DEJAEGHER Hugo *Polytech Lille 2018/2109*

ELEMVA Brandon *IMA 4e année*

RAPPORT DE PROJET P14 : “VOITURE AUTONOME EN MODÈLE RÉDUIT”

Encadrants : ***VANTROYS Thomas***

REDON Xavier

BOE Alexandre

POLYTECH LILLE

Université de Lille

REMERCIEMENTS

Nous remercions l'aide de nos encadrants Thomas VANTROYS, Xavier REDON et Alexandre BOE qui nous ont encadrés durant ce projet en nous apportant quelques conseils et en nous fournissant tout le matériel dont nous nécessaire à sa réalisation.

Nous remercions également l'équipe IronCar pour avoir eu l'idée d'organiser cette compétition et permettre ainsi à de jeunes créateurs comme nous de se lancer dans le domaine de l'intelligence artificielle.

Nous tenons particulièrement à remercier les IMA5 Erwan DUFRESNE et Eloi ZALCZER qui ont travaillé sur le même projet et qui ont contribué à l'avancement de notre projet en nous fournissant l'aide et les conseils dont nous avions besoin pour progresser.

INTRODUCTION

Le projet IronCar est inspiré de la compétition IronCar France. Il s'agit d'une compétition au cours de laquelle, plusieurs équipes se confrontent en faisant concourir des véhicules de taille réduite lors d'une course sur 2 tours sur un circuit établi par les officiels de la compétition. La particularité de cette course est qu'aucun des véhicules ne doit être piloté par qui que soit : les véhicules doivent pouvoir effectuer le parcours de façon totalement autonome.

Afin de pouvoir rendre le véhicule autonome, il faut donc développer un réseau de neurones profond (Deep Learning). L'objectif de ce réseau : le circuit est mis à notre disposition avant le début des épreuves afin de nous permettre d'effectuer des tours de piste "d'entraînement", c'est-à-dire nous allons piloter le véhicule dans un premier temps sur le circuit et tout en lui faisant effectuer des captures d'image du parcours.

Une fois l'étape de l'entraînement terminé, le véhicule est donc en mesure de refaire le parcours qu'il a gardé en mémoire (via le réseau de neurones). Cela demande, toutefois plusieurs heures de travail et un processeur doté d'une bonne puissance de calcul. À travers ce rapport, nous vous présenterons donc les différentes étapes nécessaires à la réalisation d'un mini véhicule autonome en vu de préparer la compétition IronCar France.

SOMMAIRE

PARTIE 1 : Analyse du projet

I.1 Choix du sujet

L'intelligence artificielle est un domaine phare du monde de demain. Mais c'est un sujet qui reste encore très peu exploité et étudié (sauf dans les laboratoires de recherches ou dans certaines filières d'études).

L'étude des réseaux de neurones n'a jamais été abordée au cours de cette formation, mais c'est par envie et curiosité d'apprendre à les manipuler que nous avons décidé de nous lancer dans cette aventure. Nous souhaitons nous investir dans les travaux menés dans le cadre de la conception de la voiture de demain, et globalement, les systèmes intelligents qui prendront une place importante dans l'avenir..

I.2 Analyse des concurrents

Etant donné qu'il s'agit d'une compétition, les concurrents potentiels à notre projet sont tous les autres participants souhaitant y participer et décrocher la victoire. Nous en avons ciblé 2 que l'on pourrait qualifier de "rivaux potentiels" : l'équipe Axionaut, vainqueur de l'édition de Juin 2018, et l'équipe Patate42, vainqueur de l'édition de Février 2018. Le véhicule de l'équipe Axionaut est parvenu à effectuer le parcours en 29s approximativement grâce à la qualité des données qu'ils utilisent pour la reconnaissance du circuit selon leur dire. le véhicule de Patate 42 détient, cependant, le record avec un parcours effectué en 25s. Nous pouvons également ajouter la performance réalisée par l'équipe des IMA5 sur le même projet avec un temps de 45s sur leur 2e passage.

Notre objectif était donc de parvenir à battre ces temps ou de s'en rapprocher au mieux. Bien évidemment, nous n'avons pas négligé l'efficacité et les performances des véhicules des autres équipes. Mais il semblait plus judicieux de considérer ces 3 équipes afin de nous donner un objectif final à atteindre.

I.3 Cahier des charges

En suivant le règlement de la compétition ainsi que les attentes de nos clients, nous avons établi le cahier des charges suivant, dont le véhicule modèle réduit doit répondre aux critères mentionnés :

Concernant la phase d'apprentissage :

- ➔ Pouvoir être conduit par le biais d'une manette sans fil.
- ➔ Capturer une image toute les 0.1 seconde grâce à la caméra implantée.
- ➔ Stocker dans une base de données les images capturées et leur assigner un label (un titre faisant référence à une action à effectuer)
- ➔ Appliquer des effets aléatoires aux images (ombres, miroir, luminosité par exemple) pour diversifier la base de données sans allonger la durée d'acquisition des données.

