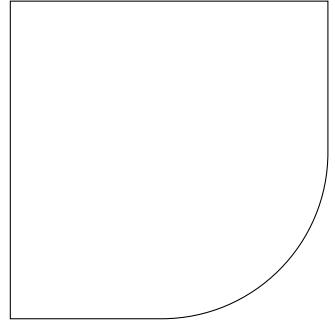


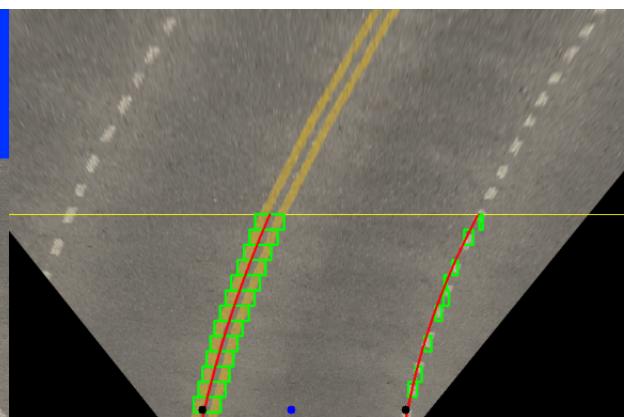
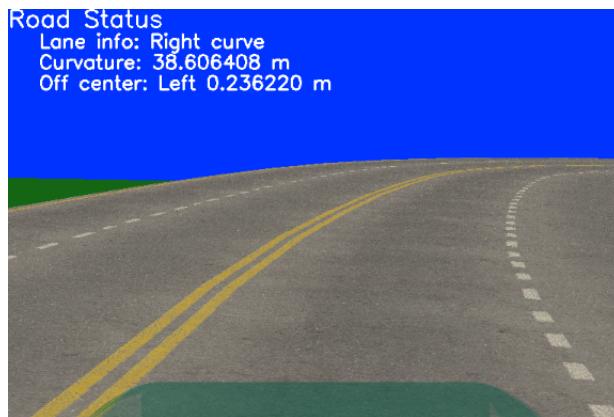
ENSTA Bretagne  
2, rue François Verny  
29806 BREST cedex  
FRANCE  
Tel +33 (0)2 98 34 88  
00  
[www.ensta-bretagne.  
.fr](http://www.ensta-bretagne.fr)



Middleware  
UE 4.1  
24 avril 2020



# Véhicule autonome suiveur de lignes



Colin Baumgard, Ludovic Diguet, Hamid Hacene,  
Corentin Lemoine, Antonin Lizé

# Table des matières

|  |           |
|--|-----------|
| <b>Introduction</b>  | <b>2</b>  |
| 1    Présentation du projet . . . . .                        | 2         |
| 1.1    Problématique . . . . .                               | 2         |
| 1.2    Objectifs . . . . .                                   | 2         |
| 1.3    Cahier des charges . . . . .                          | 3         |
| 2    Choix techniques . . . . .                              | 4         |
| 3    Support matériel . . . . .                              | 6         |
| 3.1    Présentation de la voiture . . . . .                  | 6         |
| 3.2    Adaptation de l'architecture pour le projet . . . . . | 7         |
| 4    Simulation sous V-REP . . . . .                         | 7         |
| 4.1    Modélisation de la voiture et du terrain . . . . .    | 7         |
| 4.2    Modèle cinématique et interface avec ROS . . . . .    | 9         |
| 5    Drivers et nodes de bas-niveau sur la voiture . . . . . | 10        |
| 6    Pipeline de contrôle . . . . .                          | 11        |
| 6.1    Autour du suivi de lignes . . . . .                   | 11        |
| 6.2    Les étapes de la pipeline . . . . .                   | 12        |
| 7    Résultats . . . . .                                     | 17        |
| 8    Perspectives d'améliorations . . . . .                  | 17        |
| <b>Conclusion</b>  | <b>18</b> |
| <b>A Complément Architecture Électronique</b>                | <b>19</b> |

# Introduction

Ce projet s'inscrit dans le cadre des enseignements de l'ENSTA BRETAGNE dispensés aux élèves de deuxième année de la filière *Robotique Autonome* au sein de l'U.E 4.1 : *Middleware*.

Ce rapport présente le travail effectué par l'équipe "La CJC ROS". La répartition du travail au sein du groupe peut être retrouvée sur le dépôt git à l'adresse suivante : [la CJC<sup>1</sup>](#).

## 1 Présentation du projet

### 1.1 Problématique

Nous avons pu acquérir au fil du semestre quatre une connaissance théorique et pratique du *Middleware ROS* (Robot Operating System). Pour rappel, un *Middleware* est un outil qui permet d'échanger des informations entre différentes applications et programmes via des techniques et des messages standardisés. ROS est un *Middleware* développé par l'**OPEN ROBOTICS**.

Le but de ce projet au sein de l'U.E est de maîtriser les concepts fondamentaux de ROS et d'arriver à les implémenter sur un robot. Il s'agit d'appliquer une démarche d'ingénierie afin de proposer des solutions adéquates en prenant en compte les différentes dimensions techniques du problème : *hardware* et *software*.

La problématique de départ se résume ainsi à "rendre un véhicule autonome pour accomplir un tour complet de la piste d'athlétisme de l'école". Les véhicules mis à disposition sont des voitures à l'échelle 1/10<sup>ème</sup> construites par différents groupes au semestre précédent.

### 1.2 Objectifs

L'objectif principal de notre équipe est de répondre à la problématique tout en proposant une approche différente des autres groupes pour apporter une diversité aux contenus des projets.

Ainsi, nous nous sommes fixé ces trois objectifs supplémentaires :

- Proposer une approche réaliste qui s'inspire des voitures autonomes existantes sur le marché ou en développement ;

---

1. [https://github.com/HamidHacene/laCJC\\_ROS](https://github.com/HamidHacene/laCJC_ROS)



(a) Terrain de l'*ENSTA Bretagne*

(b) Voiture utilisée pour le projet

FIGURE 1 – Problématique du projet

- Développer essentiellement en C++ pour garantir la portabilité des algorithmes et l'implémentation sur d'autres structures ;
- Utiliser le plus possible les solutions et algorithmes proposés dans les *packages* ROS afin d'exploiter au maximum les possibilités offertes par ce *Middleware*.

Compte tenu de l'évolution de la pandémie du *Covid-19*, et suite à la fermeture de l'*ENSTA Bretagne*, nous avons revu la problématique et les objectifs du projets. À cause d'une avarie de matériel, l'objectif principal ne peut être atteint : il s'agit désormais de construire une simulation pour valider les différents algorithmes qui seront proposés.

### 1.3 Cahier des charges

Comme la figure 2 le montre ci-dessous, nous avons dans un premier temps cherché à élaborer un cahier des charges fonctionnel afin de hiérarchiser et de définir les contraintes que nous allions rencontrer au cours de ce projet.

L'objectif du cahier des charges comme du diagramme pieuvre n'est pas seulement de documenter notre projet. En effet il met en évidence les éléments clés que nous avons décidé de prioriser durant ce projet.

