

Rapport de Projet de Fin d'études

P44 : IronCar

Eloi Zalczer et Erwan Dufresne

Polytech Lille, IMA SC S10

Remerciements

Nous tenions à remercier nos tuteurs de projet Thomas Vantroys, Xavier Redon et Alexandre Boé pour leurs conseils et leur aide. En particulier, nous adressons nos remerciements à M. Vantroys sans qui la compétition IronCar n'aurait probablement pas eu lieu à Lille, et qui a suivi de près notre avancée tout au long du projet.

Nous tenions également à transmettre notre sympathie aux membres du Fabricarium de l'école pour leur patience, leurs conseils et leur accueil ainsi que tout le matériel mis à disposition.

Enfin, merci également à Thierry Flamand pour nous avoir aidés lors de situations électroniques parfois délicates.

Sommaire

Introduction	3
1 Analyse du projet	4
1.1 Choix du sujet	4
1.2 Cahier des charges	5
1.3 État de l'art et positionnement par rapport à l'existant	5
1.3.1 Premiers concurrents : les autres participants à l'IRONCAR	5
1.3.2 Seconds concurrents : les voitures autonomes	6
2 Réalisation du projet	8
2.1 Choix techniques et matériels	8
2.1.1 Architecture informatique : Serveur et application Web	8
2.1.2 Intelligence artificielle et deep learning	8
2.1.3 Système embarqué	10
2.1.4 Choix mécaniques	11
2.2 Travaux effectués	12
2.2.1 Application et commande manuelle	12
2.2.2 Apprentissage machine	14
2.2.3 Réalisations mécaniques et électroniques	17
3 Bilan et perspectives d'amélioration	22
3.1 Analyse des performances	22
3.2 Erreurs et problèmes rencontrés	24
3.2.1 Métrique et taux d'apprentissage	24
3.2.2 Mécanique	25
3.3 Résultats et bilan général	26
Conclusion	29

Introduction

Ce rapport relate le travail réalisé par le binôme Eloi Zalczer et Erwan Dufresne dans le cadre de leur projet de fin d'études, en tant qu'étudiants ingénieurs en Informatique, micro électronique et automatisme, à Polytech Lille.

Le projet concerne la réalisation d'une voiture miniature autonome, s'orientant via un entraînement préalable d'un réseaux de neurones. L'apprentissage profond, ou deep learning en anglais, est la technique d'intelligence artificielle retenue pour réaliser cette tâche et rendre ainsi la voiture totalement indépendante.

Divisé en trois grandes parties, ce rapport replace dans un premier temps le contexte du projet ainsi que ses enjeux, puis s'étend ensuite sur les différentes tâches réalisées, avant de restituer les résultats obtenus tout en offrant une perspective d'amélioration.

Chapitre 1

Analyse du projet

1.1 Choix du sujet

Chacun de nous a réalisé un semestre à l'étranger pour le début de cette cinquième année d'études. Le système scolaire et éducatif d'Amérique latine et d'Amérique du Nord étant différent de celui français, nous avons eu la possibilité de choisir nous-mêmes quelles matières nous souhaitions étudier.

Ainsi, nous pouvions chacun choisir des cours "à la carte" et façonnez une formation un peu différente que celle proposée à Polytech Lille. C'est pourquoi nous avons tous deux choisi des cours ayant un rapport de près ou de loin avec l'intelligence artificielle. Forts de ces différentes expériences plus ou moins approfondies dans ce domaine, nous jugions judicieux de les mettre en pratique lors du PFE de 5ème année. Suite à différentes propositions jugées non adéquates par les professeurs, monsieur Vantroys nous a finalement proposé un projet, original, qui nous permettrait de mettre à bien les nouvelles connaissances acquises dans le domaine de l'IA lors du semestre d'échange.

Le projet consiste à réaliser une voiture autonome miniature capable de s'orienter via de l'intelligence artificielle. Cette voiture ainsi réalisée devrait pouvoir être conforme aux différentes normes lui permettant de participer à la compétition IRONCAR. Cette compétition permet de faire s'affronter différentes équipes entre elles plusieurs fois dans l'année. Chaque équipe a le droit d'avoir une voiture avec une ou deux Raspberry Pi/Arduino, une caméra ainsi que le nécessaire électronique pour faire avancer la voiture. Le bolide doit être totalement autonome et entraîné par intelligence artificielle en apprentissage profond, *deep learning* en anglais.

La voiture est seule sur la piste et doit réaliser 3 tours de celle-ci en un temps minimum. La piste est large de 1 m 80 avec des bandes blanches latérales continues et possède également une ligne jaune pointillée centrale.

Le classement final est du type "championnat" et chaque compétition selon son importance rapporte plus ou moins de points. À la fin de l'année, l'équipe ayant le plus de points gagne. Le projet de l'Ironcar étant de progresser via l'Open Source, le gagnant est tenu de partager le projet Github contenant les fichiers qui lui ont permis de gagner. Cela permet aux autres concurrents de s'en inspirer et d'améliorer constamment les algorithmes de deep learning. Cette compétition est orientée vers le partage et la progression commune.

1.2 Cahier des charges

Notre cahier des charges était finalement assez simpliste :

- Obtenir une voiture améliorée mécaniquement, via impression 3D ou découpe laser, afin de maintenir correctement les composants ajoutés nécessaires à l'apprentissage de la voiture.
- Obtenir une voiture fonctionnelle en orientation et en mouvement (qui se déplace correctement).
- Faire en sorte que le réseau de neurones de la voiture soit fonctionnel et performant.
- Obtenir un score satisfaisant (le plus haut possible) lors de la participation de la voiture à la prochaine course Ironcar.
- Faire en sorte que le travail réalisé sur la voiture soit propre et que le bolide ait un design final agréable.

1.3 État de l'art et positionnement par rapport à l'existant

1.3.1 Premiers concurrents : les autres participants à l'IRONCAR



FIGURE 1.1 – adversaires lors de la dernière compétition IRONCAR

Les premiers concurrents directs seront les autres participants à la course de l'Ironcar. Ces différents concurrents peuvent chacun déployer le dispositif qu'ils souhaitent, selon leurs moyens et leur envie. Sous réserve que le prototype final respecte les conditions requises pour participer à la compétition, bien entendu.

Chaque équipe est généralement associée à une école. Ils ont plus ou moins de temps et de moyens à consacrer à cette compétition. Il est difficile de pouvoir analyser nos concurrents directs sans les avoir encore affrontés en compétition, cependant certains scores obtenus et certaines vidéos à l'appui témoignent d'une réussite certaine de la part de certaines équipes. Nous tâcherons d'être assez compétitifs pour pouvoir les inquiéter dans leur course.

1.3.2 Seconds concurrents : les voitures autonomes

D'autre part, il y aussi d'autre type de concurrents, notamment des grandes entreprises comme Google, Uber ou Tesla qui conceptualisent et déploient des voitures autonomes depuis plusieurs années. En effet, la voiture autonome est déjà une réalité dans certains pays, notamment en Californie aux USA. Les résultats sont plus qu'impressionnantes. On dénombre moins d'une dizaine d'accidents responsables pour les voitures autonomes ces deux dernières années. Les réseaux de neurones (deep learning) sont entraînés avec des datasets de plusieurs milliers d'heures de conduite et peuvent ainsi répondre à chaque cas possible. Ces datasets sont parfois disponibles gratuitement sur internet.



FIGURE 1.2 – Google Car

Bien que la technologie soit déjà utilisable et fiable, les recherches dans ce domaine sont en constante expansion. L'idée d'améliorer l'intelligence artificielle des véhicules continue de croître dans la tête des hauts décisionnaires des grandes entreprises comme Google.

C'est pourquoi nous pouvons actuellement rencontrer des entraînements de réseaux de neurones sujets à controverses, parfois à la limite de l'éthique morale. C'est notamment le cas avec des recherches qui sont effectuées sur "Qui doit on sacrifier ?" en cas d'accidents inévitables. Cet algorithme prend en compte le nombre, l'âge des protagonistes, mais aussi le lien d'affection éventuel avec le propriétaire de la voiture (donc conducteur potentiel). Par exemple faut-il percuter trois personnes âgées pour sauver le propriétaire quadragénaire de la voiture ainsi que sa fille en bas âge ? Des questions dérangeantes, qui suscitent beaucoup de réactions. Le MIT a publié une étude sur le sujet.