Concernant la phase de conduite autonome :

- ➔ Etre capable de rouler en autonomie sur la même piste que celle où il a réalisé son apprentissage.
- ➔ Reconnaître des virages et des lignes droites plus ou moins grandes et adapter sa vitesse en conséquence.
- ➔ Détecter une sortie de piste sans avoir recours à des capteurs autres que la caméra (optionnellement).
- ➔ Pouvoir faire tourner le code du réseau de neurones par le raspberry suffisamment rapidement pour pouvoir réagir le plus rapidement possible (proche du temps réel).

Concernant la structure du support de la Raspberry et la communication :

- ➔ Prévoir un support solide afin que la caméra ne bouge et ne tombe pas à la suite d'une secousse.
- ➔ Être capable de communiquer sans fil avec le raspberry depuis un ordinateur.

Ceux-ci sont donc les points majeurs de notre analyse de projet. La théorie ayant été explorée, nous nous sommes attelés à la conception de notre véhicule autonome. Nous allons voir dans la partie suivante les différentes étapes qui nous ont conduites au produit final.

PARTIE 2 : Réalisation du véhicule autonome

II.1 Choix techniques et matériels

Notre projet peut être décomposé en 2 parties distinctes : une partie informatique ou logiciel qui est au centre même du projet, et une partie électronique. Voyons les plus en détails.

II.1.1 Partie logicielle

Le choix du python comme langage de programmation semble le plus indiqué dans le cadre d'un réseau de neurones. En effet même s'il est loin d'être "le plus rapide", la bibliothèque Numpy lui permet de rester compétitif.

Mais c'est surtout sa syntaxe facile et concise qui nous permettra de progresser plus rapidement et aisément que dans d'autres langages. C'est d'ailleurs un langages très utilisé dans les applications relatives à l'intelligence artificielle.

En outre, en utilisant Python, nous sommes certains de trouver de nombreuses librairies qualitatives et de la documentation. On peut aussi souligner qu'il s'agit du langage de base de la Raspberry et que des bibliothèques Python sont fournies avec le Hat PWM pour le contrôle des moteurs.

II.1.2 Partie électronique

Nous avons choisi de ne pas utiliser d'Arduino en complément du raspberry puisque celui-ci prendrait de la place et nous obligerait à rajouter des câbles et à nous doter d'une meilleure alimentation externe.

De plus, il ne présente pas d'avantages particuliers en comparaison avec un raspberry pi 3 doté d'un module pour le contrôle des servomoteurs. En effet comme nous l'avons dit dans la réponse à la question difficile, une Raspberry seule ne permet pas un bon contrôle de plusieurs servomoteurs mais l'utilisation d'un Hat PWM permet de régler efficacement le problème.

II.1.3 Matériel utilisé

Voici ci-dessous, la liste du matériel que nous avons utilisé ainsi que l'utilité de chaque élément dans notre projet :

- ➔ **1 Monster Truck radiocommandé électrique à l'échelle 1/10 de la marque T2M :** c'est le véhicule que nous avons utilisé durant la totalité du projet;
- ➔ **1 manette de Xbox sans fil :** elle ne sert que lors de la phase d'entraînement du véhicule sur le circuit (pilotage manuelle et capture d'image);
- ➔ **1 Raspberry pi 3 (nous abrégerons en Rpi par la suite):** nous avons choisi de n'utiliser qu'une Rpi 3 au lieu de rajouter une Arduino pour le contrôle des moteurs comme certains groupes : une fois muni du shield moteur, la Rpi est tout à fait en mesure de gérer la commande des moteurs ainsi que le réseau de neurones;
- ➔ ~~**1 ordinateur/PC doté de suffisamment de RAM et d'un processeur performant pour la phase de prétraitement :** le temps nécessaire pour le calcul des données recueillies durant la phase de prétraitement peut être excessivement long si le PC ne dispose pas d'une bonne puissance de calcul. La Rpi n'ayant pas une bonne puissance de calcul, il faut donc gérer le prétraitement des données sur un PC suffisamment puissant (et disposant d'au moins 16 Go de RAM);~~
- ➔ **Un ordinateur avec un accès à internet (haut débit si possible) :** En effet nous avons finalement utilisé Google Colab qui se charge de tourner le code à notre place, on peut donc travailler partout sans avoir un ordinateur puissant avec les bonnes librairies, à la seule condition d'avoir un accès internet.
- ➔ **1 caméra pour raspberry grand angle pouvant produire jusqu'à 30 fps :** le but est de faire le plus de captures d'image possible du circuit (plus on a d'images meilleur est le résultat). Avec cette caméra, nous avons la possibilité d'effectuer 10 captures à la seconde voire plus;
- ➔ **1 set de jumpers mâle/femelle pour breadboard :** pour relier les moteurs au shield de la Raspberry;
- ➔ **1 shield moteur pour Raspberry pi 3 :** afin d'éviter de s'encombrer d'une Arduino supplémentaire (et le résultat obtenu avec et sans l'Arduino étant globalement le

même), nous avons prévu un shield moteur pour le contrôle des servos moteur du véhicule par le Rpi;

