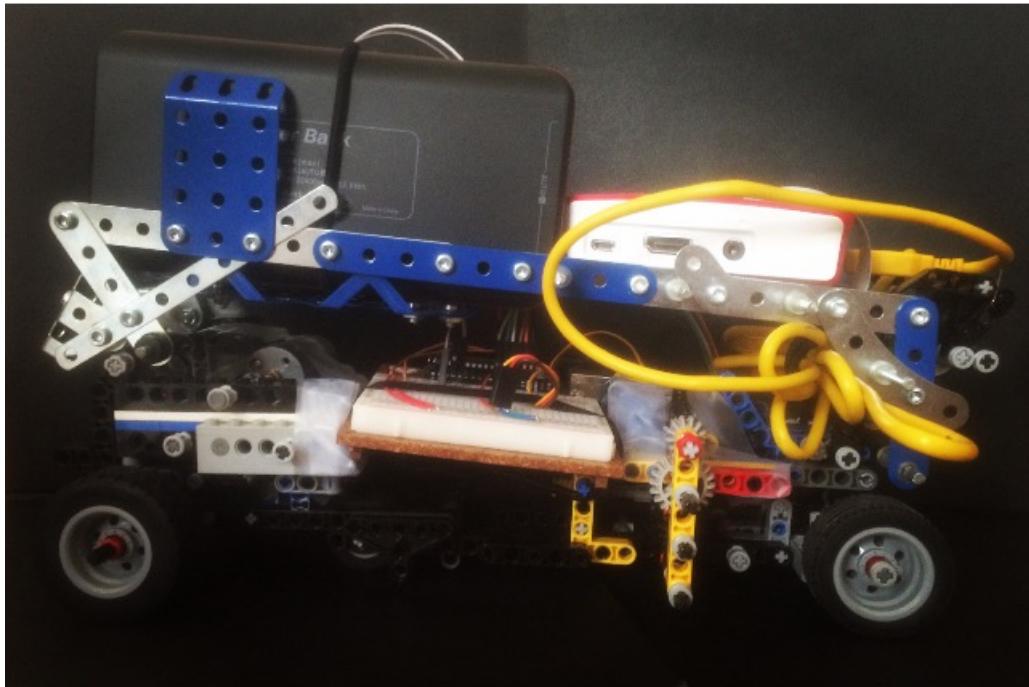


---

# **TIPE**

## **Voiture Autonome**

---



Amaury DE MIGUEL

Pierre-Louis GUILLOU

2018 - 2019



Avec l'utilisation croissantes de technologies liées à l'intelligence artificielle, il est important de comprendre cet intérêt et de pouvoir le mettre en pratique dans le cadre du transport. C'est pourquoi l'objectif de ce travail d'initiative personnelle encadré est de miniaturiser le concept de la voiture autonome avec d'une part le modèle du véhicule et de l'autre la réalisation d'un programme d'intelligence artificielle. Ainsi, le but final est de pouvoir présenter un modèle miniature de voiture capable de suivre le tracé d'une route et de réagir face à la signalétique relative au code de la route ou encore la présence de piétons en travers.

With the increasing use of artificial intelligence technologies, it is important to understand this interest and be able to put it into practice in transports. This is why the objective of this supervised personal initiative work is to miniaturise the concept of the autonomous car, with on the one hand the model of the vehicle and on the other hand the realization of an artificial intelligence program. Thus, the final goal is to be able to present a miniature model of car able to follow the route of a road and react in front of a traffic signage or the presence of pedestrians across.

# Sommaire

<b>Introduction</b>	<b>1</b>
<b>1. Élaboration de la voiture</b>	<b>3</b>
1.1 Cahier des charges . . . . .	3
1.2 La propulsion . . . . .	4
1.2.1 Le moteur . . . . .	4
1.2.2 Le modèle . . . . .	6
1.3 La direction . . . . .	9
1.3.1 Le moteur . . . . .	9
1.3.2 Le modèle . . . . .	9
1.4 Le fonctionnement . . . . .	10
1.4.1 La commande des moteurs . . . . .	10
1.4.2 Le module antenne . . . . .	11
<b>2. Données d'entraînement</b>	<b>13</b>
2.1 L'intérêt du Raspberry . . . . .	13
2.2 Le lien Arduino-Raspeberry . . . . .	14
2.3 La récupération des données . . . . .	15
<b>3. Intelligence Artificielle</b>	<b>16</b>
3.1 Généralités . . . . .	16
3.2 Le principe des réseaux de neurones . . . . .	18
3.2.1 La régression linéaire . . . . .	18
3.2.2 La régression logistique . . . . .	21
3.2.3 Les fonctions d'activation . . . . .	26
3.2.4 Les filtres . . . . .	28
3.3 Adaptation au problème . . . . .	34
3.4 Entraînement pour les réseaux profonds . . . . .	37

3.4.1	La "vectorisation" . . . . .	38
3.4.2	La descente de gradient . . . . .	39
3.4.3	Entraînement du programme . . . . .	40
<b>Conclusion</b>		<b>42</b>
<b>Sources</b>		<b>43</b>



# Introduction

L'intelligence artificielle est au coeur des considérations actuelles. Ce terme est employé de plus en plus fréquemment dans le domaine de la recherche dans le but de changer les habitudes de la société. Ainsi, on pourrait définir l'intelligence artificielle comme un ensemble de techniques visant à simuler les méthodes de pensées de l'Homme. Parmi les projets mis en oeuvre à l'aide de l'intelligence artificielle, il en existe un qui est au coeur de l'actualité scientifique : la voiture autonome. Plus précisément, dans ce cadre d'utilisation, on parle d'apprentissage profond ("deep learning"). La machine va parvenir à établir des liens entre des données plus complexes et de prédire le résultat de nouvelles données. C'est dans cette optique de confrontation à des situations réelles que ce concept d'apprentissage prend son sens pour la voiture autonome. Ainsi, l'objectif est de proposer un moyen de transport qui peut se priver d'un conducteur. Le principe est en l'apparence simple ; en effet, une caméra à l'avant du véhicule permet de percevoir l'environnement puis un système programmé en apprentissage profond va se charger d'agir en fonction après avoir assimilé l'ensemble des règles nécessaire pour une conduite adéquate. Comme représenté sur la figure 1, ce système est programmé pour repérer les différentes silhouettes qui se présentent sur l'image. Tous ces objets sont ensuite délimités par des rectangles distincts. Enfin, il les classe selon leur nature, comme par exemple le cycliste et son vélo au premier plan, et peut ainsi agir en fonction de la situation présente.

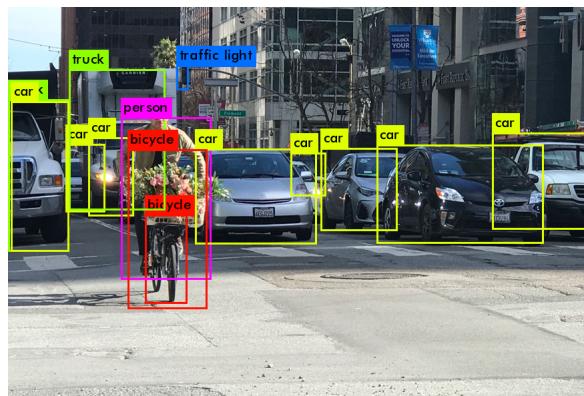


FIGURE 1 – Détection de l'environnement [1]

Dans le cadre du thème du transport, on a donc choisi de s'intéresser à ce concept innovant en l'adaptant à une échelle plus simple, et dans des dimensions plus accessibles. L'objectif de ce projet est donc de concevoir une voiture miniature capable de se diriger sur un circuit donné et d'éviter des obstacles. Le principe utilisé pour la conduite autonome reprend les idées de l'apprentissage profond mais dans des dimensions plus adapté au problème posé.

Ainsi, comment miniaturiser le concept de la voiture autonome ? L'élaboration se fait en deux parties : tout d'abord, il s'agit de concevoir le modèle miniature de la voiture, et ensuite, la partie informatique a pour but de mettre en place le programme qui se chargera de la conduite autonome.

# 1. Élaboration de la voiture

## 1.1 Cahier des charges

Pour la réalisation du projet, l'objectif est de concevoir entièrement le modèle de voiture miniature. Il faut donc définir nos attentes quant à la conception et élaborer un cahier des charges permettant de donner une idée de ce à quoi la voiture doit finalement ressembler. Ainsi, la base est de partir sur un modèle établi qu'il sera possible de modifier en fonction des différentes caractéristiques à respecter.

On se propose à l'origine de prendre le modèle de voiture Lego de la figure 2 à échelle 1/10 qui sera ensuite modifié pour accueillir les différents composants nécessaires pour la faire rouler et aussi permettre la conduite autonome.



FIGURE 1.2 – Modèle de base pour le projet (Illustration par LEGO®)

Cette voiture constitue donc la base sur laquelle la voiture miniature sera construite. Elle possède ainsi l'avantage d'être largement modifiable grâce au principe de Lego mais aussi un contrôle de la direction à l'avant qui est beaucoup plus pratique pour la réalisation du projet.

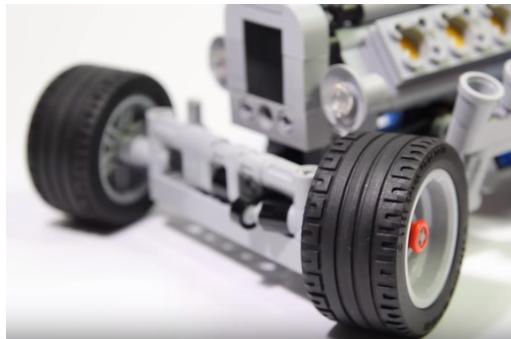


FIGURE 1.3 – Liaison avant (Illustration par BrickBuilder)

Son intérêt sera expliqué dans les parties suivantes. En partant de cela, il s'agit donc de construire toute la structure qui permet d'atteindre l'objectif fixé.

## 1.2 La propulsion

### 1.2.1 Le moteur

Dans cette partie, il s'agit de s'intéresser à comment faire rouler la voiture. Après avoir réfléchi à l'ensemble des éléments qui serait nécessaire pour construire cette voiture autonome et pouvoir être envisageable avec les Lego, on peut évaluer une masse approximative de  $1,5 \text{ kg}$ . De plus, par choix, on se place à une vitesse linéaire de  $0,6 \text{ m.s}^{-1}$  environ pour un déplacement normal de la voiture. Il s'agit donc d'évaluer le couple moteur nécessaire pour subvenir à ces besoins. Pour simplifier les calculs, on ne s'intéresse uniquement qu'à l'action du poids. En effet, il s'agit uniquement d'obtenir un ordre de grandeur de la puissance nécessaire pour faire avancer la voiture.

$$\text{Vitesse linéaire : } v = 0,6\text{m.s}^{-1}$$

$$\text{Rayon des roues : } r = 2,5\text{cm}$$

$$\omega = \frac{v}{r} = 24\text{rad.s}^{-1}$$

$$F_{\text{poids}} = mg = 14,7\text{N}$$

$$P = F_{\text{poids}} \cdot v = 8,82\text{N.m.s}^{-1}$$

Or par définition on a aussi  $P = C\omega$  avec  $C$  le couple recherché

$$\text{D'où } C = \frac{F_{\text{poids}} \cdot v}{\omega} = 0,37\text{Nm}$$

On a donc un ordre de grandeur pour le couple moteur. Au niveau de la rotation, les caractéristiques des moteurs en vente sont en tour par minute. Avec la vitesse de rotation évalué, on en déduit un ordre de grandeur de la vitesse recommandée en tour par minute qui est de 230. Parmi les différents moteurs présents sur le marché, le moteur choisi ne correspond pas scrupuleusement aux caractéristiques établies ci-dessus. En effet, l'idée est de prendre un moteur plus puissant

et d'adapter ensuite la structure de la voiture pour convenir à nos besoins. Ainsi, avec un couple bien plus élevé que celui recommandé par les calculs, on se trouve certains de venir à bout des forces qui pourraient s'opposer au déplacement du véhicule.

Après recherches sur des sites de vente de moteur miniatures, le moteur finalement utilisé est le suivant. On a donc les caractéristiques suivantes : couple de  $0.77N.m$  et vitesse de rotation de 485 tours par minute.



FIGURE 1.4 – Moteur de la propulsion (Illustration par RobotShop)

Il faut ensuite s'intéresser de l'alimentation de ce moteur. En effet, pour permettre à la voiture d'être "autonome", il faut qu'elle puisse se déplacer sans problème hors secteur; d'où le rajout d'une batterie à la structure. Afin qu'il soit à un bon rendement de capacité, il faut pouvoir lui fournir 12 Volts. La batterie doit aussi avoir un poids raisonnable pour respecter le poids repère évalué. Ainsi, la batterie qui correspond le mieux aux attentes est la suivante.



FIGURE 1.5 – Batterie (Illustration par AJC®Battery)

Avec son volume de  $217 cm^3$  et son poids de  $0.6 kg$ , c'est le parfait composant pour une voiture miniature avec peu de charge à transporter.

### 1.2.2 Le modèle

On peut observer immédiatement que les composants nécessaires ne peuvent pas rentrer dans la version actuelle de la voiture. Il faut donc maintenant adapter la voiture. Dans un premier temps, il faut permettre au moteur de s'adapter aux engrenages Lego. Pour cela, l'idée est d'utiliser une pièce qui s'enclenche avec le moteur et peut accueillir un type d'engrenage Lego. Ensuite, pour être sûr de la solidité du montage, il suffit de figer le tout avec une colle forte qui assure cette rigidité nécessaire. Pour ce faire, l'idée est de fixer ensemble tous les éléments via un montage Lego pour être certain que l'ensemble ne bougera pas pendant la durée de séchage.



FIGURE 1.6 – Structure de collage du moteur

Ainsi, l'idée première est d'utiliser la structure existante pour la rotation des roues de la voiture. En effet, le modèle Lego présente une construction avantageuse pour ce genre de manipulations.

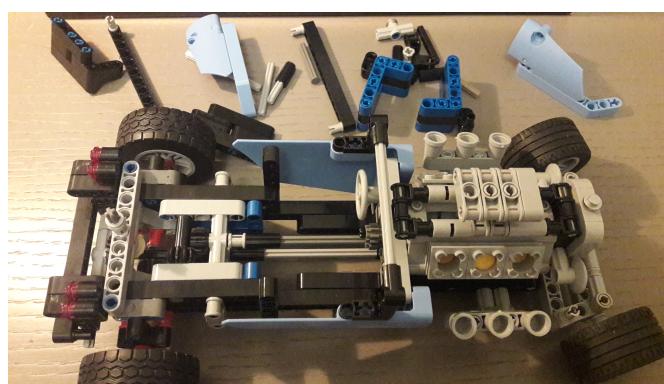


FIGURE 1.7 – Stade initial du modèle Lego

Il suffit en effet de relier le moteur aux engrenages noirs pour motoriser les roues. Cependant, après quelques tests peu concluant, cette architecture ne pouvait pas convenir. En effet, la force délivrée par le moteur était trop importante pour permettre au système Lego de suivre le rythme. Cela est dû au fait que les pièces

n'avaient pas un contact optimal pour bien transmettre la rotation jusqu'aux roues. Le fait que les engrenages ne soient qu'à moitié en lien faisait sauter les liaisons donc rendait incapable la motorisation. De plus, la place n'est pas suffisante pour accueillir les autres composants.

