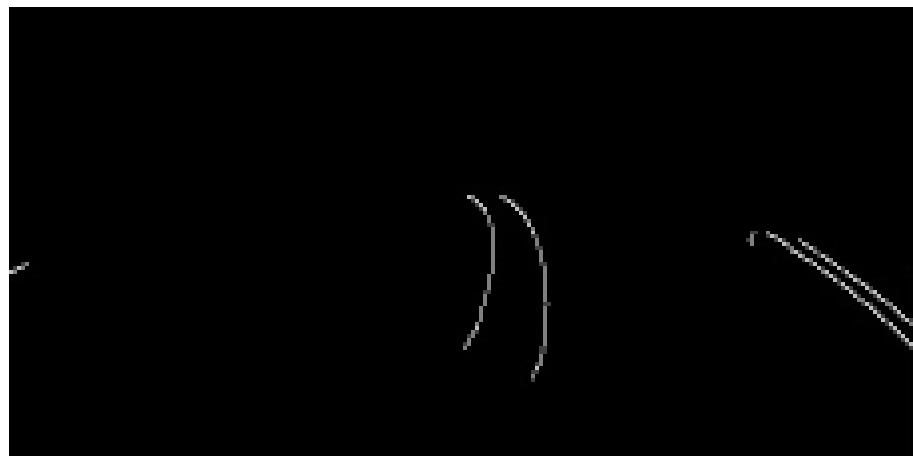


FIGURE 6.1 – Une détection de bord de qualité correcte

7

Digressions

Cette partie met en exergue des points sur lesquels nous avons tenté des choses mais qui n'ont à elles seules débouchées sur rien.

Nous avons tenté d'autres méthodes d'analyse d'image. On a tenté de calibrer la caméra pour enlever les distorsions causées par le *fish eye*. Nous avons aussi tenté de supprimer la perspective en passant en *bird eye*. Enfin, on a voulu améliorer l'image, plus particulièrement le contraste entre le sol et les bandes.

7.1 Calibration de la caméra

La calibration passe par 2 étapes. La première est la recherche de la calibration. La deuxième est l'application de celle-ci sur de nouvelles images.

7.1.1 Création de la matrice de calibration

Pour créer cette matrice, il faut faire un grand nombre de photos avec un échiquier à différentes distances et angles de la caméra.

Ensuite, ces images doivent être analysées pour détecter les coins des cases et voir leur courbure. Le nombre de photos avec des coins détectés doit être d'au moins 20 avant d'avoir des résultats confiants. Pour donner un ordre d'idée, rien n'a été détecté sur la moitié des photos prises.

Le nombre de cases présentes est un paramètre à donner à la fonction de détection sous la forme d'un tuple (lignes, colonnes), par exemple (4, 3). Pour la même image, il n'est pas rare de ne pas arriver à détecter (3, 3) mais d'arriver à détecter (4, 5) ou autre chose. Afin de maximiser le nombre de résultats trouvés, nous avons fait une boucle pour chaque image de 5x5 à 3x3 cases qui est le nombre de cases minimales.

En donnant la liste des coins trouvés à la fonction de calibration, on récupère les matrices pour enlever la distortion.

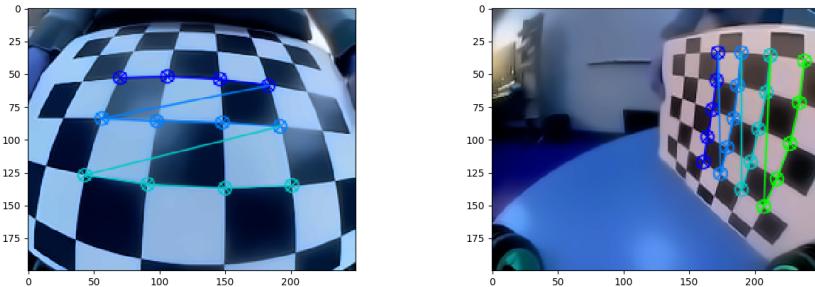


FIGURE 7.1 – Photos d'échiquier pour la calibration

7.1.2 Distortion des images

La distorsion en elle même est juste un *map* de l'image selon x et y.