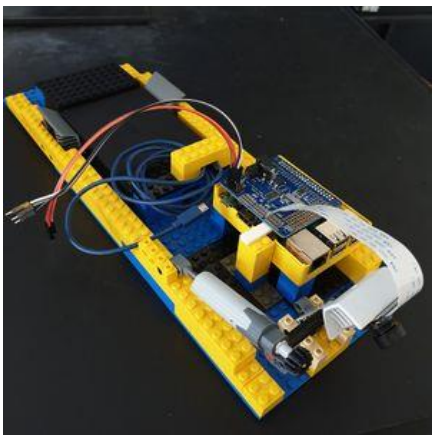
- ➔ **1 batterie externe capable de fournir 5V et au moins 2A pour l'alimentation de la raspberry :** Nous avons choisi une batterie d'un peu plus de 20 000 mAh pour être tranquille;
- ➔ **1 cable USB/micro USB pour relier la raspberry au pc.**

Il est possible de remplacer la manette de Xbox par une manette de Wii ou par une application mobile ou sur une page web. Le principal intérêt de la manette Xbox (au même titre que d'autres modèles) est de posséder un Joystick ce qui facilite l'apprentissage quand on veut utiliser plus de 3 directions.

II.2 Missions réalisées

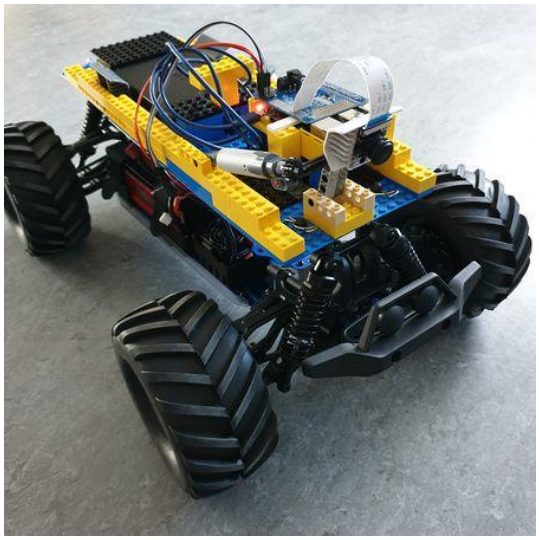
II.2.1 Partie mécanique

Cette partie est très brève : Une fois le matériel reçu et notre plan d'action en tête, nous avons donc pu attaquer la partie réalisation, répartie sur plusieurs semaines de travail.
Commençons par présenter l'aspect visuel de notre véhicule. Nous avons réalisé un support en LEGO afin de pouvoir y déposer la Raspberry et sa caméra ainsi que la batterie nécessaire à son alimentation.



Support pour l'ensemble batterie, Rpi et picamera

La caméra est posée sur un support réglable qui nous permet d'ajuster son angle par rapport au sol (elle doit bien évidemment rester à une hauteur supérieure à 10 cm du sol). De cette façon, lors de la phase de test, cela nous a permis de changer la position de la caméra afin de déterminer si les erreurs de suivi du parcours provenaient des images capturées, donc de la position de la caméra, ou d'autre chose.
La solidité et la fiabilité de la structure sont bien adaptées au type d'exercice auxquel est soumis le véhicule. Nous avons donc obtenu le montage final suivant :



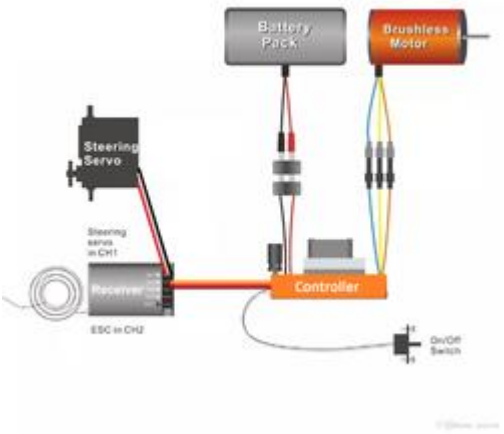
Visuel final de notre véhicule

Ceci est donc la structure globale de notre voiture. Intéressons-nous aux points de vue hardware et software à présent.

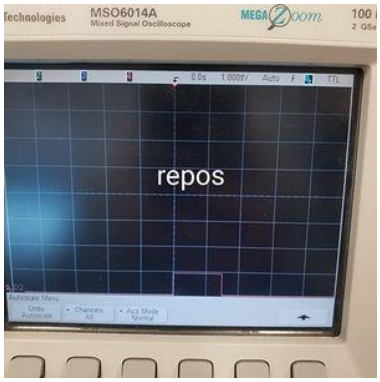
II.2.2 Partie électronique

La partie électronique est essentiellement centrée autour des liaisons entre le Shield de la Raspberry, la Raspberry et les servomoteurs du véhicule. Elle n'est en réalité pas très compliquée : Il suffit de débrancher les broches du servomoteur et du contrôleur de vitesse du récepteur fm de la voiture, et de venir les connecter directement sur le shield du raspberry sur les 2 premiers emplacements (en respectant bien les positions du GND, 5V et S), en utilisant des câbles de breadboard male/femelle pour combler le manque de longueur.