Rendre autonome la voiture était donc notre priorité maximale et c'est avec cette hiérarchisation des priorités que nous avons choisi de construire ce projet autour d'une tâche en particulier : *le traitement d'image qui permet un suivi de ligne performant*.

| Fonction | Désignation  | Critères  | Niveau | Fléxibilité |
|----------|--|---|--------|-------------|
| FP1      | Permettre que la voiture fasse un tour de circuit en autonomie | Vitesse   | 10km/h | F3: ±5km/h  |
|          |  | Précision max par rapport au milieu de la piste | 15cm   | F0          |
| FC1      | Doit avoir un poids adapté                                     | Poids   | 4kg    | F3          |
| FC2      | Doit permettre la maîtrise de la vitesse                       | Commande PWM                                    |        | F0          |
| FC3      | Ne doit pas consommer trop de batterie                         |   |        | F2          |
| FC4      | Doit gommer les aspérités du terrain                           | Souplesse de la structure et des composants     |        | F2          |

FIGURE 2 – Cahier des Charges Fonctionnel

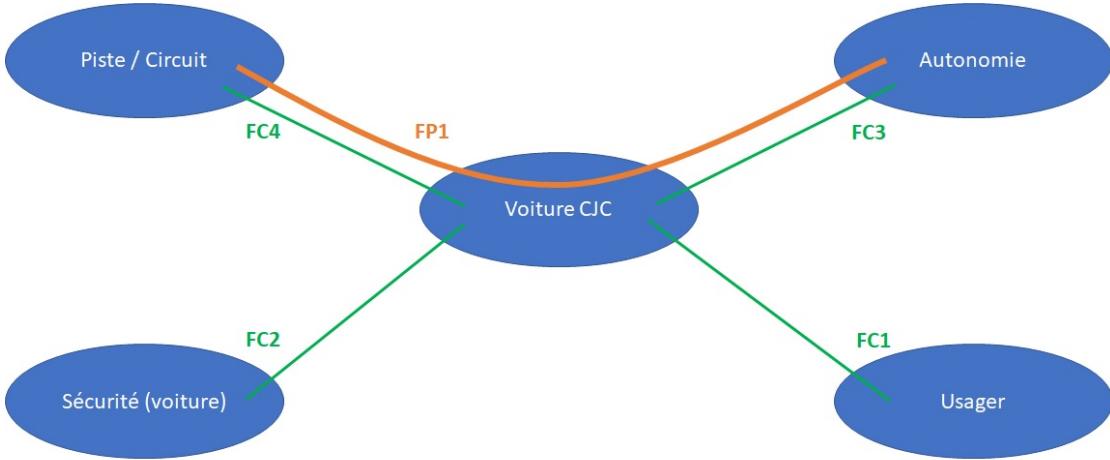


FIGURE 3 – Diagramme pieuvre

## 2 Choix techniques

Afin d'automatiser la voiture, nous avons mené une réflexion sur les différentes solutions matérielles à choisir afin d'atteindre les objectifs.

D'abord, nous avons choisi une *Raspberry Pi 3* comme ordinateur de bord du véhicule. Ce choix est motivé par plusieurs facteurs : espace limité dans la voiture, prix abordable et surtout le fait que cet outil bénéficie d'une large communauté d'utilisateurs, ce qui facilite le support et la documentation.

Afin de répondre à la problématique, nous avons décidé d'équiper la voiture avec les capteurs suivants : centrale inertie, récepteur GNSS, un module caméra d'une *Raspberry Pi*. Malgré le choix de notre équipe de se restreindre, dans un premier temps, à des méthodes d'asservissement visuel pures pour piloter le véhicule, nous avons anticipé les futures étapes du projet et les éventuelles améliorations en intégrant dès le départ tous les autres capteurs dans notre architecture matérielle.

Le module caméra de la *Raspberry Pi* suffit à priori pour réaliser la mission et répondre à la problématique. En effet, la connaissance de la position du véhicule sur une carte n'est pas essentielle pour faire un tour complet de la piste, sous l'hypothèse : "la piste d'essai est circulaire et continue".

Nous avons ainsi construit l'architecture fonctionnelle générale suivante : on peut y voir les différents blocs (contrôle, interfaces et capteurs).

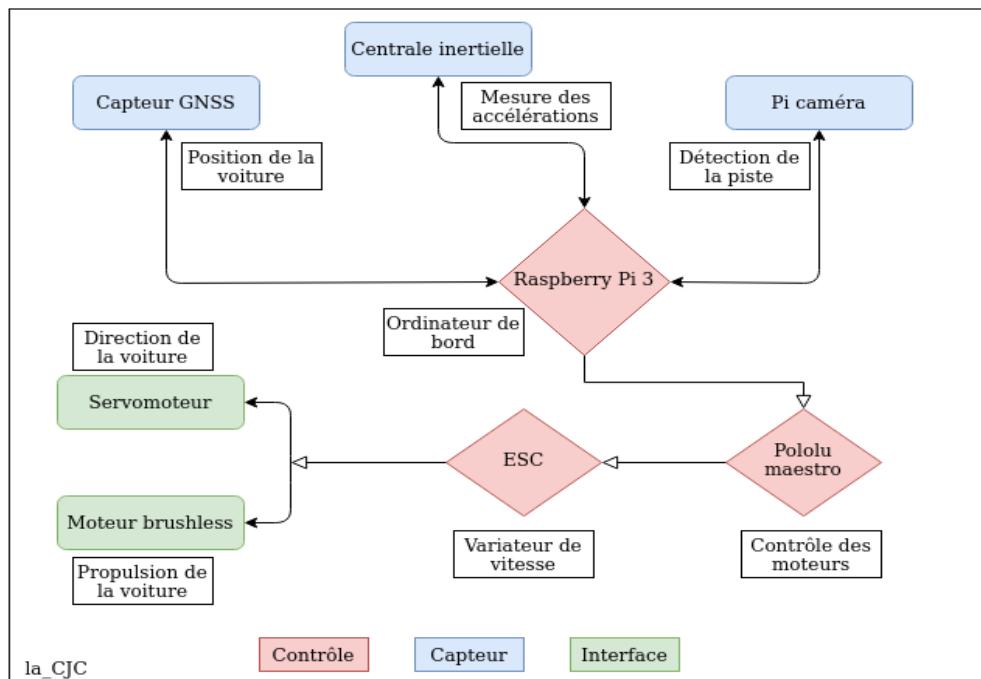


FIGURE 4 – Architecture fonctionnelle générale

*Vous pourrez retrouver en Annexe 3 les Architectures précisant les tensions qui circulent dans la voiture ainsi que les branchements qui permettent de faire circuler informations et alimentation.*

## 3 Support matériel

### 3.1 Présentation de la voiture

Le véhicule utilisé dans ce projet a été construit durant le projet atelier CNC de l'UE 3.4. La première partie de fabrication du véhicule était la conception de quelques pièces mécaniques (support des roulements arrière, support du servomoteur, etc...) sous *Autodesk Inventor* à partir d'un modèle de base fourni par les professeurs de la filière.

La deuxième partie, consistait à fabriquer la voiture en manipulant les outillages du laboratoire de robotique de l'*ENSTA Bretagne*. Les plans des pièces ont été harmonisés entre les différents groupes, mais nous avons décidé de remodeler certaines parties, notamment :

- Modification de la conception des supports de roulements : fabriqués en impression 3D, ils n'avaient pas la bonne dimension ;
- Ajout d'une pièce pour assurer la liaison entre le servomoteur et le bras de direction avant du véhicule.