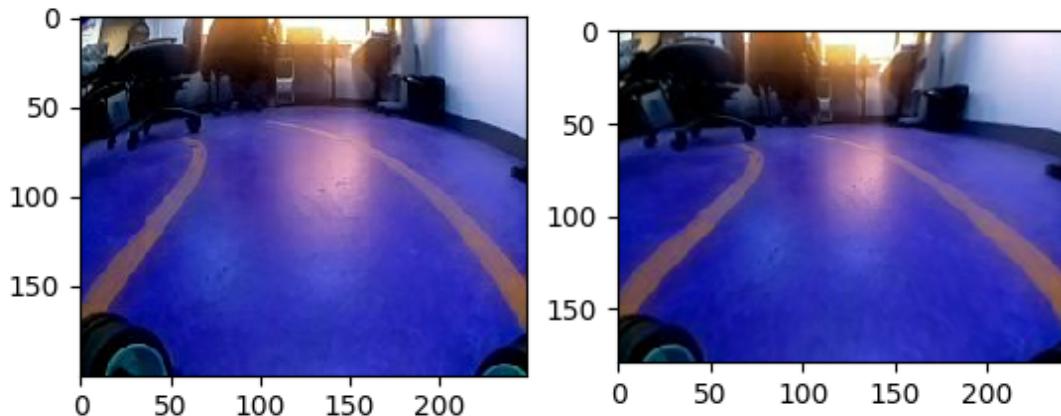


FIGURE 7.2 – Une image après distortion

7.1.3 Intégration

Au long du projet, nous nous sommes rendu compte que ce calcul était assez peu important comparé au temps de calcul requis. On a donc abandonné l'idée.

7.2 Bird eye

Nous avions une image avec une perspective et l'idée était d'obtenir une image plus compréhensible en supprimant cette perspective.

La méthode *bird eye* permet de choisir des points de départ et ensuite de tordre l'image pour qu'elle arrive aux points d'arrivée. Concrètement, elle simule une vue de haut de la situation actuelle en déformant l'image.

Le problème principal de cette transformation est la dégradation importante de la qualité. Elle amplifiait par exemple les reflets et imperfections dans la partie haute de l'image. En plus pour conserver la résolution de chaque pixel, il fallait augmenter de manière importante la taille de l'image et laisser des zones noires sur les côtés. Ces bandes noires étaient dues à la torsion de l'image.

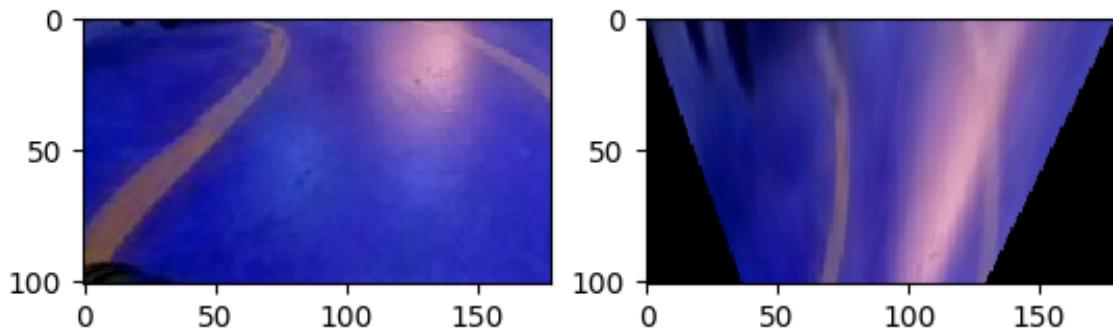


FIGURE 7.3 – La transformation *bird eye*

7.3 HSV

Lors des premières prises de photos, il y avait de nombreux reflets et des couleurs de mauvaises qualités. Une piste abordée a été de passer d'une représentation des couleurs RGB à HSV¹. Cependant en affichant chaque canal du HSV, aucun ne montrait une séparation satisfaisante entre le sol et les lignes.