Chapitre 2

Réalisation du projet

2.1 Choix techniques et matériels

2.1.1 Architecture informatique : Serveur et application Web

Enfin, nous avions besoin d'une interface utilisateur pour commander la voiture manuellement. Nous ne voulions pas surcharger la Raspberry Pi en lui ajoutant un serveur, nous avons donc décidé de le déporter sur un PC externe. Nous verrons plus loin que l'optimisation des performances du réseau de neurones a été un de nos challenges, et faire tourner un serveur en parallèle du réseau de neurones n'aurait pas été raisonnable. Nous avons décidé d'utiliser un serveur Node.js car nous étions familiers avec cette technologie et qu'elle nous permettait d'utiliser des websockets de façon très simple. Pour l'application Web, nous sommes restés sur du JavaScript basique car la complexité du travail ne justifiait pas l'ajout d'un framework.

2.1.2 Intelligence artificielle et deep learning

Python et ses librairies

Si le choix de Python pour l'apprentissage machine était évident, celui de la librairie à utiliser était plus difficile. De nombreux frameworks d'apprentissage profond sont disponibles, et chacun présente ses propres avantages. Le leader TensorFlow, développé par le géant Google, est très ancré dans la communauté et utilisé par pratiquement tous les projets IronCar jusqu'à présent. Cependant, la librairie a l'inconvénient de proposer une API très obscure et de donner une impression de boîte noire. Notre choix s'est donc d'abord porté sur PyTorch, une librairie récente développée par Facebook, qui commence à creuser sa place et a l'avantage d'offrir une API très "pythonesque". Les deux systèmes proposent des fonctionnalités équivalentes, et sont tous les deux amplement suffisants pour l'utilisation que nous souhaitons en faire. Il n'existe pas de version publiée de PyTorch pour architecture ARMv7, nous avons donc dû compiler manuellement la librairie pour la Raspberry Pi. Cette opération nous a pris plusieurs heures mais s'est finalement bien déroulée. Nous avons effectué la compilation directement sur la Raspberry Pi, en prenant du recul il aurait probablement été plus simple et plus rapide de cross-compiler sur une autre machine.

Nous sommes restés sur ce choix jusqu'à être forcés de changer pour des raisons de performances. Il nous était en effet impossible d'obtenir un nombre suffisant d'inférences du

réseau par seconde avec PyTorch. Les performances sont détaillées dans la section [Analyse des performances](#) du rapport. Face à ce problème, nous avons été obligés de nous tourner vers une autre technologie. En suivant la trace de projets IronCar précédents, nous avons décidé d'utiliser Keras. Il s'agit d'une librairie de haut niveau conçue pour pouvoir fonctionner comme une surcouche de plusieurs systèmes. Cela nous a donc permis d'utiliser un backend TensorFlow en évitant les problèmes de syntaxe. Keras supporte nativement le format de fichiers que nous utilisons et offre des options de pré-traitement et de gestion des jeux de données qui nous ont globalement permis de simplifier notre code par rapport à PyTorch.

Stockage des images

L'utilisation d'images en apprentissage automatique est très courante, cependant les méthodes de stockage des données varient selon les cas. La méthode la plus simple est bien entendu de stocker les images sous forme de fichiers JPEG directement lors de l'enregistrement. Tous les frameworks actuels proposent des API de haute performance pour l'importation de jeux de données sous cette forme. Cependant, cela pose certains problèmes. Principalement, la compression JPEG peut provoquer de légers artefacts visuels qui, bien qu'invisibles à l'oeil nu, peuvent avoir des effets importants sur un réseau de neurones. Lorsqu'une commande sera inférée automatiquement pendant la conduite automatique, l'image utilisée sera brute et risquerait de présenter des couleurs ou des motifs légèrement différents qui pourraient être mal interprétés. Par ailleurs, nous avons également besoin de stocker la commande associée à chaque image pour effectuer l'entraînement ensuite. Cette information pourrait être stockée dans le nom du fichier, mais ce cas n'est pas nécessairement prévu par les librairies. Nous perdrons alors l'avantage des algorithmes d'importation prévus à cet effet.

En explorant d'autres possibilités, notre choix s'est porté sur le format de fichiers HDF5 (*Hierarchical Data Format*). Il s'agit d'un format conteneur développé par le *HDF Group* qui permet notamment de stocker des données massives sous forme de tableaux multidimensionnels, éventuellement liés par groupes. Ce format a l'avantage d'être supporté par la totalité des librairies principales, nativement ou via le module python *h5py*. Nous avons ainsi pu stocker les images sous forme brute, soit un tableau d'entiers sur 8 bits de taille 200x66x3. Les 200 pixels de largeur et 66 de hauteur ont été choisis en s'inspirant de l'article [3]DeepPiCar et sont essentiels à l'architecture du réseau. La troisième dimension correspond simplement aux trois couches rouge, vert et bleu de l'image. Parallèlement, nous avons également pu stocker les commandes dans le même fichier en créant simplement un second dataset, ce qui nous permet d'accéder très simplement aux deux données.

Google Colaboratory

Tout entraînement de réseau de neurones nécessite une forte puissance de calcul. En particulier, notre système est basé sur un réseau convolutionnel profond, ce qui implique un grand nombre de poids à entraîner. L'entraînement sur la Raspberry Pi directement était exclu, et utiliser un PC portable classique aurait également pris beaucoup trop de temps. C'est pourquoi nous avons décidé d'utiliser la plateforme Google Colaboratory. Il s'agit d'un environnement de notebook [2]Jupyter qui s'exécute directement dans le Cloud et a l'avantage de mettre des GPU à disposition des utilisateurs. La plupart des librairies y

sont pré-installées, et il est possible d'en ajouter simplement. Ainsi, nous avons pu utiliser la plateforme pour entraîner aussi bien des architectures PyTorch et Keras. Dans les deux cas, le gain de temps lié à l'utilisation des GPU est d'environ 20x. La plateforme est liée à Google Drive, il nous a donc suffit d'uploader nos jeux de données pour pouvoir les utiliser, et d'exporter nos réseaux entraînés en fin de session. Cette technologie nous a permis de gagner énormément de temps.

2.1.3 Système embarqué

La voiture était livrée avec sa télécommande permettant de la guider à distance. Nous avons bien entendu décidé de la supprimer afin de rendre la voiture totalement autonome. Le système informatique du projet est assez complexe et utilise différentes technologies. Le premier choix que nous avons dû effectuer était celui de l'architecture matérielle, car c'est celle-ci qui nous a guidés. Nous avons ainsi rapidement décidé d'utiliser une Raspberry Pi 3B+ comme cœur du projet. C'est sur cette plateforme que doit s'effectuer le contrôle automatique de la voiture et donc toutes les opérations relatives au réseau de neurones. C'est donc tout naturellement que nous avons choisi d'utiliser Python pour cette partie car c'est le langage de référence dans le domaine de l'apprentissage automatique et qu'il propose énormément de possibilités d'acquisition, traitement et stockage d'images. Pour contrôler la voiture, nous avions également besoin d'un système qui communique avec les moteurs. Nous avons décidé d'utiliser une carte Arduino UNO pour générer les PWM, car nous avions peur que la Raspberry Pi ne soit pas capable de générer un signal d'aussi bonne qualité. Un modèle MEGA de l'arduino n'aura pas été judicieux, cette dernière n'étant pas plus performante au niveau de la PWM. De plus le modèle UNO est plus petit et plus adéquat à notre projet embarqué. Cette carte sera alimentée via le câble de la liaison série, depuis la Raspberry elle même.



FIGURE 2.1 – Raspberry et Arduino UNO fixées

Il nous suffisait alors d'alimenter la Raspberry pour alimenter l'arduino. Dans un souci de gain de place et de facilité, nous avons utilisé une batterie externe, initialement prévue pour téléphone portable. De la forme d'un disque dur, facilement fixable et d'une capacité de 5200 mAh, elle répondait parfaitement à nos besoins.

Enfin, bien que nous ayons effectué nos premiers tests avec une caméra pour raspberry dite "classique", nous nous sommes rapidement aperçus qu'il était indispensable d'utiliser une caméra grand angle. La piste de la course étant large de 1 mètre 80, il est essentiel de prendre le maximum d'informations possibles par image. Une caméra ordinaire ne permettant pas de couvrir une zone suffisante, nous alors donc opté pour un modèle de caméra avec un angle de vue de 160 degrés.