Concernant l'alimentation, on se rend compte que le shield n'a en réalité pas besoin d'être alimenté par le boitier à pile que nous avions prévu. En effet, de la même manière que le récepteur fm, il s'alimente directement depuis la batterie de la voiture, via le contrôleur de vitesse, et redistribue le courant au servomoteur de la direction.

	
Shield moteur Rpi utilisé	***Connexion des moteurs au contrôleur***

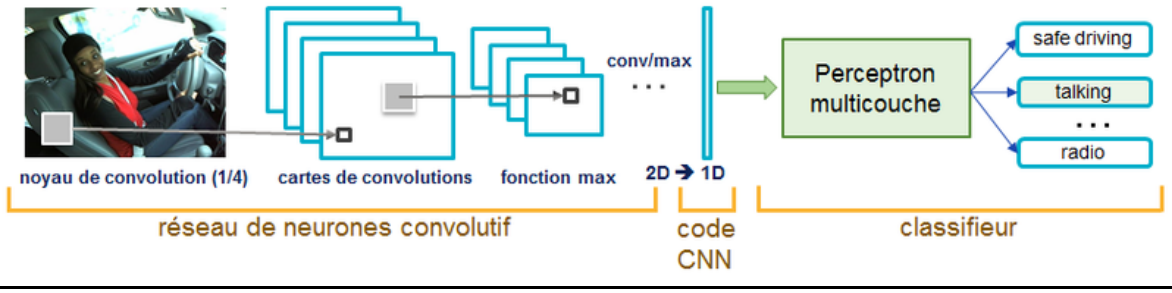
Nous avons effectué différents tests au cours desquels nous avons pu constater que, malgré que nous soyons en mesure de bien contrôler la direction, nous ne pouvions faire tourner le moteur principal qu'en plein régime ou en marche avant. La voiture étant totalement incontrôlable en plein régime il était nécessaire de pouvoir la faire rouler le plus lentement possible afin d'en garder le contrôle. C'est pourquoi, afin d'identifier le problème, nous avons décidé d'effectuer des analyses via l'analyseur de spectre, afin d'observer et de comprendre l'allure des signaux reçus par le contrôleur de vitesse. Nous avons obtenu ceci :

		
Signal véhicule à vitesse max	***Signal véhicule au repos***	***Signal véhicule en marche arrière***

Les résultats obtenus par l'analyseur de spectre nous ont montré la forme des signaux PWM lors de la position repos, lors de la marche avant (à vitesse max) et lors de la marche arrière de la voiture. Nous n'avons pas répertorié les résultats concernant les directions mais comme nous avons pu le réaliser, la forme des signaux obtenus est exactement la même qu'il s'agisse des directions ou de la vitese, ce qui était en réalité prévisible, mais nous a permis d'avoir une piste fiable dans la résolution de notre soucis de gestion de la vitesse des moteurs du véhicule.
De plus, le moteur que nous cherchons à contrôler n'est pas un servomoteur. Il s'agit d'un moteur contrôlé par le contrôleur moteur de la voiture, qui lui-même reçoit un ordre du récepteur FM (ou du shield dans notre cas). Cet ordre n'est autre qu'un signal PWM classique, de la même forme que celui reçu par le servomoteur contrôlant la direction.
Connectés au shield moteur, nous avons donc le servomoteur pour les directions ainsi que le moteur brushless (sans balai) pour la vitesse du véhicule (voir schéma précédent).
Ceci résume notre réalisation dans la partie électronique. Voyons à présent le coeur du projet : la partie informatique.

II.2.3 Partie informatique

- **Approche Théorique du Deep Learning**



Description d'un réseau de neurone convolutif

Avant de commencer la description du travail effectué, il est essentiel de savoir en partie comment fonctionne un réseau de neurones.

Les réseaux de neurones convolutifs (ou CNN pour Convolutional Neural Network) sont les modèles les plus performants pour la classification d'images. À l'entrée du réseau, nous insérons une image qui sera mise sous forme de matrice de pixels à laquelle il attribue 2 dimensions pour les niveaux de gris et une 3e pour les couleurs RGB. Le réseau se décompose en 2 parties :

- **Une partie *convolutive* :** c'est une sorte d'extracteur de caractéristiques des images. En d'autres termes, une image est passée à travers plusieurs filtres d'affilé créant ainsi de nouvelles images appelées *cartes de convolutions*. Certains filtres intermédiaires réduisent la résolution de l'image par une opération de maximum local. Ainsi, les cartes de convolutions sont mises à plat et concaténées en un vecteur de caractéristiques, appelé code CNN;
- **Une partie *perception multicouche* à laquelle est connecté le code CNN :** c'est un type de réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement, les couches sont entièrement connectées entre elles : c'est un réseau de propagation (chaque couche est constituée d'un nombre variable de neurones, les

neurones de la dernière couche étant les sorties du système global). Les valeurs numériques obtenues sont généralement normalisées entre 0 et 1.