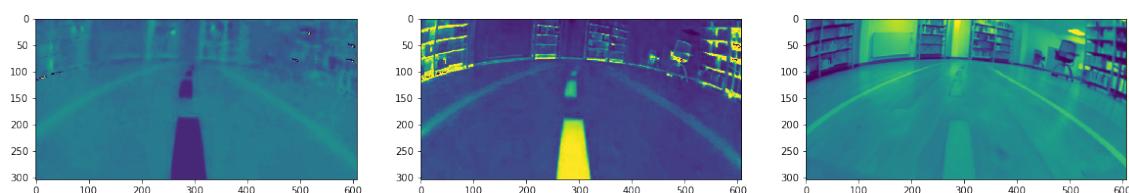


FIGURE 7.4 – Les 3 canaux du HSV

1. https://fr.wikipedia.org/wiki/Tint%C3%A9te_Saturation_Valeur

Cinquième partie

Prise de décision faible

1

L'idée

Une étape supplémentaire d'analyse d'image permettait de rassembler des pixels en lignes droites. L'idée était de faire une prédiction à partir de ces segment détectés.

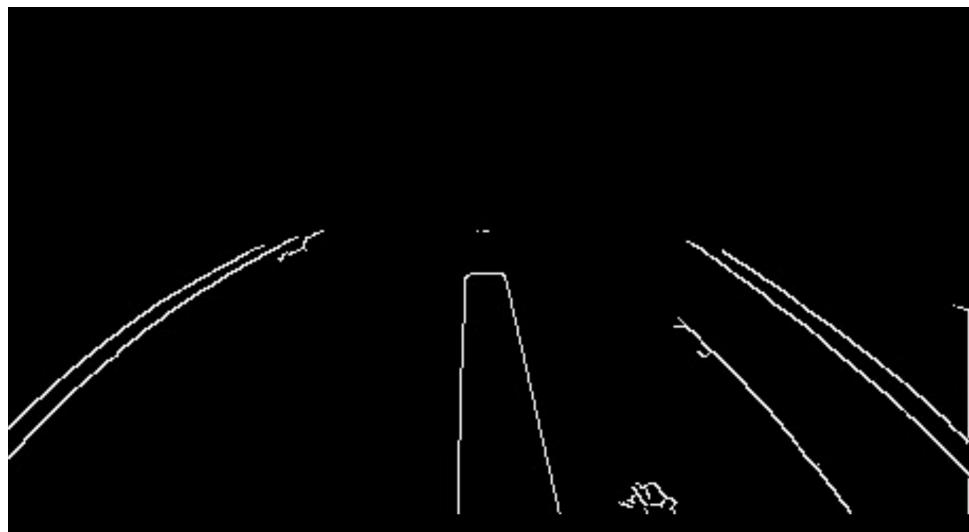


FIGURE 1.1 – Détection de lignes

2

Hough line

Cette étape supplémentaire est la fonction *Houghline*. Il existe 2 implémentations de ce traitement. La plus rapide d'exécution est celle dite probabiliste.

La liste des paramètres peut être trouvée ici. Les paramètres que l'on a le plus changés sont *minLineLength* et *maxLineGap*. Ils permettent respectivement de choisir la longueur minimale d'une ligne avant qu'elle soit acceptée et de choisir l'espace maximal avant de regrouper 2 lignes en une seule.

```
1     lines = cv2.HoughLinesP(  
2         image , 3, np.pi/180, 100, np.array([]),  
3         minLineLength=50, maxLineGap=30  
4     )
```

FIGURE 2.1 – La fonction HoughLinesP pour la détection de segments

3

Point de convergence

La méthode la plus intuitive que l'on a eue a été de regarder à quelle position le point de fuite se trouvait en haut de l'image. Ensuite, une moyenne pondérée par la longueur de la ligne est faite et donne un point objectif.