En revanche, la batterie de base de la voiture pour alimenter les moteurs sera conservée car le système était bien entendu optimal et performant, aucun intérêt de le modifier.

2.1.4 Choix mécaniques

La voiture a été choisie par le tuteur de projet. Il s'agit d'une voiture T2M Pirate XT-S, très robuste et puissante. Modèle tout terrain, nous étions sûrs de n'avoir aucune difficulté liée à la surface de la piste.

Nous savions que nous devrions modifier mécaniquement la voiture afin de permettre une fixation stable et ordonnée des différents éléments que nous avions à rajouter. Le choix d'une amélioration via ajout d'une plaque découpée au laser était de loin le plus évident. En effet, facilement réalisable, rapide à découper et facile d'accès au Fabricarium, cette solution tombait sous le sens.

Pourtant la matière découpée devait être étudiée un peu plus en profondeur. Découpons nous dans une planche en contre-plaqué, au risque d'avoir un résultat final voûté de par la non platitude naturelle du bois ? Ou alors optons nous pour un matériel plus épais et plus robuste comme le plexiglas, mais ainsi plus lourd, au risque de perdre en vitesse lors de la course ? Finalement, aux vues des aptitudes de traction de notre engin, nous choisirons la seconde option, plus robuste et plus sûre pour la fixation de nos éléments électroniques sensibles.

Enfin, pour renforcer le tout et rendre le résultat final plus esthétique, nous souhaitons réaliser une carrosserie. Nous avons décidé de faire ceci en impression 3D puisque cela nous permettait de faire des formes géométriques plus propres et plus complexes. De plus le PLA est une matière relativement légère. Enfin, cette partie n'étant pas indispensable pour le fonctionnement du projet, nous pouvions attendre l'impression totale des pièces sans pour autant retarder l'avancement global du réseau de neurones.

2.2 Travaux effectués

2.2.1 Application et commande manuelle

Front End

Une des premières réalisations du projet, dans la première semaine, a été l'application Web permettant la conduite manuelle de la voiture. Nous avons opté pour un design très simple afin de produire une application intuitive et pratique. Le fonctionnement est simple. Un canvas JavaScript est utilisé pour simuler un joystick dans un plan. La dimension horizontale permet de contrôler la direction et la dimension verticale la vitesse. Les commandes correspondantes sont calculées à partir des coordonnées de la souris. L'application présente aussi deux boutons permettant de démarrer/arrêter la voiture, et de démarrer/arrêter l'enregistrement des images, ainsi qu'un champ permettant de régler la vitesse maximale de la voiture.

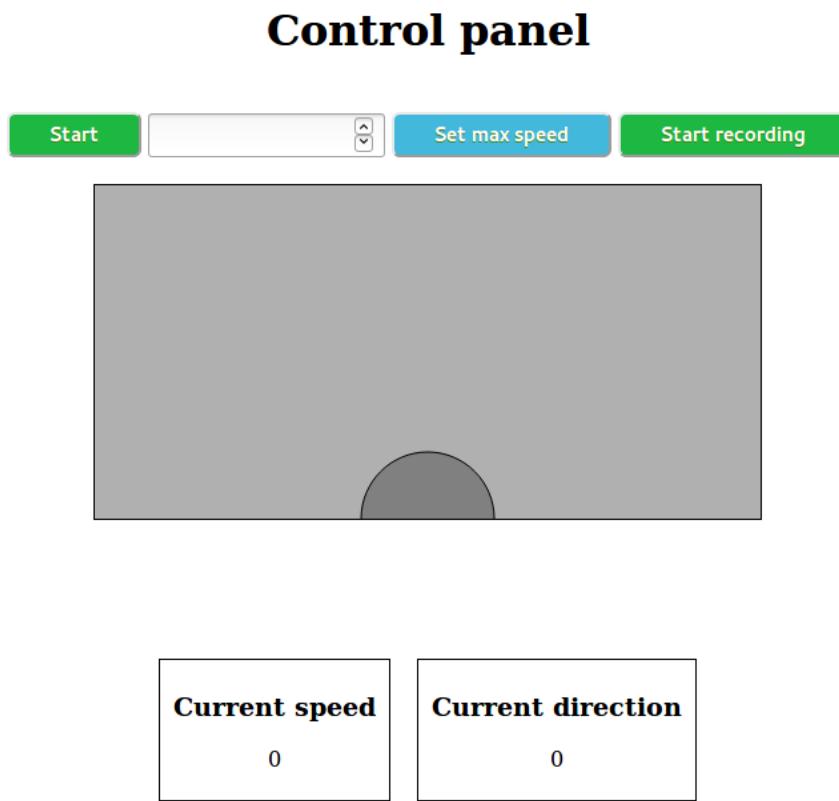


FIGURE 2.2 – Design de l'application

Dans les premières versions, les commandes étaient envoyées de façon périodique vers le serveur. Cependant, ce comportement avait tendance à saturer inutilement le port série à l'autre bout de la chaîne de contrôle. Nous avons alors décidé de renvoyer la commande uniquement lorsque celle-ci changeait. Cette nouvelle approche a posé plus de problèmes que nous ne l'avions pensé. En premier lieu, cela impliquait que les variations ne devaient pas être trop fréquentes. C'est à ce moment que nous avons décidé d'utiliser des valeurs entières entre -30 et 30, au lieu de valeurs décimales. Cela permettait au moins au système d'être robuste à des légères variations de pixels et de ne pas trop surcharger la chaîne de

contrôle.

Une autre série de modifications ont été apportées lorsque nous sommes passés à la véritable phase d'utilisation de l'application, c'est à dire la phase d'entraînement du réseau de neurones. En particulier, il manquait un certain nombre de systèmes de tolérance à l'erreur. Par exemple, si l'utilisateur lâchait totalement la commande, la voiture continuait d'avancer à vitesse constante jusqu'à ce qu'une nouvelle commande soit envoyée. Ce genre de problèmes ont été patchés aussi bien du côté front-end que du côté embarqué pour assurer une redondance et garantir la sécurité de la voiture. Enfin, nous avons modifié le code de l'application pour être capable de supporter une utilisation sur mobile. Durant les entraînements, nous avons en effet contrôlé la voiture depuis nos téléphones, ce qui s'est avéré beaucoup plus pratique et instinctif.

Application embarquée

La priorité de l'application embarquée était de favoriser la légèreté pour obtenir la meilleure vitesse d'inférence du réseau de neurones. Nous avons donc conçu la totalité du code avec cette problématique en tête. La Raspberry Pi reçoit les commandes du serveur via Websockets et les renvoie à l'Arduino par l'intermédiaire du port série. Pour la gestion des websockets, nous avons utilisé la librairie python-socketio. Cette dernière a l'avantage d'offrir un fonctionnement totalement asynchrone basé sur des événements, et donc de bloquer très peu le processus. De plus, la librairie est disponible en JavaScript et Python, ce qui fait que nous avons pu l'utiliser dans tous nos codes sans nous soucier de la compatibilité.

Pour la capture des images, nous avons utilisé le package picamera, exclusivement conçu pour Raspberry Pi et qui offre les meilleures performances sur cette plateforme (comparé à OpenCV par exemple). L'acquisition des images se fait dans un thread séparé et une API permet au programme principal de récupérer systématiquement l'image la plus récemment acquise. Ce système permet à la caméra de fonctionner totalement indépendamment sans jamais bloquer le processus principal. Aucune synchronisation de threads n'est nécessaire car l'affectation de la variable de retour est une opération atomique, et la fréquence de fonctionnement de la caméra est de toute façon supérieure à celle d'inférence du réseau. Durant l'enregistrement des images, le fichier HDF5 est écrit à l'arrêt de la voiture pour ne pas bloquer le système durant son exécution.