Il faut donc revoir toute l'architecture relative aux roues arrières pour permettre une transmission correcte entre la sortie du moteur et les roues. Le premier point est de relier les deux roues arrière pour s'assurer que les deux roulent à la même vitesse sans problème ce qui n'était pas le cas dans le modèle de base en figure 7. Le principal problème de cette architecture est le contact entre les deux engrenages qui ne se fait qu'à moitié. La nouvelle architecture doit contourner ce problème et permettre un contact optimal entre tous les engrenages Lego depuis le moteur jusqu'aux roues.

De plus, il faut compter le fait que le moteur tourne trop vite pour les besoins de la voiture. Ainsi, on utilise le rapport de réduction des engrenages Lego avec les roues dentées suivantes.

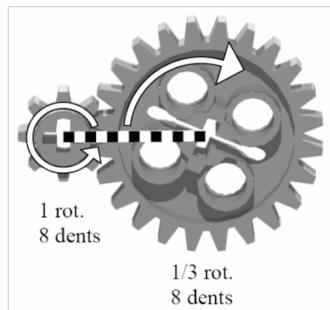


FIGURE 1.8 – Engrenages Lego (Illustration par La Mounière)

Le rapport de réduction entre ces deux roues dentées est de 3 :1. Le reste du montage conserve la vitesse transmises ensuite par la grande roue dentée représentée. On obtient donc une rotation 162 tours par minute. Pour atteindre les 230 tours par minute, on utilise les autres types de roues dentées avec un plus faible rapport de réduction.



FIGURE 1.9 – Autre type d'engrenages Lego (Illustration par eBay)

Ainsi, en utilisant ces propriétés relatives aux engrenages Lego, on peut obtenir une vitesse de rotation des roues proche de la valeur recherché. On peut donc observer un aperçu de l'architecture qui est utilisé pour l'arrière de la voiture. Le moteur se trouve posé sur une surface plane qui permet de consolider les liens entre les engrenages. La structure est solidement fixée pour éviter les problèmes rencontrés précédemment.

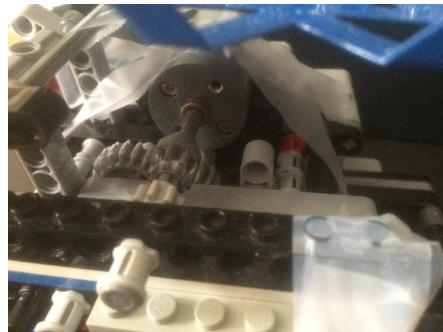


FIGURE 1.10 – Structure pour la propulsion

Après avoir installé le moteur, il s'agit de faire de même pour la batterie qui va l'alimenter. Du fait de ses dimensions, il est nécessaire de rallonger la voiture en son centre en écartant les deux parties avant et arrière ce qui donne le résultat représentée sur la figure suivante.



FIGURE 1.11 – Redimensionnement de la base

Une fois la voiture suffisamment longue pour pouvoir porter la batterie, il faut rajouter une structure de support qui doit être fixée sur la base roulante. Pour construire cette superstructure, on se sert de pièces de métal issues d'un modèle de voiture en Mécano. Ces pièces fournissent une base solide et facilement modulable grâce au système Mécano. La structure représentée en figure 12 est donc un support idéal pour la batterie et de surcroît pour le reste des composants utiles au projet. De plus, pour s'assurer d'un contact optimal des roues sur le sol, on place cette structure de manière à ce que son poids soit supporté par les quatre roues.

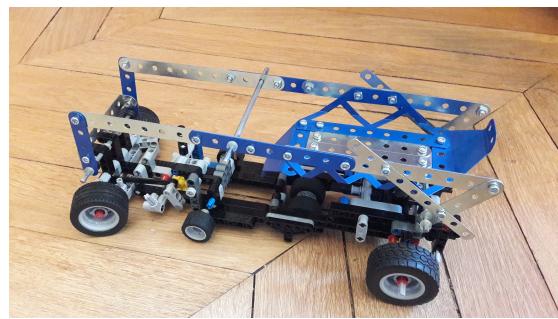


FIGURE 1.12 – Support métallique

Le modèle enfin obtenu donne un bon aperçu du rendu final du projet. Ces modifications sont nécessaire pour envisager le bon fonctionnement de la voiture. Une fois les tests concluant, il est possible de s'intéresser à la deuxième partie de la construction de la voiture.

## 1.3 La direction

### 1.3.1 Le moteur

C'est ici que la liaison avant prend tout son intérêt. En effet, elle permet la rotation des deux roues en agissant uniquement sur une liaison. Cela permet l'utilisation d'un seul servo-moteur pour se charger de diriger la voiture. De plus, cela représente un avantage pour le programme de conduite autonome qui aura moins de données à prendre en compte.



FIGURE 1.13 – Servo-moteur (Illustration par RobotShop)

A la différence de la propulsion, le choix du servo-moteur se fait par l'intermédiaire de tests avec un premier moteur. Ainsi, en effectuant des essais pour vérifier le bon fonctionnement de la liaison avant dans la pratique, le couple du servo-moteur utilisé s'est avéré insuffisant pour une valeur de  $0,2\text{Nm}$ . En effet, le moteur n'atteignait plus précisément les positions demandées à force d'utilisation. L'idée est donc de choisir un moteur avec un couple plus puissant pour pouvoir assurer un fonctionnement optimal. On prend donc un moteur d'un couple de  $0,3\text{Nm}$ .

L'alimentation de ce moteur se fait au travers d'une carte Arduino qui permet aussi d'envoyer la commande de la rotation dans un sens donné. Ce système sera détaillé dans la partie fonctionnement pour expliquer toute la démarche permettant d'effectuer de manière idéal la création des données nécessaires pour la partie informatique.

### 1.3.2 Le modèle

De même que pour la propulsion, la structure de base ne convenait pas pour placer correctement ce servo-moteur : la direction était contrôlée par un mécan-

sisme se situant à l'arrière de la voiture. Ainsi, dès les premières modifications apportées à la voiture, ce mécanisme a été enlevé pour pouvoir uniquement se concentrer sur l'avant de la voiture. Cette modification permet de consolider en séparant à l'avant la direction et à l'arrière la propulsion.

Enfin, pour le relier à la voiture, il s'agit donc de reprendre l'idée développée pour la propulsion. D'une part, on colle une pièce Lego sur l'extrémité du servo-moteur, et d'autre part on aligne sur le même plan tous les engrenages pour transmettre la rotation depuis le moteur jusqu'aux roues.



FIGURE 1.14 – Structure pour le servo-moteur

On obtient donc une structure solide qui permet de transmettre au mieux les efforts du moteur lui même solidement attaché sur la voiture.

## 1.4 Le fonctionnement

Après avoir motorisé la voiture, il faut faire en sorte de pouvoir la commander. Pour cela on utilise les cartes Arduino avec commande à distance qui va permettre de gérer plus facilement les données utilisées pour la programmation du réseau de neurones.

### 1.4.1 La commande des moteurs

L'objectif est donc de contrôler les moteurs pour pouvoir à la fois orienter la voiture et choisir sa vitesse. Pour cela, on utilise une carte Arduino Uno présentée en figure 15. Cette carte est donc constituée d'un micro-contrôleur ATmega328. Ce micro contrôleur est le coeur de la carte Arduino. Il rassemble l'essentiel d'un ordinateur : un processeur, une mémoire, des unités périphériques et des interfaces d'entrée/sortie. La carte Arduino permet d'accéder plus facilement au micro contrôleur via l'interface USB et les pin digitaux ou analogiques. Ainsi, après avoir écrit le code il suffit de le téléverser via la prise USB et d'utiliser les pins pour le montage. Les pins analogues ont un intérêt pour connaître le voltage relatif à ses bornes ensuite converti en une valeur entre 0 et 1023 pour le codage tandis que les pins digitaux traitent d'une information binaire s'il y a une tension ou non. [2]

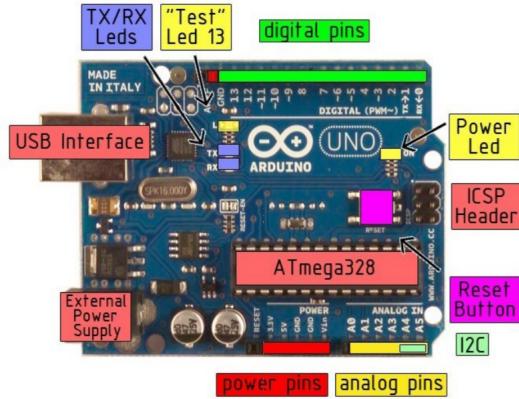


FIGURE 1.15 – Détail carte Arduino [2]

C'est grâce à ce composant qu'on peut accéder au contrôle des deux moteurs. La programmation peut se retrouver en annexe pour plus de détail concernant l'utilisation de cette carte Arduino.

#### 1.4.2 Le module antenne

Après avoir commandé les moteurs, l'objectif est de pouvoir diriger la voiture à distance. En effet, par souci de simplicité, il est plus judicieux d'utiliser le module de commande à distance nRF24L01 spécialement conçu pour Arduino.



FIGURE 1.16 – Antenne (Illustration par letmeknow)

Ce module est donc composé d'une antenne radio qui émet et reçoit à la fréquence de 2.4GHz. On la commande donc à l'aide d'une deuxième carte Arduino qui se chargera de récolter les informations obtenues par les joysticks puis les transmettra à l'antenne émettrice. L'antenne réceptrice située sur la carte Arduino de la voiture va récupérer ces infos pour laisser ensuite cette deuxième carte se charger des commandes à délivrer aux deux moteurs. Pour la programmation, il suffit juste de préciser quelle antenne se charge d'émettre ou de recevoir puisque il n'y a communication que dans un seul sens. De même, il est nécessaire de préciser un "baudrate" qui est tout simplement la vitesse de transmission des informations par l'antenne propre à la carte Arduino.

Une fois le module antenne en place, on dispose d'une première Arduino qui va récupérer les commandes des joysticks jouant le rôle de manette. Elle envoie en-

suite ces informations à la deuxième Arduino sur la voiture via le module antenne pour commander les moteurs. On peut avoir en figure un schéma du montage pour cette première carte.

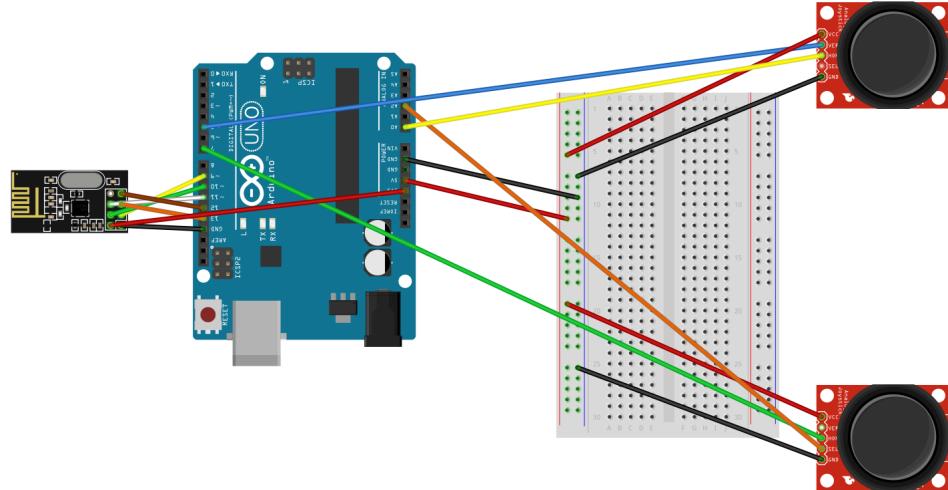


FIGURE 1.17 – Montage pour les joysticks

Pour la seconde carte, il s'agit de la relier aux deux différents moteurs. Il faut donc faire attention au voltage de chaque composant pour ne pas en abîmer un. Le seul problème provient du moteur de propulsion alimenté en 12V tandis que l'Arduino est alimenté en 5V. On utilise donc un module relais particulier pour éviter de suralimenter la carte. On a donc le montage suivant.

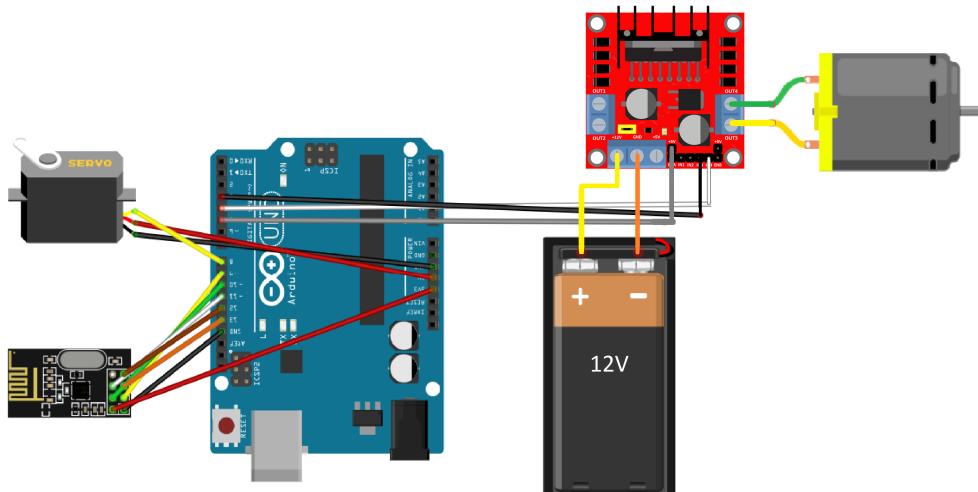


FIGURE 1.18 – Montage pour le moteur propulsion

## 2. Données d'entraînement

### 2.1 L'intérêt du Raspberry

Afin de faire fonctionner un programme d'intelligence artificiel, il est nécessaire de disposer d'un ordinateur. En effet, toute la programmation relative au caractère autonome de la voiture est implémentée en langage Python. De plus, il faut une caméra pour permettre tout simplement à la voiture de repérer ce qu'il y a devant elle. C'est là tout l'intérêt du Raspberry.

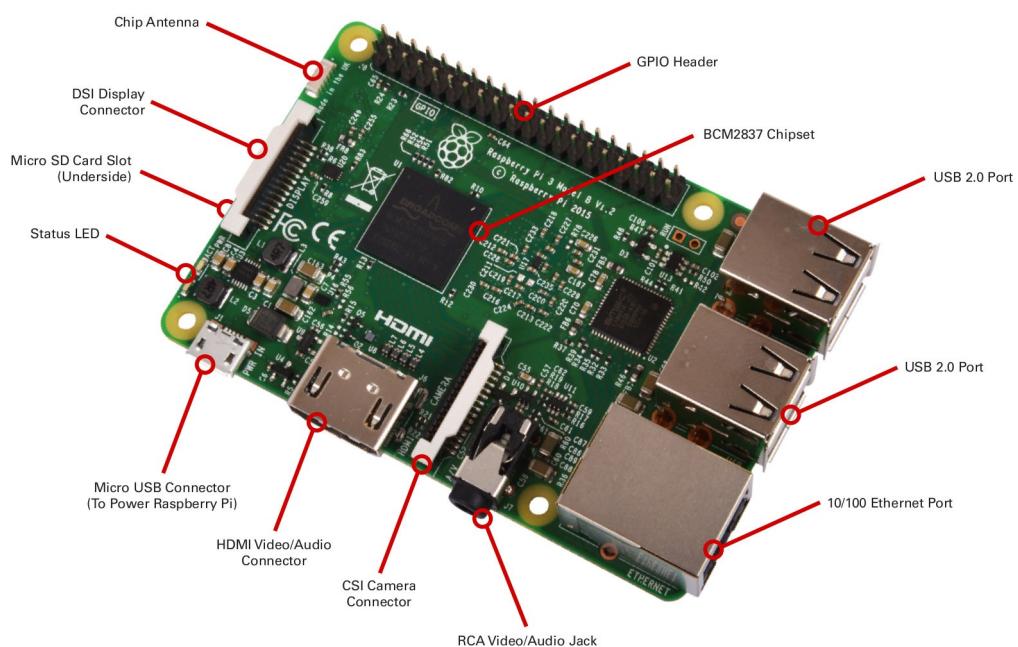


FIGURE 2.19 – Le Raspberry (Illustration par CNXSOFT)

Ce composant possède tous les intérêts d'un ordinateur mais en miniature. Il est donc beaucoup plus facile de pouvoir l'intégrer dans la structure préexistante de la voiture. Il possède aussi différentes extensions qui se connectent via ses ports. C'est de cette manière qu'on peut y rajouter une caméra.