Ce point d'objectif est passé dans une fonction que l'on pourrait appeler fonction d'activation. Elle transforme la coordonnée en x du point normalisé (entre -1 et 1) en un angle avec la fonction arc tangente. Il y a ensuite un coefficient pour correspondre mieux à la commande de l'interface qui va de -1 à 1.

Il est nécessaire d'avoir un angle et non une coordonnée d'un point puisque c'est une direction que la voiture appliquera.

```
1      x = (x-width/2)/width
2      direction = 2.5 * arctan(x)
```

FIGURE 3.1 – Fonction d'activation

4

Problème d'optimisation de prise de décision

Les résultats furent assez variables. Les conditions initiales avaient un très grand impact. En positionnant la voiture à un endroit précis du circuit avec un certain angle, la voiture était capable de faire un grand nombre de tours (plus de 10) sans sortir du circuit. Cependant dans l'immense majorité, la voiture sortait du circuit.

Nous voulions un modèle plus robuste et plus universel. Qu'il soit capable de prendre la bonne décision malgré une situation inhabituelle ou lors d'une mauvaise détection de bords.

Sixième partie

Prise de décision forte

1

L'idée

Pour la prise de décision forte, nous avons opté pour du *machine learning* labélisé. Plus particulièrement un réseau de neurones avec des couches convolutives. Notre objectif était d'être plus robuste quand le résultat du pré-traitement était mauvais. Par exemple, quand une seule ligne est détectée.

2

Les données

2.1 Capture de vidéos

La prise de décision se fait uniquement par l'intermédiaire du flux vidéo. Les étapes de pré-traitement sont importantes pour réduire le plus possible la taille du réseau de neurones.

2.2 Le labeling

Il fallait créer un *dataset* qui corresponde bien aux situations que la voiture allait rencontrer. Nous avons enregistré des vidéos à la sortie de la sélection de la ROI dans la chaîne de pré-traitement. Il y a eu près de 15 minutes de vidéo à 30 images par seconde. Ce qui donne presque 30 000 photos.

2.2.1 Les labels

Chaque image aura 2 labels, 2 valeurs réelles. Le premier label est l'angle avec comme 0 la direction tout droit, -1 tout à gauche et 1 tout à droite. Les directions extrêmes sont choisies avec l'angle que la voiture peut faire, soit environ 45 degrés. Le second label est la vitesse comprise entre 0 et 1.

2.2.2 La méthode

Avec 30 000 photos et 4 personnes. Il fallait trouver une méthode pour labéliser rapidement et correctement. Rentrer des valeurs à la main non seulement est une méthode très lente mais surtout comme savoir que -0.5 est une bonne valeur pour la direction ou non ?

Le développement d'un logiciel permettant d'avoir une méthode visuelle a permis de les analyser très rapidement. Les images arrivent dans le même ordre que dans la vidéo. Chaque

nouvelle image n'a que peu de modifications avec la précédente. Ce qui nécessite peu de temps pour comprendre et choisir les labels. Nous faisions environ 2 images par seconde. Donc à 4 pour 30 000 photos, nous mettions un peu plus d'une heure.

L'interface

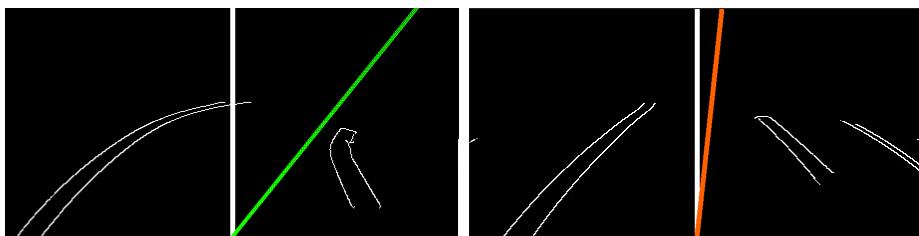


FIGURE 2.1 – 2 exemples de labeling

Après plusieurs séances de labeling, nous sommes arrivés à cette interface utilisant des coordonnées polaires. On voit sur la fenêtre :