Les commandes reçues depuis l'application Web sont au format d'objets JavaScript, et sont automatiquement converties en dictionnaires Python. Avant de pouvoir les renvoyer vers l'Arduino, ces données doivent être converties. Pour optimiser l'utilisation du port série, les données envoyées sont composées de deux octets de commande suivis d'un caractère de fin de ligne pour signifier la fin de l'envoi. Le premier octet contient la commande en direction, et le second la commande en vitesse. Il aurait été possible de se contenter de 6 bits pour chacune des commandes, la plage étant de 60 valeurs, cependant nous avons jugé que le gain n'en valait pas la peine et risquait au contraire de poser des problèmes de synchronisation du port série. Avant d'être envoyées, la valeur 90 est ajoutée à la direction car la position "tout droit" des roues de la voiture correspond à une commande de 90. La même valeur est ajoutée à la commande de vitesse car le moteur commence à tourner à partir de la valeur 100. Cela laisse donc à l'utilisateur une certaine marge avant de déclencher les moteurs principaux, qui sont surprenamment puissants.

2.2.2 Apprentissage machine

Architectures de réseaux

Le type de réseau de neurones utilisé pour le pilotage automatique de la voiture est un réseau convolutionnel profond. Ce genre de réseau est généralement composé de deux parties principales. Les premières couches du réseau sont les couches de convolution, c'est à dire qui convoluent à l'image un noyau bi-dimensionnel de dimensions données. Ces couches permettent de réduire la dimension des données et appliquent la même opération en chaque point de l'image. Les poids à entraîner sont les coefficients du noyau. Empiler plusieurs de ces couches revient à travailler à plusieurs échelles différentes et donc pouvoir analyser des détails plus ou moins fins. En sortie de la partie convolutionnelle, les données ont été réduites à une dimension et sont donc adaptées à la seconde partie du réseau. Cette deuxième partie est un ensemble de couches *denses*, c'est à dire un réseau où chaque neurone est connecté à la totalité des neurones des couches précédentes et suivantes. Cette partie effectue l'interprétation des "briques" de sortie de la partie convolutionnelle et donne la sortie finale du réseau. La conception de ces couches est particulièrement ardue car leur complexité doit être proportionnelle à la complexité du problème. S'il y a trop de couches, le réseau risque de faire du sur-apprentissage, c'est à dire apprendre par cœur le jeu d'entraînement. Dans le cas contraire, le réseau peut faire du sous-apprentissage et donner des résultats imprécis.

Pour pallier ce genre de problèmes, il existe différentes méthodes. Nous avons parlé plus tôt des fonctions de régularisation. Il est très fréquent d'utiliser des fonctions ReLU entre chaque couche dense. Cela permet de limiter les valeurs à une plage voulue et d'éviter une divergence du réseau. Une autre méthode très courante est l'utilisation de couches *Dropout*. Leur principe est de désactiver aléatoirement certains neurones à chaque nouvelle donnée d'entraînement. Cela permet de mieux répartir l'apprentissage parmi les neurones et d'augmenter la capacité de représentation du réseau. Tous ces éléments (nombre de couches, taille des couches, fonctions de régularisation, nombre d'époques d'entraînement...) sont appelés les hyper-paramètres du problème. Leur optimisation est difficile et se fait souvent au jugé. Parfois, il est possible d'utiliser une recherche en grille qui teste chacune des configurations possibles. Cependant, ce processus est extrêmement long et coûteux, c'est pourquoi nous avons utilisé une architecture déjà existante et testée.

$$\text{Fonction ReLU : } f(x) = \begin{cases} 0 & \text{si } x < 0, \\ x & \text{sinon.} \end{cases}$$

L'architecture que nous avons utilisée est une version simplifiée de celle donnée dans l'article de recherche [3]DeepPiCar. Ce réseau est lui-même inspiré de l'architecture DAVE-2, conçue par NVIDIA pour construire leur propre voiture autonome. Le réseau travaille à partir d'images de taille 200x66 et possède une unique sortie qui correspond à un angle de direction du volant. L'architecture est assez simple. Elle consiste en 5 couches convolutionnelles suivies de 4 couches denses.

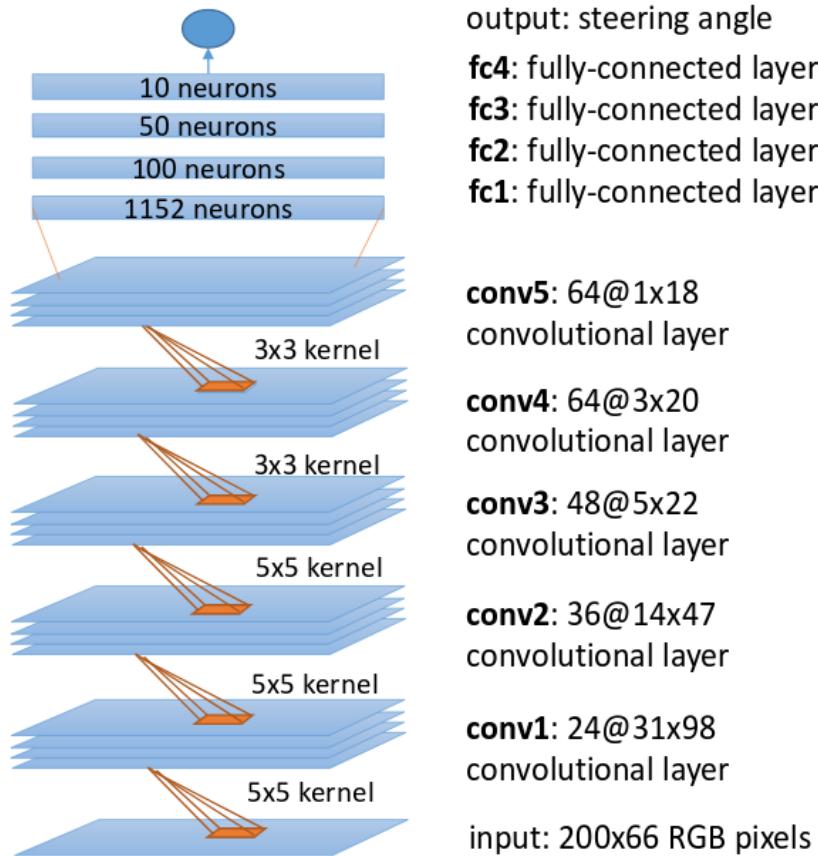


FIGURE 2.3 – Architecture du réseau DeepPiCar

La sortie de la convolution est composée de 64 calques, comprenant chacun 18 valeurs. Ces données sont ensuite réduites à une dimension pour obtenir une série de 1152 valeurs, qui sont passées aux couches suivantes. Le GitHub de DeepPiCar présente plusieurs implémentations, dont une avec une couche dense supplémentaire de taille 1164. Nous avons tenté d'utiliser cette architecture, cependant la quantité de poids associée ralentissait énormément le réseau. Cette couche à elle seule représentait plus de 1.3 million de poids. Nous avons donc décidé d'utiliser une architecture plus simple. En sortie du réseau, une fonction arc-tangente permet de convertir le résultat d'une plage de $[-1, 1]$ à la plage voulue, dans notre cas $[-30, 30]$. Les seules modifications que nous avons apportées se situent au niveau du nombre de calques des couches convolutionnelles, pour des raisons de performance. Les architectures sont présentées de façon plus avancée dans la partie [Analyse des performances](#).

Simulation

Avant d'utiliser des données réelles pour entraîner notre réseau de neurones, nous avons dû trouver un moyen de générer des données artificiellement pour tester les performances de nos architectures. Par chance, nous avons trouvé un [1]projet sur GitHub nous permettant exactement de faire ceci. Le code est assez simple et est fourni avec un certain nombre d'exemples d'utilisation. Il suffit alors de renseigner le nombre d'images à générer et le programme se charge de les créer aléatoirement suivant les directives données. Le projet est un travail étudiant dédié à la compétition IronCar et n'a visiblement pas été testé

largement. Nous avons par exemple trouvé un bug au niveau des dimensions des images générées, pour lequel nous avons heureusement pu trouver une solution rapidement. Nous avons ouvert une *Issue* sur le dépôt GitHub pour en prévenir les créateurs.

Une autre limite du générateur est la gestion de la vitesse. En effet, la direction correspondant à chaque image est encodée dans le nom du fichier, mais la vitesse est constante et égale à la valeur moyenne de 0.5. C'est une des raisons pour laquelle la vitesse n'est pas une sortie du réseau de neurones. Pour adapter le format des données, nous avons écrit un script permettant de lire le titre des images et de convertir la direction générée (globalement entre -2 et 2) en un format utilisable par notre réseau. Les données de sortie sont mises au format HDF5 utilisé partout ailleurs dans notre projet. Avec le générateur, nous pouvons générer indépendamment deux jeux de données avec les mêmes caractéristiques et n'avons donc pas eu besoin de séparer les données en un jeu d'entraînement et un jeu de validation.