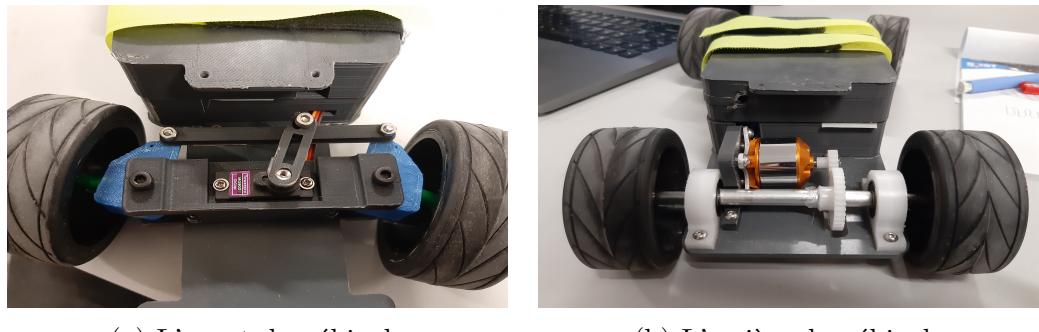


FIGURE 5 – Zoom sur la voiture

Pour la partie électronique, la voiture a été équipée dans un premier temps d'une batterie, d'un moteur *brushless*, d'un servomoteur et d'un récepteur pour la téléopération avec une télécommande. Différents tests ont été ensuite réalisés afin de vérifier la robustesse de la conception. Quelques problèmes ont été ainsi observés : le rapport de réduction était trop faible, et comme la voiture est très légère, il est très difficile de la conduire. Une solution à ce problème serait de rajouter un engrenage supplémentaire afin d'augmenter le rapport de réduction. Il est également envisageable d'augmenter la charge utile de la voiture (l'ajout des éléments nécessaires pour rendre le véhicule peut en effet résoudre ce problème, mais les tests n'ont pas été réalisés).

### 3.2 Adaptation de l'architecture pour le projet

Étant donné les solutions techniques choisies par l'équipe, une adaptation de l'architecture de la voiture s'impose :

- Le traitement d'images et l'asservissement visuel est au cœur de notre solution. C'est pour cela que la conception d'un support caméra fiable est très importante. Le support conçu, permet ainsi de régler la position de la caméra mais également son angle d'inclinaison très rapidement sous *Autodesk Inventor*. Nous avons désormais une solution de prototypage très rapide à fabriquer pour pouvoir mener les différents tests afin de trouver la meilleure position possible pour la caméra.
- Le support du capteur GNSS permet lui de surélever le capteur au dessus de la voiture afin d'éviter au maximum les interférences. Son positionnement est également réfléchi de telle sorte à ne pas trop déplacer le centre de gravité du véhicule vers l'arrière. Ceci est réalisé grâce au désaxage de la tige qui surélève le capteur. Sa proximité avec le support carte au milieu de la voiture, rend les branchements également plus simples.

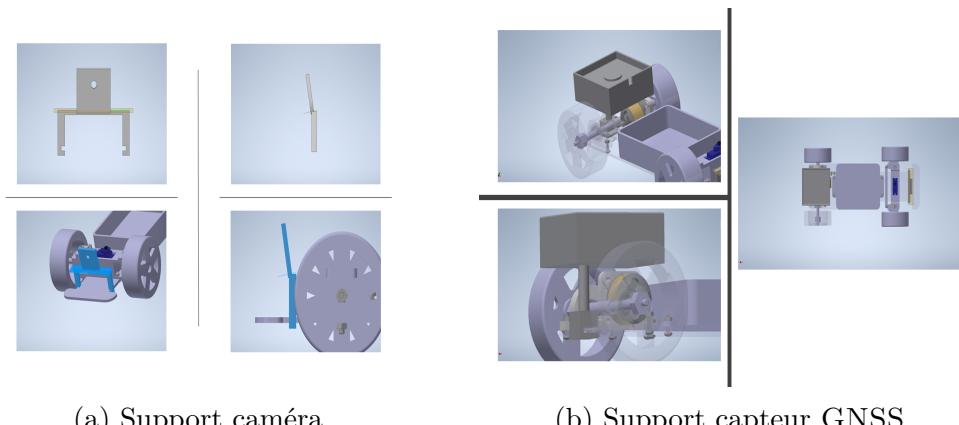


FIGURE 6 – Pièces ajoutées

## 4 Simulation sous V-REP

### 4.1 Modélisation de la voiture et du terrain

Les circonstances actuelles ne nous permettent pas de travailler sur le projet en présentiel. La solution de repli qui a été adoptée est de construire une simulation de tout l'environnement du problème (voiture + terrain) afin de poursuivre le projet. Les outils de simulation permettent en effet de s'abstraire du matériel et de valider des solutions techniques avant de les intégrer en réalité.

La simulation a été construite dans le cours de M.BENOÎT ZERR. Le tutoriel peut être consulté en suivant ce [lien<sup>2</sup>](#).

Le modèle dynamique est construit avec des formes simples (cylindres et parallélépipèdes) afin de réduire le temps de calcul lors de la simulation. Les pièces de la CAO ont été rajoutées par dessus afin d'avoir un aspect visuel qui permet de distinguer les différentes parties.

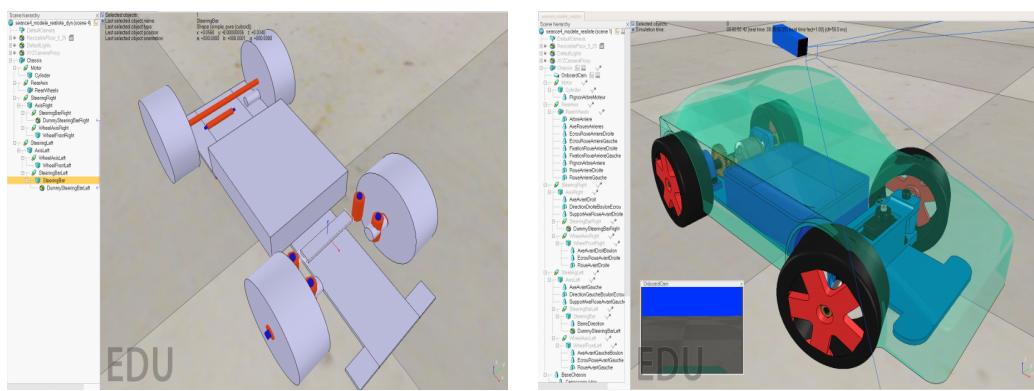


FIGURE 7 – Modélisation de la voiture

Dans le tutoriel précédent, il s'agit également de créer l'environnement dans lequel la voiture évoluera. Nous avons décidé de choisir comme revêtement de la route, une texture qui se rapproche le plus d'une vraie route. En effet, le traitement sur ce genre de texture est difficile compte tenu du bruit très élevé dans l'image : lignes pointillées, lignes doubles, bifurcations et le nombre de voies. Nous sommes partis du principe que si notre solution répond à la problématique sur une vraie route, il serait simple de l'adapter pour un usage sur une piste d'athlétisme.



