



PÔLE VÉHICULE AUTONOME

Autonomous Driving Cup

Auteurs :

Loïc DUHAMEL
Thomas LEFEBVRE
António LOISON
François POINSIGNON

Référents :

Anthony KOLAR
Raul DE LACERDA

15 juin 2019

Table des matières

1	Introduction	1
2	Contexte et enjeux	1
2.1	Enjeux sociétaux	1
2.2	Enjeux technologiques	2
3	Cahier des charges et objectifs	2
4	Organisation du travail	3
5	Travail effectué	3
5.1	Architecture algorithmique	3
5.2	Suivi de lignes	5
5.2.1	Approche du problème	5
5.2.2	Étude des solutions existantes	5
5.2.3	Les différentes implémentations existantes	6
5.2.4	Solution choisie et explication du fonctionnement	9
5.2.5	Tâches réalisées	13
5.2.6	Axes d'amélioration	14
5.3	Obstacles	15
5.3.1	Problématique, technologie employée	15
5.3.2	Traitements de l'image	16
5.3.3	Détection d'obstacles	18
5.3.4	Comportement du véhicule	18
5.3.5	Problématiques restantes	20
5.4	Détection et reconnaissance de panneaux	20
5.4.1	Detection des formes	21
5.4.2	Reconnaissance des panneaux	22
5.4.3	Axes d'amélioration	26
5.5	Algorithme de Décision	26
5.5.1	Formalisme des ordres	26
5.5.2	Règles de décision	27
5.5.3	Hiérarchisation des ordres	28
5.6	Hardware	28
5.6.1	Prise en main de la raspberry	28
5.6.2	Utilisation du ZumoShield	30
5.6.3	Petits calculs et création de l'algorithme	30
5.6.4	Raspberry Compute Module 3	31
6	Conclusion et ouverture	32

1 Introduction

Le projet de S6 est un point-clé du nouveau cursus de CentraleSupélec. En effet, ce projet permet de choisir de travailler sur un sujet précis, concret et personnel. De plus, ce projet peut être lié à une entreprise et donc avoir un effet professionnalisant. Rencontres avec des clients, recherches bibliographiques et travaux en autonomie sont des éléments importants que peut apporter le projet de S6.

Dans ce cadre, nous nous sommes regroupés en une équipe de quatre élèves de première année. Lors du choix en février, nous avons préféré ne pas nous engager dans un projet mêlant premières et deuxièmes années car nous voulions une équipe homogène, sans différence de niveau, pour assurer une bonne cohésion.

Au moment de choisir notre pôle projet, nous nous sommes tous les quatre directement intéressés au pôle « véhicule autonome ». En effet, en plus de l'intérêt tout particulier que nous portons à ce domaine de recherche, nous pensons que c'est un secteur d'avenir, qui est amené à beaucoup évoluer lors des prochaines décennies. Étant en pleine réflexion quant à notre futur, nous avons jugé que ce pôle représentait une bonne façon de se donner une première idée des problématiques liées à ce secteur.

Par la suite, nous avons pu connaître, par l'intermédiaire des professeurs, l'existence d'une compétition de programmation de véhicule autonome : l'Audi Autonomous Driving Cup (AADC). Ce projet, complexe et ambitieux, nous a tout de suite séduit. Notre équipe s'est alors constituée d'António Loison, de François Poinsignon, de Loïc Duhamel et de Thomas Lefebvre.

2 Contexte et enjeux

2.1 Enjeux sociétaux

Avec près de 1,25 million de morts et jusqu'à 50 millions de blessés dans le monde[1], les voitures restent l'un des moyens de transport les moins sûrs au monde alors que c'est l'un des plus utilisés. La sécurisation de ce transport est donc un des grands enjeux du XXI^e siècle. De plus, la voiture est l'un des plus grands pollueurs du monde. Elle représentait 26% des émissions de *CO*₂ en 2009 aux Etats-Unis [2]. Face à cet enjeu environnemental, l'humanité se doit d'agir pour utiliser ce moyen de transport si pratique de manière plus intelligente.

Les voitures, et plus généralement les véhicules autonomes sont donc un enjeu crucial des mobilités de demain. En effet, les demandes de mobilités ne vont faire que s'amplifier et les problèmes environnementaux et urbaines risquent d'empirer si on ne trouve pas de nouvelles façons de se déplacer.

La voiture autonome est une des solutions aux problèmes. Grâce à sa prise en main entièrement automatisée, les risques d'accidents sont très amoindris. D'autre part, l'absence de conducteur crée un nouveau paradigme d'utilisation. Son coût d'utilisation occasionnel serait beaucoup plus faible que les coûts actuels. Ainsi, au lieu d'appartenir et d'être utilisée par une seule personne, la voiture pourrait être partagée intelligemment par un groupe d'individus. Au lieu de passer la plupart de

son temps au garage, l'utilisation de la voiture serait optimisée pour transporter un maximum de personnes sur une journée. La production de véhicules et le nombre de véhicules en circulation seraient donc réduits.

2.2 Enjeux technologiques

Le véhicule autonome est un système très complexe faisant appel à de nombreuses briques technologiques pour son développement. D'une part, le véhicule doit être capable de comprendre l'environnement autour de lui. Cela nécessite de multiples capteurs et des logiciels permettant d'interpréter les données issus de ces instruments. D'autre part, grâce à l'intelligence artificielle, le véhicule doit pouvoir prendre des décisions tout en garantissant la sécurité des passagers à bord et des usagers de la route aux alentours. Le milieu dans lequel évolue le véhicule est très imprévisible et il se doit d'avoir des logiciels assez robustes pour se prémunir vis à vis de tous les événements possibles. Néanmoins, la puissance de calcul du véhicule est limitée. Une grande part de l'énergie consommée par les véhicules autonomes, aujourd'hui, est utilisée pour le traitement informatique. Il est donc important que les programmes soient optimisés au maximum pour réduire la consommation d'énergie dans le contexte de la crise environnementale mondiale.

Dans cette optique, le pôle projet véhicule autonome représente un secteur d'avenir. Il est en effet nécessaire de former des ingénieur.e.s pour continuer la recherche et développer des solutions d'avenir. L'école CentraleSupélec s'engage donc dans cette voie en encourageant les projets liés à cette discipline. De plus, le secteur de l'automobile, qui se remet difficilement de la crise de 2008-2009 [3] a bon espoir de redorer son blason grâce au déploiement très proche des véhicules autonomes [4]. C'est donc un secteur qui aujourd'hui recrute des ingénieurs, et, en tant que bâtisseurs du monde de demain, il est nécessaire que nous nous formions à ces technologies d'avenir.

Enfin, dans une optique plus générale, notre travail sur l'intelligence artificielle et la reconnaissance d'image nous forme à des technologies d'avenir, qui sont utilisées massivement dans de nombreux secteurs (industries, nouvelles technologies, IoT...).

3 Cahier des charges et objectifs

Lorsque nous avons commencé à travailler sur ce projet, nous avions un objectif clair et précis : participer (voire gagner) l'AADC. Cette compétition, organisée par Audi, vise à ce que des étudiants de différentes nations programment un modèle réduit de voiture Audi, possédant de nombreux capteurs, pour qu'elle soit capable de suivre une route, de détecter et de comprendre des panneaux, ou de s'arrêter en cas d'obstacle. Les inscriptions devaient avoir lieu début avril, pour une phase finale se passant en novembre 2018.

Malheureusement, nous n'avons pas pu participer à cette compétition. Lorsqu'à la mi-avril le site d'inscription n'était toujours pas ouvert, nous leur avons envoyé un mail. Ils nous ont alors expliqué que la compétition n'aurait pas lieu cette année. Très déçus, mais néanmoins toujours très motivés par la programmation de véhicule autonome, nous avons alors décidé de créer nous même notre propre voiture autonome.

Nous avons donc travaillé sur une raspberry pi, reliée à une Arduino contrôlant un Zumo Shield [3]. Nous avions pour objectif de conserver le même cahier des charges que pour l'AADC, mais en construisant nous même la voiture et la surface de test.

En se basant sur les règlements de compétitions autour de véhicules autonomes, nous avons identifié plusieurs axes de travail :

- Le suivi de lignes, en travaillant sur du traitement d'images.
- Comprendre l'intelligence artificielle d'un véhicule autonome, notamment via des réseaux de neurones, pour détecter et comprendre les panneaux.
- La détection d'obstacle et le choix d'une stratégie à adopter si obstacle il y a.
- L'implémentation hardware

A partir de là nous nous sommes fixés 3 objectifs à atteindre avec notre prototype :

- Être capables de suivre une ligne avec notre véhicule
- Déetecter et reconnaître les panneaux les plus communs
- Déetecter un obstacle et éventuellement être capable de l'éviter

4 Organisation du travail

Au début, tout le monde travaillait plus ou moins sur tout, nous avons notamment tous commencé des formations sur les réseaux neuronaux. Nous nous sommes par la suite répartis les tâches de façon plus précise : Loïc était chargé de travailler sur le suivi de lignes. António s'occupait de la détection de panneaux, avec des réseaux de neurones. François travaillait à la détection d'obstacles et au bon respect du code la route (avec un algorithme de décisions). Thomas mettait notamment en place le hardware, avec la prise en main de la Raspberry Pi.

5 Travail effectué

5.1 Architecture algorithmique

Il nous a d'abord fallu théoriser la structure des algorithmes de notre véhicule. L'organisation retenue est exposée en figure 1.

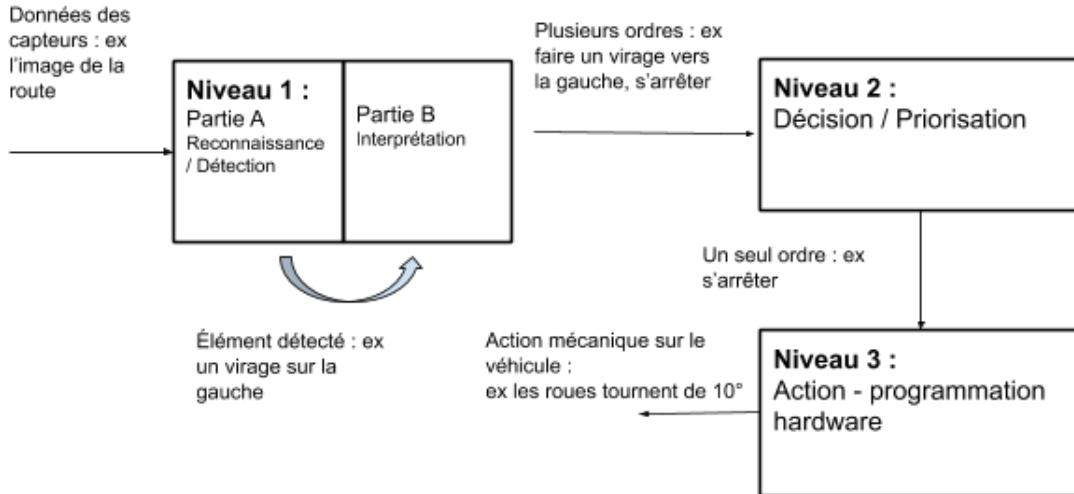


FIGURE 1 – Architecture algorithmique retenue

Les algorithmes de niveau 1 prennent en entrée les données par les capteurs et renvoient un *ordre élémentaire* associé à l'information qu'ils ont observé. Ces algorithmes sont eux-mêmes en deux parties :

- Une première partie de reconnaissance et détection qui consiste à repérer et isoler les *features* d'intérêt
- Une seconde partie qui interprète ces *features* et renvoie un ordre associé.

Par exemple, l'algorithme de suivi de lignes que nous évoquerons par la suite prend en entrée une image, la traite, en extrait les lignes de la route (partie A) puis renvoie l'ordre élémentaire de tourner d'un certain angle afin de suivre le tracé de la route (partie B).