FIGURE 2.20 – Caméra Raspberry (Illustration par Raspberry)

Après avoir installé la caméra et le nécessaire pour pouvoir utiliser Python sur le Raspberry, il s'agit de connecter le Raspberry à la carte Arduino de commande des moteurs. C'est un point crucial dans cette conduite autonome. En effet, à l'aide de la caméra, le programme d'intelligence artificielle aura pour objectif de prédire la commande à imposer aux moteurs en fonction de la situation qui se présente devant la voiture, puis, via cette connexion, cette commande sera transmise à la carte Arduino qui pourra modifier ou non l'état des moteurs.

## 2.2 Le lien Arduino-Raspeberry

Grâce à l'installation de Python sur le Raspberry, il est possible d'accéder à une bibliothèque particulière appelée “pyserial”. Cette bibliothèque de Python permet comme son nom l'indique de traiter des données en “serial” qui signifie série. Cela fait référence à un type de transmission appelé “transmission série”. Ainsi, dans ce type de transmission, les données sont envoyées “en série”, les unes après les autres sur une seule voie. Cette bibliothèque prend donc tout son intérêt dans le lien entre la carte Arduino et le Raspberry. En effet, on relie dans un premier temps les deux composants via leur port USB respectif.

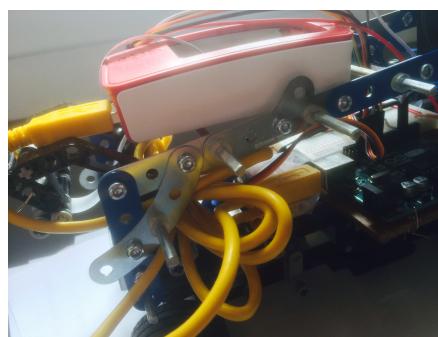


FIGURE 2.21 – Lien physique Arduino-Raspberry

Ensuite, c'est la bibliothèque pyserial qui permet la communication de données entre Arduino et Raspberry. Elle a un grand intérêt car elle peut recevoir des données en serial ce qui sera utile pour la récupération des données mais aussi en

transmettre ce qui permettra de communiquer les commandes à donner aux moteurs après la prédiction du programme d'intelligence artificielle. De même que pour la transmission de données entre les deux cartes Arduinos pour la commande à distance, il faut préciser le même "baudrate" défini par Arduino pour lire et transmettre correctement les données.

## 2.3 La récupération des données

La récupération des données se fait tout naturellement après avoir effectué toutes les manipulations précédentes. Ainsi, l'objectif de ces données est de servir de base pour toute la programmation de la partie intelligence artificielle. Il faut donc récupérer les images issues de la caméra d'une part, et les stocker en fonction de l'état des deux moteurs. Ainsi, au moment de stocker la figure 22, la voiture fait face à une ligne droite. Le moteur de la direction est donc dans la position "centre", et celui de la propulsion est à sa valeur maximale. On enregistre alors cette image dans un dossier "centre". Il en va de même pour les directions "droite" et "gauche" qu'on enregistre respectivement dans des dossiers du même nom.

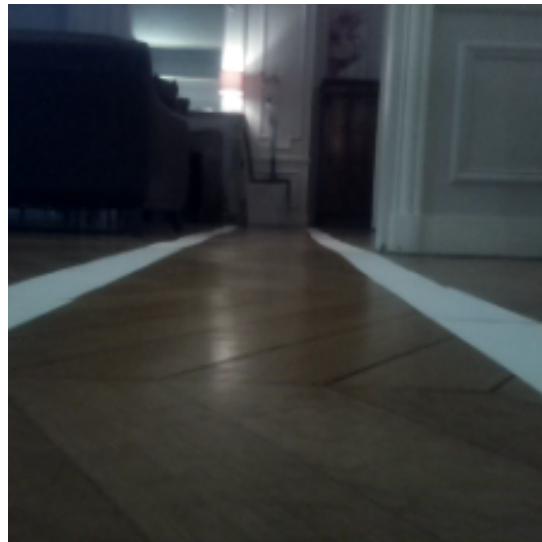


FIGURE 2.22 – Image traitée de la caméra

Les images sont modifiées pour bien avoir une taille de  $224 \times 224$  afin d'être conforme au programme d'intelligence artificielle qui sera précisé par la suite.

L'objectif est donc d'utiliser la manette élaborée précédemment pour commander à distance la voiture pour récupérer un grand nombre de photos de ce genre toutes classées dans leur fichier correspondant. Ces données sont la base pour permettre au programme d'intelligence artificielle de conduire la voiture face à une route similaire. Une fois un nombre suffisant de données récupérées, la carte Arduino de commande à distance n'a plus d'intérêt et seule subsiste la carte de commande des moteurs maintenant reliée au Raspberry. Il s'agit donc de mettre en place ce programme qui a pour but de rendre la voiture autonome dans sa conduite.

# 3. Intelligence Artificielle

## 3.1 Généralités

Cette partie s'intéresse à tout le processus autour du programme qui sera utilisé à la fin par la voiture lors de son déplacement. Depuis l'élaboration du programme à son entraînement à partir des données récoltées précédemment, il s'agit de mettre en place tous les principes relatifs à l'intelligence artificielle et à son utilisation pour le problème posé.

L'intelligence artificielle est donc cette volonté de mettre en oeuvre un programme qui a pour but de transcrire des comportements humains. Cette notion peut s'étendre en fonction de son utilisation et des capacités d'adaptation du programme face à différents problèmes. Ainsi, on distingue trois niveaux qui caractérise le degré d'intervention du programmeur sur les paramètres qui constituent cette intelligence artificielle.

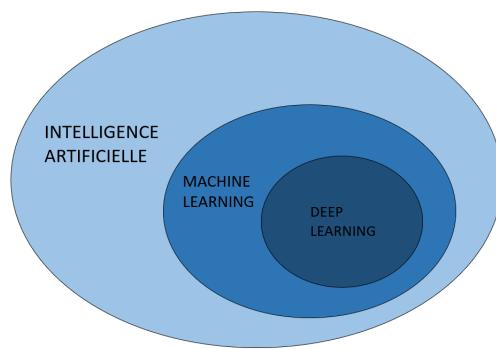


FIGURE 3.23 – Degré de complexité de l'IA

INTELLIGENCE ARTIFICIELLE CLASSIQUE lorsque le programmeur donne tous les éléments pour permettre la réponse attendu du programme comme quand on fait face à un ordinateur dans un jeu simple. Dans ce cas, le programmeur donne toutes les possibilités et règles au programme qui n'a plus qu'à choisir en fonction de chaque situation bien définie en amont.

APPRENTISSAGE AUTOMATIQUE ou “Machine Learning” qui est le type d'intelligence artificielle qui caractérise le mieux la programmation de cette voiture autonome miniature. Cela se fait donc en deux temps : le programme en lui même puis son entraînement. C'est cette phase d'entraînement qui est décisive et fait la

différence face à la programmation décrite précédemment. Le programmeur crée un ensemble de données avec des caractéristiques claires qui seront des exemples sur lesquels le programme va se baser pour ensuite tenter de donner une réponse adéquate face à des situations totalement nouvelles. Au travers de différentes méthodes qui seront explicitées par la suite, le programme va trouver par lui-même la manière de répondre face au problème qui lui sera présenté.

**APPRENTISSAGE PROFOND** ou “Deep Learning” pousse l’apprentissage automatique plus loin encore, c’est l’étude des réseaux de neurones. L’objectif reste le même que précédent : la capacité de pouvoir analyser un ensemble de données utilisées en entraînement pour prédire un nouveau résultat face à une situation nouvelle. Cependant, l’intérêt réside dans la nature des données de l’entraînement : en apprentissage automatique, les données sont précédemment traitées pour que le programme puisse plus facilement établir les liens entre entrées et sorties tandis qu’en apprentissage profond, le fait de mettre en place un réseau plus profond permet au programme de suffisamment analyser les données pour en tirer de lui-même les éléments nécessaires aux prédictions futures.

On se place donc dans le cadre de l’apprentissage automatique pour établir le programme qui sera ensuite utilisé pour le projet. L’objectif est donc d’élaborer un algorithme capable de commander les moteurs de la voiture en fonction du trajet dessiné par des lignes qui indiquent la route. Pour obtenir les résultats escomptés, il faut tout d’abord élaborer le programme qui aura pour rôle de faire ces liens entre l’image de la route et les consignes à envoyer aux moteurs. Ensuite, le but est d’inculquer au réseau de neurones la réponse à donner en fonction d’un certain nombre de situations de références. Cela constitue toute la partie entraînement du réseau de neurones. C’est grâce à cela qu’il va créer le lien cohérent entre la situation en face de la voiture et ce qui est recommandé par le code de la route.

Un neurone autrement appelé unité logistique est tout simplement un ensemble d’opérations élémentaires. Il est ensuite possible d’associer des neurones en parallèle pour obtenir ce qu’on appelle une couche. Les couches se succèdent enfin pour donner un réseau de neurones. Les liens entre les unités logistiques de chaque couche peuvent aussi varier en fonction du réseau de neurones. On peut donc représenter de manière schématique l’aspect d’un réseau de neurones.

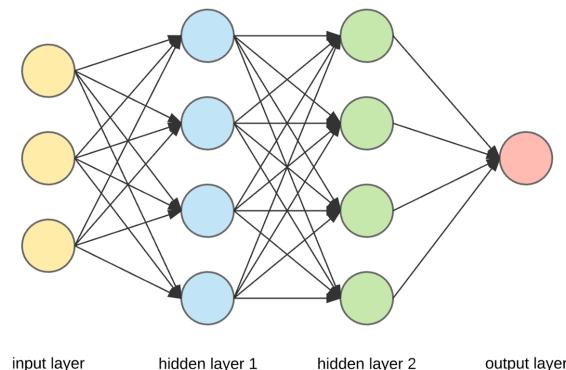


FIGURE 3.24 – Réseau de neurones artificiel [3]

## 3.2 Le principe des réseaux de neurones

Dans un premier temps, l'objectif est de s'intéresser aux réseaux de neurones dans leur globalité pour spécifier ensuite les concepts qui seront prépondérants dans le programme nécessaire à la voiture autonome. Ainsi, il est possible de définir un réseau de neurones comme étant un modèle de programmation qui est inspiré du modèle et du fonctionnement simplifiés des neurones du cerveau humain. Son objectif par la suite est essentiellement d'apprendre à accomplir une certaine tâche à partir de l'analyse de données. On s'intéresse ici à la base de ces concepts au travers d'exemples simples de neurones.

### 3.2.1 La régression linéaire

On prend un réseau de neurones réduit à un seul neurone. Il est totalement défini par la représentation suivante. C'est donc une simple fonction affine. La programmation qui suit consiste en l'entraînement de ce neurone pour que la valeur de  $a$  et de  $b$  soit optimale pour généraliser l'ensemble de données de départ. Pour simplifier, on pose  $b = 0$  dans le cas suivant.

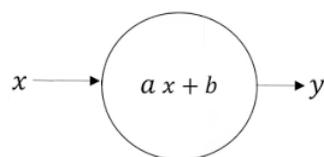
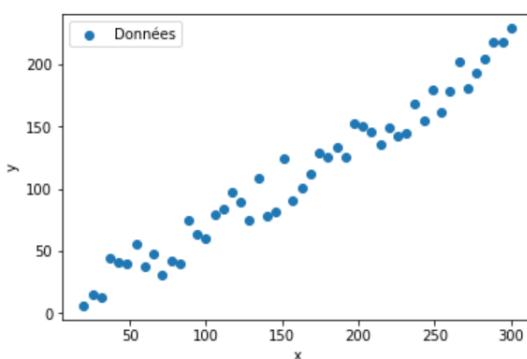


FIGURE 3.25 – Neurone

On suppose qu'on a le graphique suivant qui lie deux variables quelconques  $x$  et  $y$ . Le but est donc d'entraîner le neurone pour qu'il trouve la meilleure relation linéaire entre ces deux variables. Une fois entraîné, il sera capable de faire des prédictions à partir d'une donnée nouvelle avec une certaine précision. La méthode d'entraînement découle directement de la régression linéaire.