FIGURE 8 – Texture de la route

---

2. <https://www.ensta-bretagne.fr/zerr/dokuwiki/doku.php?id=vrep:mainv1>

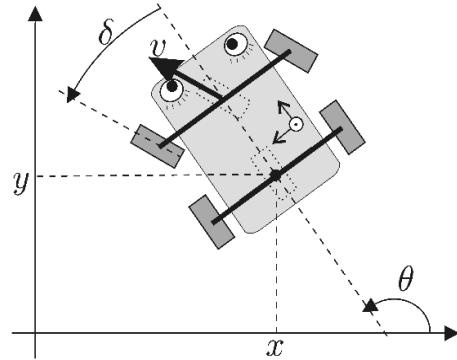
## 4.2 Modèle cinématique et interface avec ROS

Sur la voiture, nous possédons deux commandes : l'accélération des roues arrières (qui sont motrices) et la vitesse de rotation du volant (servomoteur). Nous noterons  $\delta$  l'angle entre les roues avant et l'axe de la voiture, et  $\theta$  l'angle que fait la voiture par rapport à l'horizontale, et  $(x, y)$  les coordonnées du milieu de l'essieu arrière. Les variables d'état de notre système sont constituées par :

- **les coordonnées de position** : les coordonnées  $(x, y)$  du centre de l'essieu arrière, l'orientation  $\theta$  de la voiture, et l'angle  $\delta$  des roues avant ;
- **la coordonnée cinématique  $v$**  : représentant la vitesse du milieu de l'essieu avant (ceci est une approximation pour simplifier les équations : pour notre voiture, la vitesse est appliquée sur l'essieu arrière).

$$\begin{cases} \dot{x} = v \cos(\delta) \cos(\theta) \\ \dot{y} = v \cos(\delta) \sin(\theta) \\ \dot{\theta} = \frac{v \sin(\delta)}{L} \\ \dot{v} = u_1 \\ \dot{\delta} = u_2 \end{cases}$$

(a) Équations d'état



(b) Modèle cinématique

FIGURE 9 – Modélisation cinématique de la voiture

Nous avons ainsi deux variables à contrôler :

- **l'accélération des roues arrières  $u_1$**  : pour faire le lien avec la modélisation dynamique sous V-REP, cette commande doit correspondre à l'accélération des deux roues arrières. Mais pour simplifier le modèle et la simulation, nous allons considérer une vitesse  $v$  constante. Ceci implique donc un changement dans les équations d'états :

$$\begin{cases} \dot{x} = u_1 \cos(\delta) \cos(\theta) \\ \dot{y} = u_1 \cos(\delta) \sin(\theta) \\ \dot{\theta} = \frac{v \sin(\delta)}{L} \\ \dot{\delta} = u_2 \end{cases}$$

- **la vitesse de rotation du volant  $u_2$**  : sur V-REP, cette commande va correspondre à la vitesse de rotation du servomoteur (en réalité, cette commande va contrôler deux liaisons pivots afin de simuler un servomoteur).

Le travail précédent nous permet ainsi de réaliser l'interface entre ROS et V-REP (en langage *lua*). En plus des *topics* relatifs aux deux commandes, nous avons également codé l'interface qui permet de simuler une caméra sur notre voiture. L'image obtenu est ainsi publiée sur un autre *topic* afin d'être traitée dans un *node* ROS.

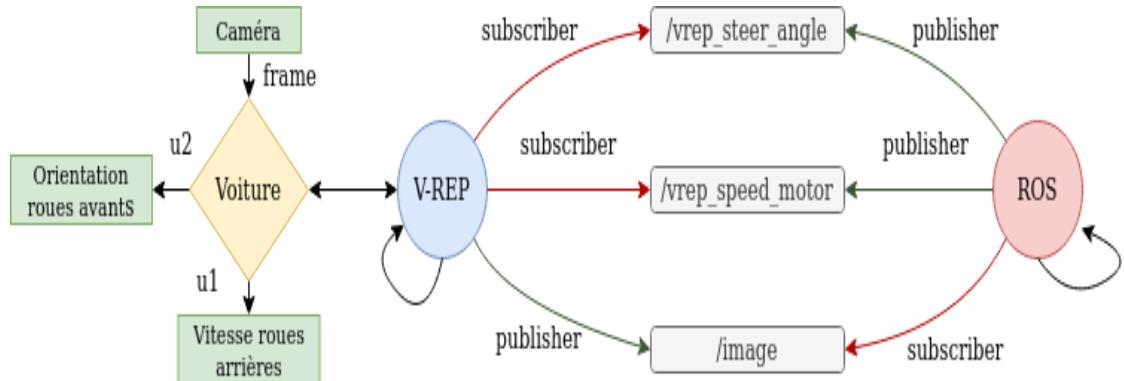


FIGURE 10 – Interface entre ROS et V-REP

## 5 Drivers et nodes de bas-niveau sur la voiture

En adéquation avec la simulation sous V-REP, nous avons développé des nodes ROS pour interfaçer les différents capteurs et actionneurs.

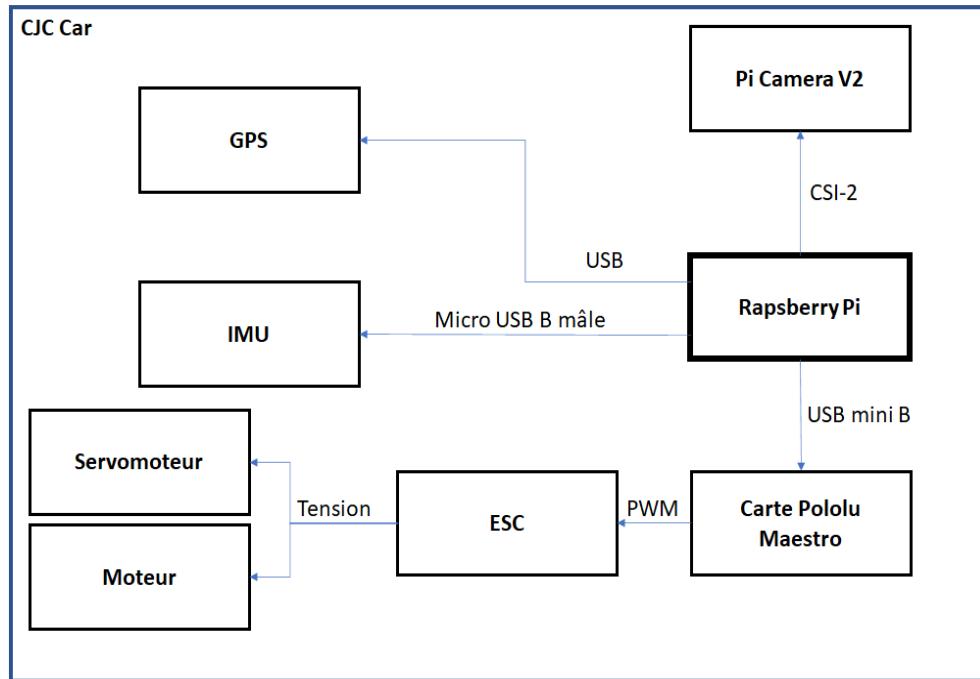


FIGURE 11 – Architecture fonctionnelle d'information

Les drivers sont implémentés en utilisant le plus possible les nodes développés par la communauté ROS. Le GNSS et l'IMU sont représentés mais à notre stade d'avance de ce projet, ils ne sont pas utilisés. Le package *cv\_camera* nous permet de publier une image sur un topic ROS issue de la caméra de la Raspberry pi. Pour l'envoi de signaux PWM vers l'ESC et le servo de direction, nous avons développé un node Python utilisant une bibliothèque pour contrôler la carte Maestro via la connexion série USB. Ce dernier node s'abonne aux mêmes topics que notre simulation V-REP pour permettre à terme de faire du *Hardware in the loop*.