L'algorithme de niveau 2 est l'algorithme de décision. Il prend en entrée plusieurs ordres élémentaires issus des algorithmes de niveau 1 et renvoie en sortie un ordre unique, qui correspond à l'action que devra effectuer la voiture. Pour cela, il faut hiérarchiser l'importance de chaque ordre. Un exemple un peu caricatural serait d'une part la détection d'un panneau de fin de limite de vitesse, qui conduirait à un ordre élémentaire d'accélération, et d'autre part (en simultané) le repérage un piéton sur la chaussée : il faut alors bien sûr ralentir ou l'éviter. Ce second ordre est prioritaire sur le premier : l'algorithme de décision ordonne ainsi à la voiture de l'effectuer plutôt que l'autre. Nous entrerons plus dans les détails dans la section consacrée.

Les algorithmes de niveau 3 sont les algorithmes dits "hardware". Ils prennent en entrée l'ordre donné par l'algorithme de décision et s'assurent que celui-ci est bien converti en une action mécanique de la voiture. Par exemple, après réception d'un ordre de tourner de 10° vers la droite, l'algorithme hardware qui commande les roues va modifier leur vitesse de rotation afin que le virage s'effectue en pratique.

Nous allons à présent présenter chaque algorithme de niveau 1 présenté jusqu'à présent.

5.2 Suivi de lignes

5.2.1 Approche du problème

Tâches à effectuer Nous avons d'abord élaboré un ensemble de tâches pouvant être réalisées. Cette liste était à l'origine basée sur les critères d'évaluation du challenge Audi, mais a évolué lors de l'étape de reformulation des objectifs. Nous avons finalement abouti à la liste suivante :

- Déetecter une ligne bien nette sur une photo
- Déterminer l'ordre de direction correspondant (moyenné temporellement pour plus de fluidité)
- Déetecter une ligne sur la caméra embarquée
- Tourner à une intersection
- Reconnaître les tailles des lignes (dépassement, type de route, ...)
- Déetecter les lignes dans des conditions non optimales :
 - Courbes fortes
 - Plusieurs voies, zébras, ligne continue et discontinue (dépassement que sur une voie), ...
 - Bruit : feuilles, neige, ...
 - Ombres
 - Reflets
 - Pente

Caractéristiques attendues du code

- **Modularité** : il nous a semblé nécessaire de pouvoir segmenter le code en modules spécialisés dans une tâche donnée et de minimiser le recouvrement d'information entre ceux-ci.
- D'une part, pour limiter les répercussions de potentielles mauvaises conditions extérieures (luminosité trop forte ou ombres trop marquées par exemple) ou internes (capteurs mal étalonnés, caméra salie, ...).
- D'autre part, cela permet de coder chaque segment dans le langage le plus adapté à la tâche et de l'exécuter sur un hardware approprié. Le code a été écrit en python et en C, et la partie en python sera par exemple bientôt implémentée en C++, afin de diminuer le temps d'exécution des programmes.
- Ensuite, cela rend l'amélioration future des segments plus simple (moins d'interdépendance implique moins de code à faire évoluer).
- Et enfin, cela permet de minimiser les flux d'information interne, très coûteux en temps et surtout en énergie.
- **Indépendance maximale par rapport à la plateforme** : les paramètres de chaque fonction doivent pouvoir être adaptés à tous types de *hardware*, afin de pouvoir passer très aisément d'une plateforme à une autre.

5.2.2 Étude des solutions existantes

Dans un premier temps, nous avons mené une longue recherche bibliographique, tant pour se renseigner sur les solutions existantes et sur la base mathématique sous-jacente que pour nous former sur les moyens informatiques de les implémenter. Il est alors apparu deux grandes familles de solutions :

- une première basée sur du traitement d'image (utilisée par exemple par RALPH[5], AuRoRA[6] et beaucoup d'autres)
- et une autre basée sur des réseaux neuronaux (ALVINN[7], R-CNN[8][9], YOLO[10], ...)

Afin d'y voir plus clair pour nous permettre de choisir entre les deux, nous avons récapitulé les différences dans le tableau suivant :

	<i>image processing</i>	<i>machine learning</i>
avantages	plus rapide, moins lourd plus explicite	plus précis plus facilement généralisable
inconvénients	Nécessite des règles exhaustives temps de développement plus long non généralisable	plus lent nécessite une base de données conséquente

TABLE 1 – tableau comparatif des différentes implémentations possibles

Cependant, les conditions d'entraînement du réseau neuronal implique que les conditions doivent être les mêmes pendant l'apprentissage et lors de la conduite au risque de fausser les entrées. Par exemple, cela nécessite que la caméra soit positionnée toujours exactement de la même manière, ce qui d'une part va à l'encontre de la règle d'indépendance de la plateforme que nous nous étions fixé, et d'autre part implique que la base de données d'entraînement ne peut pas être une base préexistante mais doit être créée par nos soins pour avoir des images issues de la même caméra, avec la même orientation. De plus, même si un réseau neuronal est plus efficace que le traitement d'image après entraînement, nous avons utilisé OpenCV pour les fonctions d'*image processing*, qui sont parallélisées et donc très efficaces sur le hardware que l'on était sensé avoir à notre disposition à l'origine, puisqu'il incluait un GPU. Finalement, nous avons décidé que le manque d'une base de données adéquate était trop handicapant pour pouvoir nous lancer dans l'entraînement d'un réseau neuronal pour la détection de ligne.

5.2.3 Les différentes implémentations existantes

Nous avons identifié là encore deux types implémentations différentes : RALPH [5], et les algorithmes de détection du bord du marquage au sol, en anglais *edge detectors*.

RALPH (Rapidly Adapting Lane Position Handler) : Il s'agit d'un programme, développé par une équipe de chercheurs de l'université de Carnegie-Mellon, qui détermine la courbure de la route en plusieurs étapes :

1. pré-traitement de l'image : transformation en noir et blanc, floutage Gaussien pour réduire le bruit, ...
2. l'algorithme émet une hypothèse sur la courbure de la route, corrélée aux sorties précédentes
3. à partir de cette courbure, on "redresse" l'image prétraité (cf 2)
4. calcul du "taux de redressage" de l'image (cf 3)
5. maximisation dans le voisinage de l'hypothèse initiale

Un des avantages majeurs de RALPH est sa rapidité d'exécution (en cas d'*overflow* de la RAM, il suffit de réduire la taille du voisinage dans lequel on recherche le maximum de redressement). De plus, comme il traite l'image ligne par ligne sans calculs conséquents et indépendamment, l'algorithme est très facilement parallélisable, et donc adapté au traitement rapide par un GPU.

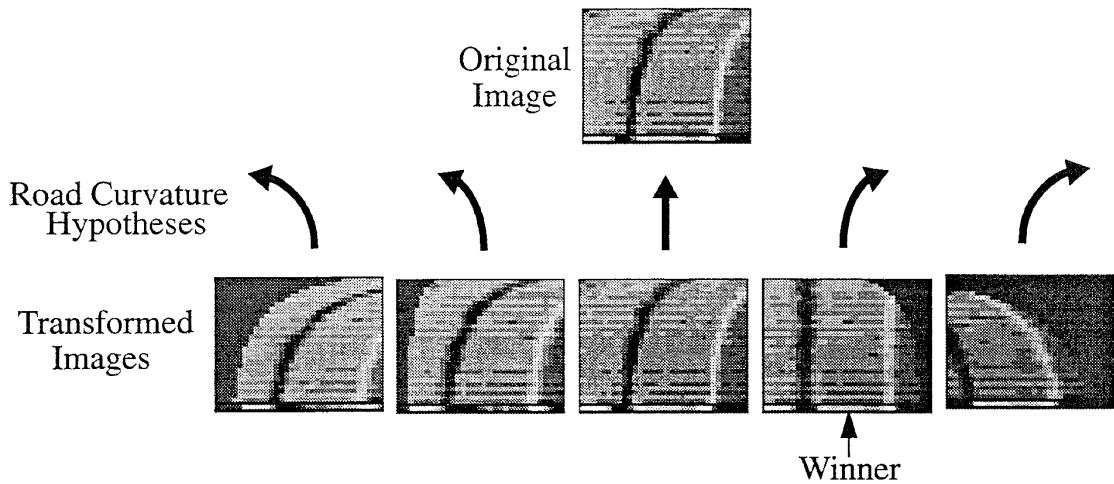


FIGURE 2 – détermination de la courbure de la route en "redressant" l'image ligne par ligne

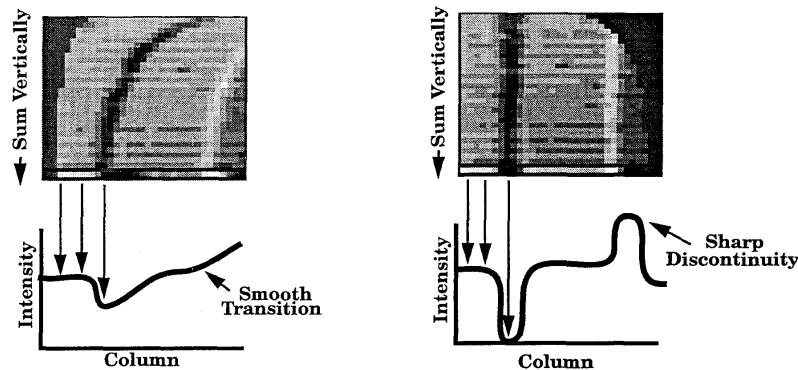


FIGURE 3 – calcul du "taux de redressage" de l'image

L'autre avantage majeur de RALPH est sa capacité à fonctionner sans qu'il n'y ait de lignes nettes : de simples traces parallèles à la trajectoire (usure chaussée, trace dans la neige, ...) suffisent à lui faire comprendre la direction globale de la route. Néanmoins, son implémentation mathématique n'est pas aisée.

edge detectors : Il s'agit des algorithmes les plus fréquemment utilisés, de par leur simplicité. Ce sont des algorithmes basés sur une suite de traitements visant à mettre en évidence des bords en ligne droite, comme celui des lignes blanches matérialisant la chaussée. La suite la plus courante de traitement est la suivante :

1. pré-traitement de l'image : transformation de l'image en noir et blanc, floutage Gaussien pour réduire le bruit, ...
2. transformation de Canny : détecte les bords
 - (a) calcul du gradient de l'intensité de l'image, avec un masque de convolution dans chaque

direction x et y

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

(b) calcul de l'intensité et la direction du gradient avec

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

- (c) *Non-maximum suppression* : enlève les pixels qui ne sont pas considérés comme un bord pour ne garder que de fines lignes.
- (d) *Hysteresis* : pour la dernière étape, on n'utilise non pas un mais deux seuils d'acceptation d'un pixel comme un bord :
 - si la valeur du gradient en ce point est supérieure au seuil haut, le pixel est considéré comme appartenant à un bord
 - si la valeur est inférieure au seuil bas, on le rejette
 - si la valeur est entre les deux, on ne l'accepte que s'il est voisin d'un pixel appartenant à un bord

Cela permet une sensibilité plus fine, et ainsi rejette moins de pixels des bords, qui ont donc plus tendance à être continus.
- 3. Transformation de Hough : il s'agit de détecter les lignes droites dans l'ensemble des bords détectés précédemment.

Pour cela, on écrit l'équation de la droite en coordonnées polaires

$$y = \left(-\frac{\cos \theta}{\sin \theta}\right)x + \left(\frac{r}{\sin \theta}\right), \text{i.e. } r = x \cos \theta + y \sin \theta$$

Pour tout point (x_0, y_0) , la famille des droites qui passent par ce point est : $r_\theta = x_0 \cos \theta + y_0 \sin \theta$, donc chaque couple (r_θ, θ) représente une droite passant par (x_0, y_0)

- 4. pour un point (x_0, y_0) de l'image, on trace la courbe associé à la famille des droites qui passent par ce point. C'est une sinusoïde dans le plan (θ, r) .