Pour s'assurer de la validité du résultat, on fixe le coefficient de la droite linéaire à 0,7. L'objectif est donc de retrouver ce résultat après entraînement du

neurone. On implémente dans Python pour donner en sortie la valeur du coefficient obtenu ainsi que l'évolution de l'erreur dans le calcul des prédictions sur les données de départ.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Création des données
5 pente = 0.7
6 ecart_bruit = 20
7 min_x = 20
8 max_x = 300
9 num_point = 50
10
11 # On génère des données d'entrées x
12 x = np.linspace(min_x, max_x, num = num_point)
13 # On génère une sortie y proportionnelle à chaque entrée
14 y = x*pente + np.random.randint(-ecart_bruit, ecart_bruit + 1, size = x.shape)
15
16 # Régression Linéaire
17 lr = 0.00003
18 iterations = 20
19 m = int(x.shape[0])
20 # On initialise aléatoirement la pente du neurone
21 a = 0
22
23 history = np.zeros(iterations)
24
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
32         # Calcul de l'erreur
33         Erreur += (p - y[i])**2
34         # Dérivée de l'erreur par rapport à p
35         dp = 2*(p - y[i])
36         # Dérivée de l'erreur par rapport à a
37         da += dp*x[i] # chain rule
38
39     da = da/m
40
41     a = a - lr*da
42
43     Erreur = Erreur/(2*m)
44
45     history[iteration] = Erreur
46
47 print("Erreur :", history, "a =", a)
48 pred = a*x

```

Tout d'abord on pose une valeur aléatoire de  $a$ , cela permet simplement de donner un point de départ pour le programme. On définit aussi le nombre d'itérations qui est le nombre de fois que le programme va analyser les données et donc rectifier  $a$ .

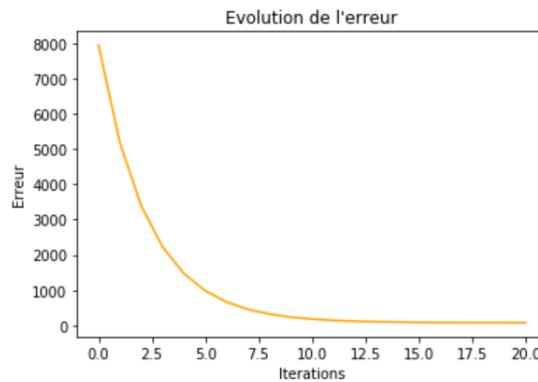
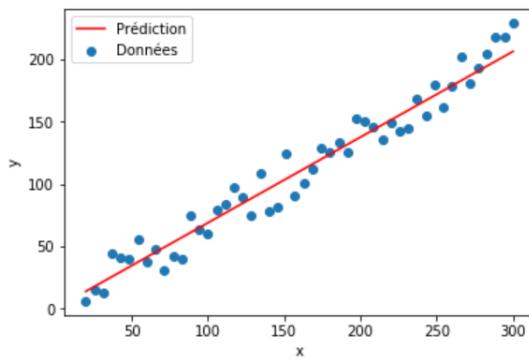
On se place donc dans une boucle d'itérations. Pour une boucle on pose donc la fonction d'*Erreur* à la valeur 0 tout comme la dérivée par rapport à  $a$  dans la relation  $p = ax$ . C'est la condition initiale. On rentre ensuite dans une boucle qui se permet de passer en revue l'ensemble des données de départ. On a donc la relation recherchée  $p = ax$  avec  $x$  de l'ensemble de données et  $p$  qui résulte de la relation linéaire recherchée en  $x$  et  $y$ . On définit ensuite la fonction *Erreur* qui est le cœur de la méthode des moindres carrés. Ainsi, après avoir itéré sur les  $m$  données de départ on obtient cette fonction caractéristique :

$$Erreur = \sum_{i=1}^m (y_i - p_i)^2$$

La méthode correspond donc à minimiser cette fonction. Pour cela, on définit la variable  $dp$  qui correspond à  $\frac{dE}{dp}$  et ensuite  $da$  incrémenté à chaque fois. On obtient donc une première valeur de l'Erreur et de  $da$  qui fait ensuite varier  $a$ . Ainsi, à chaque itération l'*Erreur* va donc évoluer selon le graphique représenté après qui montre cette décroissance attendue typique de la méthode des moindres carrés. En effet, les points  $p$  définis par  $ax$  n'auront plus la même valeur et vont même se rapprocher de la valeur idéale qui est  $y$  ce qui est à l'origine de cette diminution de l'erreur. De même pour le  $dp$  qui va influencer le  $da$  pour se rapprocher de la droite linéaire qui s'accorde le mieux avec les données.

A la fin, on a aussi la commande  $da = \frac{da}{m}$ . Cela permet tout simplement de ramener à l'échelle d'un seul exemple le résultat obtenu pour  $m$  exemples. C'est ce qu'on appelle la normalisation. Il en va de même pour l'*Erreur* pour qu'elle aie une valeur cohérente avec l'échelle de valeur des données.

On obtient finalement au bout de 20 itérations une *Erreur* qui se trouve beaucoup plus faible qu'à l'origine et surtout un coefficient  $a$  qui vaut 0,69 très proche de la valeur attendue. A partir de ce coefficient  $a$ , il est donc possible de faire une prédition plutôt fidèle avec de nouvelles données du simple fait de cette relation linéaire.



On peut en effet se demander pourquoi cette dérivée partielle intervient et en quoi son action sur la variable permet d'aboutir à un résultat correct. Cela sera abordé par la suite quand il s'agira d'expliquer l'entraînement du réseau de neurones utilisé pour la voiture. Cette méthode porte le nom de descente de gradient

qui comme son nom l'indique utilise les propriétés du gradient pour obtenir le résultat attendu ici. On a ici un cas particulier à une dimension de la descente de gradient.

On observe aussi une variable assez inattendu notée  $lr$ . Cette variable est importante pour les réseaux de neurones et est appelée “learning rate” ou taux d’apprentissage. Dans la plupart des cas, elle s’obtient de façon empirique jusqu’à ce que les résultats soit satisfaisant en se plaçant initialement à 0,001. Elle est uniquement utilisée lors de l’incrémentation de  $a$ . En effet, elle caractérise la vitesse d’évolution de  $a$ . Ainsi, si ce taux est trop important, l’évolution de  $a$  sera trop importante aussi et donc il sera impossible de tomber sur la valeur recherchée, on parle d’“overshooting”. A l’inverse s’il est très faible, il faudra beaucoup d’itérations pour arriver à une bonne valeur de  $a$  mais il est plus certain d’obtenir des résultats satisfaisants.

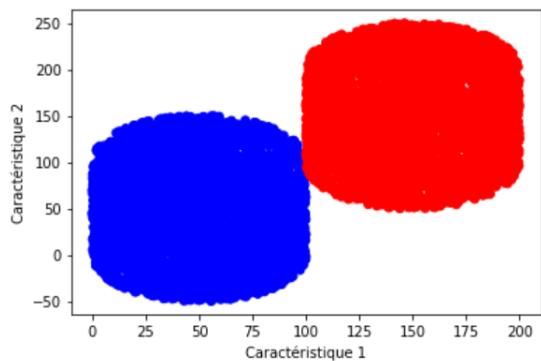
On a ici un cas pratique d’utilisation des réseaux de neurones et comment ils fonctionnement sur un exemple simple. Ainsi, le neurone se résume en une fonction qui après entraînement va faire des prédictions en renvoyant simplement la valeur par la relation linéaire.

### 3.2.2 La régression logistique

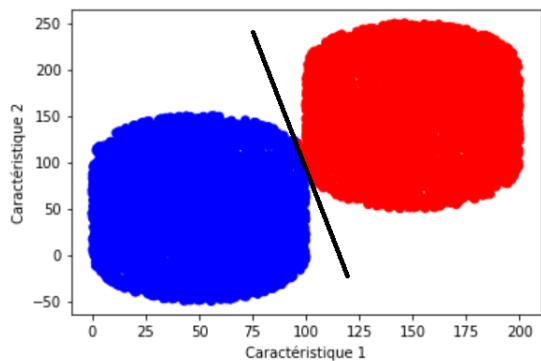
Ainsi, après un premier aperçu du principe de l’intelligence artificielle, on peut prendre un autre type de problème basique. L’objectif ici est toujours de relier un entrée à une sortie à la différence que cette sortie est binaire. On se place donc dans une situation où le réseau de neurones va renvoyer une sortie du type succès/échec sous forme d’une valeur continue entre 0 et 1. Cela peut s’interpréter comme la probabilité de succès de l’entrée.

On se place donc dans un nouveau contexte : on dispose de deux groupes distincts représentés en figure suivante. Ainsi, les groupes sont classés selon deux caractéristiques. Le principe de cette régression logistique est de décrire une courbe qui permet de séparer les deux groupes. Ainsi, avec la fonction correspondante il sera possible de déterminer à quel groupe appartient un nouvel élément inconnu. Il est aussi intéressant d’observer la manière dont ce classement sera effectué. En effet, sur le graphique des données, on remarque que les deux groupes se chevauchent ce qui va nécessairement avoir des conséquences sur la classification. De même que pour la régression linéaire, la programmation effectuée constitue tout l’entraînement du neurone.

Les données d’entrées ne sont donc plus composées d’une simple abscisse mais bien d’un couple  $(x,y)$  issu des caractéristiques bleue et rouge. La programmation à mettre en place reste similaire à celle de la régression linéaire à la différence qu’il s’agit de déterminer la fonction à deux variables qui classe le mieux les données de base.



A première vue, la fonction idéale serait une droite linéaire qui décrirait la courbe représentée sur la figure suivante. En effet, on a une séparation claire des deux groupes et donc des deux zones qui correspondent à chaque groupe. L'objectif est donc de retomber sur une droite similaire pour illustrer l'efficacité de cette méthode.



On obtient donc cette programmation pour la régression logistique. Ainsi, comme précédemment, on initialise les deux poids de manière aléatoire, chaque poids va jouer sur une variable du couple  $(x, y)$ .

```

1 import numpy as np
2 from random import random, seed
3
4 def my_rand():
5     return 1 if random() < 0.5 else -1
6
7 def my_rand_tab(x):
8     for i in range(x.shape[0]):
9         x[i] = my_rand()*x[i]
10    return x
11
12 def my_norm (x):
13     mean = np.mean(x)
14     std = np.max(x) - np.min(x)
15     return ((x - mean) / std), mean, std
16
17 # Création des données
18
19 centre_1 = np.array([50, 50]) # centre du cercle
20 r_1 = 50 # rayon
21 bruit_1 = 50 # bruit
22 nb_donnees_1 = 4000 # nombre de points
23 centre_2 = np.array([150, 150]) # idem ... (pour un 2e cercle)
24 r_2 = 50
25 bruit_2 = 50
26 nb_donnees_2 = 4000
27
28 # Abscisses
29 x1 = np.linspace(centre_1[0] - r_1, centre_1[0] + r_1, nb_donnees_1)
30 x2 = np.linspace(centre_2[0] - r_2, centre_2[0] + r_2, nb_donnees_2)
31
32 # Ordonnées
33 y1 = my_rand_tab(np.sqrt((r_1**2) - (x1 - centre_1[0])**2)) + centre_1[1]
34     + np.random.randint(-bruit_1, bruit_1 + 1, size = x1.shape)
35 y2 = my_rand_tab(np.sqrt((r_2**2) - (x2 - centre_2[0])**2)) + centre_2[1]
36     + np.random.randint(-bruit_2, bruit_2 + 1, size = x2.shape)
37
38 # On rassemble le set 1 avec le set 2
39 x = np.concatenate((x1, x2))
40 y = np.concatenate((y1, y2))
41
42 # (x1, y1) sont labélisées 0
43 t1 = np.zeros(x1.shape)
44 # (x2, y2) sont labélisées 1
45 t2 = np.ones(x2.shape)
46 target = np.concatenate((t1, t2))
47 print(target)
48
49 # Normalisation
50 x, x_mean, x_std = my_norm(x)
51 y, y_mean, y_std = my_norm(y)
52
53 # Paramètres
54 lr0 = 0.00003
55 iterations = 40
56
57 # Nombre d'exemples
58 m = int(x.shape[0])
59
60 # On initialise aléatoirement nos poids
61 wx1 = 0.001*random()
62 wy1 = 0.001*random()
63 b = 0.001*random()
64
65 print("wx1 =", wx1, "wy1 =", wy1, "b =", b)
66
67 # Pour voir l'évolution de l'erreur
68 history = np.zeros(iterations)
69
70 # Notre fonction d'activation
71 def sigmoid(x):
72     return 1 / (1 + np.exp(-x))

```