## 6 Pipeline de contrôle

Désormais, nous avons tous les éléments nécessaires afin de rendre la voiture autonome. Dans cette partie, nous allons détailler la *pipeline de contrôle* : d'une *frame* issue de la caméra jusqu'au calcul de  $u_1$  et  $u_2$ .

### 6.1 Autour du suivi de lignes

Depuis quelques années, de nombreux chercheurs travaillent sur des projets concernant l'aide à la conduite automobile par vision artificielle. Beaucoup de projets ont vu le jour et les techniques développées sont très variées.

Deux approches peuvent répondre à notre problématique :

- Positionner la voiture sur une ligne de la piste et faire en sorte de suivre cette ligne en la gardant toujours au milieu sous le châssis de la voiture. Cette approche peut s'avérer efficace pour faire uniquement un suivi de ligne. Cependant, nous avons vu quelques inconvénients à utiliser cette méthode : d'abord la largeur de la ligne blanche de la piste mesure un peu plus de la moitié de la largeur du châssis de la voiture (5cm). Ceci pourrait poser un soucis si la focale de la caméra n'est pas suffisamment petite (on aurait une image presque toute blanche, ce qui compliquerait la détection et le contrôle). De plus, les algorithmes à développer ne serviront pas à d'autres situation (conduite en milieu urbain par exemple). En revanche, cette méthode fonctionne très bien en simulation.
- Positionner la voiture sur une voie de la route et détecter les deux lignes délimitant cette voie. Cette méthode se rapproche un peu plus de ce qui se fait dans l'industrie automobile en terme de conduite autonome (cf voiture autonome *Tesla*). Nous avons donc opté pour cette deuxième solution.

## 6.2 Les étapes de la pipeline

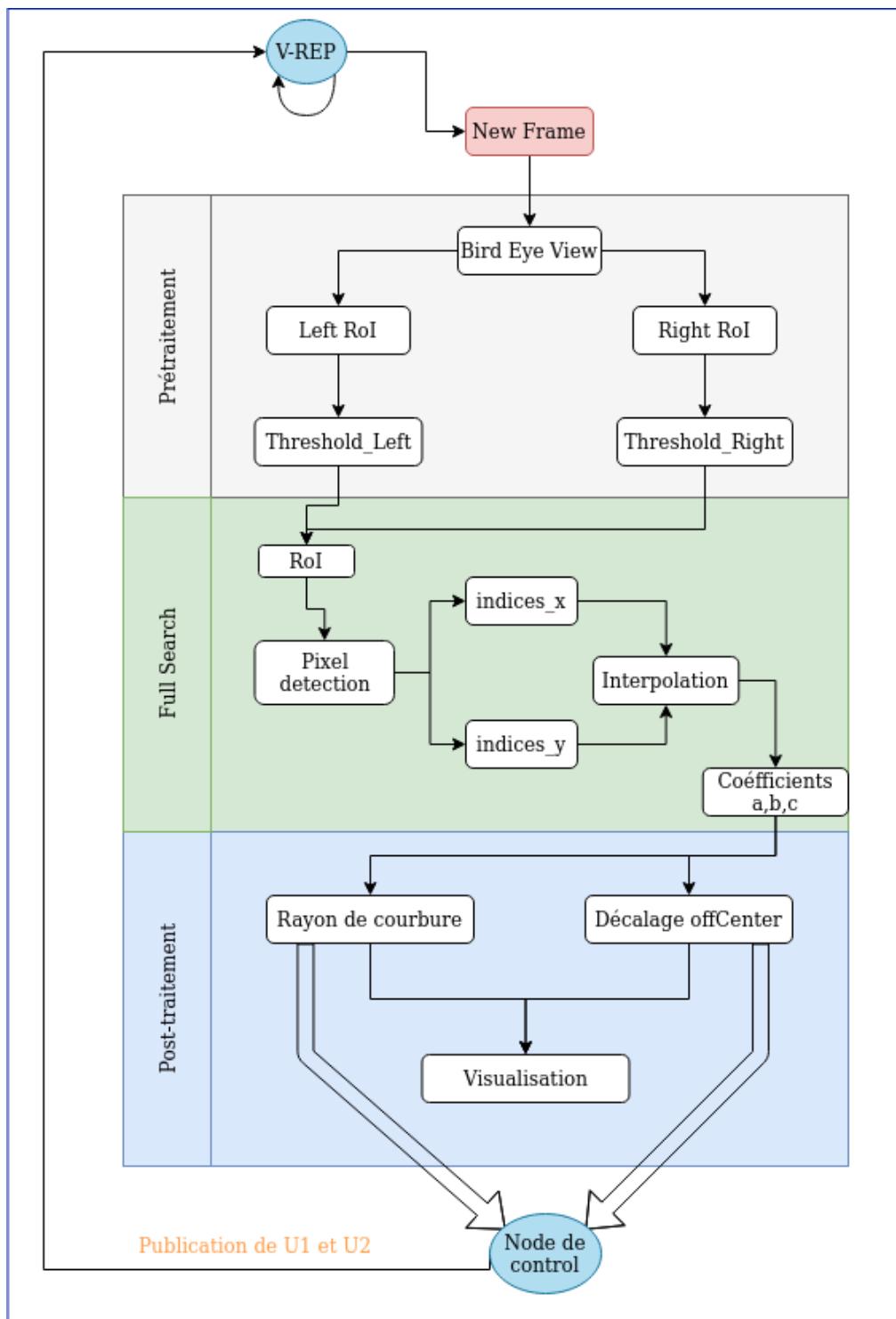


FIGURE 12 – Pipeline de contrôle

## Prétraitement

1. **Bird Eye View** : cette vue permet d'opérer une transformation de l'image source afin d'avoir une perspective de la route. Elle règle le problème des lignes parallèles qui se coupent à l'horizon par exemple.