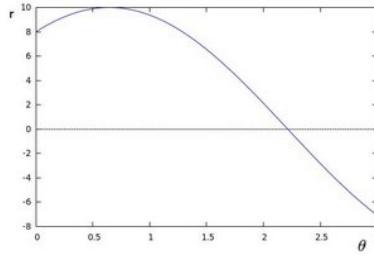


FIGURE 4 – courbe $\theta = f(r)$ pour un point quelconque de l'image

- 5. On répète l'opération pour tous les points catégorisés comme des bords par Canny. Si plusieurs courbes se coupent en un point (θ, r) donné, cela signifie qu'autant de points du bord sont alignés. On ajuste simplement le seuil de votes (i.e. le nombre de droites minimales devant s'intersecter) pour affirmer la présence de cette droite dans l'image.

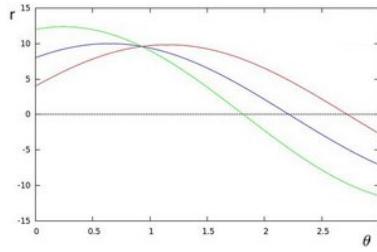


FIGURE 5 – courbe $\theta = f(r)$ pour trois points alignés de l'image

On comprend donc que le traitement de Hough est très conséquent, et nécessite une certaine puissance de calcul. Mais cela reste le traitement le plus répandu et l'un des plus efficaces, car c'est cette combinaison Canny+Hough qui donne les résultats les plus concluants de tous les algorithmes de détection de bords. Néanmoins, il repose sur l'hypothèse que la ligne matérialisant le bord de la route sera tout le temps présente et visible, ce qui n'est pas forcément le cas en réalité mais suffit tant pour l'Audi Cup que pour les conditions que nous avons nous même recréées plus tard.

Nous avons là aussi récapitulé le pour et le contre de chaque approche pour nous permettre de choisir :

	RALPH	Edge Detectors
principe	calcule la déformation pour rendre la ligne droite	Canny + Hough
avantages	Fonctionne même sans lignes.	Exclusivement du traitement d'image
inconvénients	Plus lourd mathématiquement Moins de contrôle sur le résultat	Moins robuste

TABLE 2 – tableau comparatif des différentes implémentations possibles

5.2.4 Solution choisie et explication du fonctionnement

Nous avons donc décidé d'implémenter dans un premier temps la stratégie *edges detection*, car RALPH est très peu documenté et beaucoup plus long à élaborer.

L'algorithme du traitement d'image en soi a été détaillé plus haut. Le pseudocode suivant détaille l'ensemble des traitements auxiliaires nécessaires.

Algorithm 1 Algorithme de suivi de ligne

```
0: for frame in CameraStream do
0:   convert frame into an array image
0:   roiImage  $\leftarrow$  regionOfInterest(image)
0:   colorFilteredImage  $\leftarrow$  filterColors(roiImage)
0:   grayedImage  $\leftarrow$  grayscale(colorFilteredImage)
0:   smoothedImage  $\leftarrow$  gaussianBlur(grayedImage)
0:   edges  $\leftarrow$  Canny(smoothedImage)
0:   lines  $\leftarrow$  HoughLines(edges)
0:   lanesEquations  $\leftarrow$  getLanesEquations(lines)
0:   vanishingPoint  $\leftarrow$  getVanishingPoint(lanesEquations)
0:    $\theta \leftarrow$  getTheta(vanishingPoint)
0: end for=0
```

Exemple d'*outputs* des fonctions ci-dessus.

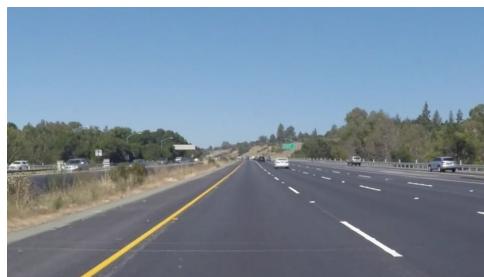


FIGURE 6 – image d'origine

1. `regionOfInterest()` : crée une "région d'intérêt" trapézoïdale autour de la position supposée de la route, afin de réduire la proportion de l'image traitée.

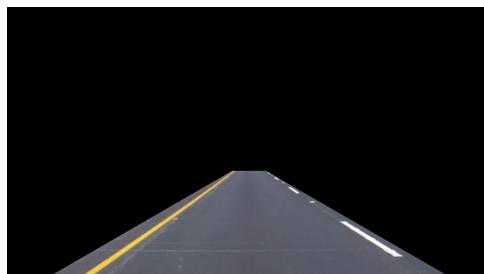


FIGURE 7 – isolement de la région d'intérêt

2. `filterColors()` : Crée un masque de couleur qui ne garde que les pixels blancs et jaunes de la même teinte que le marquage au sol (à étalonner préalablement en fonction de la caméra). Cela enlève tous les bords parasites d'ombres, fissures sur la routes, etc.

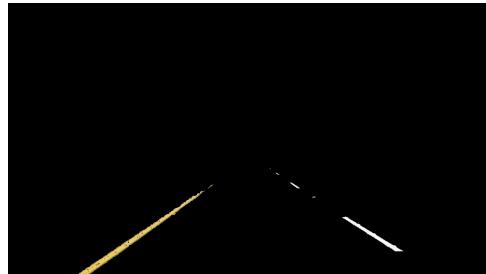


FIGURE 8 – isolement des couleurs correspondant à des lignes

3. `grayscale()` : convertit l'image en noir et blanc (ce qui diminue la taille de l'*input* par 3)

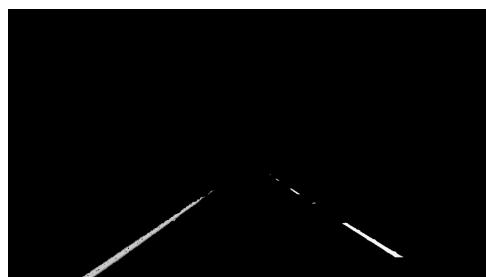


FIGURE 9 – compression du volume de données à traiter

4. `gaussianBlur()` : floute l'image pour rendre les contours plus doux et débruiter l'image



FIGURE 10 – Suppression du bruit

5. `Canny()` : exécute l'algorithme de Canny exposé plus haut, et renvoie un `array` de tous les points appartenant à un bord

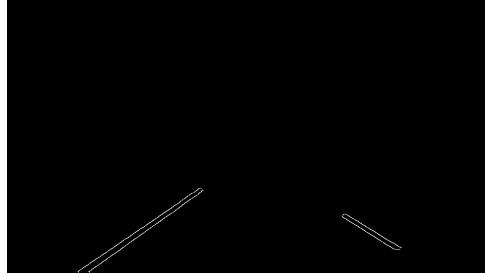


FIGURE 11 – Détection des bords

6. `HoughLines()` : exécute l'algorithme de Canny exposé plus haut, à la différence près que l'on execute `cv2.HoughLinesP()`, une transformée de Hough probabilisée, une variante plus rapide de `cv2.HoughLines()`. L'algorithme ne renvoie donc pas des couples (r, θ) mais un `array` $[x_1, y_1, x_2, y_2]$ contenant les points extrêmes des segments détectés.

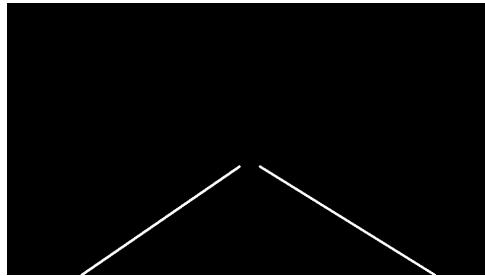


FIGURE 12 – Déduction de la position de la voie

Ensuite, on fait appel à des fonctions détaillées ci-dessous :

On filtre les segments renvoyées par `HoughLinesP` en enlevant les lignes de pente trop faible pour être vraisemblable, et en les rangeant parmi les lignes de droite ou de gauche. Ensuite, on concatène toutes les abscisses d'un côté, et on effectue une régression linéaire avec les y concaténés du même côté. Et on effectue la même chose pour l'autre côté. On en déduit ainsi les équations de chaque ligne, et on peut en déduire la position de la voie.

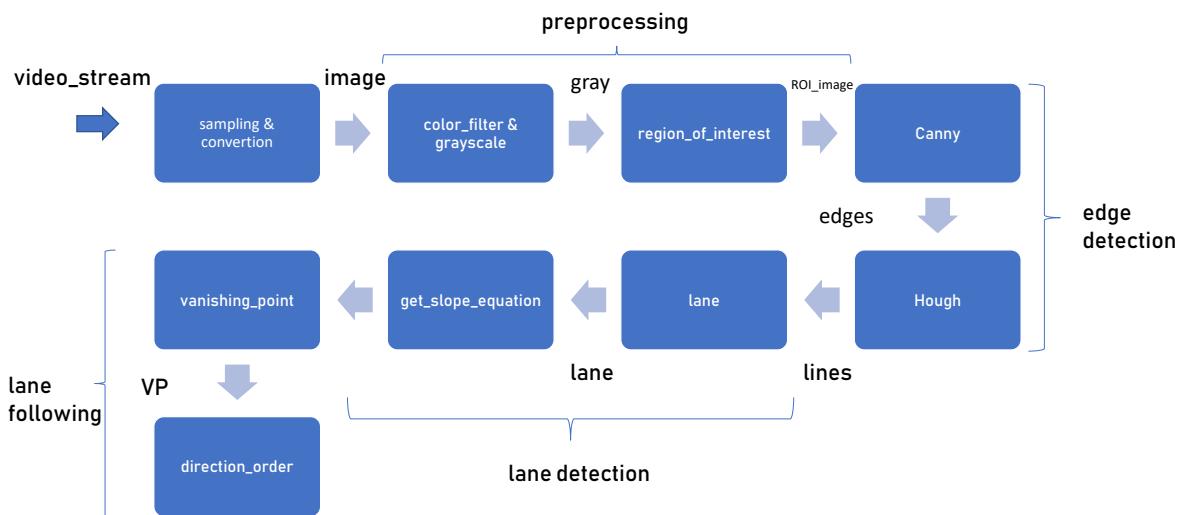
Pour obtenir θ , l'angle de l'ordre de direction, il suffit d'obtenir les coordonnées du point de fuite de la trajectoire en trouvant le point d'intersection des deux lignes. Et on en déduit enfin θ avec un peu de trigonométrie.

Algorithm 2 getLanesEquations(lines)

```

0: newLines = [ ]
0: slopes = [ ]
0: for line in lines do
0:    $[x_1, y_1, x_2, y_2] \leftarrow line$ 
0:   slope  $\leftarrow (y_2 - y_1)/(x_2 - x_1)$ 
0:   if abs(slope) > slopeThreshold then
0:     add slope to slopes
0:     add line to newLines
0:   end if
0: end for
0: rightLines = [ ]
0: leftLines = [ ]
0: for i, line in enumerate(newLines) do
0:    $x_1, y_1, x_2, y_2 \leftarrow line$ 
0:   if slopes[i] > 0 and  $x_1, x_2$  dans la partie droite de l'image then
0:     add line to rightLines
0:   else if slopes[i] < 0 and  $x_1, x_2$  dans la partie gauche de l'image then
0:     add line to rightLines
0:   end if
0: end for
0: effectuer une régression linéaire des  $x_1$  et  $x_2$  de rightLines sur les  $y_1$  et  $y_2$  de rightLines
0: effectuer une régression linéaire des  $x_1$  et  $x_2$  de leftLines sur les  $y_1$  et  $y_2$  de leftLines =0

```



4

FIGURE 13 – Schéma récapitulatif du fonctionnement de l'algorithme

5.2.5 Tâches réalisées

Parmi les objectifs fixés en avril, les suivants ont été atteints :

- Déetecter une ligne bien nette sur une photo
- Déterminer l'ordre de direction correspondant (moyenné pour plus de fluidité)
- Déetecter une ligne sur la caméra embarquée

De plus, afin d'avoir plus de visibilité sur l'influence de chaque paramètre, nous avons développé une interface graphique permettant de renvoyer le résultat de chaque transformation, qui nous a pris beaucoup de temps à mettre au point.