```

73
74 # Entrainement
75 for iteration in range(iterations):
76     if iteration % 5 == 0 :
77         print("iteration :", iteration)
78     # Remise à zéro des dérivées partielles
79     Erreur = 0
80     dwx1 = 0
81     dwy1 = 0
82     db = 0
83     # On réduit le Learning rate petit à petit pour faciliter la convergence
84     lr = lr*(1/(1 + 0.5*iteration))
85     for i in range(m):
86         # Calcul de la prédiction
87         a = wx1*x[i] + wy1*y[i] + b
88         p = sigmoid(a)
89         # Calcul de l'erreur
90         Erreur += -(target[i]*np.log(0.005+p) + (1-target[i])*np.log(1.005-p))
91         # Dérivée de l'erreur par rapport à a
92         da = p*(1 - p)
93         # Dérivée de l'erreur par rapport à w1, w2 et b
94         dwx1 += da*x[i]
95         dwy1 += da*y[i]
96         db += da
97
98     # On se ramène à l'échelle d'un exemple
99     dwx1 = dwx1/m
100    dwy1 = dwy1/m
101    db = db/m
102
103    # Mise à jour des poids
104    wx1 = wx1 - lr*dwx1
105    wy1 = wy1 - lr*dwy1
106    b = b - lr*db
107
108    Erreur = Erreur/(2*m)
109
110    # On sauvegarde l'état actuel de l'erreur
111    history[iteration] = Erreur
112
113 print("Erreur : ", history)
114 print(" ")
115 print("wx1 =", wx1, "wy1 =", wy1, "b =", b)
116
117 # La fonction sigmoid va donner une valeur continue entre 0 et 1
118 # On veut soit 0 soit 1
119 threshold = 0.5
120
121 # Calcul de la prédiction (sur le training set)
122 pred_temp = sigmoid(wx1*x + wy1*y + b)
123 pred = (pred_temp > threshold).astype(np.int32)
124
125 # Calcul de la précision
126 diff = np.abs(target - pred)
127 acc = (1 - (np.sum(diff)/m))*100
128 print(pred)
129 print(f"""
130 acc : {round(acc, 2)}
132 """
133 ""))

```

On a la présence d'une fonction particulière appelée "sigmoïde" définie en ligne 71 de la manière suivante.

$$\sigma : \left( \begin{array}{ccc} \mathbb{R} & \rightarrow & [0, 1] \\ x & \mapsto & \frac{1}{1+e^{-x}} \end{array} \right)$$

Cette fonction dite "d'activation" représente l'enjeu de cette régression. En effet, comme en témoigne ses variations, c'est elle qui va permettre de classer les données selon chaque groupe en donnant une valeur plus ou moins proche de 0 ou 1 qui est lié à l'un ou l'autre groupe. Cependant, la valeur rendu par cette fonction peut aussi se trouver dans l'intervalle  $[0, 1]$ . C'est donc un nouveau paramètre appelé "threshold" signifiant seuil qui va donner la limite pour décider si cette valeur intermédiaire vaut 0 ou 1. Cette fonction caractérise l'opération effectué par le neurone ici. On parle donc d'unité logistique lorsqu'une unité est munie d'une fonction d'activation en sortie. La courbe représentative permet de mieux comprendre l'intérêt de cette fonction.

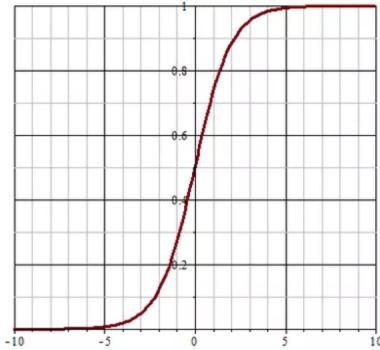


FIGURE 3.26 – Courbe représentative

Ensuite, on effectue de la même manière un certain nombre d’itérations pour affiner le résultat à chaque fois et chaque itération se fait sur tous les exemples de l’ensemble des données de départ.

On pose aussi l'*Erreur*,  $dw_1$ ,  $dw_2$  et  $db$  nulle comme point départ pour ces dérivées partielles. De même, il est judicieux de réduire le taux d’apprentissage pour les mêmes raisons qu’en régression linéaire ; c’est en effet un moyen plus simple de faciliter la convergence sans chercher indéfiniment un taux qui convient.

On calcule ensuite la prédiction  $a$  selon la relation  $a = w_1x + w_2y + b$ . Le  $a$  présent ici joue un rôle différent de celui de la régression linéaire. En effet, il joue le rôle des valeurs prédites auquel les poids  $w_1$  et  $w_2$  doivent permettre d’approcher tout au long des itérations.

On peut aussi remarquer qu’en notant  $X = \begin{pmatrix} x \\ y \end{pmatrix}$  et  $W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$  la relation devient alors :  $a = \langle W | X \rangle + b$

On retrouve donc la fonction prédiction de la régression linéaire avec la notation  $p$  qui devient  $a$  et le produit réel qui est devenu un produit scalaire. Cette formule généralise le principe de régression. C’est de cette manière qu’on peut traiter des données classées avec  $n$  caractéristiques en prenant un vecteur  $X$  de taille  $n$ . On parle alors de "vectorisation", concept qui sera explicité par la suite.

Une fois cette prédiction calculée, on utilise la fonction d’activation  $\sigma$  pour obtenir une probabilité d’être dans l’un ou l’autre groupe. De même, on utilise une fonction d’erreur qui s’avère plus adaptée à ce problème que celle de la méthode des moindres carrés. En effet, à la fin on obtient pour  $p$  dans  $[0, 1]$  la fonction :

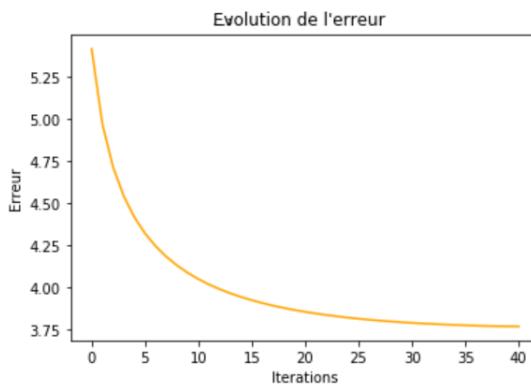
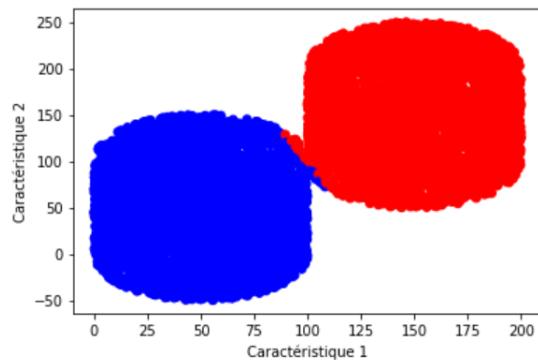
$$\text{Erreur} = - \sum_{i=1}^m y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

On la trouve sous le nom de fonction d’entropie croisée. Elle convient plus à ce genre de problème binaire. Ainsi, lorsque  $y$  vaut 0, pour minimiser l'*Erreur*, il faut que  $p$  soit petit aussi donc proche de 0 et de même avec  $y$  qui vaut 1, il faut  $p$  grand donc proche de 1.

Ensuite, on retrouve le même concept des dérivées partielles qui permettent de faire évoluer les poids au fur et à mesure qu’on analyse l’ensemble de données.

Ainsi, on a de la même manière la descente de gradient qui intervient et s'effectue cette fois-ci en deux dimensions. De même, il s'agit d'effectuer une normalisation de ces variables pour revenir à l'échelle d'un exemple pour enfin incrémenter les poids.

On obtient donc les résultats suivants pour 40 itérations. On observe que l'erreur va en diminuant ce qui est un bon indicateur de la performance du programme. De même, on peut voir la fonction de classification qui résulte de cet algorithme. Ainsi, elle est suffisamment proche du modèle idéal pour être considéré comme correcte. La zone de chevauchement a bien été prise en compte pour évaluer le modèle ce qui oblige le fait de classer des éléments bleus dans le groupe rouge et inversement. Pour éviter cette classification qui peut apparaître imparfaite, il suffit de rajouter plus de poids et d'augmenter le degré de la fonction de classification.



### 3.2.3 Les fonctions d'activation

On en a rencontré un exemple dans la régression logistique et ainsi pu entrevoir leur intérêt pour les réseaux de neurones. Cette fonction a donc pour but de donner la sortie d'un neurone souvent de manière binaire comme en régression logistique. Son terme renvoie à l'étude des réseaux de neurones biologique lorsqu'on parle de "seuil d'activation". Ainsi, cette fonction permet de déterminer si la réponse est 0 ou 1 principalement. Chaque problème peut recourir à une fonction d'activation bien précise qui résulte donc d'un choix du programmeur.

Un autre intérêt de ces fonctions d'activations est d'enlever le caractère linéaire des neurones. En effet, on a vu pour la régression logistique qu'on pouvait ramener cette unité à une forme de relation linéaire  $a = \langle W|X \rangle + b$ . En se rameant à un cas à une dimension, on prend donc un réseau de neurones "profond" car constitué de deux couches mais sans aucune fonction d'activation (fonction identité en sortie). On peut alors observer le résultat suivant.

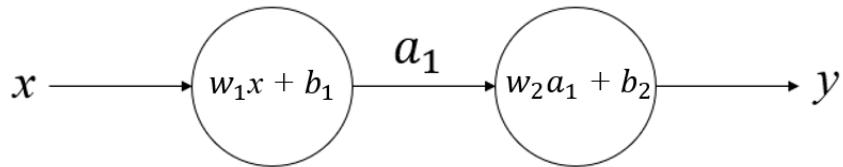


FIGURE 3.27 – Réseau de neurones

On a donc  $a = w_1x + b_1$  et  $y = w_2a_1 + b_2$

D'où  $y = w_2(w_1x + b_1) + b_2$

i.e.  $y = (w_1w_2)x + (w_2b_1 + b_2)$

En notant  $w' = w_1w_2$  et  $b' = w_2b_1 + b_2$  on a finalement  $y = w'x + b'$

Ainsi, cela revient à une seule unité au lieu de deux. On perd donc tout l'intérêt de la profondeur du réseau de neurones. C'est pourquoi cette fonction d'activation a un rôle essentiel dans les réseaux de neurones.

En reprenant la représentation du réseau de neurones de la figure13, on voit donc que pour chaque neurone ou unité logistique d'un même couche est attribué une certaine fonction d'activation qui va dépendre de la tâche de la couche. On a vu que cette fonction peut renvoyer 0 ou 1. Cela permet donc de décider si un neurone est à un rôle dans le réseau de neurones ou non d'où ce parallèle avec la biologie où des neurones sont activés ou non en fonction de la tâche à exécuter.

On peut présenter ici les fonctions d'activations parmi les plus utilisées suivant les différents besoins.

Nom	Graphe	Équation
Identité/Rampe		$f(x) = x$
Marche/Heaviside		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistique (ou marche douce, ou sigmoïde )		$f(x) = \frac{1}{1 + e^{-x}}$
Tangente Hyperbolique (Tanh)		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
Arc Tangente (ArcTan ou Tan <sup>-1</sup> )		$f(x) = \tan^{-1}(x)$
Signe doux		$f(x) = \frac{x}{1 +  x }$
Unité de Rectification Linéaire (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

FIGURE 3.28 – Fonctions d’activation (Illustration par Wikipédia)

### 3.2.4 Les filtres

Pour ce projet, on sait qu’il faut que le réseau de neurones analyse des images issues de la caméra à l’avant de la voiture. Ainsi, il s’agit effectivement d’utiliser les mêmes outils qu’en traitement d’image. Parmi ces outils en existe un qui a un grand intérêt pour les réseaux de neurones : les filtres. L’objectif n’est pas d’expliquer tous les filtres utiles en intelligence artificielle mais ceux qui ont leur intérêt pour notre réseau de neurones.

#### La convolution

Le produit de convolution est un opérateur bilinéaire et un produit commutatif, généralement noté “\*”, qui à deux fonctions  $f$  et  $g$  sur un même domaine infini, fait correspondre une autre fonction  $h$  sur ce domaine, qui en tout point de celui-ci est égale à l’intégrale sur l’entièreté du domaine d’une des deux fonctions autour de ce point, pondérée par l’autre fonction autour de l’origine.

Cette opération s’effectue dans le cas de fonctions continues mais aussi de fonctions discrètes avec la relation suivante :

$$h(i) = (f * g)(i) = \sum_{k=-\infty}^{+\infty} f_{i-k} g_k$$

C'est dans ce cas qu'on se place car avec cette relation il est possible de l'appliquer aux images en les considérant comme des tableaux de valeurs en fonction des pixels qui la constitue.

On se place donc avec une image de dimension 1 qui se représente sous la forme de la fonction  $f$  suivante.

10	50	60	10	20	40	30
----	----	----	----	----	----	----

On prend aussi une deuxième fonction  $g$  définie de la manière suivante.

1/3	1/3	1/3
-----	-----	-----

Alors si on calcule  $h(2)$  on obtient d'après la définition :

$$h(2) = \frac{50 + 60 + 10}{3} = 40$$

On obtient donc la fonction  $h$  définie de la manière suivante.

40	40	30	20	30
----	----	----	----	----

D'après Python, Cours, Volume 2 par J-M. Saint-Jalm.

On peut remarquer que le tableau qui résulte de la convolution a diminué en taille. En effet le tableau d'entrée est de taille  $6 \times 6$ , le filtre de taille  $3 \times 3$  et le tableau résultant de taille  $4 \times 4$ . Plus généralement, cette réduction est régie par la loi suivante.

*Soit une image de taille  $m \times n$*

*Soit un filtre de taille  $f \times g$*

*Alors la convolution des deux donne une image de taille  $(m-f+1) \times (n-g+1)$*

De cela découle le concept des filtres de convolution dans l'analyse d'images pour obtenir différents résultats selon les besoins. Ainsi, le filtre suivant en figure 29 permet de faire ressortir la composante verticale qui est bien délimitée entre la zone des 10 et des 0. De là en ressort une image réduite avec cette zone de 30 qui confirme la détection des contours. Ce filtre a un grand intérêt dans l'analyse d'images, on le retrouve sous le nom de filtre de Prewitt ; on calcule donc le gradient d'intensité lumineuse d'une image pour en faire ressortir les contours.

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline \end{array}$$

FIGURE 3.29 – Produit de convolution d’une image avec un filtre de contour [4]

On peut prendre un exemple plus concret avec une image en nuances de gris d’un temple aztèque pour avoir une unique couche de pixels. De même, on utilise le filtre de Prewitt pour détecter les contours et on peut observer l’efficacité de cette opération. Le filtre met donc en valeur toutes les lignes de contour du temple pour faire ressortir ses formes dans la nouvelle image.

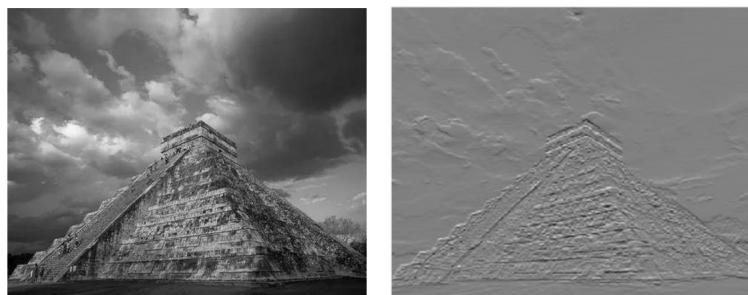


FIGURE 3.30 – Utilisation du filtre sur image

Ce principe est prépondérant en intelligence artificielle notamment dans le domaine de la “vision ordinateur” qui signifie tout simplement donner à l’ordinateur les moyens de “voir” une image par exemple même s’il ne prend en compte que des tableaux de valeurs. Cela signifie que l’ordinateur possède des informations précises sur l’image qu’il peut ensuite utiliser pour d’autres opérations. Dans le cas de la voiture autonome, ce traitement d’images a un grand intérêt car c’est ce genre d’opérations que devra effectuer le programme pour repérer l’état de la route. Ainsi, c’est de cette manière qu’il pourra détecter si la route à suivre est une ligne droite ou bien un virage.

Dans le cas d’utilisation de ces filtres, les images couleurs sont donc composées de trois couches : rouge, verte et bleue. Ainsi, il ne s’agit pas de détecter des caractéristiques sur une surface comme précédemment mais sur un volume.

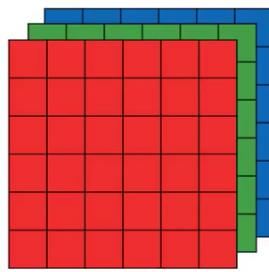


FIGURE 3.31 – Image couleur en terme de tableau [4]

Face à ce genre d'objets, il faut mettre en place de nouveaux filtres pour pouvoir toujours appliquer la convolution sur toute l'image. On a donc des filtres avec volume et on parle de convolution sur volume. Il est évident que pour que la convolution soit bien définie, la profondeur du filtre doit être la même que celle du tableau d'entrée. Ainsi, comme représenté par le jaune sur la figure 32, on applique le même filtre simultanément sur chaque couche. On se ramène donc à un calcul normal de convolution de deux tableau mais fait autant de fois que la profondeur l'indique.

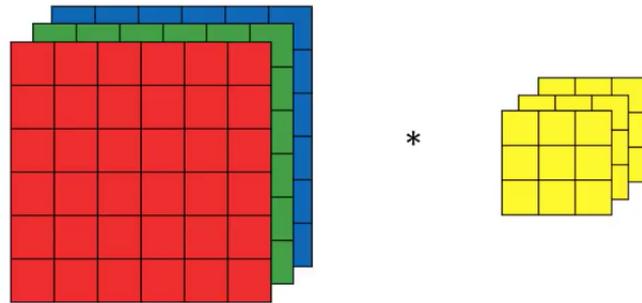


FIGURE 3.32 – Convolution sur volume [4]

Avec cette opération de convolution, on peut aussi faire appel à un paramètre entier naturel appelé “stride” qui signifie l'enjambée. Comme son nom l'indique, ce paramètre modifie la manière dont le filtre va agir sur le tableau de valeur considéré. En effet, plus ce paramètre est grand et plus le filtre va enjamber des cases du tableau pendant la convolution.

On se place donc dans le cas suivant d'un tableau de valeurs de taille  $7 \times 7$  convolué avec un filtre quelconque de taille  $3 \times 3$ . Ainsi, pour le début, il s'agit d'une simple convolution comme vue précédemment puis le paramètre d'enjambement fera effet

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

\*

3	4	4
1	0	2
-1	0	3

FIGURE 3.33 – Convolution avec "stride" [4]

On cherche donc à déterminer le résultat. Ainsi, pour le premier élément du tableau résultat on effectue l'opération :

$$2 \times 3 + 6 \times 1 + 3 \times (-1) + 3 \times 4 + 6 \times 0 + 4 \times 0 + 7 \times 4 + 9 \times 2 + 8 \times 3 = 91$$

2	3	7	4	4	6	2	9		
6	1	6	0	9	2	8	7	4	3
3	-1	4	0	8	3	3	8	9	7
7	8	3	6	6	6	3	4		
4	2	1	8	3	4	6			
3	2	4	1	9	8	3			
0	1	3	9	2	1	4			

\*

3	4	4
1	0	2
-1	0	3

=

q1		

FIGURE 3.34 – Convolution avec "stride" [4]

C'est à ce moment que l'enjambement prend tout son sens. En effet, au lieu de décaler le filtre sur la case suivante, on le place à deux cases.

2	3	7	3	4	4	6	4	2	9
6	6	9	1	8	0	7	2	4	3
3	4	8	-1	3	0	8	3	9	7
7	8	3	6	6	6	3	4		
4	2	1	8	3	4	6			
3	2	4	1	9	8	3			
0	1	3	9	2	1	4			

\*

3	4	4
1	0	2
-1	0	3

FIGURE 3.35 – Convolution avec "stride" [4]

Cette fois-ci, on effectue le même genre de calcul ce qui donne 100. On recommence en se déplaçant de deux cases à chaque fois jusqu'à atteindre la fin du tableau d'entrée. Une fois arrivée en fin de tableau, on effectue le même enjambement des lignes et on recommence.

Sur cet exemple, on observe en figure 34 que la taille du tableau en sortie ne suit plus la loi établie précédemment car  $4 - 3 + 1 = 2 \neq 3$ . C'est ici l'intérêt de

ce paramètre, il permet de modifier selon une nouvelle loi la taille du tableau de sortie.

*Soit une image de taille  $m \times n$*

*Soit un filtre de taille  $f \times g$*

*Alors pour un enjambement  $s$ , la convolution des deux donne une image de taille  $\frac{m-f}{s} \times \frac{n-g}{s}$*

Il est donc nécessaire de s'assurer que la taille du tableau résultat est bien un entier pour que le tableau puisse effectivement exister.

### Les filtres de regroupement

On peut donc décliner les filtres en fonction des besoins de l'analyse des images. Parmi ceux existants, il en est un qui a son intérêt pour la suite : le filtre dit "pooling" qui signifie regroupement. Ainsi, il peut prendre différentes formes selon les besoins en effectuant une moyenne des pixels, ou bien en ne retenant que celui de valeur maximale. Son utilité découle du fait que la convolution avec un filtre réduit la taille de l'image en entrée. En effet, avec ce genre de filtre, on conserve une grande partie de l'information contenue dans l'image. C'est pourquoi appliquer ces filtres revient tout simplement à réduire la taille d'une image tout en gardant ses caractéristiques essentielles.

L'intérêt de ces filtres de regroupement pour l'intelligence artificielle permet d'effectuer un plus grand nombre d'opérations sur cette image réduite pendant le même intervalle de temps que si on gardait la taille d'origine. C'est pour cela qu'ils ont une place particulière dans la réalisation du programme pour la voiture autonome miniature.

On peut prendre le cas du filtre moyennant ou "average pooling" qui a son intérêt pour le programme à établir. On a donc le tableau suivant qu'on délimite selon quatre régions ensuite réparties sur le tableau de sortie. Ainsi, on prend la moyenne de chaque région ce qui donne bien un tableau de plus petite dimension. On peut remarquer l'utilisation du paramètre d'enjambement avec pour valeur 2.

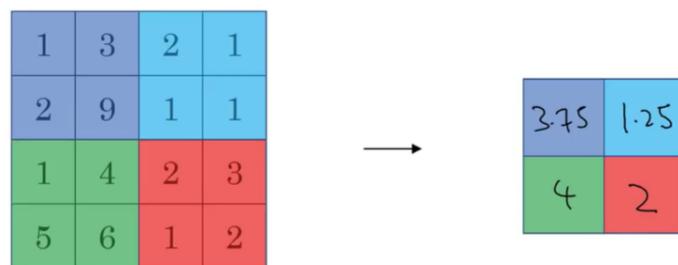


FIGURE 3.36 – Filtre de moyenne avec stride [4]

Il peut être difficile de se rendre véritablement compte de la conservation de l'information sur un tableau de si petite taille. On prend donc l'image précédente dont on fait la convolution avec un filtre moyennant.



FIGURE 3.37 – Filtre de moyenne sur image

On observe donc que ce filtre crée un flou qui se comprend comme une réduction de la quantité de données. Cependant, l'aspect de l'image est toujours présent, on distingue bien le temple aztèque.

Finalement, les réseaux de neurones peuvent se ramener à des fonctions plus ou moins complexes par ce fait de relier des entrées à des sorties via un graphe. Ce graphe est plus ou moins difficile à appréhender en fonction de la complexité du programme. Tout l'intérêt réside en la mise en place de cette fonction en généralisant des exemples de bases pour pouvoir ensuite obtenir de nouveaux résultats.

*Pour plus d'informations sur l'utilisation des filtres en intelligence artificielle et sur d'autres notions non explicitées dans cet écrit, le lecteur est invité à visiter la chaîne YouTube Deeplearning.ai. Il pourra y trouver les cours en libre accès du chercheur en intelligence artificielle Andrew Ng dont cet écrit est largement inspiré.*

### 3.3 Adaptation au problème

Dans cette partie, le but est de mettre en place le réseau de neurones adapté au problème posé. La puissance de calcul ainsi que la mémoire vive du Raspberry étant bien inférieures à celles d'un ordinateur classique, il faut un réseau de neurones relativement peu profond. Pour ce faire, on choisit parmi les architectures existantes, celles qui peuvent s'exécuter au mieux en considérant ces limites physiques.

Le réseau de neurones le plus adapté est le "MobileNet" et plus précisément sa deuxième version. En effet, rien que sa première version est un modèle efficace pour ce genre de problèmes. Ainsi, il a pour objectif d'accélérer les étapes convolutives, en diminuant la complexité du programme tout en essayant conserver au mieux les performances.

Ainsi, "une convolution standard filtre et combine les entrées dans un nouvel ensemble de sortie" [5]. Comme vu précédemment, les convolutions peuvent s'effectuer sur des volumes avec une profondeur quelconque. Pour cela, les filtres en jeu sont donc des filtres de volume avec la même profondeur. Comme illustré précédemment en figure 32 avec une profondeur de trois couches, on effectue sur chacune des couches la convolution avec le même filtre représenté en jaune. En terme de coût, on peut prendre une entrée de taille  $D_F \times D_F \times M$  et un filtre ou

"neurone convolutionnel" de taille  $D_K \times D_K \times M$ . Alors, on effectue tout simplement  $D_K^2 \times D_F^2 \times M$  opérations pour effectuer la convolution. Ce résultat est ensuite multiplier par le nombre de filtres présents soit le nombre de fois qu'on effectue des convolutions standard ce qui donne pour  $N$  filtres un coût total de  $D_K^2 \times D_F^2 \times M \times N$  opérations. Il s'agit de comparer ce résultat à celui de la convolution utilisée par le MobileNet.

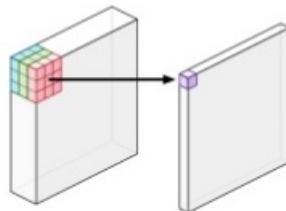


FIGURE 3.38 – Convolution standard sur volume pour un filtre [6]

Pour le MobileNet, l'approche est totalement différente. Les convolutions se font en deux temps. D'une part avec la "depthwise convolution" et d'autre part avec la "pointwise convolution". On parle alors de "depthwise separable convolution" ou convolution séparable en profondeur.

Ainsi, dans un premier temps, au lieu de considérer le même filtre pour chaque couche, on a un filtre sans profondeur qui s'applique à une couche particulière. Cette méthode porte le nom de "depthwise convolution" ou convolution en profondeur.

Cette première étape peut ressembler à la convolution standard à cette différence très notable que les filtres ne sont que des tableaux en deux dimensions. Ainsi, pour un filtre de dimensions  $D_K \times D_K \times M$  en convolution standard, on a  $M$  filtres de taille  $D_K \times D_K \times 1$  en "depthwise convolution".

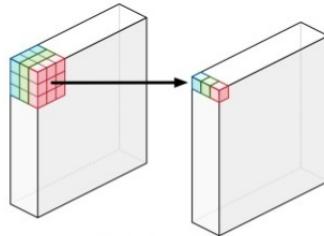


FIGURE 3.39 – Depthwise convolution [6]

Lors de cette première étape, en reprenant les mêmes tailles pour les objets que précédemment, on effectue donc  $D_K^2 \times D_F^2 \times M$  opérations.

Dans un second temps, on effectue à nouveau des convolutions mais uniquement avec des filtres de dimension  $1 \times 1 \times M$ . C'est ce qu'on appelle la "pointwise convolution" ou convolution en point.

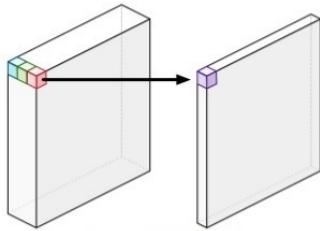


FIGURE 3.40 – Pointwise convolution [6]

En sortie de la "depthwise convolution", on peut noter que la taille du tableau d'entrée va nécessairement varier comme conséquence directe de l'opération de convolution. Il est donc possible, en théorie, de faire appel à une méthode appelée "padding" ou remplissage qui va rajouter des valeurs, souvent 0, en bordure du tenseur, tableau de valeurs à plusieurs dimensions, pour conserver les dimensions d'entrée. Ainsi, les dimensions étant inchangée, on effectue lors de cette "pointwise convolution"  $D_F^2 \times M \times N$  opérations pour les  $N$  filtres.

Cette étape supplémentaire peut paraître à première vue inutile mais c'est pourtant tout l'intérêt du MobileNet. En effet, le nombre total d'opérations effectuées sur le tableau de base est finalement  $D_K^2 \times D_F^2 \times M + D_F^2 \times M \times N$ .

On peut donc maintenant comparer les deux résultats obtenus en déterminant leur quotient.

$$\frac{D_K^2 \times D_F^2 \times M + D_F^2 \times M \times N}{D_K^2 \times D_F^2 \times M \times N} = \frac{1}{N} + \frac{1}{D_K^2}$$

Ce quotient nous donne bien que le nombre d'opérations effectuées par la deuxième méthode est bien inférieur à celui de la première. De plus, on a une relation qui indique que plus on a de filtres ou plus le filtre est de dimensions importantes et plus cette deuxième méthode est efficace par rapport à la première.

C'est dans cette optique de diminution du nombre de calculs que le modèle du MobileNet a été choisi. En outre, le choix s'est tourné vers sa deuxième version qui tend vers toujours plus d'efficacité ce qui la prédomine face à sa première version.

Dans la conception des réseaux de neurones très profonds, on retrouve souvent le concept suivant. On crée un bloc composé de couches de neurones, ces couches étant caractérisées par des paramètres. Ce bloc est répété plusieurs fois à la suite dans le réseau en variant légèrement les paramètres au fur et à mesure. Le MobileNet ne fait pas exception.

Pour expliquer ce concept relatif au MobileNet, lorsqu'on mentionne la taille du tenseur, cela fait uniquement référence à sa dernière dimension qui est la profondeur. En effet, largeur et hauteur restent inchangées. L'intérêt de ne considérer que sa profondeur permet d'observer son évolution de manière claire tout au long du passage de ce tenseur dans un bloc. Ainsi, le bloc de référence du MobileNet prend en entrée un tenseur de taille relativement faible, par exemple 16. A l'intérieur de ce bloc, le tenseur est convolué avec un filtre de dimension  $1 \times 1 \times 64$ .

Lors de cette opération, on passe d'un tenseur de taille 16 à un tenseur de taille 64. On applique ensuite la "depthwise separable convolution" du MobileNet. Enfin, on applique un nouveau filtre de dimension  $1 \times 1 \times 16$ . Alors, la dimension du tenseur en sortie est bien la même que celle en entrée.

Cette opération peut se comprendre grâce à l'analogie suivante. Le tenseur peut être vu comme un fichier compressé. La première convolution agit donc comme un logiciel de décompression qui extrait les informations contenues dans le tenseur. La deuxième est une opération de convolution "classique". La dernière agit comme un logiciel de compression qui ne garde que les informations importantes du résultat des opérations effectuées.

L'intérêt de cette méthode est que le tenseur qui passe d'un bloc au suivant est de dimension relativement faible. Ceci accélère donc le temps de calcul. Cela aide d'autant plus le MobileNet à atteindre son objectif d'efficacité, et fait toute la différence avec sa première version.

Une fois le modèle choisi, il est possible d'en disposer via une bibliothèque disponible sur Python sous le nom de "Keras". C'est une interface qui permet d'accéder à un certain nombre de modèles de réseaux de neurones. Pour faire appel au modèle du MobileNet V2 on aura la commande suivante :