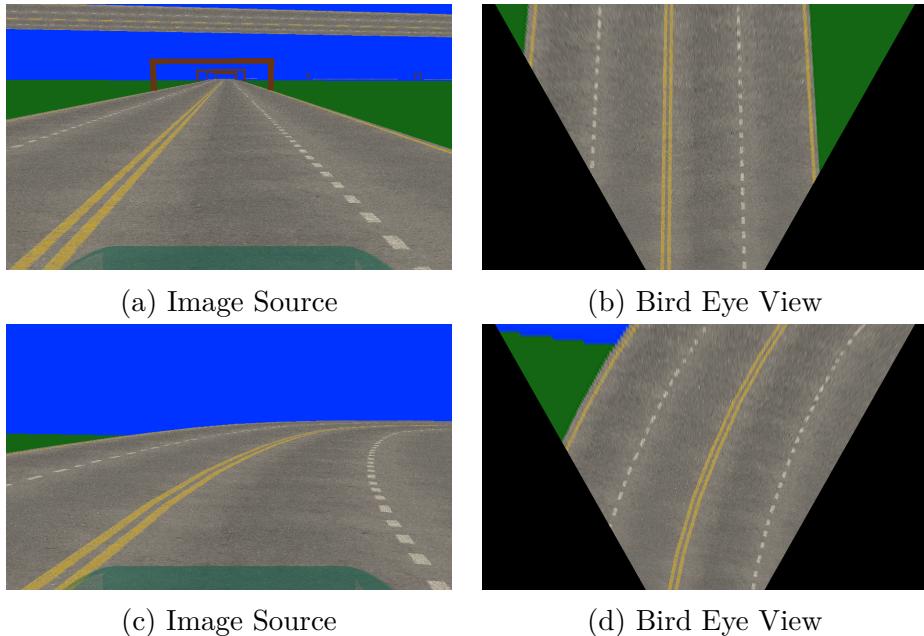


FIGURE 13 – Bird Eye View

2. **Extraction des régions d'intérêt** : cette étape permet de diviser l'image issue de l'étape précédente en deux régions : gauche et droite.
3. **Filtrage** : on applique deux filtres différents pour les deux régions afin d'extraire les *pixels* qui nous intéressent. En effet, la ligne gauche de la voie est complètement différente de celle de droite (ligne double jaune, ligne en pointillées blanches).

À la fin de cette étape, nous avons filtré l'image de la route et elle est donc prête pour l'étape suivante.

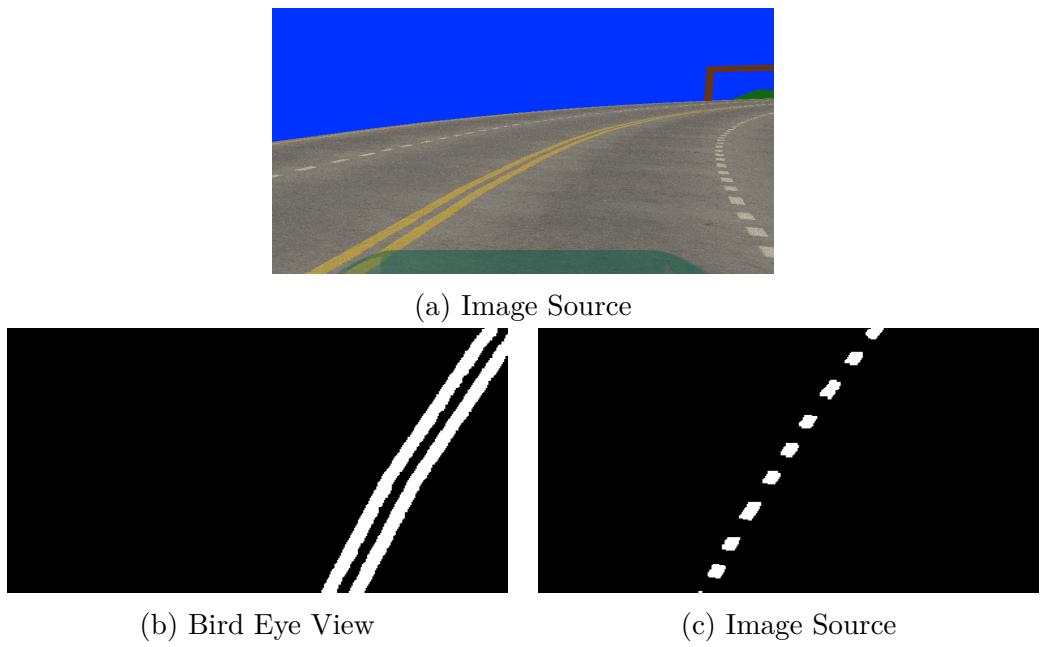


FIGURE 14 – Extraction des ROI

### **Full Search**

Cette algorithme a pour objectif de détecter les *pixels* qui constituent chaque ligne.

1. **Détection** : on divise l'image issue de la précédente étape en  $n$  sous-fenêtres. Ensuite, on parcourt avec deux indices l'image (indices sur  $x$ ) : un qui part de 0 vers la droite, et l'autre de la droite (largeur de l'image) vers la gauche. À chaque itération, on retient les deux indices uniquement s'ils sont à l'intérieur de la fenêtre (limitée par deux indices sur  $y$ ), auquel cas on calcule la somme des *pixels* blancs sur les colonnes données par chacun de ces indices. Si cette somme est supérieure à un seuil, on arrête d'itérer. À la fin de la boucle, on obtient un encadrement de notre ligne avec deux listes d'indices.
2. **Interpolation** : avec les deux listes précédentes, on effectue une interpolation polynomiale pour approximer la ligne avec un polynôme du second degré. On obtient ainsi les coefficients de ce polynôme.

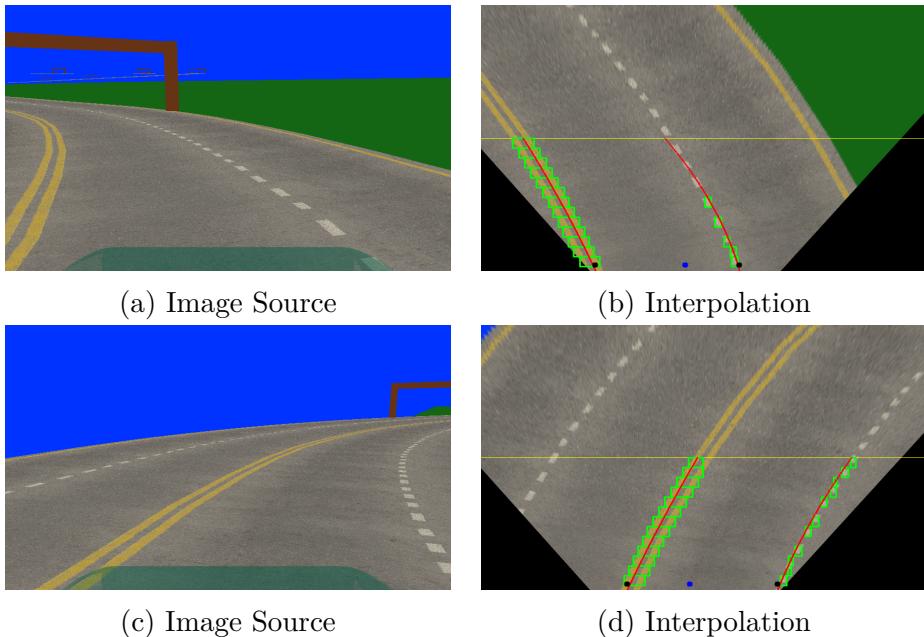
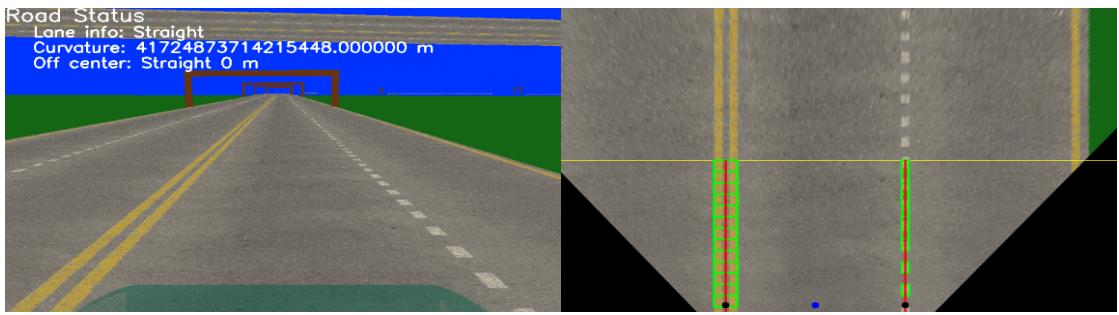