Les autres tâches sont en cours de réflexion et seront implémentées dès la rentrée prochaine. Notamment pour le dépassement, `overtakingHandler`, nous avons choisi l'architecture algorithmique suivante :

1. bouger la ROI sur ligne d'à côté
2. détecter les lignes
3. vérifier que pas d'obstacle sur la voie
4. contrôle angle mort
5. activer le clignotant
6. ajuster θ en fonction de la vitesse et de l'urgence de la manoeuvre

Ce n'est qu'une question de temps avant qu'il ne soit fonctionnel.

5.2.6 Axes d'amélioration

Rendre le code robuste aux erreurs Pour l'instant, dès qu'une erreur survient, par exemple lorsqu'un des programmes renvoie une liste vide car il n'a détecté aucune ligne, l'architecture globale crash. Il s'agit donc de prendre en compte à l'avenir toutes ces exceptions pour permettre au Zumo de continuer d'avancer.

Lignes Le problème majeur de la solution choisie réside dans le fait que face à des courbures trop marquées, et donc des lignes pas droites, HoughLines est incapable d'identifier la voie. Une solution incomplète serait de faire une régression polynomiale au lieu de linéaire sur l'ensemble des segments renvoyés par Hough.

Dans l'implémentation de la solution aussi il s'est avéré que certaines de nos hypothèses étaient trop simplificatrices, comme par exemple la distinction droite/gauche pour une ligne en fonction de sa pente. Quand le rayon de braquage est trop important, cela pose des problèmes (car la pente de droite devient négative par exemple).

Enfin, l'algorithme est peu robuste s'il n'y a plus de ligne. Nous avions négligé ce problème car nous pensions qu'il se poserait surtout en présence de neige ou de feuilles sur la chaussée, mais il s'avère que c'est aussi problématique sur des carrefours, où là trajectoire n'est plus matérialisée. à terme, il s'agira donc d'implémenter un système de secours pour remplacer celui ci lorsqu'il ne détecte pas de ligne. RALPH conviendrait très bien, et cela résoudrait par la même occasion le problème soulevé au paragraphe précédent.

Hough Cette transformée n'étant pas exploitable en l'état, nous avons commencé à développer une solution de remplacement, consistant à moyenne les pixels renvoyés par Canny. Les résultats seraient satisfaisant dans notre cadre simplifié, mais l'algorithme serait encore moins robuste en milieu urbain ou bruité. `filterColors` : cette fonction induit un manque de robustesse face aux reflets, aux bandes sales, ... Un grand soin est à accorder au calibrage du seuil inférieur du filtre, au risque d'éliminer une proportion très grande d'informations, ou au contraire de prendre en compte des *data* n'ayant rien à voir avec la ligne.

Choix des paramètres du système Puisque l'algorithme peut détecter un très grand nombre de lignes sur une image quelconque, les paramètres sont à faire varier très finement. Le problème majeur réside dans le fait que nous n'avons pas vraiment développé de méthodes plus précise que de les adapter au jugé, alors que ceux-ci dépendent de nombreux facteurs extérieurs, comme les conditions d'éclairage par exemple. Il s'agit donc premièrement de développer une phase d'étalonnage automatique préalable pour se ramener aux conditions connues (ajuster l'ISO de la caméra notamment), mais aussi de se documenter sur des moyens de chiffrer l'adaptation des paramètres pour avoir des critères plus précis que l'empirisme.

Limitation du hardware Les programmes ont été choisis pour leurs performances sur GPU, alors que la plateforme n'en n'a pas actuellement. Il faudra donc d'une part trouver un moyen d'effectuer les calculs sur un PU et d'autre part optimiser grandement les temps de calcul. Une piste pour cela est la programmation en C++ au lieu du python, et nous avons déjà commencé à nous former sur ce langage.

En tout cas les résultats obtenus avec des calculs sur la Raspberry Pi ne sont pas encore satisfaisant, notamment lors de l'exécution de la transformée de Hough.

5.3 Obstacles

Une autre problématique très fréquente auquel un véhicule autonome peut faire face est celle de la détection d'un obstacle. Par obstacle, nous entendons tout ce qui gêne la trajectoire de la voiture, donc un objet complètement ou en partie sur la chaussée. Cela peut être une autre voiture, une barrière de péage, ou un piéton. Dans un premier temps, nous expliquons comment l'algorithme parvient à détecter ces obstacles. Dans un second temps, nous nous intéresserons au comportement qu'il faut aborder après détection : arrêt ou évitement. La structure algorithmique est en partie inspirée de [11]

5.3.1 Problématique, technologie employée

Pour détecter les possibles obstacles sur la chaussée, Il existe deux technologies principalement : le LIDAR (télédétection laser) et la vision stéréo, qui consiste à utiliser deux caméras classiques et combiner leur information pour créer une unique image en 3D. [12]

Parmi les avantages du LIDAR, on compte la capacité de détecter des formes à faible visibilité (notamment de nuit), une meilleure robustesse au bruit et des cartes de profondeur plus précises. Mais cette technologie a aussi un coût qui se compte en dizaines de milliers d'euros pour les voitures autonomes actuelles. Cette technologie peut aussi potentiellement représenter un danger pour la santé humaine, les scans des laser pouvant causer des dommages à nos rétines.

La vision stéréo est une technologie beaucoup plus simple, et donc moins coûteuse (seulement deux caméras ordinaires sont nécessaires). Il s'agit de combiner les images provenant de deux caméras excentrées de part et d'autre de l'axe médian de la voiture pour créer une "carte de profondeur" (*depth map*) de l'image. Les profondeurs sont calculées à l'aide de la distance du décalage relatif de chaque objet sur chacune des caméras. Malheureusement, cette technologie a plus de difficulté à identifier les obstacles dans les lieux peu éclairés et demande également un travail plus important en calibrage. C'est néanmoins celle ci qu'on va retenir pour notre voiture, notamment pour sa simplicité.

Afin d'implémenter cette technologie, nous avons dû nous procurer deux caméras Raspberry Pi que nous avons monté sur une Raspberry Compute Module 3 , version plus puissante de la Pi classique qui malheureusement ne savait pas traiter les entrées de deux caméras en simultané. Nous avons donc monté deux caméras sur un support fixe temporaire en carton relié au CM3. Plus de détails sur ce nouveau hardware peuvent être trouvés dans la section consacrée plus bas.

5.3.2 Traitement de l'image

La première étape consiste à calculer la "carte de profondeur" à partir des deux images reçues de chacune des caméra. Pour cela, nous utilisons une fonction de la librairie OpenCV dédiée. L'image en sortie de cette étape est en fausses couleurs, chaque couleur correspondant à une certaine profondeur (et donc à une certaine distance au véhicule). Cette image est fortement bruitée. Ce bruit est très dangereux car il correspond à de faux obstacles, et en les détectant, le véhicule peut adopter un comportement chaotique. Il est donc important de le filtrer. Pour ce faire, on utilise un filtre gaussien qui permet de lisser l'image et en retire les irrégularités.

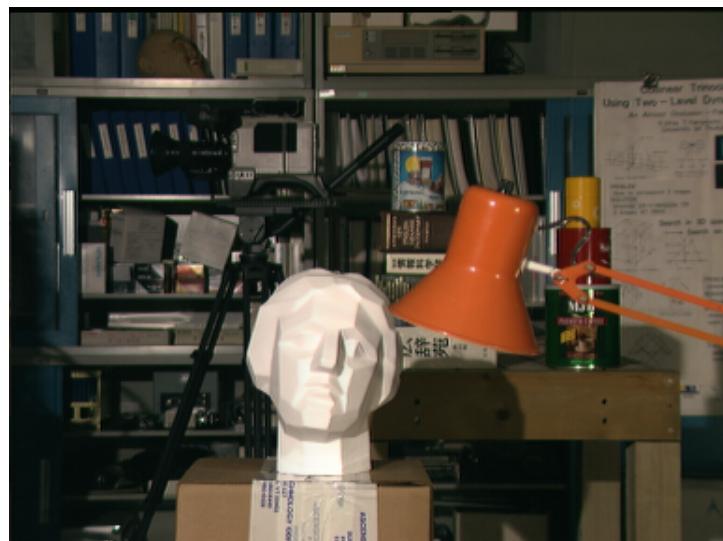


FIGURE 14 – Image de départ. La lampe et la statue sont au premier plan.

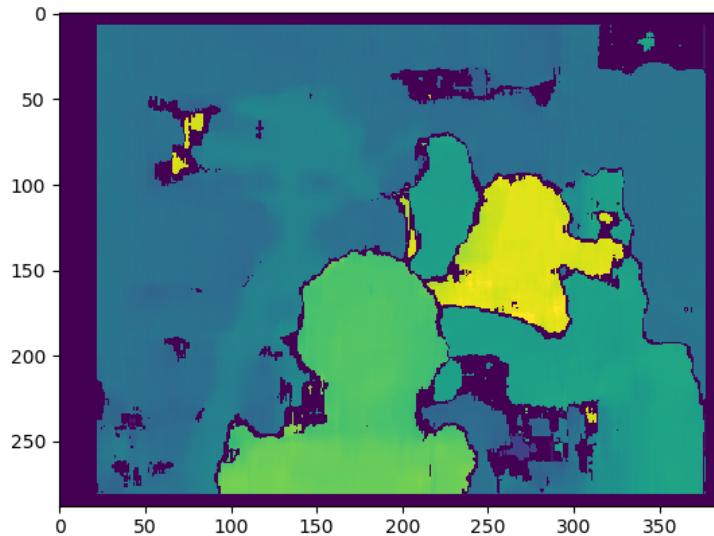


FIGURE 15 – Carte des profondeurs sans traitement

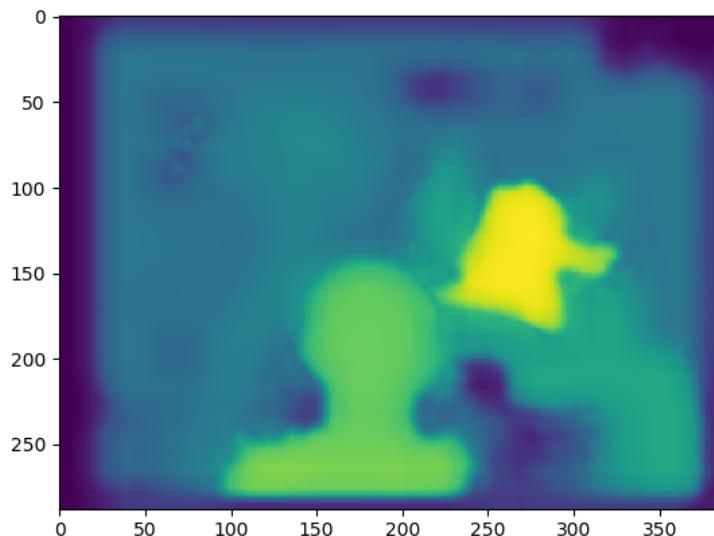


FIGURE 16 – Carte des profondeurs avec filtre gaussien

Note : les tests de vision stéréo ont été faits sur une image test classique.. Nous aurions aussi pu utiliser des images de route plus proches de notre objectif. Néanmoins, la fonction de vision stéréo d'OpenCV demande de renseigner des paramètres à la main qui dépendent de la nature de l'image. N'ayant pas encore fait les test sur hardware, il faudra probablement recalibrer tous ces paramètres pour s'adapter à la résolution et à la qualité des caméras de Raspberry, donc l'origine de l'image n'a pas beaucoup d'importance à ce stade.