```
keras.applications.mobilenet_v2.MobileNet_v2
```

Une fois le modèle chargé, il est possible de le modifier en fonction des besoins. C'est de cette manière qu'on peut ajuster le réseau de neurones pour qu'il soit spécifiquement adapté au problème posé. Ainsi, une fois le modèle modifié, on peut afficher avec Keras un récapitulatif des neurones présents dans le réseau. Il est donc possible de voir en annexe le tableau de tous les neurones utilisés pour le programme final.

Le modèle du MobileNet V2 peut s'expliquer de la manière suivante : une entrée adaptée pour les images avec un enchaînement de layers pour chaque valeur du tableau puis 15 blocs qui effectuent au fur et à mesure les opérations sur les images et enfin une sortie sous forme de classification comme la régression linéaire jusqu'à 1000 différentes classes.

Après différents essais pour les besoins du projet sur le programme, on a le modèle suivant : l'entrée reste inchangée puis on utilise les 3 premiers bloc du MobileNet V2 et enfin une sortie personnalisée qui permet de classifier selon 3 classes pour chaque direction à prendre.

## 3.4 Entrainement pour les réseaux profonds

On a abordé précédemment l'entraînement de réseaux simples au travers de la régression linéaire et logistique. Cette partie vise donc à généraliser ce concept pour des réseaux de neurones plus complexes. Les entrées ne sont plus de simples variables à une ou deux dimensions mais peuvent atteindre un nombre quelconque de dimensions.

### 3.4.1 La "vectorisation"

Ce concept a pour principal objectif de réduire le nombre de boucles "pour" dans l'entraînement d'un réseau de neurones. En effet, on a pu observer que l'entraînement des réseaux simples se faisait par des itérations sur le nombre d'exemples présents dans l'ensemble des données d'entraînement puis sur le nombre de fois voulu pour analyser cet ensemble de données. Pour des réseaux simples, cette technique ne pose pas de problèmes car le nombre d'opérations totales à effectuer reste faible comparé à la puissance de calculs des ordinateurs. Cependant, cela devient vite obsolète lorsqu'il s'agit de réseaux plus profonds. On a pu constaté précédemment que pour une simple convolution sur un réseau traitant d'images, le nombre d'opérations est très important. C'est là tout l'intérêt de la vectorisation.

Ainsi, on a pu observer pour la régression linéaire que le calcul effectué peut se ramener à un produit scalaire en ajustant bien les notations. C'est cette simple idée qui permet de gagner en efficacité de manière significative par rapport à une boucle "for". On peut donc illustrer cela dans le cas de matrices pour un neurone qui aurait le même genre de fonction que la régression linéaire.

$$W = \begin{pmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \dots & w_{n,n} \end{pmatrix} \text{ Matrice des poids du neurone}$$

$$X = \begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,n} \end{pmatrix} \text{ Matrice des données d'entrée}$$

Alors les calculs à effectuer se ramènent à  $a = \langle W | X \rangle = \text{Tr}(W^T X)$

Cela évite de faire une première boucles pour sur les colonnes puis une seconde sur les lignes ce qui peut rapidement devenir lourd plus  $n$  est grand. Il peut donc être intéressant de comparer les performances des deux méthodes.