FIGURE 15 – Détection de la voie

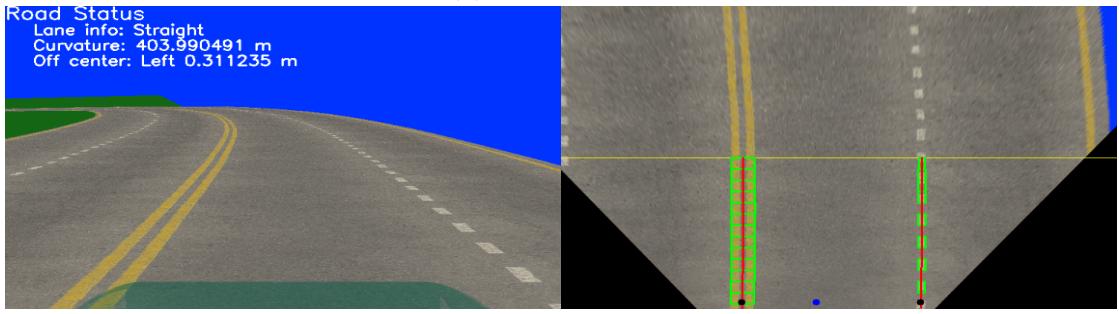
### Post-traitement

Cette étape permet d'extraire les informations qui nous permettront de contrôler la voiture. Grâce à l'étape précédente et à l'interpolation polynomiale, nous pouvons proposer une nouvelle méthode pour contrôler la voiture :

1. **Décalage par rapport au centre de la voie** : on fait un calcul supplémentaire pour déterminer la position de la voiture dans l'image (en exploitant la présence du capot dans l'image brute notamment). Une fois cette position connue, on calcule le décalage de la voiture par rapport au centre de la voie : en fonction de la détection de la ligne de gauche et de droite, on arrive à calculer avec précision cette donnée. Le décalage par rapport au centre est ensuite utilisé pour alimenter le *node* de contrôle qui calcule l'angle à appliquer sur les roues afin de rester au milieu de la voie.
2. **Rayon de courbure** : nous calculons également le rayon de courbure de la voie. Ceci nous permet de savoir dans quel sens on tourne mais également d'adapter la vitesse de la voiture. Plus le rayon de courbure devient petit, plus le virage est serré. Il faut donc réduire la vitesse de la voiture. Pour l'instant, cette donnée n'est pas exploitée dans la suite. En effet, comme il s'agit d'un calcul de dérivée, la donnée brute est trop bruitée. Il faut ainsi appliquer un filtre pour l'exploiter.



(a) Road status 1



(b) Road status 2

FIGURE 16 – Résultat de la pipeline

## Noeud de contrôle

Nous avons utilisé comme interface de contrôle *rqt* afin de publier un message contenant la vitesse de la voiture. Le package *PID* de ROS a été également utilisé afin de fournir les commandes  $u_1$  et  $u_2$ . Ce noeud est simplement lancé dans le fichier *.launch* après une opération de *remap* pour rediriger les topics ROS correspondants.

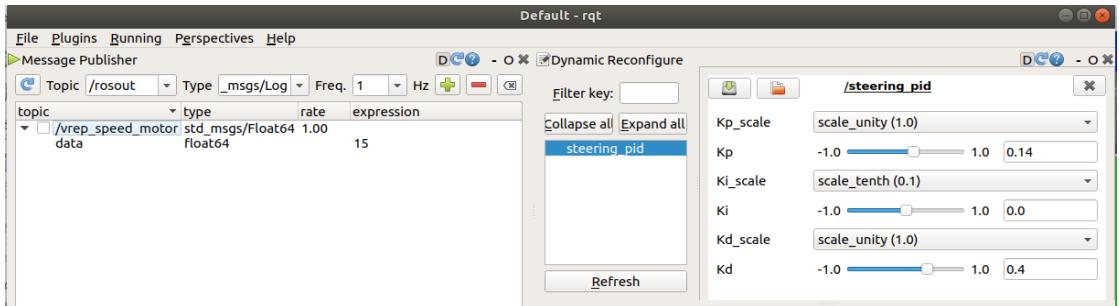


FIGURE 17 – Interface de contrôle

## 7 Résultats

Le traitement précédent n'a été testé qu'en simulation, cependant les résultats obtenus sont prometteurs quant à son utilisation sur un terrain réel : la voiture arrive à faire le tour du terrain de façon autonome.

Un lien d'une vidéo de démonstration est disponible sur le dépôt *GitHub* du projet.

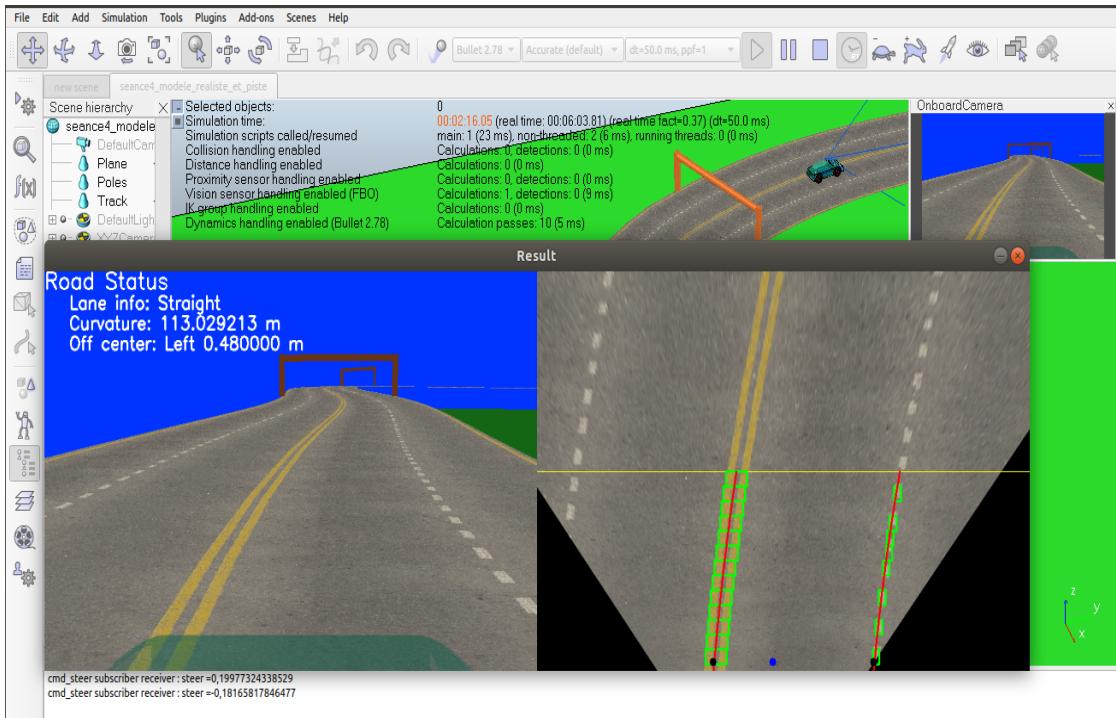


FIGURE 18 – Interface V-REP

## 8 Perspectives d'améliorations

D'un point de vue mécanique, nous avions parlé du problème lié à la réduction de vitesse en sortie du moteur qui n'était pas assez grande et qui rendait le contrôle de la voiture difficile. Ceci pourrait être réglé en ajoutant étage de réduction avec un autre engrenage. Pour cela vu le peu de place restant à l'arrière de la voiture, il faudrait redessiner tout le système de réduction pour pouvoir rajouter cet engrenage.