- Une ligne blanche indiquant la direction de la voiture si elle allait tout droite, la direction 0.
- Une ligne infinie passant par la position actuelle de la voiture et la souris servant pour la label de la direction.
- Une couleur pour la ligne infinie en fonction de la distance entre la position de la voiture et la souris. Elle est bleu quand elle est proche de 0 ou de 1. Ensuite elle évolue de vert à rouge en allant de 0 à 1.

2.3 Digressions

Au début, on utilisait des coordonnées cartésiennes avec en abscisses, la direction et en ordonnées, la vitesse. Le réseau convergeait bien mais lors de l'application sur la voiture, le comportement à l'approche de certains virages était inadapté. En voyant le virage, nous cliquions instinctivement dans la direction où les lignes allaient. Donc quand le virage pointait en haut à gauche, nous cliquions dans le coin, grandes valeurs en abscisse et en ordonnée. Ce qui donnait que la voiture tournait le plus possible avec la plus grande vitesse. D'où le passage en coordonnées polaires.

Avec cette méthode en coordonnées cartésiennes, nous avons également été soumis aux problèmes de changement de direction en pleine ligne droite. Le modèle entraîné permettait à la voiture de considérer qu'une ligne droite était propice à une accélération. Néanmoins, lors d'une mauvaise détection des lignes, la voiture pouvait avoir envie de tourner en pleine ligne droite. Un changement de direction brusque allié à une vitesse élevée entraînait quasiment instantanément une sortie de piste aussi inattendue que surprenante.

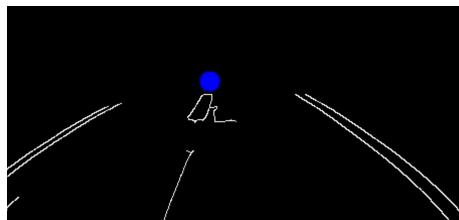


FIGURE 2.2 – L'ancienne méthode de labeling



FIGURE 2.3 – Image du circuit générée par ordinateur (avant et après recadrage)

3

Définition du modèle

Notre réseau de neurone s'est voulu le plus léger possible à embarquer sur la voiture. Notre expérience en pré-traitement de l'image nous ont finalement encore plus servi dans cette partie plutôt qu'avant.

3.1 Objectifs

La voiture a pour but d'assurer sa conduite autonome sur tous les circuits. Mais comme nous sommes dans un contexte d'informatique embarqué et en plus en temps réel, le réseau de neurones doit être le plus petit possible.

3.2 CNN

Le modèle conçu a été réalisé une première fois grâce à la bibliothèque Pytorch. Suite à des problèmes de mise en production sur la voiture, nous avons reproduit le modèle à l'identique grâce à la bibliothèque Tensorflow.

Le modèle créé comporte trois couches de convolution (CNN pour Convolutional Neural Network). Nous avons choisi un modèle convolutif car c'est la meilleure méthode pour l'analyse d'images.

Il est à noté qu'un CNN tend à être utilisé de bout en bout. Généralement, il n'y a quasiment pas de pré-traitement à faire, il est mis en place dans le CNN. Mais voulant réduire la taille, nous avons opté pour un pré-traitement assez gros. Sans tester les performances en utilisant qu'un CNN.

3.3 Modèle complet

Après les couches de convolutions, il faut *aplatir* la couche pour ensuite passer dans un réseau classique (*fully connected* et finalement avoir 2 sorties, vitesse et direction

3.4 Fonction de coût

Pour que le réseau de neurones apprenne à faire le lien entre l'image et les sorties, nous devons indiquer une fonction de coût¹, *loss function*. Nous hésitions entre optimiser en réduisant la valeur absolue de l'erreur ou l'erreur au carré. Mais comme, il fallait éviter au maximum les valeurs aberrantes, l'erreur au carré (MSE) est la plus adaptée.