De cette façon une image qui a une profondeur de 3 couches (le nombre 3 correspondant aux 3 canaux RGB) pourra ainsi résulter en une matrice d'une profondeur de 5, si le réseau convolutif est constitué de 5 filtres. Avec la technique du transfert learning, on réduit la complexité du CNN en utilisant des réseaux pré-entraînés (on exploite la connaissance acquise sur un problème de classification général pour l'appliquer de nouveau à un problème particulier).

Outre cela, le Deep Learning, repose sur des notions de probabilité. En effet, il faut être en mesure de déterminer la probabilité qu'une même occurrence se produise plusieurs fois de suite mais aussi le nombre de succès et d'échecs que peut rencontrer notre modèle afin de déterminer si oui ou non il est valide.

Précisons que la version de Python utilisée ici est Python 3.

- **Application sur notre véhicule : pilotage manuel**

Avant de débuter les travaux, nous avons commencé par configurer la Rpi en installant son OS. Cette manœuvre ayant été faite, nous avons décidé de nous occuper de la partie communication entre la Rpi, la manette et la picamera fournie avec la Rpi.

L'étape suivante du travail était la récupération des adresses de chaque bouton de la manette de Xbox sur la Rpi. Via la librairie *evdev*, il est possible de pouvoir établir la communication entre la Rpi et la manette de Xbox. Nous nous en sommes également servis afin de récupérer chaque événement associé aux différentes commandes de la manette (boutons pressés/relâchés, mouvements du Joystick). Afin d'utiliser la librairie *evdev*, nous avons saisi la ligne ci-dessous dans l'entête de notre programme dédié au pilotage manuel : `import evdev` .

Manette de Xbox utilisée pour le pilotage manuel

La manette étant à présent connectée, nous avons procédé à l'activation de la caméra de la Rpi. Pour cela, nous avons utilisé les fonctions Python de la bibliothèque *Picamera*. Notre objectif était de réaliser une prise d'images toutes les 100ms, au format adapté (jpg dans notre cas). Pour utiliser les fonctions de la bibliothèque Picamera, il faut saisir la ligne suivante au début du programme python: `from picamera import PiCamera` .

Concernant le contrôle des moteurs, nous avons utilisé ceci dans notre programme :

`kit.servo[1].angle = 120`

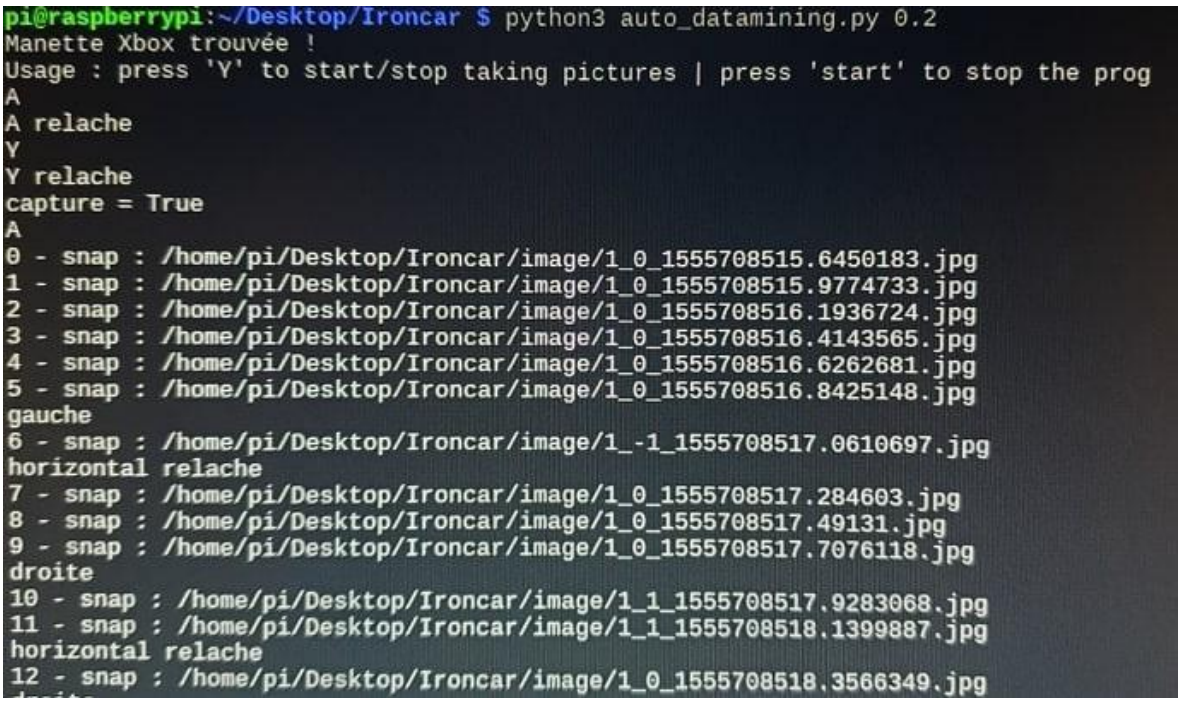
120 est une valeur d'angle pour la rotation des roues choisie entre 0° et 360° (nous éviterons tout de même d'utiliser ces 2 valeurs extrêmes pour ne pas endommager le matériel, puisque dans notre cas l'angle peut seulement varier entre 60 et 140).