5.3.3 Détection d'obstacles

Une fois l'image lissée obtenue, nous passons à la phase de détection d'obstacle proprement dite. Pour cela, nous fenêtrons l'image pour nous intéresser uniquement à la zone en face de la voiture (là où se trouve la chaussée). Par la suite, seule cette zone sera analysée afin de gagner en temps de calcul. Cette zone correspond à un rectangle centré horizontalement mais décentré vers le bas verticalement (la partie supérieure de l'image, qui comprend en général le ciel, ne nous intéresse pas)

Après ce fenêtrage, nous isolons les parties de l'image dont la couleur correspond à une distance suffisamment proche du véhicule pour que l'obstacle soit réellement un danger. De plus, la carte de profondeur est très imprécise pour les objets lointains donc les considérer n'est pas pertinent.

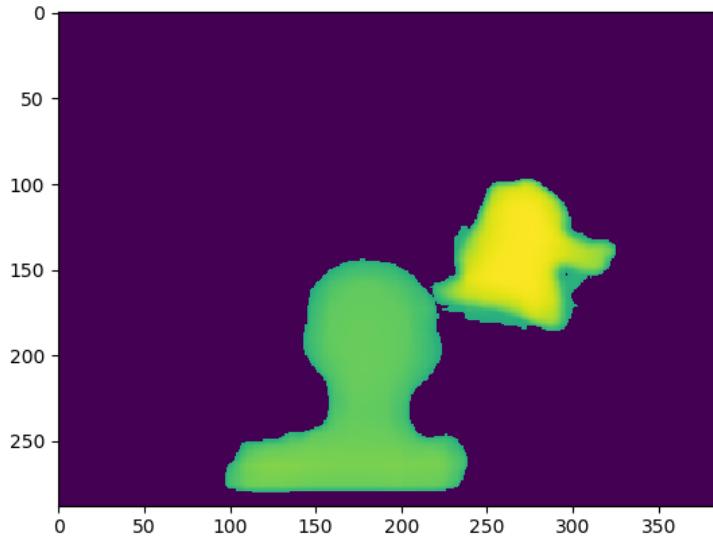


FIGURE 17 – Isolement des obstacles

Ensuite, nous repérons l'obstacle le plus proche de la voiture, c'est celui à traiter en priorité, en utilisant encore une fois la carte des couleurs. L'étape de détection est maintenant achevée et nous passons à la partie qui décidera du comportement de la voiture.

5.3.4 Comportement du véhicule

Il s'agit ensuite de calculer la vitesse relative entre la voiture et l'obstacle. Pour cela, nous utilisons la distance entre l'obstacle et la voiture préalablement obtenue dans une (ou plusieurs) frame(s) précédente(s). Le différentiel de distance et le taux de rafraîchissement de l'image nous permet finalement d'en déduire une vitesse moyenne pour la voiture.

Il reste le problème de pouvoir "suivre" l'obstacle de frame à frame, qui n'a pas été adressé. Pour cela, une solution serait d'utiliser une fonction de "tracking", mais nous ne nous sommes pas encore intéressés à cette problématique en détail.

Une fois cette vitesse relative calculée, il est donc possible d'en déduire deux autres variables qui joueront un rôle dans le comportement de la voiture. On se place à l'instant t et on considère :

- le temps pour atteindre l'obstacle, qui correspond au temps $\delta(t)$ que mettra la voiture pour atteindre l'obstacle à l'instant $t + \delta(t)$. Celui-ci se calcule simplement en divisant la distance à l'obstacle par la vitesse relative. Ce temps est infini si l'obstacle et la voiture se déplacent à la même vitesse. Si l'obstacle se déplace plus vite que la voiture, alors cette valeur n'a pas de sens, et nous renvoyons par convention l'infini également
- le temps pour arriver à la hauteur de l'obstacle, qui correspond au temps $\delta'(t)$ que mettra la voiture pour atteindre la position qu'occupait l'obstacle à l'instant t . Celui-ci se calcule en divisant la distance à l'obstacle par la vitesse instantanée (absolue) de la voiture. Ce temps doit être de deux secondes d'après le code de la route.

On veillera à ne pas confondre ces deux notions. Nous conserverons les notations $\delta(t)$ et $\delta'(t)$ pour la suite.

Piler. La première chose à faire et de s'assurer que nous n'allons pas rentrer dans l'obstacle le plus proche dans un temps critique. Nous comparons alors $\delta(t)$ à une valeur seuil, correspondant à un danger critique d'accident. Si $\delta(t)$ est inférieur à cette valeur, nous envoyons à la voiture l'ordre élémentaire de piler avec l'indice de priorité le plus élevé possible (voir la section sur l'algorithme de décision pour une explication des indices de priorité).

Éviter. Si ce n'est pas le cas, il n'y a aucun danger critique. Mais il est toujours possible que la voiture ait une vitesse importante (mais pas critique) par rapport à l'obstacle. Nous comparons donc $\delta(t)$ à un second seuil, correspondant au seuil en-dessous duquel l'évitement est préférable. Dans ce cas, il faut l'éviter, si possible. Cela est possible à condition que la route ne soit pas complètement bloquée par l'obstacle. Nous y reviendrons dans la partie "trajectoire d'évitement"

Maintenir la distance de sécurité. Les cas restants correspondent à la voiture et l'obstacle qui se rapprochent relativement faiblement (voir s'éloignent). Dans ces cas, il n'y a pas de danger immédiat, mais il faut tout de même s'assurer que $\delta'(t)$ est bien supérieur à deux secondes. Sinon, la voiture roule trop près de l'obstacle et la moindre variation de vitesse de ce dernier peut provoquer un accident. Si $\delta'(t)$ est inférieur à deux secondes, il s'agit de décélérer la voiture (avec une accélération inversement proportionnelle à la valeur initiale de $\delta'(t)$) pour respecter la distance de sécurité.

Note : un traitement similaire serait à faire pour les obstacles situés à l'arrière de la voiture, mais cela dépasse le cadre du projet.

Trajectoire d'évitement La procédure pour éviter les obstacles est complexe et encore en cours de développement. Un premier algorithme simple a été réalisé. Une fois la décision d'évitement initiée, il s'agit ensuite d'examiner les environs gauches de l'obstacle (nous considérons ici uniquement un dépassement par la gauche). Ainsi il s'agit tout d'abord de repérer le bord de l'obstacle puis d'examiner un rectangle situé à sa gauche. Si dans ce rectangle se trouve un autre obstacle à une distance faible, alors le dépassement est impossible : celui-ci est annulé et la voiture freine à la place. En revanche, si aucun autre obstacle n'est détecté dans ce rectangle, la voiture peut initier une manœuvre d'évitement.

Calcul de la trajectoire Voir figure ci-dessous. On peut maintenant facilement calculer la position horizontale cible de la voiture avec la connaissance de sa largeur, et nous connaissons sa distance à l'obstacle. On peut donc en déduire l'angle α de la nouvelle trajectoire de la voiture. Il faudra donc que la voiture se déporte de cet angle dans la voie de gauche pour parvenir à éviter l'obstacle.

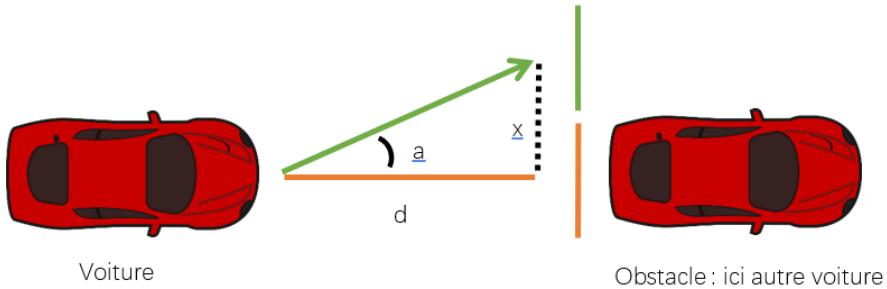


FIGURE 18 – Trajectoire de dépassement du véhicule

En général, ce n'est en fait pas aussi simple car la route peut être sinuuse. Il s'agira donc de sommer cet angle d'évitement avec l'angle de sortie de l'algorithme de suivi de lignes, afin de simplement déporter la voiture par rapport à la trajectoire suivie sans dépassement.

5.3.5 Problématiques restantes

Il subsiste des problèmes non traités : prendre en charge le tracking d'obstacles permettrait de réduire les temps de calcul et de savoir lorsqu'un obstacle disparaît du champ de vision. Dans ce cas, il faudrait annuler les manœuvres en cours. De plus, la manœuvre d'évitement est particulièrement complexe. Peut-on dépasser une seconde fois alors qu'une manœuvre de dépassement est déjà en cours ? Quand revenir sur sa voie ? Nous avons des pistes de réponses à ces questions, mais il serait inutile de se projeter trop loin sans aucune implémentation sur le hardware. L'étape suivante reste donc de faire marcher ce programme (ou une version simplifiée) sur le véhicule et observer les premiers résultats.

Mais pour cela, nous avons besoin d'un hardware adapté capable de supporter la lecture de deux images en simultané : le raspberry CM3 (voir partie hardware)

5.4 Détection et reconnaissance de panneaux

L'algorithme de reconnaissance des panneaux procède en deux grandes étapes qui permettent de diviser la tâche assez complexe de trouver et comprendre un panneau en plusieurs tâches simples, qui s'effectuent rapidement plutôt que d'appliquer un algorithme complexe qui sera plus lent. Ce programme a été codé en Python à l'aide des modules numpy et OpenCV. J'ai choisi de reconnaître les panneaux dont la forme et la couleur sont assez variés mais qui restent assez communs sur les routes.

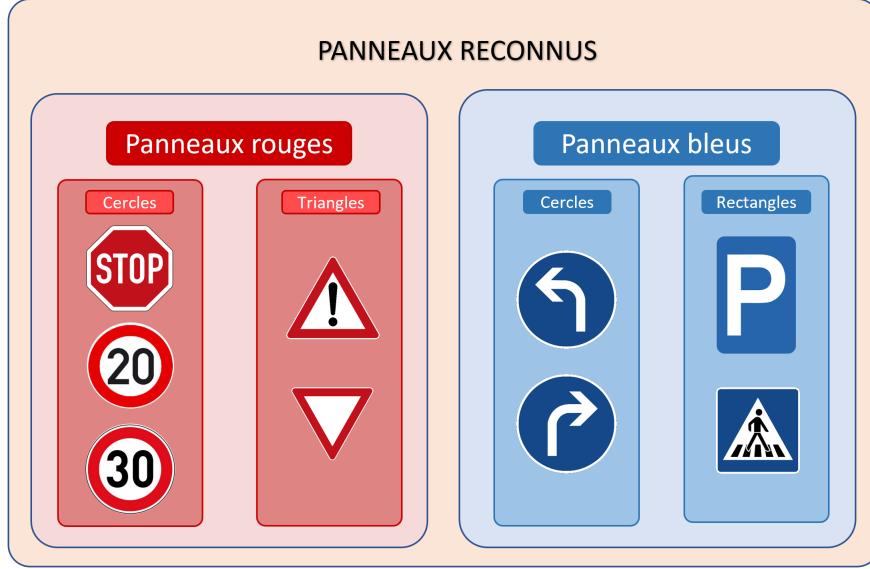


FIGURE 19 – Schéma du fonctionnement de l'algorithme

5.4.1 Detection des formes

Cet algorithme est inspiré des travaux d'Abderrahim SALHI, Brahim MINAOUI et Mohamed FAKIRA [13]. A partir d'une image brute enregistrée par la caméra avant de la voiture, on procède d'abord à une détection des polygones rouges et bleus. Afin de détecter ces polygones, l'image subit une série de traitements qui permettent de mettre en évidence les contours pour en extraire ceux qui s'approchent le plus de polygones.