Pré-traitement des données

Le pré-traitement des données est une étape essentielle dans tout processus d'apprentissage automatique. Les résultats dépendent en effet entièrement de la qualité et de la cohérence des données d'entrée. Lorsque l'on travaille sur des images en particulier, il existe certains types de pré-traitements couramment appliqués. Tout d'abord, un certain nombre de composantes des réseaux de neurones imposent de travailler sur une plage de valeurs de [0, 1]. C'est par exemple le cas de fonctions de régularisation comme ReLU (Rectified Linear Unit) ou sigmoïde qui écrasent les valeurs hors de cette plage. La première transformation que nous avons systématiquement appliquée sur toutes nos images est donc une division de toute les valeurs par 255 pour se ramener dans la plage souhaitée. Par ailleurs, nous avons également envisagé un certain nombre d'autres options. PyTorch propose une fonction *Normalize* qui permet de ramener la variance et l'écart-type d'une image à une valeur donnée. Cela permet une meilleure couverture de la plage de valeurs et globalement une amélioration des résultats. Keras ne fournit pas d'implémentation par défaut de cette normalisation, et nos tentatives d'implémentations manuelles n'ont jamais permis d'obtenir de meilleurs résultats. Nous avons également rapidement tenté de travailler sur des images en noir et blanc, sans bons résultats.

Un autre intérêt du pré-traitement des données peut être l'augmentation artificielle du volume de données. En travaillant sur le simulateur, nous pouvions générer une quantité virtuellement infinie d'images et n'avions donc pas besoin de gonfler nos données. Cependant, en passant aux données réelles, le volume des données est devenu critique. Une premier moyen évident pour augmenter leur quantité est d'appliquer une transformation en miroir. Il suffit alors d'inverser l'image et la direction correspondante de façon à ce que les deux correspondent. Pour augmenter la capacité d'adaptation du réseau, une bonne pratique est également d'apporter des variations aléatoires aux images. Par exemple, l'ajout d'un bruit gaussien aléatoire ou des variations d'exposition et de contraste pourront permettre au réseau de mieux s'adapter aux conditions de luminosité différentes de la compétition et aux variations de sensibilité de la caméra. Nous avons écrit une série de scripts, sur le même modèle, permettant d'appliquer toutes ces transformations de façon automatique. Nous avons également écrit quelques scripts utilitaires, par exemple pour regrouper deux fichiers de données ou les partitionner en un jeu d'entraînement et un jeu de validation.

Après avoir conduit la voiture sur une piste et enregistré une quantité suffisante d'images, nous avons entrepris de valider manuellement les données. En effet, notre pilotage n'était pas parfait et il est arrivé que la voiture sorte de la piste. Nous ne voulons évidemment pas que le réseau apprenne à être un chauffard, il était donc nécessaire de filtrer ces données polluées. Durant ce processus, nous nous sommes rendus compte que les commandes étaient très bruitées. En raison de la faible vitesse à laquelle nous allions, certains des ajustements que nous avons faits ne paraissaient pas pertinents. Par exemple, tourner fortement à gauche alors qu'un virage à droite arrive, pour ajuster légèrement la trajectoire. Nous avons eu peur que ce bruit ne rende l'apprentissage très inefficace. Nous avons donc décidé de nous séparer le travail. Un d'entre nous a continué la validation des données réelles, et un autre s'est attaqué à un ré-étiquetage manuel. A l'aide d'un petit programme Python, nous avons alors dessiné une trajectoire sur chaque image qui était convertie en commande. Il s'est trouvé que ces données ré-étiquetées donnaient de bien meilleurs résultats en entraînement et nous les avons utilisées quasi-exclusivement.

2.2.3 Réalisations mécaniques et électroniques

Support et fixation pour Raspberry et Arduino

Après un test de prise en main des contrôles de la voiture à l'aide de la télécommande fournie, nous sommes partis sur l'idée de ne transmettre que les informations de direction et de vitesse à la voiture. Les autres réglages possibles avec la télécommande nous ont paru optionnels.

La première étape était donc de réaliser un support solide pour recevoir la Raspberry Pi, l'Arduino ainsi que l'alimentation externe. Le choix de plexiglas épais ayant été fait au préalable pour les raisons expliquées précédemment, nous devions ensuite réaliser le schéma de découpe. Dans un premier temps il s'agissait de prendre de nombreuses mesures sur la voiture ainsi que sur les cartes elles-mêmes.

Une fois ceci fait, nous avons réalisé une modélisation de la pièce souhaitée via l'outil de modélisation en ligne ONSHAPE. L'export fut ensuite sorti sous le format .DXF pour pouvoir être modifié dans INKSCAPE et le rendre conforme aux paramètres attendus par la découpeuse laser. Enfin, l'export final a eu lieu au format .svg pour pouvoir être découpé correctement.

Les cartes Arduino et Raspberry Pi n'étant pas censées être amenés à être modifiées au niveau des câblages I/O, nous avons décidé de les rendre difficiles d'accès via une position suspendue en dessous de la plaque de plexiglas. Ceci permet également une meilleure protection des ces deux pièces fragiles en cas d'effondrement de la carrosserie par exemple suite à un choc violent. La fixation est donc destinée à être définitive, ce pourquoi nous les avons fixées avec des vis.

Alimentation

En revanche la batterie externe se doit d'être relativement accessible puisque c'est cette dernière que nous serons amenés à recharger régulièrement, voire à démonter pour la charger indépendamment du reste de la voiture, bien trop encombrante. Ainsi, nous

avons cette fois encastré le composant dans une pièce prévue à cet effet. Cette pièce est composée de plusieurs couches de plexiglas, rotatives et permettant un maintien aux quatre coins de la batterie, facilement ôtable. Le résultat est en plus très propre puisque l'ensemble fut réalisé dans un plexiglas similaire au précédent, grâce à la découpe laser, offrant ainsi un travail de découpe extrêmement précis.

Cette disposition de l'Arduino et de la Raspberry Pi sur la face inférieure de la plaque de plexiglas a également un second avantage. Effectivement, les pins entre les cartes et le système de contrôle des moteurs sont donc en face les unes des autres, ou presque. L'utilisation de simples câbles mâle-mâle sera donc suffisante pour transmettre l'information. Il n'est pas nécessaire de router une carte électronique pour cela, cette dernière prendrait plus de place que les quelques fils nécessaires.

Le seul véritable travail électronique réalisé a été de dénuder un câble USB afin d'en extraire les câbles d'alimentation (ceux du transfert de données ne nous intéressent pas dans ce cas). Nous avons ensuite soudé des pins femelles afin de pouvoir alimenter la Raspberry Pi directement via les pins 2 et 6. Nous avons mis des gaines thermiques pour solidifier le tout et garder un certain esthétisme de réalisation. Désormais le câble USB possède une longueur parfaite pour relier la batterie externe au micro PC. Ceci n'était pas le cas précédemment, le fil était bien trop long.



FIGURE 2.4 – ensemble final retenu

Carrosserie

La caméra ayant mis un certain temps avant d'être livrée, nous avons eu, durant une période, un ralentissement de l'avancement général du projet. Nous avons donc décidé de nous investir dans des parties moins essentielles mais tout aussi formatrices, plutôt que de stagner sans réels objectifs jusqu'à l'arrivée de la caméra.

C'est pourquoi nous avons entrepris de réaliser une carrosserie permettant de protéger

l'ensemble des composants de la voiture. En effet, avec l'ajout des différentes pièces, la coque initiale n'était plus du tout adaptée au bolide.

Après avoir parcouru les différentes images d'internet ayant un rapport de près ou de loin avec le mot "4x4", nous avons appris l'existence de cars scolaires tout terrain. Cela répondait parfaitement à nos besoins.

Effectivement, l'ensemble avait une carrosserie complexe mais malgré tout modélisable, avec notre niveau de débutant en CAO.

De plus, la forme surélevée de la partie "passagers" du bus, nous permet donc de disposer de beaucoup d'espace pour y ajouter des composants par la suite.

Enfin, ce bus scolaire tout terrain nous permet en plus de lui donner un surnom amical : le mélange parfait entre un car et un 4x4, le 4 car !