La fonction ***kit.servo[].angle*** permet donc de commander la direction du véhicule ainsi que l'angle de rotation que le véhicule prendra sur lorsqu'il devra tourner.

En utilisant des threads python nous avons regroupé les 2 programmes (contrôle par la manette et prise d'images par la Picamera) dans notre programme principal : `from threading import Thread` . Cela nous permet également de connaître l'état bouton (donc la direction) au moment de la prise d'images.

Ces étapes étant achevées il ne restait plus qu'à tester notre programme en exécutant la ligne suivante dans un terminal Linux (veillez à être dans le même dossier que le fichier Python dédié au pilotage manuel lorsque vous compilerez celui-ci) : python3 manual_drive.py <délais de capture en secondes>. À la place de <délais de capture en secondes> , il faut saisir le nombre de secondes souhaitée entre chaque capture d'image. Dans notre cas, 0.2s si on souhaite capturer 5 images par secondes (le code a été renommé entre temps pour plus de lisibilité).

Le résultat visible à l'écran est donc le suivant :



Fonctionnement du mode pilotage manuel vu du terminal

En pressant le bouton A de la manette de Xbox, on fait avancer le véhicule (en le relâchant il s'arrête). Même chose pour le bouton B sauf qu'il recule cette fois. En pressant le bouton Y on active la capture d'image, en le pressant à nouveau on la désactive. La capture s'effectue uniquement quand la voiture est en marche avant. Les images sont nommées en fonction de la direction (gauche, droite, tout droit).

Si aucune manette Xbox n'est connectée en Bluetooth à la Rpi, le code se ferme immédiatement.

- **Application sur notre véhicule : pilotage automatique**

Les images capturées doivent être prétraitées sur un ordinateur doté d'une RAM de 16 Go minimum, ce que la Rpi ne peut fournir Afin de simplifier la tâche, nous avons utilisé la plateforme ***Google Colaboratory*** qui un environnement basé sur le même concept que ***Jupyter notebook*** et exécuté dans le Cloud. Il présente les avantages suivant :

- facile à prendre en mains;
- librairies utiles au deep learning déjà installées et prêtes à l'emploi;
- environnement Python 2.7 ou 3.5 disponibles et utilisables;
- visualisation des résultats en temps réel;
- 16 Go de RAM disponibles pour l'exécution de notre code.

C'est donc sur cette plateforme que nous avons effectué le prétraitement de nos données et l'entraînement de notre réseau de neurones. Nous avons donc réalisé les tâches suivantes :

- Upload de notre dataset sur le Notebook de Google Colab :

```
!rm *.zip        //pour éviter les conflits
from google.colab import files
files.upload()
!unzip image.zip
dataset = "image"
```

- Rédaction d'une fonction qui, en fonction de leur nom, convertit les photos en 3 tableaux numpy (X : l'image, Y : la direction, Z : la vitesse) :

```
def load_photos(dataset)
  ...
  return X, Y, Z
```

- Ajout d'une partie dédiée à l'augmentation de la quantité de données pour l'entraînement du CNN. Cette partie dispose d'une fonction miroir qui permet de ce fait d'avoir exactement le même nombre d'images correspondant à des virages à droite que d'images correspondant à des virages à gauche :

```
def mirror_image(X,Y):
  ...
  return X_mirror,Y_mirror

def random_brightness(X,Y):
  ...
  return X_bright, Y_bright
```

- ➢ Implémentation d'une partie post-traitement dédiée à la validation du modèle : on affiche l'évolution des performances du réseau CNN à la fin de l'entraÎnement ainsi que le pourcentage de prédictions correctes.

Afin de pouvoir réaliser quelques vérifications sur le code, nous avons rapidement tracé au sol un virage avec du scotch blanc et pris quelques images (programme de capture mais sans la voiture) et nous les avons uploadés sur le Notebook. Nous avions donc un petit dataset de 550 images (sans augmentation) avec lequel nous avons pu vérifier le bon fonctionnement du programme.

L'objectif final de cette phase est de pouvoir établir un modèle pour notre réseau de neurones dont l'extension est .h5 et qui sera chargé dans notre programme de pilotage automatique principal. Pour lancer le pilotage automatique (une fois le modèle conçu), il faut saisir la commande suivante sur le terminal Linux :

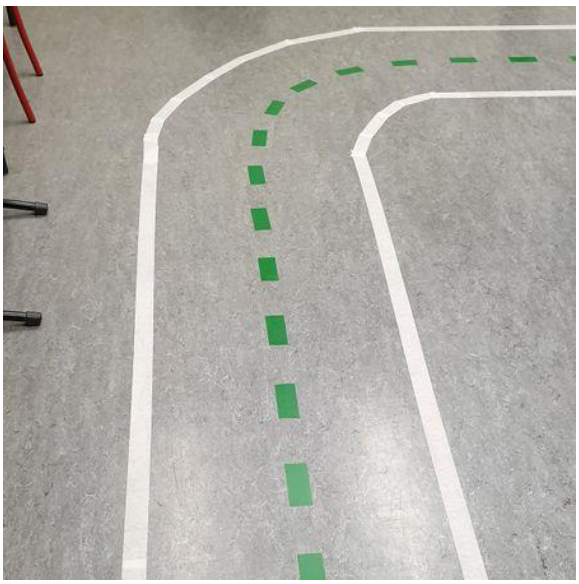
`python3 auto_drive.py <nom du modèle>` .