En effet, on doit extraire les informations importantes de l'image avant de les traiter. On convertit la matrice donnant les couleurs RGB en matrice de couleurs HSV. Ce codage des couleurs permet d'avoir une image dont les couleurs peuvent être filtrées sans dépendre de la luminosité ou la saturation de l'image. Ensuite, à partir d'une méthode de thresholding, on extrait de l'image les composantes bleue et rouge appelées masques bleu et rouge. Ensuite comme le threshold renvoie une image donnant la présence ou l'absence d'une composante, les contours peuvent être peut définis à cause du bruit de l'image brute. On applique alors un filtre Gaussien qui permet de gommer les irrégularités et lisser l'image. On peut alors appliquer l'algorithme de Canny pour détecter les bords de chaque masque. A partir de ces deux masques de bords, on prend les contours codés comme des listes de points formant chaque contour. On peut alors appliquer l'algorithme de Douglas-Peucker sur cette liste pour approcher les contours par des polygones. On applique alors une série de filtres permettant d'éliminer tous les polygones inutiles : polygones dont l'aire ou le périmètre sont trop petits, polygones non convexes, dont un des côtés est beaucoup plus grand que les autres... Avec cette liste de polygones on les trie en fonction du nombre de côtés pour les mettre dans des catégories pré-définies et avoir des réseaux de neurones plus simples à appliquer sur chaque objet. Pour les objets rouges on a deux catégories : triangle si il a 3 côtés ou cercle si il a plus de 6 côtés. Pour les bleus on garde les rectangles (4 côtés) et les cercles comme avant. Après ce classement, on stocke les images rectangulaires et réduites à une taille de 28×28 pixels pour qu'elles soient traitées par le réseau de neurones.



FIGURE 20 – Détection des polygones

5.4.2 Reconnaissance des panneaux

Pour reconnaître les panneaux parmi les objets qui sont détectés par l'algorithme de détection, on utilise un réseau de neurones simple implémenté "from scratch" avec le module numpy.

Implémentation du réseau de neurones Le réseau de neurones est implémenté comme une classe python appelée NNetwork. On peut faire varier le nombre de couches, le nombre de neurones par couches et les fonctions d'activations de chaque couche en entrant en paramètres du réseau la liste du nombre de neurones par couche et la liste des fonctions d'activations par couche. Grâce à ses paramètres on peut en déduire les attributs de l'objet NNetwork qui sont les poids, les vecteurs de chaque couches et les biais, tous implémentés comme listes de matrices, les fonctions d'activations et leur dérivées.

Ensuite, on a implémenté deux méthodes correspondants à la propagation avant et arrière du réseau.

On a des méthodes pour entraîner le réseau de neurones suivant la manière qu'on veut : nombre d'epochs fixé ou jusqu'à ce que le taux de précision soit maximal pour l'ensemble de données de validation. Dans cette deuxième méthode un paramètre peut être réglé pour que le paramètre α de descente du gradient diminue en fonction de la fonction coût.

Enfin, deux méthodes servent à enregistrer les réseaux entraînés et à charger un réseau de neurones déjà entraîné.

Comparaison des performances avec Kheras sur le dataset MNIST Nous avons utilisé la base de données du MNIST [14] (base de données classique de reconnaissance de chiffres pour comparer les performances des outils de Machine Learning). J'ai entraîné deux réseaux de neurones très proches en terme de structure. Tous les deux avaient 784 neurones en entrée (vecteur de l'image), 128 neurones en couche cachée et 10 neurones de couche de sortie permettant de classer les nombres. La fonction de coût utilisée est la fonction d'entropie catégorique, commune pour

les problèmes de classification. La seule différence est que la méthode de descente de gradient du réseau Kéras est une méthode de descente stochastique tandis que la mienne est la méthode simple. On les entraîne sur un dataset de 1000 images. Voici les résultats lorsqu'on teste la précision sur 1000 images :

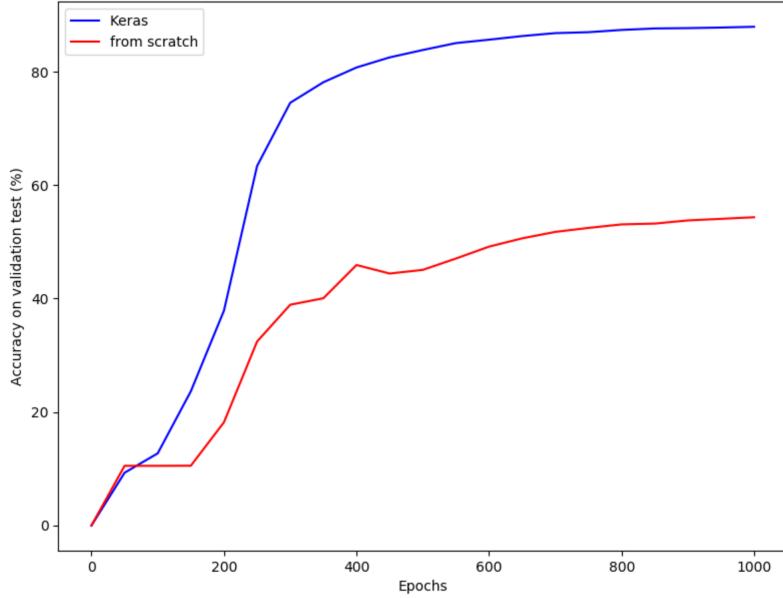


FIGURE 21 – Comparaison de la précision des reconnaissances en fonction du nombre de passages sur les données du dataset MNIST

Le réseau de neurones "from scratch" a tout de même ses limites. Il ne supporte pas bien les grands datasets. Lors du test avec les 60000 images d'entraînement de MNIST, on aboutit pas à des résultats convaincants. De plus, il s'entraîne beaucoup plus lentement que le réseau neuronal Kéras qui est bien mieux optimisé. L'intérêt principal de ce réseau est donc de pouvoir comprendre finement toutes les étapes de la reconnaissance. Mais on verra que les résultats sont assez satisfaisants pour une implémentation simple d'un premier prototype.

Utilisation pour la reconnaissance Dans notre cas, le réseau de neurones possède 3 couches (une couche d'entrées, une sous-couche et une couche de sorties). En entrée, on met un vecteur de taille $28 \times 28 = 784$ correspondant aux pixels du masque blanc de la petite image contenant l'objet. En effet c'est la seule information dont nous avons besoin car elle va distinguer les panneaux qui ont la même forme. En sortie, on a un vecteur de taille 3 ou 4 en fonction du nombre de panneaux dans chaque catégorie sachant qu'une de ces catégories correspond à la catégorie autre pour ne pas reconnaître d'objets pouvant être détectés comme des polygones mais qui ne sont pas des panneaux de la catégorie en question. La décision se fait en prenant le maximum des scores en sortie. La couche intermédiaire possède autour de 6 neurones. Ce réseau de neurones est très simple en entrée il a que l'essentiel des informations grâce à tout le module de détection précédent.



FIGURE 22 – Détection d'un panneau STOP

Données d'entraînement de la reconnaissance La base de données est constituée d'environ 100 images par panneau issues de bases de données connus (German Traffic Sign Benchmark [15] et le Belgian Traffic Sign Dataset [16]), ce qui est assez peu mais permet déjà d'avoir des résultats satisfaisants. Les images ont été séparées en trois parties : les images d'entraînement (60%), de validation (20%) et de test (20%). Les résultats sont prometteurs mais le dataset est trop petit donc beaucoup ne sont pas encore détectés. Pour être exploitées, ces données doivent subir un traitement pour qu'elles soient comprises par le réseau de neurones. Le masque blanc et noir de l'image doit être transformé en vecteur et normalisé de manière à avoir des valeurs entre 0 et 1. Des fonctions s'occupent aussi de mélanger la base de données pour pas que le réseau de neurones s'entraîne sur des données biaisées.

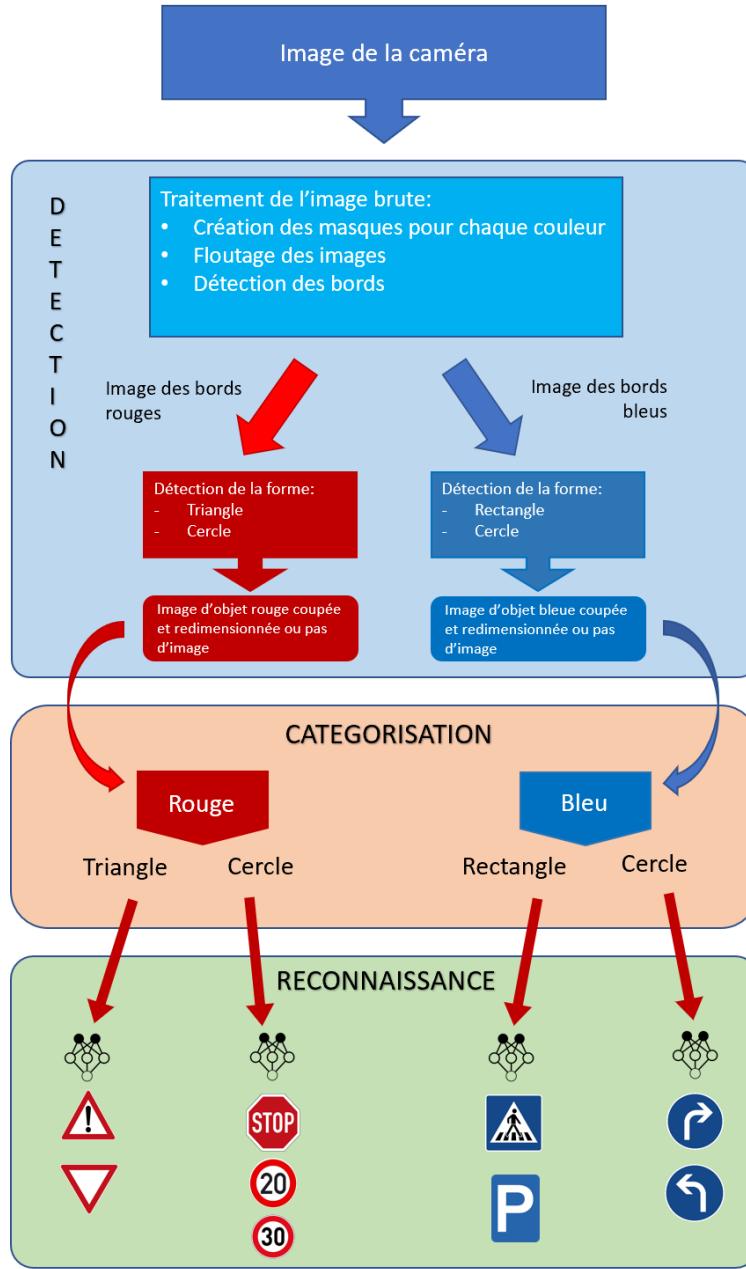


FIGURE 23 – Schéma du fonctionnement de l'algorithme

Reconnaissance en temps réel Grâce à la division en sous-tâches de l'algorithme, le programme est assez performant pour réaliser de la reconnaissance en temps réel sur un ordinateur portable classique. Il suffit d'appliquer l'algorithme à chaque frame de la caméra. On remarque qu'il y a quelques panneaux qui sont identifiés qui ne devraient pas l'être mais leur identification n'est pas régulière et pourrait donc être passée sous silence grâce à un filtrage des résultats résistant au bruit.