Nous avons alors entamé la réalisation de cette carrosserie, complexe mais pas irréalisable. Cela nous a demandé de nombreuses heures de modélisation. Nous avions en effet sous-estimé la tâche de modélisation malheureusement. N'étant que de simples novices dans le domaines de la CAO, certaines actions parfois simples nous demandaient plusieurs dizaines de minutes. Heureusement, après de nombreux essais, de nombreuses versions et de très nombreuses mesures sur la voiture, nous avons enfin réussi à obtenir une version correcte de la carrosserie du 4xCar.

Bien que les impressions 3D soient performantes de nos jours, nous ne pouvions pas prendre le risque d'imprimer le modèle ainsi réalisé en une seule fois. De plus, les imprimantes disponibles au Fabricarium ne nous le permettaient simplement pas, de par la taille du fichier modélisé. Ainsi nous sommes partis sur l'idée de répartir la coque en plusieurs impressions que nous assemblerions plus tard, toutes ensemble. Aux vues du nombre d'échecs d'impressions des différentes pièces, nous venions alors de faire sans doute l'un des meilleurs choix de tout le projet. En additionnant tous les temps d'impressions, nous devons être à environ 150 heures d'impressions 3D !

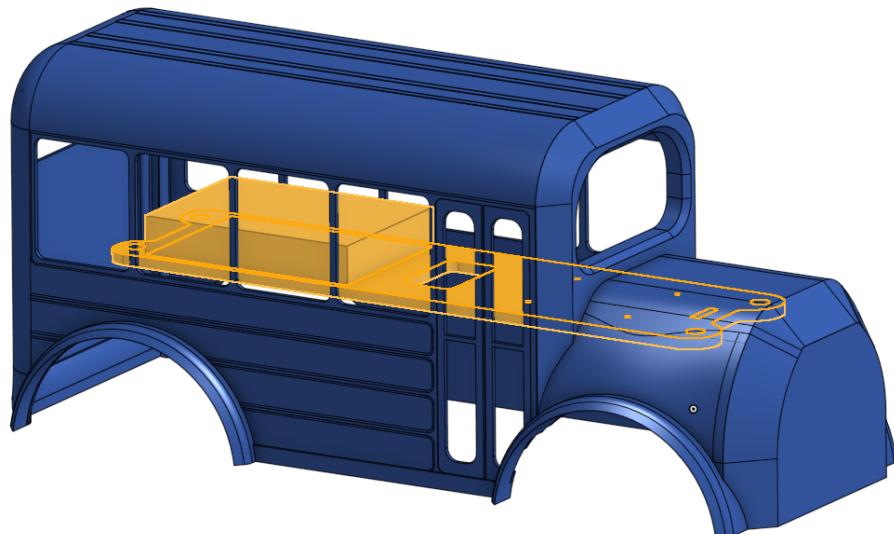


FIGURE 2.5 – Modèle 3D assemblés de la carrosserie et de la plaque de plexiglas

Support caméra

La dernière partie mécanique indispensable à notre projet était le support de la caméra. C'est cette partie qui à le plus évolué au cours du projet. Nous pensions d'abord à un support classique type bras mécanique. Avant de nous apercevoir qu'un trou initialement présent dans le châssis de la voiture pourrait faire l'affaire.

Puis finalement, il fallu changer de modèle de caméra pour passer sur un angle plus large. Après de nombreux essais et diverses combinaisons expérimentées, nous sommes arrivés à la conclusion que la solution la mieux adaptée serait de mettre la caméra directement sur la carrosserie de la voiture finale.

Fixation de la carrosserie et design

Ensuite, après l'assemblage final, nous nous sommes focalisés sur sa fixation. Celle-ci se devait d'être impeccable et parfaitement stable pour éviter que la caméra fixée directement dessus, ne soit sujet aux vibrations et ainsi au bruit, pouvant erronner nos données d'entraînement. Le tout devait cependant être enlevable facilement pour pouvoir accéder à la batterie externe notamment.

Notre binôme a donc opté pour l'option d'aimants, permettant une force suffisamment puissante pour maintenir la carrosserie en place et également assez facilement ôtable par une main humaine, pour ouvrir l'ensemble et accéder aux composants du car.

La mise en place des éléments de fixation fut laborieuse et complexe parfois mais le résultat final est satisfaisant et surtout stable, c'est ce qui nous importe le plus !



FIGURE 2.6 – carrosserie assemblée

Pour finir, en attendant la caméra et pour terminer le côté agréable à l'oeil ainsi commencé, nous avons choisi de nous la jouer "Pimp my 4xCar". Nous avons repeint le tout

en jaune "bus scolaire" et lui avons ajouté des décos à l'effigie de l'école. Nous avons même ajouté des fenêtres en plexiglas pour appuyer le réalisme du bus.

Piste d'entraînement

Un travail également assez chronophage que nous avons dû réaliser, est la création de piste pour entraîner la voiture.

Effectivement il était nécessaire de créer des pistes conformes à la réglementation de la course pour pouvoir exercer le réseau de neurones. N'ayant aucune information sur le terrain des compétitions, ou sur le type de lignes réalisées(craies, peinture, etc.) hormis leur couleur et leur distance de séparation, nous avons donc décidé d'entraîner notre voiture sur des pistes faites sur bitume grâce à du scotch coloré.

Nous avons, en tout, créer quatre pistes d'entraînement pour faire rouler le 4xCar.

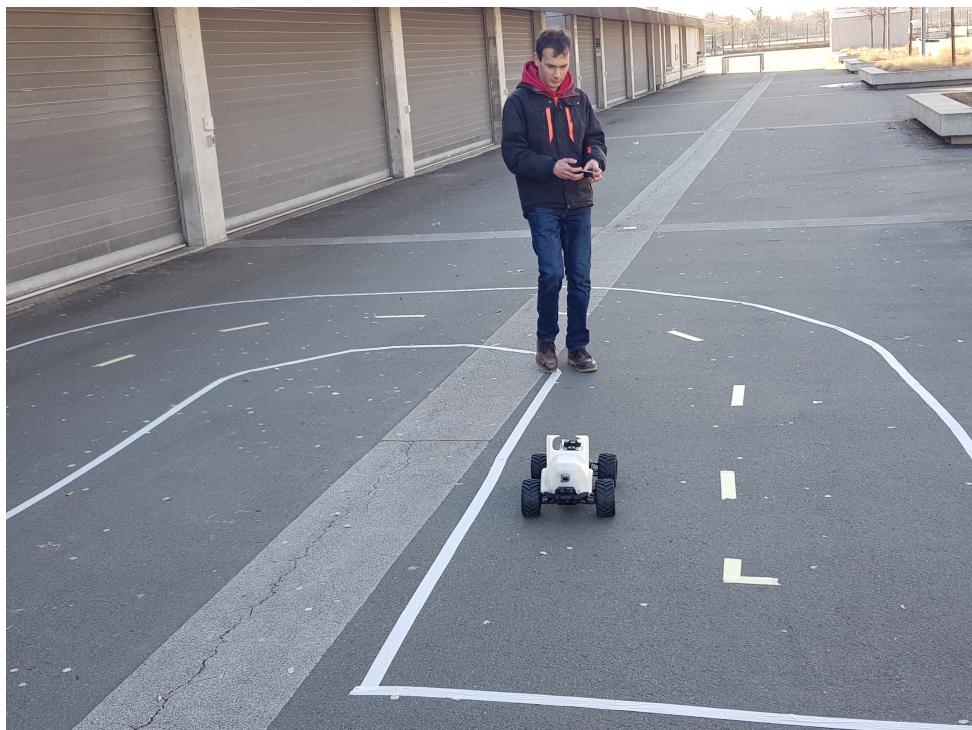


FIGURE 2.7 – Eloi entraînant la voiture sur notre piste

Finalement, le résultat final est plus que satisfaisant et il s'inscrit pleinement dans la démarche de faire ce projet jusqu'au bout. Pas seulement avoir quelque chose de fonctionnel, mais également avoir un prototype final qui pourrait être présenté à une entreprise sans avoir à pâlir de son aspect rudimentaire et provisoire.