```

1 import time
2 import numpy as np
3
4 #On crée des vecteurs aléatoires de grande dimension
5 a = np.random.rand(1000000)
6 b = np.random.rand(1000000)
7
8 tic = time.time()
9 c = np.dot(a,b) #Produit scalaire
10 tac = time.time()
11 print(c)
12 print("Vectorisation " + str(1000*(tac-tic)) + "ms")
13
14 c = 0
15 tic = time.time()
16 for i in range(1000000):
17     c += a[i]*b[i] #Même calcul en passant par une boucle
18 tac = time.time()
19 print(c)
20 print("Boucle pour " + str(1000*(tac-tic)) + "ms")
249653.54034214703
Vectorisation 1.9962787628173828ms
249653.54034214708
Boucle pour 749.0277290344238ms

```

On a effectivement un différence de taille. En effet, pour obtenir le même résultat, la vectorisation met en ordre de grandeur  $10^3$  fois moins de temps que la boucle pour. Adaptée à l'échelle du traitement d'images, cette méthode est donc un moyen efficace de gagner du temps sur l'analyse de données.

### 3.4.2 La descente de gradient

Cette méthode est une des raisons qui explique la performance des réseaux de neurones. En effet, c'est de cette manière qu'il est possible de minimiser les erreurs lorsqu'il s'agit d'analyser les données pour ensuite effectuer des prédictions. Elle se base sur les propriétés du gradient.

*Le gradient en un point d'un fonction correspond au vecteur des dérivées partielles de cette fonction évaluées en ce point.*

Soit  $f: \mathbb{R}^n \rightarrow \mathbb{R}$

Soit  $a \in \mathbb{R}^n$

On appelle gradient de  $f$  en  $a$  le vecteur  $(\frac{\partial f}{\partial x_1}(a), \dots, \frac{\partial f}{\partial x_n}(a))$  noté  $\nabla f(a)$

Parmi les propriétés de ce vecteur, certaines prennent tout leur intérêt dans l'entraînement des réseaux de neurones.

1.  $\nabla f(a)$  est normal à la tangente de la courbe de  $f$  en  $a$
2.  $\nabla f(a)$  est orienté vers les valeurs croissantes de  $f$

La première propriété signifie que le gradient est un indicateur de l'évolution de la fonction  $f$ . La seconde est celle qui permet la descente de gradient. En effet, en plus d'indiquer les variations de  $f$ , le gradient donne la direction des valeurs croissantes. Afin de minimiser cette fonction, il suffit simplement de prendre l'opposé du gradient pour se diriger petit à petit vers le minimum.

La descente de gradient à toute dimension s'opère donc de la manière suivante :

- On prend  $a_0$  quelconque sur la courbe de  $f$
- On choisit un pas  $\lambda$  qui correspond au "learning rate" vu précédemment.
- On crée une suite  $(a_n)$  telle que  $\forall n \in \mathbb{N} a_n = a_{n-1} + \lambda \cdot \nabla f(a_{n-1})$

D'après le cours de Mathématiques de V. Rohart.

Cette méthode peut s'illustrer avec la figure 41. On a donc un point pris au maximum de  $f$  puis au fur et à mesure qu'on opère cette descente de gradient, le point rose a une image par  $f$  qui tend vers le minimum.

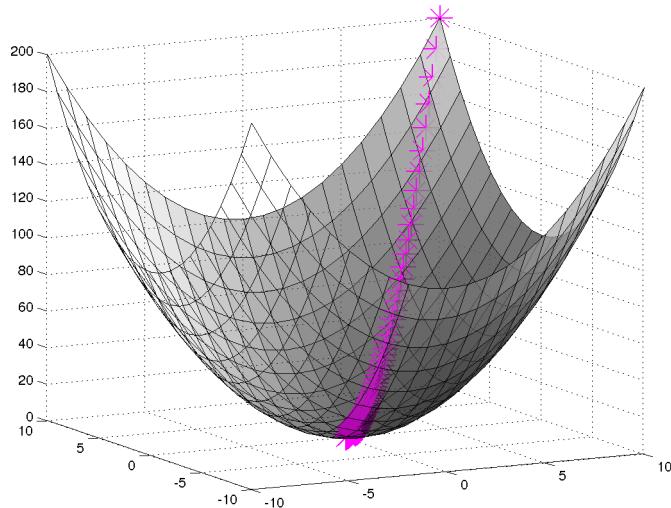


FIGURE 3.41 – Descente de gradient (Illustration par MathWorks)

### 3.4.3 Entrainement du programme

Le programme étant élaborée, c'est cette dernière étape décisive qui va permettre d'aboutir au résultat attendu. Ainsi, avec toutes les données récupérées, il s'agit pour le programme de les analyser pour en tirer les caractéristiques nécessaires pour s'adapter à un circuit inconnu. Grâce à la vectorisation, on a directement l'image en entrée comme tableau de valeur et on effectue les calculs comme vu précédemment.

On peut appuyer le fait que plus il a de données à sa disposition et plus le programme peut réagir face à des situations nouvelles. Dans le but de doubler le nombre de données acquises, il est possible de faire un effet miroir sur toutes les photos et disposer d'un nouveau set pour chaque type gauche, centre et droit.

De même, les photos étant presque toutes prises dans le même environnement, un inconvénient peut survenir : le programme est efficace uniquement s'il reconnaît cet environnement particulier. En effet, c'est un cas typique d'"overfitting". Le programme a plus de données sur les alentours que sur la route. Pour empêcher cela, on applique donc un cache noir sur chaque photo qui est ensuite analysée. Ainsi, il aura en toute situation ce cache qui lui permettra de fonctionner en n'importe quel environnement.

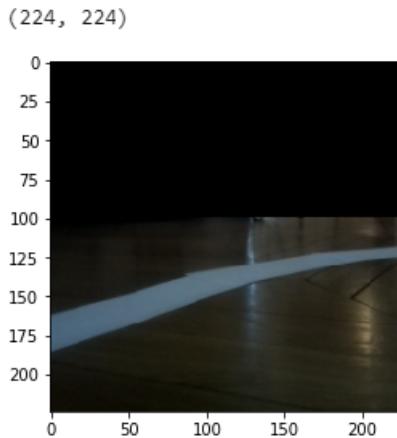


FIGURE 3.42 – Application du cache

On observe que le sol entre les lignes de la route ne peut être enlevé. Le programme va donc nécessairement trop analyser ce sol particulier qui peut limiter la généralisation. Pour diminuer ce problème, il suffit de prendre des données sur différents sols pour compléter l'entraînement du programme.

Après cet entraînement, la voiture est capable de suivre le tracé d'une route composée de deux lignes blanches. Il s'agit donc maintenant de lui permettre de réagir face à d'autres situations comme des piétons sur la route ou des panneaux de signalisation. Pour ce faire, il existe différentes manières d'aborder ces nouveaux problèmes. Il est possible d'entraîner le modèle existant avec de nouvelles données relatives aux nouvelles situations, ou alors d'approfondir le modèle en prenant non plus 3 blocs mais 6 du MobileNet V2 ou enfin en entraînant un autre réseau de neurones qui se chargera de détecter ces obstacles et agira avant le modèle existant pour donner la commande s'arrêter ou de continuer à avancer.

# Conclusion

Tout l'enjeu de cette voiture autonome miniature réside dans l'utilisation des réseaux de neurones. Ils disposent d'une adaptation qui permet un large éventail d'applications. La reconnaissance d'images effectuée ici en est une des plus fréquentes. Cependant, une fois l'image reconnue, les possibilités sont multiples. En effet, il s'agit uniquement ici de commander des moteurs en fonction de la réponse du réseau face à l'image, mais ce n'est qu'une seule application de cette reconnaissance d'image. De plus, l'étude de ces réseaux de neurones découle directement de concepts mathématiques qui prennent un intérêt nouveau, parfois inattendu. Étant donné que cette technologie prend toujours plus d'ampleur, il est intéressant de pouvoir l'aborder ici et de la mettre directement à profit dans un projet très concret et très actuel. Ce travail d'initiative personnelle encadré est donc une occasion unique de pouvoir étudier de près ces concepts et d'aboutir à cette voiture miniature, fruit de plusieurs mois de travaux à la fois théoriques et pratiques. Afin d'apporter une réponse à la problématique de départ, la miniaturisation du concept de voiture autonome est le résultat d'une prise en main complète de l'électronique nécessaire, ainsi que d'une compréhension fine des concepts des réseaux de neurones et de leur utilisation.

Nous tenons à remercier les professeurs encadrant M. Jean-Marie Saint-Jalm et M. Camille Gaudilliere pour leur accompagnement tout au long de la conception de ce projet. Nous voulons aussi souligner l'aide apportée par notre camarade de classe Armand Barral qui a su résoudre des problèmes parfois épineux.

# Sources

- [1] J. Hui. Real-Time Object Detection, Accès 19/05/19.
- [2] L. Reynier. Kesaco Arduino [PDF], 2010.
- [3] A. Dertat. Applied Deep Learning Part 1, Accès 19/05/19.
- [4] A. Ng. Deeplearning.ai, Accès 19/05/19.
- [5] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam. MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications [PDF], 2017.
- [6] J. Lee. MobileNet PR054 SlideShare, Accès 19/05/19.

Arduino. Arduino Home, Accès 19/05/19.

J-M. Saint-Jalm. Python, Cours, Volume 2, 2018.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L-C. Chen. MobileNetV2 : Inverted Residuals and Linear Bottlenecks [PDF], 2019.

OpenClassroom. Programmez vos premiers montages avec Arduino, Accès 19/05/19.

P-L. Pröve. MobileNetV2 : Inverted Residuals and Linear Bottlenecks, Accès 19/05/19.

R. Deora. MobileNet Research Paper Walkthrough, Accès 19/05/19.

V. Rohart. Cours de Mathématiques, 2019.

```
[ ] # Dowloading libraries on the cloud machine
!pip install numpy
!pip install scipy

[ ] # Importing libraries
from matplotlib import pyplot as plt
import scipy.misc
import numpy as np
import os

[ ] # Accessing Google Drive
from google.colab import drive
drive.mount('/content/gdrive')

[ ] # Path to our raw data
path_data = '/content/gdrive/My Drive/mobile_net_tipe/data/train/'
```

## ▼ Flip & Save

```
[ ] # gauche à droite
path_img = path_data + 'gauche'
path_save = path_data + 'droite'

img_list = os.listdir(path_img)
print(len(img_list))
m = 2800
compt = 0

for img_file in img_list[:m]:
    arr = plt.imread(path_img + '/' + img_file)
    arr = np.fliplr(arr)
    scipy.misc.toimage(arr).save(path_save + '/' + str(m + compt) + '.png')
    compt += 1
    if compt % 500 == 0:
        print(compt)

[ ] # droite à gauche
path_img = path_data + 'droite'
path_save = path_data + 'gauche'

img_list = os.listdir(path_img)
m = 2800
compt = 0

for img_file in img_list[:m]:
    arr = plt.imread(path_img + '/' + img_file)
    arr = np.fliplr(arr)
    scipy.misc.toimage(arr).save(path_save + '/' + str(m + compt) + '.png')
    compt += 1
    if compt % 500 == 0:
        print(compt)

[ ] # centre à centre
path_img = path_data + 'centre'
path_save = path_data + 'centre'

img_list = os.listdir(path_img)
m = 2800
compt = 0

for img_file in img_list[:m]:
    arr = plt.imread(path_img + '/' + img_file)
    arr = np.fliplr(arr)
    scipy.misc.toimage(arr).save(path_save + '/' + str(m + compt) + '.png')
    compt += 1
    if compt % 500 == 0:
        print(compt)
```

## ▼ Dark

### ▼ Exemple

```
[ ] path_img = path_data + 'gauche'
    img_list = os.listdir(path_img)
    img_file = img_list[0]
    arr = plt.imread(path_img + '/' + img_file)
    dark = np.ones(arr.shape)
    m = int((dark.shape[0]) * 0.45)
    for i in range(m):
        dark[i] = np.zeros(dark.shape[1:])
    arr = np.fliplr(arr)
    print(arr.shape)
    plt.imshow(arr)
    plt.show()
    plt.imshow(dark)
    plt.show()
    plt.imshow(arr*dark)
    plt.show()
```

## ▼ Traitement

```
[ ] path_data = '/content/gdrive/My Drive/mobile_net_tipe/data/train/'
path_save = '/content/gdrive/My Drive/mobile_net_tipe/data_dark/train/'

[ ] # train
files = ['droite']
dark = np.ones((224, 224, 3))
n = int((dark.shape[0]) * 0.45)
for i in range(n):
    dark[i] = np.zeros(dark.shape[1:])
for file in files:
    print(file)
    path_img = path_data + file
    path_new = path_save + file
    img_list = os.listdir(path_img)
    m = len(img_list)
    compt = 0
    for img_file in img_list:
        arr = plt.imread(path_img + '/' + img_file)
        res = arr*dark
        scipy.misc.toimage(res).save(path_new + '/' + img_file)
        compt += 1
        if compt % 200 == 0:
            print(str(compt) + ' / ' + str(m))
```

```
[ ] path_data = '/content/gdrive/My Drive/mobile_net_tipe/data/valid/'  
path_save = '/content/gdrive/My Drive/mobile_net_tipe/data_dark/valid/'
```

```
[ ] # valid  
files = ['gauche', 'centre', 'droite']  
dark = np.ones((224, 224, 3))  
n = int((dark.shape[0]) * 0.45)  
for i in range(n):  
    dark[i] = np.zeros(dark.shape[1:])  
for file in files:  
    print(file)  
    path_img = path_data + file  
    path_new = path_save + file  
    img_list = os.listdir(path_img)  
    m = len(img_list)  
    compt = 0  
    for img_file in img_list:  
        arr = plt.imread(path_img + '/' + img_file)  
        res = arr*dark  
        scipy.misc.toimage(res).save(path_new + '/' + img_file)  
        compt += 1  
        if compt % 200 == 0:  
            print(str(compt) + ' / ' + str(m))
```

```
[ ] path_data = '/content/gdrive/My Drive/mobile_net_tipe/data/test/'  
path_save = '/content/gdrive/My Drive/mobile_net_tipe/data_dark/test/'
```