1. https://en.wikipedia.org/wiki/Loss_function

4

Entraînement et évaluation

4.1 Entraînements

Le modèle convergeait assez rapidement. Il fallait environ 30 époques avant que la fonction de coût sur les données de validation ne baisse plus. Certes sur les données d'entraînement, elle continuait de descendre mais il commençait à sur-apprendre (overfitting).

La valeur de la fonction de coût n'indiquait en rien sur les performances possibles sur circuit. Elle indique à quel point le modèle arrive à prédire les labels.

Si le comportement indiqué lors du labeling est mauvais mais régulier alors le modèle va converger mais la voiture n'aura pas un bon comportement. Nous avons eu ce cas, nous avons choisi la consigne suivante : prendre les virages larges. Sauf que si la voiture entre dans le virage par l'intérieur alors elle décide d'aller vers l'extérieur et donc tourne dans le sens opposé au virage.

Mais en plus, si la voiture reste bien entre les lignes avec un MSE de 0.01. Quand est-il si la MSE est de 0.05 ou 0.1 ? Il est possible que de passer sur un réseau plus petit en acceptant une MSE plus grande change peu les performances de la voiture

Septième partie

Interface et Déploiement

1

Objectifs

L'interface permet de transmettre les prises de décision à la voiture de manière abstraite. Les commandes se retrouvent standardisées. Donc les modèles devront toujours avoir les mêmes sorties malgré toute modification du matériel ou de quelconque ajustement.

2

L'accéléromètre

Le binome 2 a réalisé l'implémentation du module mpu-5060, qui était déjà existant mais non-implémenté. Ce dernier a une double fonction : gyroscope et accéléromètre.

Pour que ce module fonctionne il faut déjà que I2C soit actif (son activation se fait directement via le fichier config de la Raspberry : sudo raspi-config). Il faut également que ic2-tools ainsi que smbus (python-smbus) soient installés.

Le câblage entre la Raspberry et le module est plutôt simple : les broches SDA et SCL sont branchées à leurs homologues et on alimente la carte en 3,3V (certains sites internet font état d'une possible alimentation en 5V, qui après test s'avère dangereuse pour le module). Les autres broches ne sont pas connectées.

Pour pouvoir vérifier l'installation et les branchements il suffit d'exécuter dans un terminal sudo i2cdetect -y 1. Si tout a été fait correctement, un 68 devrait apparaître dans la matrice (ligne 60, colonne e).

Initialement, nous pensions utiliser la fonction d'accéléromètre de ce module. Finalement, nous ne l'utilisons pas mais cela peut être une future piste de réflexion intéressante.

3

Gestion de la direction et de la vitesse

3.1 Orientée objet

Toutes les commandes se font via un même objet. Une fois créé et initialisé, à tout moment une commande peut être envoyé et l'objet se chargera d'envoyer la commande au "hat" de la raspberry pi.

3.1.1 Initialisation

Les paramètres d'initialisation ne se font pas dynamiquement. Les attributs décrivant les valeurs de PWM sont ceux qui devront le plus souvent être modifiés.

3.1.2 L'envoi de commande

Les 2 méthodes de commandes sont :

```
1     def set_direction(self, value, force=False)
2     def set_speed(self, value, force=False)
```

Pour la direction, la plage de valeurs possibles est [-1 ; 1] avec 0 pour aller tout droit. La vitesse se règle de 0 à 1, avec 0 comme vitesse de point de départ et 1 la vitesse maximale. Les valeurs seront écrêtées si elles sont en dehors de la plage

3.1.3 Côté développeur

L'application des commandes se fait à travers un *thread* à part. Non pas pour appliquer continuellement le même PWM mais pour appliquer une fonction entre le paramètre souhaité (target) et le paramètre appliqué (current) avec une composante temps. De base, la fréquence de mise à jour est de maximum 100 fois par seconde.