Chapitre 3

Bilan et perspectives d'amélioration

3.1 Analyse des performances

Nous avons testé de nombreuses versions de l'architecture DeepPiCar afin d'essayer d'optimiser aussi bien le temps d'inférence et la précision des résultats. Avant de comparer les performances, il est nécessaire de bien préciser les métriques utilisées. Pour le temps d'inférence, toutes les mesures ont été effectuées sur la Raspberry 3B+, avec le processus lancé en priorité maximale. Le code n'a a priori pas changé, sauf entre la version PyTorch et la version Keras mais ces modifications n'ont pas dû affecter la performance générale. Pour les mesures de performance, nous avons utilisé 3 métriques différentes. Comme dit précédemment, nous ne cherchons pas à obtenir en sortie la commande exacte de la consigne mais une commande s'en rapprochant suffisamment pour amener à une trajectoire correcte. Ce qui nous intéresse est donc la distance entre la consigne et le résultat inféré. Nos trois métriques se présentent de la façon suivante, soit r le résultat inféré, c la consigne et avec $n = 3, 5, 8, 10$.

$$f(x) = \begin{cases} 1 & \text{si } r \in [c - n, c + n], \\ 0 & \text{sinon.} \end{cases}$$

Dans un premier temps, nous avons testé les architectures en utilisant PyTorch. Chacun des réseaux a été entraîné sur 10 époques, ce qui nous a permis d'obtenir un premier aperçu des performances.

	Couches convolutionnelles	Couches denses	Précisions	Temps d'inférence
Modèle 1	(3->9) calques, 5x5, pas de 2 ; (9->12) calques, 5x5, pas de 2 ; (12->18) calques, 5x5, pas de 2 ; (18->24) calques, 3x3 ; (24->32) calques, 3x3	(576 ->50) ; (50 ->10) ; (10 ->1)	$\pm 3 : 0.854$ $\pm 5 : 0.9155$ $\pm 8 : 0.951$ $\pm 10 : 0.958$	0.078s
Modèle 2	(3->9) calques, 5x5, pas de 2 ; (9->12) calques, 5x5, pas de 2 ; (12->18) calques, 5x5, pas de 2 ; (18->24) calques, 3x3 ; (24->64) calques, 3x3	(1152 ->50) ; (50 ->10) ; (10 ->1)	$\pm 3 : 0.8425$ $\pm 5 : 0.9185$ $\pm 8 : 0.955$ $\pm 10 : 0.9546$	0.08s
Modèle 3	(3->9) calques, 5x5, pas de 2 ; (9->12) calques, 5x5, pas de 2 ; (12->18) calques, 5x5, pas de 2 ; (18->24) calques, 3x3 ; (24->64) calques, 3x3	(1152 ->100) ; (100 ->50) ; (50 ->10) ; (10 ->1) ;	$\pm 3 : 0.8735$ $\pm 5 : 0.9245$ $\pm 8 : 0.948$ $\pm 10 : 0.9535$	0.08s
Modèle 4	(3->12) calques, 5x5, pas de 2 ; (12->18) calques, 5x5, pas de 2 ; (18->24) calques, 5x5, pas de 2 ; (24->32) calques, 3x3 ; (32->64) calques, 3x3	(1152 ->100) ; (100 ->50) ; (50 ->10) ; (10 ->1)	$\pm 3 : 0.818$ $\pm 5 : 0.902$ $\pm 8 : 0.938$ $\pm 10 : 0.9515$	0.13s

Les performances étaient assez encourageantes, cependant les temps d'inférence étaient largement trop longs. Aucune architecture ne nous permettait d'atteindre notre objectif de 20 images/seconde, c'est pour cela que nous sommes passés à Keras. La meilleure vitesse de la librairie nous a permis d'avoir une plus grande liberté au niveau du nombre de calques des couches convolutionnelles. Nous avons donc testé les architectures suivantes, d'abord avec les données du simulateur. Pour Keras, nous avons arrêté d'enregistrer précisément les temps d'inférence car l'objectif était de toute façon atteint.

	Couches convolutionnelles	Couches denses	Pré-traitements	Précisions
Modèle 1	(3->24) calques, 5x5, pas de 2 ; (24->36) calques, 5x5, pas de 2 ; (36->48) calques, 5x5, pas de 2 ; (48->64) calques, 3x3 ; (64->64) calques, 3x3	(1152 ->1164) ; (1164 ->100) ; (100 ->50) ; (50 ->10) ; (10 ->1) ;	Normalisation	$\pm 3 : 0.325$ $\pm 5 : 0.417$ $\pm 8 : 0.531$ $\pm 10 : 0.618$
Modèle 2	(3->24) calques, 5x5, pas de 2 ; (24->36) calques, 5x5, pas de 2 ; (36->48) calques, 5x5, pas de 2 ; (48->64) calques, 3x3 ; (64->64) calques, 3x3	(1152 ->1164) ; (1164 ->100) ; (100 ->50) ; (50 ->10) ; (10 ->1) ;	Aucun	$\pm 3 : 0.713$ $\pm 5 : 0.8725$ $\pm 8 : 0.928$ $\pm 10 : 0.951$
Modèle 3	(3->24) calques, 5x5, pas de 2 ; (24->36) calques, 5x5, pas de 2 ; (36->48) calques, 5x5, pas de 2 ; (48->64) calques, 3x3 ; (64->64) calques, 3x3	(1152 ->100) ; (100 ->50) ; (50 ->10) ; (10 ->1) ;	Aucun	$\pm 3 : 0.737$ $\pm 5 : 0.858$ $\pm 8 : 0.925$ $\pm 10 : 0.947$

On remarque de façon évidente que le pré-traitement n'apporte pas de bons résultats

dans cette configuration. Nous avons donc laissé tomber la normalisation. L'architecture simplifiée du modèle 3 obtenant pratiquement les mêmes résultats, nous avons décidé d'adopter ce réseau définitivement. De façon générale, les résultats ne sont pas très bons, mais ils sont principalement dus aux données générées qui sont parfois difficilement interprétables, même par un humain. Lorsque nous avons introduit les données réelles, les résultats se sont nettement améliorés.

	Nombre d'époques	Quantité de données	Précisions	Valeurs aberrantes (différence >20)
Modèle 1	50	Entraînement : 25364 Validation : 5108	$\pm 3 : 0.702$ $\pm 5 : 0.884$ $\pm 8 : 0.977$ $\pm 10 : 0.992$	0
Modèle 2	200	Entraînement : 35472 Validation : 6888	$\pm 3 : 0.787$ $\pm 5 : 0.942$ $\pm 8 : 0.995$ $\pm 10 : 0.999$	0

Lors du premier entraînement, les résultats étaient mitigés et nous avons découvert un problème en le testant plus tard. Le jeu de données que nous avions utilisé n'était pas équilibré. Il y avait environ 20% plus d'images de virages à droite que de virages à gauche, ce qui faisait que la voiture avait une tendance à aller à droite. Dans le deuxième cas, nous nous sommes efforcés d'utiliser un jeu le plus équilibré possible. La différence se voit déjà sur les résultats théoriques. Nous avons utilisé plus d'époques, mais les précisions à ± 8 et ± 10 sont quasi-parfaites. Par ailleurs, dans chacun des cas, on remarque qu'il n'y a aucune valeur vraiment aberrante, ce qui est très positif. Cela signifie que le réseau interprète correctement la totalité des images. Malgré ces résultats excellents, nous pouvons écarter la théorie du sur-apprentissage. Si le problème se produisait, le réseau aurait la même précision pour toutes les tolérances car il donnerait systématiquement la valeur exacte.

3.2 Erreurs et problèmes rencontrés

Nous avons bien entendu fait quelques erreurs ou emprunté certaines voies sans issues, cela faisant pleinement partie de la phase d'exploration de solutions.

3.2.1 Métrique et taux d'apprentissage

L'entraînement de réseaux de neurones est une science inexacte et qui repose fortement sur le jugement humain. Les librairies et les architectures sont en effet devenues tellement complexes qu'il n'est plus possible d'anticiper les résultats. Cette faculté de jugement s'acquiert par l'expérience, ce dont nous manquons cruellement. Dans certains cas, elle peut être remplacée par une recherche automatique des hyper-paramètres optimaux, mais cette procédure demande énormément de temps et de puissance de calcul et peut être complexe à implémenter. L'entraînement des réseaux se fait grâce à un algorithme d'optimisation qui accepte deux principaux paramètres. Le premier est la métrique à optimiser. Dans notre cas, nous avons fait le choix le plus courant d'optimiser la précision de la sortie. La précision est calculée en fonction de la proportion des sorties égales à la commande originale. Cependant, dans notre cas, nous ne cherchons pas forcément à obtenir exactement

le même résultat mais simplement une commande assez similaire. Nous obtenions donc toujours des précisions très faibles.