5.4.3 Axes d'amélioration

La base de données d'apprentissage doit être augmenté. En effet, les réseaux de neurones entraînés ne sont pas assez robustes pour détecter les panneaux dans le cas où la luminosité n'est pas favorable ou les panneaux ne sont pas bien en face et bien verticaux face à la caméra. Il faudra donc réaliser nos propres saisies d'images pour enrichir la base de données.

Globalement, le code peut être largement optimisé. Le langage Python est très utile pour le codage d'un premier jet de l'algorithme car il rend la programmation assez naturelle. Mais cette facilité de prise en main entraîne une difficulté supplémentaire de compréhension de la part de l'ordinateur. Les algorithmes pourraient donc être implémentés en C++, ce qui accélérerait grandement les calculs. On pourra intégrer un GPU sur notre prototype et faire appel à la programmation parallèle pour accélérer encore plus les calculs si jamais on est à cours de puissance.

Il faudrait aussi utiliser des réseaux de neurones plus performants comme ceux du module Keras car ceux que nous avons implémenté se retrouvent vite en difficulté lorsqu'il s'agit de reconnaître plus de panneaux et lorsque la base de données a beaucoup d'images.

5.5 Algorithme de Décision

Nous allons maintenant expliquer l'algorithme de niveau 2 de notre architecture.

Lorsqu'un humain conduit une voiture, il reçoit beaucoup d'informations qu'il doit traiter en simultané : un feu au rouge, un virage, un véhicule qui nous dépasse sur la droite... Un conducteur saura agir en fonction de la pertinence de chaque information et prendre une décision cohérente qui ne mettra pas en danger autrui. Pour le véhicule autonome, le problème est le même, mais c'est maintenant l'ordinateur qui doit prendre une décision à partir des différentes données et ordres (parfois incompatibles) des capteurs qu'il reçoit. C'est tout l'objectif de l'algorithme détaillé ci-dessous.

5.5.1 Formalisme des ordres

Concrètement, les algorithmes de niveau 1 renverront leurs ordres élémentaires sous la forme de tuples à trois éléments de la forme (Type d'ordre, valeur(s) quantitative(s), priorité).

- Les deux types d'ordre sont : ralentir, accélérer (ordres d'allure) et tourner à droite, tourner à gauche (ordres de direction).
- Le ou les valeurs quantitatives représentent, dans le cas d'un ordre d'allure, un couple (accélération, vitesse cible) et dans le cas d'une ordre de direction, un couple (rayon de braquage, angle cible).
- Enfin, l'indice de priorité sert à hiérarchiser les ordres selon leur importance. Plus celui ci-est bas, plus il aura l'ascendant sur d'autres ordres reçus en simultané. **Cet indice est lié à l'algorithme et non à l'ordre.** Nous y reviendrons.

Par exemple un ordre élémentaire peut être de la forme (A,(-1,2),2). Il signifie alors : ralentir de -1 m/s^2 , jusqu'à atteindre la vitesse cible de 2 m/s, avec une priorité de 2.

Chaque ordre a également une durée de vie. Si cette durée de vie expire alors que l'ordre n'a pas pu être effectué (parce qu'il y avait d'autres ordres plus importants à réaliser pendant ce temps), alors l'ordre est tout simplement annulé. La durée de vie d'un ordre pourrait également être rallongée après sa première émission. Les modalités d'implémentation de cette fonctionnalité n'ont pas encore été décidées et plusieurs approches sont possibles, mais la durée de vie reste une donnée importante pour chaque ordre.

5.5.2 Règles de décision

Nous détaillons maintenant les trois règles permettant à l'algorithme de hiérarchiser les ordres élémentaires pour ne renvoyer qu'un seul ordre principal.

Règle 1 : Compatibilité Un ordre d'allure et un ordre de direction sont compatibles sauf si l'accélération est trop importante et / ou que l'angle du virage est trop importante (valeurs à calibrer). Deux ordres de direction opposés ne sont pas compatibles. Exemple : on reçoit deux ordres (A, (-5,30), 2) et (D, 5°, 1). Les deux ordres sont a priori compatibles donc on effectue les deux en simultané (on ralentit de 2 m/s² et on tourne de 5°).

Règle 2 : Traitement Séquentiel Lorsque deux (ou plus) ordres non compatibles sont actifs simultanément, on applique l'ordre avec l'indice de priorité le plus petit. Une fois cet ordre achevé, on effectue les autres ordres non compatibles séquentiellement par indice de priorité croissant, à condition qu'ils soient toujours actifs (ie vivants, voir la donnée de leur durée de vie) Exemple : on reçoit simultanément (R, (-2, 30), 4) et (non R, 2). On applique le 2e ordre et on ne ralentit pas. Dès que cet ordre n'est plus actif, on passe au premier ordre à condition qu'il soit toujours actif.

Règle 3 : Décélération maximale Lorsque deux (ou plus) ordres de ralentissement ou d'accélération compatibles (tous de même signe) et *d'indice de priorité égaux* sont reçus mais avec des valeurs quantitatives différentes, la priorité est donnée à l'ordre avec la valeur d'accélération la plus basse. SI les indices de priorité sont différents, on applique la règle de priorité. Exemple : on reçoit simultanément (R, (-2,40), 3) et (R, (-5,50), 3). C'est l'ordre avec la valeur la plus basse d'accélération qui est prioritaire, donc ici le 2e. Dans cette situation, le premier ordre est ensuite effectué dès que l'on arrive à 50km/h, à condition qu'il soit toujours actif

Note : Cette règle n'est pas forcément nécessaire avec notre formalisme, mais permet de réduire le nombre d'indice de priorité. En effet les ordres d'allure pourraient à terme provenir de beaucoup d'algorithmes différents et tous les hiérarchiser a priori n'est pas forcément pertinent. Cette règle permet donc d'établir un autre critère dans ce cas.

Ces règles et ce formalisme permettraient alors une décision très efficace et peu coûteuse en calcul (seulement des clauses "if" et pas de boucle utilisée). L'algorithme se base sur l'idée que l'importance qu'on attribue à un ordre élémentaire vient de sa provenance, ce qui est largement vrai : si l'algorithme de détection d'obstacle repère un piéton sur la chaussée et ordonne de piler, nous avons bien affaire à un ordre plus important qu'un ordre demandant d'accélérer pour respecter la distance de sécurité avec la voiture qui nous suit.

5.5.3 Hiérarchisation des ordres

Afin d'utiliser ces règles, il faut d'abord attribuer à chaque ordre élémentaire un indice de priorité. Nous avons commencé cette tâche pour les ordres de décélération / freinage, ici un exemple de classement par ordre d'importance décroissante :

- Indice 1 : détecter un véhicule prioritaire, un obstacle sur la route ;
- Indice 2 : détecter une autre voiture en marche normale ;
- Indice 3 : Stop, cédez le passage, passages piétons, lignes, passages piétons, marquage au sol généralement ;
- Indice 4 : priorité, entrée de ville.

Pour le reste, nous préférons attendre les premiers tests concluants avec le hardware pour nous y avancer plus.

De plus, de telles hiérarchisations sont parfois écrites de manière explicite dans le code de la route. Par exemple, en cas d'ambiguïté entre le marquage au sol (signalisation horizontale) et les panneaux (signalisation verticale), il est indiqué que le marquage au sol est toujours prioritaire. Cette règle a bien sûr été prise en compte dans notre algorithme.

Note : Ce genre d'algorithme est peu documenté dans la littérature. Cela est peut-être dû au fait qu'il existe énormément d'approches différentes possibles, et que c'est un domaine encore en développement, car cela soulève des questions de législation souvent peu claires. Il existe également des approches basées sur des processus de décision markoviens [17]. Mais nous avons préféré garder une approche déterministe ici, quitte à la modifier si besoin une fois l'implémentation sur le hardware.

Cet algorithme renvoie donc un unique ordre aux algorithmes de niveau 3 (hardware) qui se convertira donc en action concrète du véhicule.

5.6 Hardware

L'implémentation hardware fut une tâche longue et complexe. En effet, l'objectif du projet était d'obtenir un prototype viable à la fin de l'année, il était pour cela nécessaire d'avoir un mini véhicule capable de suivre une route, de détecter et de comprendre les panneaux, et de prendre des décisions en cas de détection d'obstacle. Nous avons travaillé notamment sur des raspberry Pi, reliées à une Arduino commandant un ZumoShield. Ce dernier est un moteur relié à des chenilles, produit par Polulu. Il a pour objectif d'être le système de propulsion du véhicule.

5.6.1 Prise en main de la raspberry

Le premier gros défi que nous avons dû relever fut la prise en main de la Raspberry. Nous avons reçu notre première carte juste avant les vacances de Pâques, ce qui nous laissait environ un mois pour mettre en place le hardware. Aucun de nous n'avait déjà travaillé avec ce type de système, c'est pourquoi les débuts furent un peu complexes.

Nous avions plusieurs axes de travail concernant la Raspberry : la mise en place des bibliothèques nécessaires aux différents algorithmes, la mise en place de la communication avec Arduino et la prise en main des Pi Camera.

Implémentation des bibliothèques La découverte d'un système linux est un point important dans le parcours d'un ingénieur. En effet, ce type de système d'exploitation est couramment utilisé en électronique ou en gestion de serveurs, outils qu'un ingénieur peut être amené à utiliser. Malheureusement nous n'étions pas du tout formé à utiliser le Raspbian, une distribution Debian spécialement conçue pour la Raspberry. Par la suite, nous avons dû télécharger des logiciels et des bibliothèques sur la plateforme. En effet, pour fonctionner nos algorithmes nécessitent notamment OpenCV, une bibliothèque graphique fonctionnant sous Python. L'installation de cette bibliothèque fut assez long et fastidieux, nous dûmes nous y reprendre à plusieurs fois avant que cela fonctionne. La première fois, l'installation a planté parce que la mémoire de la Raspberry était pleine !

Nous avons aussi dû installer Arduino ainsi que la bibliothèque ZumoShield pour pouvoir utiliser le support mobile.

Communication avec Arduino L'objectif de l'algorithme du suivi de ligne était de renvoyer un angle de direction θ , et ainsi de faire tourner la voiture de cet angle. Pour cela, il était nécessaire de transmettre l'angle à la carte Arduino, afin que celle-ci puisse agir sur le ZumoShield pour effectuer le virage. Il a donc fallu prendre en main le module Arduino/Python serial, qui permet une communication unilatérale de la Raspberry vers l'Arduino. [18]

Pi Camera L'utilisation des Pi Camera était aussi un point important de notre projet. En effet, ces petites caméras permettaient à notre voiture de "voir", ce qui est un point important lorsqu'on veut suivre une route ! Sur notre première Raspberry, une seule caméra pouvait être branchée, ce qui ne permettait pas de faire de la vision stéréo. Nous avons donc dû créer des scripts python pour pouvoir utiliser la camera, qu'elle soit capable d'envoyer en temps réel un flux d'information à la carte. [19] Cette étape fut assez plaisante, puisqu'elle représentait un avancement concret de notre travail, et permettait d'envisager de nouvelles pistes de recherches. En effet, à partir de ces images, nous avons pu mettre à l'épreuve nos algorithmes. C'est grâce à cela que nous avons pu nous rendre compte que nos programmes nécessitaient d'être retravaillés, afin de pouvoir être utilisés par la raspberry.