Algorithm 1: Application des commandes

```

while True do
    current speed = compute speed(current speed, target speed);
    apply speed(current speed);
    current dir = compute dir(current dir, target dir);
    apply dir(current dir);
end

```

3.2 Calibration

Pour calibrer le changement de direction, nous avons testé empiriquement toutes les valeurs de PWM afin de déterminer des seuils.

```

1      from car import Car
2      Car().calibrate()

```

Lors de l'exécution de la méthode, le premier PWM à calibrer sera celui de la direction. Les 3 valeurs à trouver sont celles qui font :

- tourner le plus à gauche
- tourner le plus à droite
- aller tout droit

Il est important de mettre la valeur limite, celle qui ne fera pas forcer les servos.

Ensuite pour la gestion de la vitesse, il y a à nouveau 3 valeurs de PWM :

- la mise en arrêt
- le point de départ à l'arrêt
- la vitesse maximale

2 points sont important à prendre en compte. Premièrement, la voiture peut continuer à avancer avec un PWM inférieur au point de départ avec l'inertie. Deuxièmement, le point de départ n'est pas le même si les roues avant sont droites ou complètement tournées.

3.3 Smoothing

Notre voiture est capable de prendre environ 20 décisions par seconde. En fonction des images reçues par la caméra, la décision prise par la voiture peut passer d'un virage à droite à un virage à gauche très brusquement.

Afin de pallier ce problème, nous avons mis en place une méthode d'inertie de changement de direction afin de lisser les changements de direction de la voiture et stabiliser en partie la conduite.

Pour des valeurs (*target* et *current*) de directions proches, il n'y a que peu de modification. Cependant lors d'un changement brusque de la valeur *target*, la valeur *current* va mettre un certain temps avant d'être égale à *target*.

4

Caméra

La caméra est reliée physiquement à la Raspberry par une nappe de câbles. Un support imprimé en 3D permet de la tenir. Nous utilisons le package Picamera.

4.1 Optimisation de la récup

Il y a plusieurs façon de récupérer une image avec Picamera. Nous avons fait en sorte de récupérer une image non compressée sans attente.

4.2 La classe PiRGBAnalysis

A chaque fois qu'une frame est disponible, la méthode *analyze* de l'objet PiRGBAnalysis est appelée. Ainsi, l'image actuelle change de manière continue et est mise à disposition pour le reste du programme rapidement.

5

Problèmes physiques

Lors de la journée de présentation du projet, la voiture a percuté une étagère de la bibliothèque et le pied de la caméra s'est brisé aux deux bases. Le pied tient actuellement avec de la colle. Sans choc et sans tenir la voiture par le support de caméra, il n'y a pas de raison que le pied se casse.

Pendant la deuxième semaine du projet, le servo moteur de la direction a cessé de fonctionner. Celui ci a été remplacé avec l'aide d'un professeur de l'école. Le servo de direction n'est donc plus celui d'origine. Les pins 4 et 5 du hat de la Raspberry y étaient connectées, nous pensons qu'elles ont été endommagées suite à cet incident.

Suite à l'accident du dernier jour du projet, des pins sont tordues sur le hat de la Raspberry. Nous pensons qu'un remplacement du hat serait judicieux pour reprendre le projet avec un matériel en bon état.



6

Sauvegarde d'un modèle

La sauvegarde des poids du réseau se fait avec le format HDF5. Comme ce sont les poids qui sont enregistrés, il faut redéfinir la structure du réseau de neurones lors du chargement et le construire avant.

Huitième partie

Des pistes d'amélioration

1

Le matériel

Il est à prévoir de repenser et reconcevoir le pied de la caméra afin de stabiliser la hauteur et l'orientation de la caméra. La modélisation peut se faire en 3D à l'aide d'un outil tel que Blender et l'impression peut être faite dans les locaux de l'école.

Une fois ce support fait, il faudra sans doute reprendre une vidéo autour d'un circuit et labéliser de nouveau les images pour ré-entraîner le modèle.