Un bien meilleur indicateur de l'entraînement de notre réseau est la fonction d'erreur (*loss function* en anglais). Cette fonction est un indicateur de la différence entre la sortie obtenue et la sortie souhaitée. Tant que cet indicateur diminue, le réseau est donc en train d'apprendre. Le second paramètre principal de l'optimiseur utilisé est le taux d'apprentissage. Il s'agit d'un coefficient donné à la fonction d'erreur permettant de pondérer la rétro-propagation des erreurs. Plus le taux d'apprentissage est faible, moins le réseau apprendra à chaque nouvelle itération. Cependant, choisir une valeur trop grande est aussi dangereux. Si le réseau apprend trop vite, il risque de dépasser l'optimum et de ne pas réussir à apprendre efficacement. Au contraire, une valeur trop faible rend l'entraînement beaucoup trop long et le réseau n'a pas le temps d'apprendre. Avec PyTorch, nous avons trouvé que la valeur 10^{-3} de taux d'apprentissage donnait de bons résultats. Sur Keras, nous sommes restés bloquées plusieurs heures avant de nous rendre compte que la seule valeur donnant de bons résultats était 10^{-4} . Une valeur supérieure ne permettait pas de trouver l'optimum.

3.2.2 Mécanique

Pour la partie mécanique, nous avons dû explorer différentes pistes afin de trouver la solution que nous jugions optimale.

Pour ce qui est de la découpeuse laser, chacune des réalisations était fonctionnelle immédiatement. Nous avions pris grand soin de faire les mesures les plus précises et les plus rigoureuses possibles avant d'entamer la découpe.



FIGURE 3.1 – Échec d'impression de la face supérieure du car

En revanche la partie impression 3D fut laborieuse. Entre les erreurs d'impressions dues à un bug de l'imprimante, la bobine n'ayant plus assez de PLA et les pièces cassées suites à certains chocs lors de l'assemblage... Nous avons consacré beaucoup de temps à cette partie mécanique.

De plus, la fixation de la carrosserie relève, en certains points, plus du bricolage et du rafistolage que d'un travail d'ingénieur.

Malgré tout, l'ensemble fonctionne et nous permet de confirmer une fois de plus ce fameux dicton "si ça paraît stupide mais que ça fonctionne, alors ça ne l'est pas" !

3.3 Résultats et bilan général

Le 4xCar

Le prototype final est ainsi satisfaisant. Il est robuste, stable et propre.



FIGURE 3.2 – Bus version finale

De plus il est capable de s'orienter seul et de suivre la piste de manière autonome grâce à son entraînement. L'orientation des roues est en adéquation avec la courbure de la piste et la voiture cherche à rester réellement sur le centre de la piste.

Toutefois, il est vrai que parfois, la voiture sort de piste sans que nous en sachions exactement la raison. Nous avons par exemple remarqué que les virages à 90° avaient tendance à poser des problèmes. Il faudra donc essayer d'améliorer l'entraînement avant la participation à la compétition IRONCAR.

Apports personnels

Premièrement, ce PFE nous a permis de gérer un projet dans son intégralité, en totale autonomie. De plus, la répartition du travail entre les deux membres du binôme était équitable, chacun s'impliquant dans la partie du projet où il était le plus à l'aise tout en s'informant sur les avancées de son partenaire, dans un but d'apprentissage de nouvelles compétences. Le duo était investi et complémentaire.

Bien qu'évidentes, les notions d'adaptabilité et de gestion de projet sont essentielles dans le quotidien d'un ingénieur. Ce PFE nous a donc permis de renforcer ces aptitudes, qui seront dans quelques semaines, notre quotidien.

Ensuite, comme nous le souhaitions, le sujet de ce travail était parfaitement adapté à la mise en pratique de notions d'intelligence artificielle étudiées à l'étranger, de manière un peu trop théorique. En effet, l'IA étant au cœur de toutes les attentions ces dernières années, il nous semblait essentiel d'essayer, de ne serait-ce que d'effleurer du doigt certaines de ces applications. Nos compétences en Deep learning et Machine Learning ont bien entendu été confirmées et améliorées. Construire un réseaux de neurones nous paraissait abstrait il y a quelques mois, aujourd'hui nous avons su en proposer un fonctionnel.

De plus, le 4xCar nous a permis de consolider les bases solides que nous avions dans le domaine informatique mais également mécanique. Il nous a permis d'utiliser les machines mises à disposition au Fabricarium et de gagner en compétences au niveau de l'utilisation de ces dernières.

Enfin, il nous parait primordial de préciser que ce projet nous a énormément plu en général ! Conduire une voiture téléguidée pendant plusieurs dizaines de minutes, que ce soit pour un entraînement de réseau de neurones ou pour réaliser un saut sur un circuit tout terrain, cela reste avant tout du plaisir. Il est tellement plaisant de pouvoir travailler sur des projets qui nous rappellent nos nombreuses heures passées à jouer durant nos tendres années. En cinquième année, pouvoir conduire une voiture comme celle-ci tout en justifiant cela par un "c'est pour notre projet", est une chance ! Avoir un projet aussi ludique et aussi concret nous a énormément satisfaits !

Perspectives d'améliorations

Bien que le résultat global final du projet soit satisfaisant, il est bien entendu améliorable en divers points.

Une vitesse variable en fonction de la piste, selon les virages ou les lignes droites serait par exemple une amélioration utile pour gagner du temps au chronomètre tout en limitant le risque de sortie de piste. Nous tenterons de mettre ceci en place avant la compétition.

De plus, l'intelligence artificielle étant basée sur un entraînement préalable, il est toujours possible d'entraîner davantage la voiture. On peut très bien imaginer des entraînements sur diverses autres pistes, de différentes formes et longueurs bien entendu. Mais on peut également faire varier des éléments plus subtils, comme la luminosité, la matière de la piste, l'intensité des couleurs des lignes de celle-ci ou encore la manière de faire ces lignes (craies, scotch, peinture, etc.) ? Chaque prise en compte d'un élément peut consti-

tuer un apprentissage plus robuste et plus puissant, permettant à la voiture de s'adapter à n'importe quelle conditions.

Enfin, le jour de la compétition IRONCAR, il y aura sans doute des facteurs auxquels nous n'aurons pas pensé. Puisqu'il s'agit de la première participation de l'école à ce type de course, nous savons que certains aspects n'auront pas été anticipés, et qu'il faudra prévoir certaines modifications par la suite, dans le but de viser de meilleurs scores dans les participations futures.

Conclusion

Ce projet nous a permis de mettre en application et de nous familiariser avec certaines notions d'intelligence artificielle étudiées lors de semestres à l'étranger. Le machine Learning est un domaine qui nécessite beaucoup d'intuition, et cette intuition s'acquiert grâce à l'expérience. Ce projet nous permettra donc d'être plus performants sur d'éventuels travaux futurs.

Il nous a également permis de consolider nos connaissances en informatique, en modélisation et en réalisation mécanique. Nous avons dû concevoir un système alliant chacun de ces domaines pour fonctionner en harmonie, et mettre en commun nos compétences pour optimiser notre efficacité.

Nous savons que le projet sera probablement repris par d'autres groupes dans des années futures, et nous nous sommes efforcés de fournir un travail facile à reprendre et de communiquer avec les groupes en 4A et 3A. Nous espérons que notre travail servira de première brique aux futures victoires de Polytech Lille en compétitions IronCar.

Enfin, nous avons pu, lors de ce PFE, mener intégralement un projet de ses fondements à sa présentation finale en passant par sa réalisation. Chacun de ces points est essentiel, puisque c'est ce que nous serons amenés à faire tout au long de notre carrière d'ingénieur.



FIGURE 3.3 – Le 4xCAR terminé

Bibliographie

- [1] Easy-to-use road simulator for little self-driving cars. [`https://github.com/vinzeebreak/road_simulator`](https://github.com/vinzeebreak/road_simulator). Accessed : 25-02-2019.
- [2] Project jupyter. [`https://jupyter.org/`](https://jupyter.org/). Accessed : 25-02-2019.
- [3] Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, and Heechul Yun. Deeppicar : A low-cost deep neural network-based autonomous car. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.