```
[ ] # test  
files = ['gauche', 'centre', 'droite']  
dark = np.ones((224, 224, 3))  
n = int((dark.shape[0]) * 0.45)  
for i in range(n):  
    dark[i] = np.zeros(dark.shape[1:])  
for file in files:  
    print(file)  
    path_img = path_data + file  
    path_new = path_save + file  
    img_list = os.listdir(path_img)  
    m = len(img_list)  
    compt = 0  
    for img_file in img_list:  
        arr = plt.imread(path_img + '/' + img_file)  
        res = arr*dark  
        scipy.misc.toimage(res).save(path_new + '/' + img_file)  
        compt += 1  
        if compt % 200 == 0:  
            print(str(compt) + ' / ' + str(m))
```

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_1 (InputLayer)	(None, 224, 224, 3)	0	
Conv1_pad (ZeroPadding2D)	(None, 225, 225, 3)	0	input_1[0][0]
Conv1 (Conv2D)	(None, 112, 112, 32)	864	Conv1_pad[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (DepthwiseConv2D) (None, 112, 112, 32)	288	Conv1_relu[0][0]	
expanded_conv_depthwise_BN (BatchNormalization) (None, 112, 112, 32)	128	expanded_conv_depthwise[0][0]	
expanded_conv_depthwise_relu (ReLU) (None, 112, 112, 32)	0	expanded_conv_depthwise_BN[0][0]	
expanded_conv_project (Conv2D) (None, 112, 112, 16)	512	expanded_conv_depthwise_relu[0][0]	
expanded_conv_project_BN (BatchNormalization) (None, 112, 112, 16)	64	expanded_conv_project[0][0]	
block_1_expand (Conv2D)	(None, 112, 112, 96)	1536	expanded_conv_project_BN[0][0]
block_1_expand_BN (BatchNormalization) (None, 112, 112, 96)	384	block_1_expand[0][0]	
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0	block_1_expand_BN[0][0]
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	block_1_expand_relu[0][0]
block_1_depthwise (DepthwiseConv2D) (None, 56, 56, 96)	864	block_1_pad[0][0]	
block_1_depthwise_BN (BatchNormalization) (None, 56, 56, 96)	384	block_1_depthwise[0][0]	
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	0	block_1_depthwise_BN[0][0]
block_1_project (Conv2D)	(None, 56, 56, 24)	2304	block_1_depthwise_relu[0][0]
block_1_project_BN (BatchNormalization) (None, 56, 56, 24)	96	block_1_project[0][0]	
block_2_expand (Conv2D)	(None, 56, 56, 144)	3456	block_1_project_BN[0][0]
block_2_expand_BN (BatchNormalization) (None, 56, 56, 144)	576	block_2_expand[0][0]	
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_2_expand_BN[0][0]
block_2_depthwise (DepthwiseConv2D) (None, 56, 56, 144)	1296	block_2_expand_relu[0][0]	
block_2_depthwise_BN (BatchNormalization) (None, 56, 56, 144)	576	block_2_depthwise[0][0]	
block_2_depthwise_relu (ReLU)	(None, 56, 56, 144)	0	block_2_depthwise_BN[0][0]
block_2_project (Conv2D)	(None, 56, 56, 24)	3456	block_2_depthwise_relu[0][0]
block_2_project_BN (BatchNormalization) (None, 56, 56, 24)	96	block_2_project[0][0]	
block_2_add (Add)	(None, 56, 56, 24)	0	block_1_project_BN[0][0] block_2_project_BN[0][0]
block_3_expand (Conv2D)	(None, 56, 56, 144)	3456	block_2_add[0][0]
block_3_expand_BN (BatchNormalization) (None, 56, 56, 144)	576	block_3_expand[0][0]	
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_3_expand_BN[0][0]
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	0	block_3_expand_relu[0][0]
block_3_depthwise (DepthwiseConv2D) (None, 28, 28, 144)	1296	block_3_pad[0][0]	
block_3_depthwise_BN (BatchNormalization) (None, 28, 28, 144)	576	block_3_depthwise[0][0]	
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	0	block_3_depthwise_BN[0][0]
block_3_project (Conv2D)	(None, 28, 28, 32)	4608	block_3_depthwise_relu[0][0]
block_3_project_BN (BatchNormalization) (None, 28, 28, 32)	128	block_3_project[0][0]	
conv2d_8 (Conv2D)	(None, 11, 11, 128)	200832	block_3_project_BN[0][0]
batch_normalization_8 (BatchNormalization) (None, 11, 11, 128)	512	conv2d_8[0][0]	
activation_8 (Activation)	(None, 11, 11, 128)	0	batch_normalization_8[0][0]
global_average_pooling2d_9 (GlobalAveragePooling2D) (None, 128)	0	activation_8[0][0]	
dense_9 (Dense)	(None, 32)	4128	global_average_pooling2d_9[0][0]
dropout_8 (Dropout)	(None, 32)	0	dense_9[0][0]
dense_10 (Dense)	(None, 1)	33	dropout_8[0][0]
output (ReLU)	(None, 1)	0	dense_10[0][0]
<hr/>			
Total params: 233,153			
Trainable params: 231,041			
Non-trainable params: 2,112			

```
[ ] # Downloading libraries on the cloud machine
!pip install tensorflow-gpu
!pip install keras
!pip install numpy
!pip install scikit-learn
```

```
[ ] # Loading the libraries
import numpy as np
import pickle
import os
import time

import tensorflow as tf
import keras
from keras.models import Model, Sequential
from keras.layers import Input, Conv2D, MaxPooling2D, MaxPool2D, Dropout, Flatten, Dense
from keras.layers import Activation, BatchNormalization, GlobalAveragePooling2D, ReLU
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array

from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
%matplotlib inline
```

```
[ ] # Connection to Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
[ ] path_data = '/content/gdrive/My Drive/mobile_net_tipe/data_dark'
```

```
[ ] train_path = path_data + '/train'
valid_path = path_data + '/valid'
test_path = path_data + '/test'
```

```
[ ] # Creating the data generators
train_batches = ImageDataGenerator(preprocessing_function =
                                    keras.applications.mobilenet_v2.preprocess_input)
          .flow_from_directory(train_path, target_size = (224, 224),
                               batch_size = 64, class_mode = 'sparse')

valid_batches = ImageDataGenerator(preprocessing_function =
                                    keras.applications.mobilenet_v2.preprocess_input)
          .flow_from_directory(valid_path, target_size = (224, 224),
                               batch_size = 30, class_mode = 'sparse')

test_batches = ImageDataGenerator(preprocessing_function =
                                    keras.applications.mobilenet_v2.preprocess_input)
          .flow_from_directory(test_path, target_size = (224, 224),
                               batch_size = 30, shuffle = False, class_mode = 'sparse')
```

↳ Found 16800 images belonging to 3 classes.  
 Found 300 images belonging to 3 classes.  
 Found 300 images belonging to 3 classes.

```
[ ] # Loading th MobileNet graph
mobile = keras.applications.mobilenet_v2.MobileNetV2()
```

```
[ ] # Printing the representation of the MobileNet
mobile.summary()
```

```
[ ] # Creating our Deep Neural Network
x = mobile.layers[36].output

x = Conv2D(128, kernel_size = 7, strides = 2)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = GlobalAveragePooling2D()(x)

x = Dense(32, activation = 'relu')(x)
x = Dropout(0.4)(x)

x = Dense(1)(x)
output = ReLU(max_value = 2, name = 'output')(x)
```

```
[ ] # Creating the model
my_model = Model(inputs = mobile.input, outputs = output)
```

```
[ ] # Printing the representation of our model
my_model.summary()
```

```

from imutils.video import VideoStream
from threading import Thread
import imutils
import cv2
import numpy as np
import time
import os

fichier = "data/..." # fichier ou sauvegarder les donnees

# temps max de fonctionnement du programme
max_record_time = 600 # in seconds
# intervalle de sauvegarde des donnees
intervalle = 10 # in 100th of second
# attente pour avoir le temps de preparer la voiture
dodo = 30
# nombre de donnees a prendre
nb_imgs = 3000

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(dodo)

compteur = 0 # pour ne pas se perdre dans la boucle while
print('Go !')
start = int(round(time.time())) # plus que max_record_time secondes d'execution

# on enregistre les images dans un ordre aleatoire
order = np.arange(nb_imgs)
np.random.shuffle(order)

# repere pour les intervalles
start2 = int(round(time.time()*100))

while time.time() - start < max_record_time:

    # Waiting 2 seconds because the first datas are corrupted
    # At least "intervalle" 100th of second between each capture
    if (time.time() - start) >= 2 and (int(round(time.time()*100)) > (start2 + intervalle)):

        frame = vs.read() # prise de la photo

        # on resize
        frame = imutils.resize(frame, height = 224)
        frame = frame[:, 37:-37, :]

        # on enregistre
        cv2.imwrite("{}/{}.png".format(fichier, int(order[compteur])), frame.astype("uint8"))

        # pour ne pas se perdre dans le comptage des donnees
        compteur += 1

        # a "nb_imgs" donnees on arrete la prise de donnees
        if compteur >= nb_imgs:
            break

        # repere pour les intervalles
        start2 = int(round(time.time()*100))

    start -= 2

    # Printing some infos
    print("Time : " + str((int(round(time.time()*100)) - start*100) / 100) +
          "s      Nb data : " + str(compteur) )
    print( str(round((float((round(time.time()*100)) - start*100) / 100.) / compteur, 3)) +
          " seconds per data")

# on ferme le tout proprement
print("[INFO] cleaning up...")
cv2.destroyAllWindows()
vs.stop()

```

```

from imutils.video import VideoStream
from threading import Thread
from keras.models import load_model
from keras.applications.mobilenet_v2 import preprocess_input
import imutils
import cv2
import numpy as np
import serial
import time
import os

MODEL_PATH = 'models/model_final_3dir_relu2_v0016_t002.hdf5'

# Donnees pour la connexion a l'arduino
com_arduino = '/dev/ttyACM0'
baudrate_arduino = 9600

# Limite de temps d'execution du programme
max_working_time = 150 # in seconds
intervalle = 10 # in 100th of second
# Pause avant lancement
dodo = 3

# Encoder les int en bytes
def my_encode(x):
    if x == 0:
        return b'0'
    elif x == 1:
        return b'1'
    elif x == 2:
        return b'2'
    else:
        return b'3'

# On attend l'arduino avant de continuer
# Pour rester synchronise avec elle
def wait_for_arduino(ser):
    while ser.inWaiting() < 0:
        time.sleep(.1)
    time.sleep(.1)
    arData = str(ser.readline())

# Creation du masque
dark = np.ones((224, 224, 3))
for i in range(int(224*0.45)):
    dark[i] = np.ones((224, 3))

# loading the model
print("[INFO] loading model...")
model = load_model(MODEL_PATH)

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(dodo)

```

```

# Trying to connect
connectionSuccess = False
while not connectionSuccess:
    try:
        ser = serial.Serial(com_arduino, baudrate = baudrate_arduino, timeout = 1)
        connectionSuccess = True
    except (OSError, serial.SerialException):
        sleep(.1)
        pass

print('Connected')

ser.flushInput()

print('Go !')

# Repere pour les intervalles
start = int(round(time.time()))
start2 = int(round(time.time()*100))

while time.time() - start < max_working_time:

    # Waiting 2 seconds because the first datas are corrupted
    # At least "intervalle" 100th of second between each capture
    if (time.time() - start >= 2) and (int(round(time.time()*100)) > (start2 + intervalle)):

        frame = vs.read() # Prise de la photo

        # On resize
        frame = imutils.resize(frame, height = 224)
        frame = frame[:, 37:-37, :]

        # On applique le filtre
        frame = frame*dark

        # Preprocessing
        frame = preprocess_input(frame)
        # (224, 224, 3) -> (1, 224, 224, 3)
        # car .predict() ne prend que des tenseurs a 4 dimensions
        frame = np.expand_dims(frame, axis = 0)

        # Orediction
        pred = model.predict(frame)[0]
        pred2 = int(pred)
        print(pred, pred2)

        # On l'envoie a l'arduino
        ser.write(my_encode(pred2))

        # On s'assure de rester synchro
        wait_for_arduino(ser)

        # Repere pour les intervalles
        start2 = int(round(time.time()*100))

    start -= 2

    # on ferme le tout proprement
    print("[INFO] cleaning up...")
    cv2.destroyAllWindows()
    vs.stop()

```

```

#include <SPI.h>
#include <RF24.h>
#include <Servo.h>

int autonome = 1;

int old_speed = 0;
int new_speed = 0;
int old_dir = 90;
int new_dir = 90;

// pins de commande radio
int pinCSN = 10;
int pinCE = 9;

int pinServo = 8;

// pins de commande du moteur CC
int pinMoteur = 5;
int in3 = 3;
int in4 = 4;

// initialisation des variables lors de la conduite manuelle
int angleServo = 0;
int valMoteur = 0;

// initialisation des variables lors de la conduite autonome
char centre = '1';
char droite = '2';
char gauche = '0';
char arret = '3';

// creation de l'objet Radio et de l'objet Servo
RF24 radio(pinCE, pinCSN);
Servo servo;

// adresse de communication des deux radios
const byte adresse[6] = "00001";

void setup()
{
    // initialisation de la communication arduino raspberry
    Serial.begin(9600);
    // initialisation des pins
    pinMode(pinMoteur, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
    // la voiture ne pourra pas reculer (qu'avancer)
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
    // mise a 0 du moteur cc
    analogWrite(pinMoteur, 0);
    // mise a 0 du servo moteur
    servo.attach(pinServo);
    servo.write(90);
    // demarrage de la radio
    radio.begin();
    // cette radio va lire les infos
    radio.openReadingPipe(0, adresse);
    // puissance emettrice
    radio.setPALevel(RF24_PA_LOW);
    // pret a l'emploi
    radio.startListening();
}

```

```

void loop()
{
    // conduite manuelle
    if (autonome == 0)
    {
        delay(5); // 5ms
        if (radio.available()) // attendre qu'un message soit envoyé
        {
            // recuperation des données radio
            double val_servo_dc;
            radio.read(&val_servo_dc, sizeof(val_servo_dc));
            if (val_servo_dc == 10000) // stop
            {
                Serial.println("stop");
            }
            else
            {
                // mise à jour des moteurs
                angleServo = decode_val_servo(val_servo_dc);
                valMoteur = decode_val_dc(val_servo_dc);
                int data_to_python = val_servo_dc*100;
                Serial.println(data_to_python);
                servo.write(angleServo);
                analogWrite(pinMoteur, valMoteur);
                delay(10);
            }
        }
    }
    // conduite autonome
    else if (autonome == 1)
    {
        if (Serial.available() > 0) // attendre qu'un message soit envoyé
        {
            // lecture du message
            char rasp_info = Serial.read();
            delay(5);
            // on le renvoie pour que raspberry continue son exécution
            // cette étape est comme une synchro entre arduino et raspberry
            Serial.println(rasp_info);
            // mise à jour des moteurs
            new_dir = pred_to_angle(rasp_info, gauche, droite, centre, arret);
            new_speed = pred_to_vit(rasp_info, gauche, droite, centre, arret);
            servo.write(new_dir);
            analogWrite(pinMoteur, new_speed);
            delay(10);
        }
    }
}

```

```

// mes fonctions
int decode_val_servo(double v_radio)
{
    if (v_radio == round(v_radio))
    {
        return v_radio*30;
    }
    else
    {
        int val_servo = round(v_radio);
        val_servo = 30*val_servo;
        return val_servo;
    }
}

int decode_val_dc(double v_radio)
{
    if (v_radio == round(v_radio))
    {
        return 0;
    }
    else
    {
        int val_dc = (v_radio - round(v_radio - 0.5))*1000;
        return val_dc;
    }
}

int pred_to_vit(char x, char gauche, char droite, char centre, char arret)
{
    if (x == arret)
    {
        return 0;
    }
    else if (x == centre)
    {
        return 250;
    }
    else if (x == gauche)
    {
        return 190;
    }
    else
    {
        return 190;
    }
}

int pred_to_angle(char x, char gauche, char droite, char centre, char arret)
{
    if (x == arret)
    {
        return 90;
    }
    else if (x == centre)
    {
        return 90;
    }
    else if (x == gauche)
    {
        return 150;
    }
    else
    {
        return 30;
    }
}

```

```

#include <SPI.h>
#include <RF24.h>

// pins radio
int pinCSN = 10;
int pinCE = 9;

//pins moteurs
int pinPotServo = A1;
int pinPotMoteur = A5;

// creation de l'objet radio
RF24 radio(pinCE, pinCSN);

// adresse de communication
const byte adresse[6] = "00001";

void setup()
{
    Serial.begin(9600);
    // demarrage de la radio
    radio.begin();
    // cette radio va envoyer les infos
    radio.openWritingPipe(adresse);
    // puissance emettrice
    radio.setPALevel(RF24_PA_LOW);
    // pret a l'emploi
    radio.stopListening();
}

void loop()
{
    // recuperation des valeurs des joysticks
    int valPotServo = analogRead(pinPotServo);
    int valPotMoteur = analogRead(pinPotMoteur);
    // conversion en donnees utilisables
    int angle = my_map(valPotServo);
    int valMot = map(valPotMoteur, 0, 1023, 50, 0);
    if (valMot <= 30){
        valMot = 0;
    }
    else{
        valMot = valMot - 25;
    }
    // creation du message
    double message = encode(angle, valMot);
    // envoie du message
    radio.write(&message, sizeof(message));
    Serial.println(message);
    delay(10); // 5ms
}

// mes fonctions
double encode(int angle, double valMot)
{
    return (angle + (valMot/100));
}

int my_map(int val)
{
    val = val / 205;
    val = val + 1;
    val = 6 - round(val);
    return val;
}

```