2

Le réseau de neurones

2.1 Le labeling

La labélisation des images est le point clé de la réussite des modèles. Mais nous avons assez peu expérimenté sur ce point. Nous ignorons

- si des types d'images sont négatives pour le modèle
- si les proportions des différents types d'images sont bonne
- si le labeling de chacun doit être normalisé pour homogénéiser les labels

2.2 La structure

La structure du réseau n'a quasiment pas évolué. Nous ne savons pas si les performances de la voiture seraient meilleures avec un réseau plus conséquent ou proche avec un réseau encore plus léger.

Un autre axe à améliorer est la mémorisation des états précédents pour les prises de décision futures. Ce que l'on a fait est assez basique. Un réseau de neurone pourrait peut-être évoluer vers une architecture hybride (RNN+CNN).

2.3 Le déploiement

Pour le déploiement, le réseau pourrait être exécuter en mode graphe et limiter le code python pour de meilleures performances. Ou même avant, juste chercher une potentielle méthode de sauvegarde plus appropriée.



IMT Lille Douai
École Mines-Télécom
IMT-Université de Lille

Projet - Voiture autonome

Conclusion

Pendant un mois, notre étude a porté sur le développement d'une solution amenant à la conduite autonome d'un véhicule précédemment radio-commandé. Nous étions deux binômes à nous partager le travail.

Nous avons (binôme 1) développé des systèmes différents et plus ou moins fonctionnels et les avons intégré tout au long du projet. La solution développée ayant donné les meilleurs résultats est un réseau de neurones convolutif. Avec l'aide d'un logiciel dédié que nous avons créé, nous avons labélisé les données pour permettre l'apprentissage (les deux binômes).

De par cette expérience, nous avons pu observer un cas concret d'utilisation des réseaux de neurones. A travers nos digressions, nos échecs et nos réussites, nous sommes ensemble arrivés à réaliser ce projet dans le temps, le budget et l'espace impartis. Ce projet a été pour nous l'occasion de parfaire notre sens de l'organisation, des priorités et notre investissement.

Les résultats sont très satisfaisants et la voiture a fait environ 40 tours sans sortir du circuit avant de tomber en panne de batterie lors de notre meilleure performance.

Nous espérons que le présent rapport ainsi que le code disponible sur le GitHub aideront quiconque souhaiterait reprendre ce projet pour le mener encore plus loin.

Nous remercions tous les professeurs, les doctorants et les élèves qui nous ont soutenu et accompagné pendant ce projet, particulièrement Monsieur Wannous.

Table des figures

1	La vision de la voiture RC	1
1.1	Photo de la voiture	8
1.2	Carte Raspberry	8
3.1	Circuit de la bibliothèque	15
1.1	Créer, activer un environnement virtuel	18
1.2	Mettre à jour des packages avec pip	18
1.3	L'interface de Jupyter	20
2.1	Modification de la taille du Swap	23
2.2	Installation de Tensorflow	24
1.1	L'image à la sortie de la caméra	26
2.1	L'image après le flou gaussien	27
2.2	La fonction du flou Gaussien	27
3.1	La détection de bord	28
3.2	La fonction Canny	28
4.1	La partie de l'image restante après la sélection ROI	29
4.2	Sélection de la région d'intérêt (ROI)	29
5.1	La fonction de redimensionnement	30
5.2	Recadre l'image	30
6.1	Une détection de bord de qualité correcte	31
7.1	Photos d'échiquier pour la calibration	33
7.2	Une image après distortion	33
7.3	La transformation <i>bird eye</i>	34
7.4	Les 3 canaux du HSV	34



1.1	Détection de lignes	36
2.1	La fonction HoughLinesP pour la détection de segments	37
3.1	Fonction d'activation	38
2.1	2 exemples de labeling	43
2.2	L'ancienne méthode de labeling	44
2.3	Image du circuit générée par ordinateur (avant et après recadrage)	44