5.6.2 Utilisation du ZumoShield

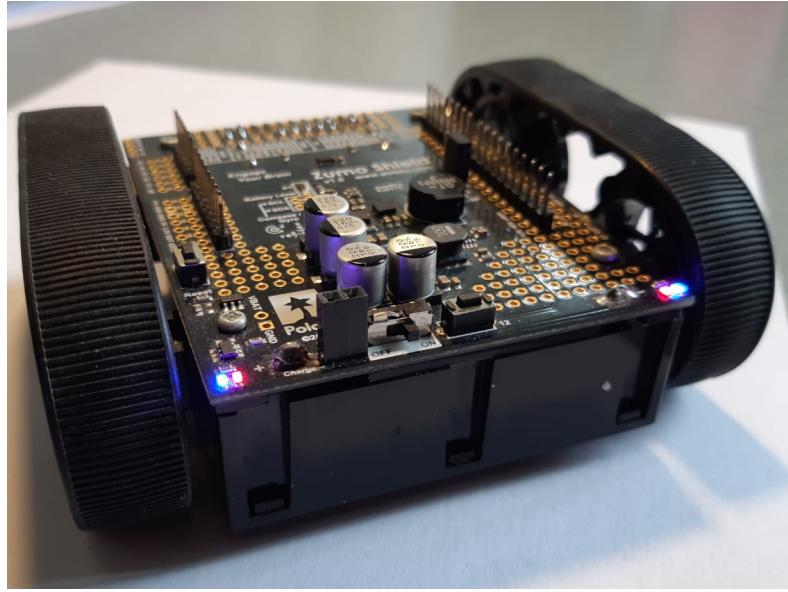


FIGURE 24 – Le ZumoShield utilisé pour notre projet

Nous avons aussi dû travailler sur l'algorithme Arduino qu'il fallait implémenter sur la carte. Celui-ci requérait l'utilisation de la bibliothèque ZumoShield. Pour faire tourner ce module, il a fallu commencer par une réflexion préalable : quels sont les paramètres nécessaires à prendre en compte pour faire tourner un véhicule ? Lorsqu'une personne conduit et veut effectuer un virage, elle tourne le volant *d'un certain angle* et pendant *un certain temps*. Il y a donc deux paramètres à prendre en compte.

Le premier, à savoir l'angle du volant, définit le rayon de braquage de la voiture, c'est-à-dire la distance entre le centre du véhicule et le point autour duquel la voiture vire. Ce paramètre s'implémente en choisissant la différence de vitesse entre les chenilles de la voiture, l'avantage du ZumoShield est alors que le rayon de braquage peut vraiment être entre 0 et $+\infty$. En effet, si les deux chenilles ne tournent pas dans le même sens, le rayon de braquage peut alors être très proche (voire égal) à zéro.

Le second paramètre à prendre en compte est donc le temps durant lequel le véhicule tourne. Une fois le rayon de braquage et la vitesse fixée, c'est directement l'angle de direction renvoyé par l'algorithme de suivi de ligne qui va imposer le temps de rotation. On fixe la convention que si l'angle est positif, c'est qu'il faut tourner vers la droite.

5.6.3 Petits calculs et création de l'algorithme

On va maintenant pouvoir s'intéresser au calcul du temps t_0 qu'il faut tourner en fonction de l'angle de direction θ . Soit v la vitesse du centre de la voiture et r le rayon de braquage désiré.

On a alors $t_0 = \frac{\theta \cdot r}{v}$.

On choisit un rayon de braquage égal à la demi-largeur du ZumoShield, soit 4,2 cm, car il semble assez naturel. Il correspond à l'arrêt d'une chenille de la voiture, et est donc facile à visualiser.

On fixe une vitesse max égale à 4 tours de roues par seconde, soit une vitesse $v = 2 \cdot \pi \cdot r_r \cdot n_{rot} = 50 \text{ cm/s}$, où $r_r = 2 \text{ cm}$ est le rayon des roues et n_{rot} le nombre de tours par seconde.

La vitesse du centre de la voiture est alors de 25 cm/s lorsque les l'une des chenilles est à l'arrêt. On peut alors calculer $t_0 = 0,17 \cdot \theta$.

On va donc implémenter l'algorithme suivant en Arduino, en ne considérant que les angles supérieurs pas trop proches de zéro, pour éviter que la voiture n'avance jamais.

Algorithm 3 Algorithme de trajectoire

```
N = 4
loop
   $\theta$  = Read Serial
  while abs( $\theta$ ) < 0,1 do
     $v_{moteur}$   $\leftarrow$  (N, N)
     $\theta$  = Read Serial
  end while
  if  $\theta$  > 0 then
     $v_{moteur}$   $\leftarrow$  (N, 0)
    wait (0, 17  $\cdot$   $\theta$ )
  else
     $v_{moteur}$   $\leftarrow$  (0, N)
    wait (0, 17  $\cdot$   $\theta$ )
  end if
end loop
=0
```

5.6.4 Raspberry Compute Module 3

L'un des axes auxquels nous nous sommes intéressés est celui de la détection d'obstacles. Dans cette optique, nous utilisons la technologie de vision stéréo (voir section consacrée). Mais pour employer cette technologie, il nous faut un système capable de traiter deux images en simultané, ce qui n'est malheureusement pas le cas de la Raspberry PI 3 sur quelle nous travaillons jusqu'à présent. Nous avons donc, mi-mai, fait l'acquisition d'une seconde Raspberry plus puissante (Compute Module 3) muni d'un "board" auquel nous pouvions brancher deux caméras. Après une longue phase de prise en main et d'installations de modules (phase rendue difficile par le fait que c'est un outil à usage industriel et donc peu documenté pour les particuliers), nous l'avons finalement rendue fonctionnelle il y a peu (deux semaines). Il reste maintenant à fixer les deux caméras sur un socle commun afin que leurs positions relatives restent fixes et de recréer un support pour pouvoir attacher le micro-ordinateur à l'Arduino et au Zumo Shield.

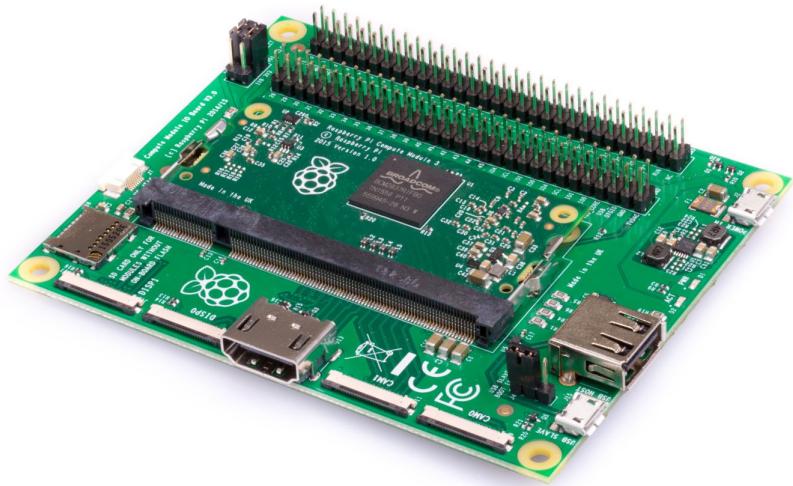


FIGURE 25 – Raspberry compute module 3 avec son IO board

6 Conclusion et ouverture

Notre travail a fournit des résultat très prometteurs pour la suite. En effet, une partie des objectifs que nous nous avions fixé ont été atteint et l'autre partie et déjà partiellement atteinte. Nous avons appris à acquérir des compétences par nous même et à cerner dans la documentation très fournie qui est disponible sur Internet les informations utiles pour notre projet. Nous avons pu expérimenté la mise en pratique d'un projet assez long en équipe et nous avons globalement bien réussi à coordonner nos tâches pour atteindre le but final. Ce projet nous a permis d'approfondir les thématiques scientifiques qui nous tenaient à cœur comme l'informatique, l'intelligence artificielle ou l'électronique. Au-delà de la simple acquisition de compétences, ce travail répond à des enjeux cruciaux pour l'humanité, qui nous touchent tous. Nous avons ressenti, par la découverte des véhicules autonomes, que cette technologie pourra bien révolutionner nos vies futures et il est important de comprendre les machines qui se cachent derrière le véhicule.

Par ailleurs, nous sommes conscients qu'il reste de nombreux points à améliorer si nous voulons mettre en oeuvre un véhicule autonome. Mais nous sommes proches d'atteindre cet objectif. Nous avons pu prendre la mesure de la complexité d'un tel projet et de nombreuses composantes d'un véhicule autonome peuvent encore être développées. Cette grandeur du champ des possibles nous attire énormément et c'est pour cela que nous souhaitons continuer ce projet l'année prochaine. De nombreuses nouvelles idées nous sont déjà venu à l'esprit. Il faudrait concevoir un système d'hardware plus puissant pour augmenter notre puissance de calcul. Il serait intéressant de construire un système GPS ou de reconnaissance simple de sa localisation grâce auquel on pourrait indiquer à quel endroit on souhaite aller. La voiture pourrait aussi se garer de manière autonome. Nous pourrions construire d'autres voitures qui pourraient interagir entre elles ou avec une unité centrale informatique à distance. On peut développer un assistant vocal permettant d'interagir avec la

voiture. Les possibilités sont infinies et peuvent nous faire découvrir encore beaucoup de nouvelles technologies pour approfondir notre formation d'ingénieur.

Références

- [1] Rapport de situation sur la sécurité routière, organisation mondiale de la santé. 2015.
- [2] Jean-Marc Jancovici. Comment évoluent actuellement les émissions de gaz à effet de serre? <https://jancovici.com/changement-climatique/les-ges-et-nous/comment-evoluent-actuellement-les-emissions-de-gaz-a-effet-de-serre/>.
- [3] https://www.lemonde.fr/economie/article/2018/11/02/un-vent-de-froid-souffle-sur-le-secteur-des-voitures-autonomes_5377816_3234.html, 2014.
- [4] <https://www.lesechos-etudes.fr/etude/le-marche-des-vehicules-autonomes/>.
- [5] Dean Pomerleau. Ralph : Rapidly adapting lateral position handler. 1995.
- [6] Dean Pomerleau Mei Chen, Todd Jochem. Aurora : A vision-based roadway departure warning system. 1997.
- [7] Todd Jochem and Dean Pomerleau. Life in the fast lane : The evolution of an adaptive vehicle control system. 1996.
- [8] R. Girshick. Fast r-cnn. 2015.
- [9] Ross Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. 2016.
- [10] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once : Unified, real-time object detection. 2015.
- [11] Aditi R Dakhane and Manish Tembhurkar. Stereo vision for autonomous vehicle routing using raspberry pi. *International Journal of Emerging Technology and Advanced Engineering*, 9001 :98–100, 02 2015.
- [12] <https://www.digitalengineering247.com/article/technologies-driving-autonomous-car/>, 2017.
- [13] Mohamed FAKIR Abderrahim SALHI, Brahim MINAOUI. Robust automatic traffic signs detection using fast polygonal approximation of digital curves. 2014.
- [14] Mnist database. <http://yann.lecun.com/exdb/mnist/>.
- [15] German traffic sign benchmark. <http://benchmark.ini.rub.de/>.
- [16] Belgian traffic sign dataset. <https://btsd.ethz.ch/shareddata/>.
- [17] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving : State-of-the-art and future research directions. *Transportation Research Part C : Emerging Technologies*, 60 :416–442, 11 2015.
- [18] Documentation officielle arduino. <https://www.arduino.cc/reference/en/language/functions/communication/serial/>.
- [19] <http://espace-raspberry-francais.fr/Composants/Utilisation-Camera-sur-Raspberry-Pi-Francais/>