PARTIE 3 : Analyse des résultats obtenus

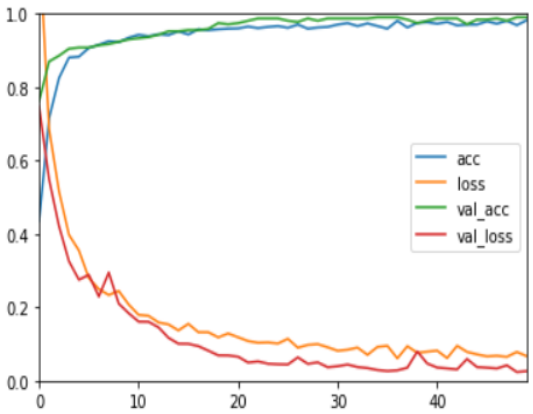
III.1 Les performances obtenues

À l'issue de notre test final, nous avons obtenu notre modèle polytest.h5, et en chargeant d'autres images pour la validation du modèle, nous avons obtenu le résultat suivant :

187 / 187 correctement prédits <=> 100.000000 % de réussite . Nous avons entraîné notre véhicule sur un circuit en forme de "U" dont les virages sont à 90° mais restent assez simples. Ce résultat n'est donc propre qu'à ce circuit de test. Le pourcentage d'erreur, en général, reste cela dit peu important car inférieur à 10%. A cause d'un soucis dont nous reparlerons plus tard, nous n'avons pas pu aller plus loin que cela. Cela dit, notre modèle est une réussite. Le véhicule fut en mesure de suivre le parcours indiqué de façon totalement autonome.



Piste d'essai du véhicule



Courbes de résultats post-traitement

III.2 Les problèmes rencontrés

1er problème :

Afin d'élaborer notre dataset d'images nous avons utilisé l'outil Tensorflow mais le test fut peu concluant. Bien que très pratique en termes de traitement d'image, Keras semble être une bonne alternative en termes de facilité de prise en main et de rapidité de traitement et de calcul.

➔ **Solution : nous nous sommes donc tournés vers l'outil Keras.**

2e problème :

Quelques problèmes de praticité nuisaient au fonctionnement optimal du code du contrôle manuel de la voiture :

A chaque connexion de la manette Xbox, le numéro de l'événement associé à la connexion était susceptible de changer

il faut donc, à chaque connexion, changer la ligne de code associée pour que cela fonctionne.

```
...
gamepad = InputDevice('/dev/input/event3')
...
```

➔ **Solution : Un peu plus tard nous avons fini par rédiger une fonction capable de lire la liste des événements, reconnaître celui associé à la manette et changer le numéro d'événement en conséquence.**

3e problème :

Pour le contrôle de la vitesse, nous utilisions :

```
kit.continuous_servo[1].throttle = 1      //1 correspond à la vitesse max, -1 à la vitesse max dans l'autre sens .
```

La fonction **kit.continuous_servo[].throttle** sert à faire tourner un servomoteur de manière continue. Or, le moteur que nous cherchons à contrôler n'est pas un servomoteur. Il s'agit d'un moteur contrôlé par le contrôleur moteur de la voiture qui, lui-même, reçoit un ordre du récepteur FM (ou du shield dans notre cas). Cet ordre n'est autre qu'un signal PWM classique, de la même forme que celui reçu par le servomoteur de la direction.

➔ **Solution : en utilisant la même fonction que pour la direction (kit.servo[].angle), nous avons pu commander la vitesse de notre voiture correctement.**

4e problème :

Il s'agit là du problème le plus important que nous avons rencontré en termes de perte de temps. Lors de nos phases de test, le véhicule ne réagissait pas de la façon escomptée. Malgré un dataset parfait, des programmes de pilotage manuel et automatique à priori fonctionnels, et des résultats très satisfaisants lors de la validation du modèle, le véhicule ne parvenait pas à suivre le parcours et agissait de manière totalement incohérente. Nous avons donc réfléchi à quelle pouvait bien être la cause de cet échec et avons émis plusieurs hypothèses :

1. l'architecture du CNN de Nvidia que nous avons récupéré est prévue pour 5 directions (gauche, gauche légèrement, tout droit,... par exemple). Or nous avons adapté sa sortie pour seulement 3 directions. Il se pourrait alors qu'il ne réagisse plus correctement;
2. L'angle de la caméra était peut-être mauvais. Mais au vu des résultats, ce détail ne peut pas justifier à lui tout seul le comportement de la voiture;
3. L'absence de pointillés sur notre parcours empêche la voiture de bien se repérer, notamment dans les virages. Encore une fois cela semblait peu probable au vu du fait que même une ligne droite n'était pas reconnue;

4. La luminosité extérieure posait problème à cause des fortes variations d'une minutes à l'autre (avec le passage des nuages par exemple). En effet puisque le réseaux met un peu plus de 20 minutes à s'entraîner, la luminosité a peut-être fortement varié et ainsi le réseaux de neurones n'est plus en mesure de reconnaître quoi que ce soit. Sachant que les concours IronCar se font en intérieur, et que les caméra de raspberry sont très sensibles à la luminosité, c'est la piste de l'éclairage que nous retenons.

➔ **<u>Solution :</u> la caméra que nous utilisons est placée à l'envers sur notre véhicule pour des questions de praticité, ce qui fait que les images qu'elle capture sont à l'envers. Dans notre programme de pilotage manuel, nous avions pris ce paramètre en compte et avions utilisé une fonction de rotation de l'image de 180° afin de la remettre à l'endroit. Ce que nous avions omis de faire dans le programme du pilotage automatique. En rectifiant cela, le véhicule fut enfin fonctionnel.**

Notons que ce problème était en réalité le plus simple à résoudre, mais puisque nous n'avons jamais émis le moindre doute quant à cette possibilité, nous avons perdu de nombreuses et précieuses heures de travail durant 2 semaines, avec lesquelles nous aurions pu améliorer au maximum notre projet.

III.3 Bilan du fonctionnement du véhicule et améliorations possibles

Voici le fonctionnement final de notre projet, en partant du principe que le matériel utilisé est similaire au notre (manette et shield Rpi) :

Réglages et pré-requis

- Il est tout d'abord nécessaire de télécharger les prérequis sur le raspberry (et sur le PC en cas de non-utilisation de Google Colab)
- Connecter les moteurs au shield et effectuer les réglages nécessaires. Il est notamment indispensable de déterminer la plage d'angle d'utilisation des servomoteurs : angle min et angle max. Ceci peut être fait à l'aide de servotest.py.
- Il faut désormais régler la caméra, c'est à dire son inclinaison, sa hauteur (de manière à voir correctement le circuit) et enfin déterminer si elle est droite ou à l'envers.
- Adapter le fichier constantes.py en fonctions des résultats précédents.

Utilisation (mode 3 directions)

- Connecter en ssh la raspberry. L'utilisation d'un écran est également possible mais moins pratique.
- Allumer la manette et lancer sur la raspberry le programme suivant en précisant un délais, 0.1 par exemple : python3 manual_drive.py <délais de capture en secondes>
- Mettre en marche la voiture, il faut alors activer la capture d'images et réaliser plusieurs tours de piste.
- *(optionnel)* Pour assurer la qualité du dataset, il est possible d'insister dans les passages difficiles en réalisant plusieurs passages. Egalement on peut ensuite supprimer les images mal labellisées en les contrôlant.

- De préférence sur Google Colab (Jupyter également possible, avec un PC puissant), lancer le code traitement.ipynb et uploader le dataset. Suivre ensuite les différentes étapes sur ce code (les fonctions se lancent par "bloc", par simple pression sur l'icône exécuter, une par une et dans l'ordre.
- Télécharger le modèle au format .h5 ainsi obtenu et le mettre sur la raspberry dans le même répertoire que les autres fonctions.
- Poser la voiture sur la piste et exécuter le code d'auto-pilotage en précisant le modèle. Le modèle met une trentaine de secondes à charger : python3 auto_drive.py <nom du modèle>.
- Si tout s'est bien passé la voiture suit désormais la piste toute seule !

Des améliorations possibles auraient été de concevoir une carrosserie digne de ce nom, mais aussi d'optimiser le code et la position de la caméra afin de gagner en précision et faire le meilleur chrono possible sur un véritable circuit. Cela aurait permis de rajouter une réelle plus-value à notre projet. Malheureusement, de manière générale, ce sont souvent les problèmes les plus simples qui font perdre le plus de temps.

CONCLUSION

Finalement notre projet est donc fonctionnel et conforme aux normes du concours IronCar sur lequel il a été inspiré. La voiture est capable de suivre une piste en autonomie avec une certaine précision après un entraînement à la manette sur cette même piste ou potentiellement une autre dont les caractéristiques seraient similaires.

Nous demeurons fiers et heureux d'avoir réussi à mener à bien ce projet, qui fut un réel enrichissement et nous a permis de nous familiariser avec le Deep Learning et quelques notions d'intelligence artificielle. Nous avons acquis des notions de base qui sauront nous être d'une grande utilité si nous souhaitons l'appliquer sur des véhicules autonomes à taille réelle et participer, ainsi, à l'évolution du monde de demain. Nous souhaitons bonne chance aux IMA3 qui prennent la relève sur ce projet.

WEBOGRAPHIE

Initiation au Deep learning : donne accès à des cours et des travaux pratiques centrés sur l'apprentissage approfondi

http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/

Pour installer Anaconda

https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-16-04

Lien de notre git

https://github.com/DejaElem/p14_Dejaegher_Elemva

Informations sur la compétition IronCar France

http://www.ironcar.org/

Plateforme Google Colab

https://colab.research.google.com/notebooks/welcome.ipynb

Documentation pour Keras

https://keras.io/