Pour ce qui est du poids de la voiture, les tests effectués avec des poids à l'avant de la voiture étaient relativement concluant. Il faudrait d'abord assembler tous les capteurs et toutes les pièces mécaniques avant de décider si il est nécessaire d'alourdir la voiture ou non.

Concernant l'utilisation du capteur GNSS et de la centrale inertuelle, nous avons délibérément choisi de ne pas les utiliser et d'effectuer seulement l'asservissement via le suivi de ligne. Comme améliorations possibles nous pourrions ainsi rajouter une odométrie visuelle avec la caméra et fusionner cette donnée avec celle du capteur GNSS et de la centrale inertuelle afin d'avoir l'état complet de la voiture.

Pour la partie asservissement visuel, nous sommes encore loin d'atteindre des résultats fiables et robustes. En effet, quelques problèmes sont d'ores et déjà réglés comme la non détection des lignes sur une frame. Cependant, beaucoup d'améliorations restent à faire :

- Effectuer un *FullSearch* uniquement sur la première frame. Ensuite, au lieu de rechercher les lignes de la voie sur toute l'image, on recherche uniquement dans une zone proche de la zone où la détection s'est effectué sur l'image d'avant. L'implémentation de la librairie qui effectue la pipeline de contrôle en orienté objet permettrait de passer facilement à cette étape ;
- Comme mentionné précédemment, le rayon de courbure n'est pas utilisé pour l'instant. Il faudrait implémenter un filtre afin d'exploiter cette donnée ;
- Comme dernière étape, au lieu d'extraire les deux régions en bas de l'image, on peut lancer la détection sur toute l'image. Cela pourrait améliorer le calcul du rayon de courbure. Mais pour cela, il faut trouver les bons filtres et prétraitements à appliquer.

Enfin, après avoir suivi les cours de systèmes embarqués, nous comprenons que le fait d'utiliser un *Linux* classique (comme *Ubuntu*) n'est pas la solution optimale pour une *Raspberry*. L'utilisation de *Buildroot* afin de concevoir et de compiler directement les briques élémentaires composant notre *Linux* embarqué, est un complément qui serait très intéressant. Nous ne savons pas si cela rendrait l'exécution des codes plus rapides, mais il serait bien de pouvoir comparer ces deux solutions différentes afin de choisir la meilleure des deux.

## Conclusion

Pour conclure, ce projet nous a permis de mettre en pratique différents cours étudiés durant cette année, tels que ceux de *ROS*, de *Traitemet d'image*, de *VREP* ou même de *Conception et réalisation*. Il s'apparente comme étant une synthèse de nos connaissances et permet à juste titre de faire le point sur nos compétences et nos capacités en tant que futur ingénieur roboticien.

## Annexe A

# Complément Architecture Électronique

Le schéma suivant permet de valider le choix des capteurs, des adaptateurs de tension ainsi que des contrôleurs.

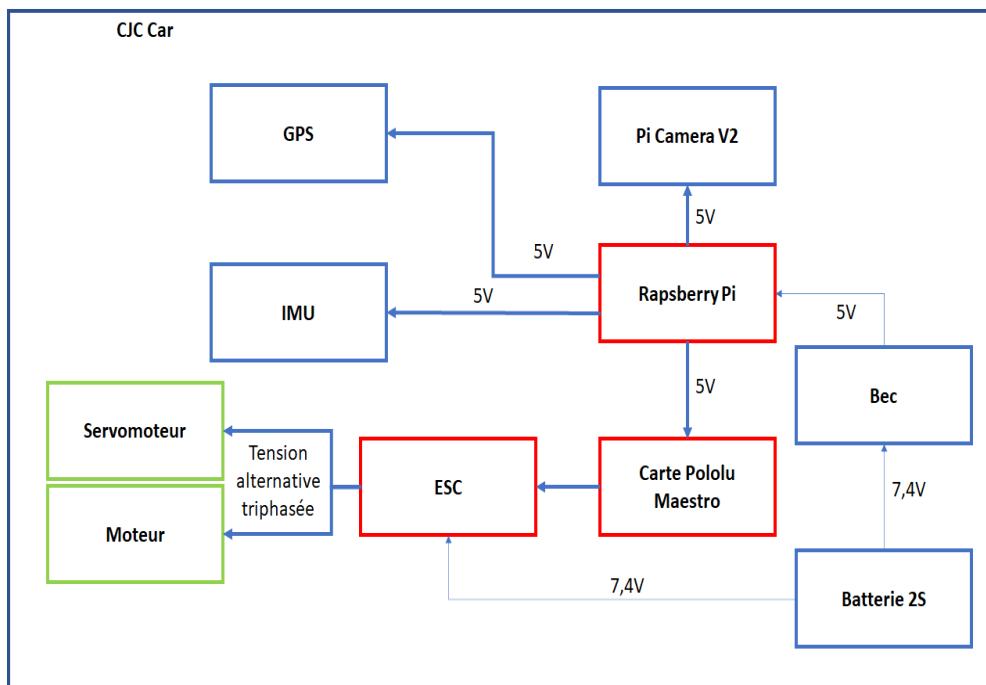


FIGURE A.1 – Architecture fonctionnelle d'alimentation

# Table des figures

|     |   |    |
|-----|---|----|
| 1   | Problématique du projet . . . . .                   | 3  |
| 2   | Cahier des Charges Fonctionnel . . . . .            | 4  |
| 3   | Diagramme pieuvre . . . . .                         | 4  |
| 4   | Architecture fonctionnelle générale . . . . .       | 5  |
| 5   | Zoom sur la voiture . . . . .                       | 6  |
| 6   | Pièces ajoutées . . . . .                           | 7  |
| 7   | Modélisation de la voiture . . . . .                | 8  |
| 8   | Texture de la route . . . . .                       | 8  |
| 9   | Modélisation cinématique de la voiture . . . . .    | 9  |
| 10  | Interface entre ROS et V-REP . . . . .              | 10 |
| 11  | Architecture fonctionnelle d'information . . . . .  | 10 |
| 12  | Pipeline de contrôle . . . . .                      | 12 |
| 13  | Bird Eye View . . . . .                             | 13 |
| 14  | Extraction des ROI . . . . .                        | 14 |
| 15  | Détection de la voie . . . . .                      | 15 |
| 16  | Résultat de la pipeline . . . . .                   | 16 |
| 17  | Interface de contrôle . . . . .                     | 16 |
| 18  | Interface V-REP . . . . .                           | 17 |
| A.1 | Architecture fonctionnelle d'alimentation . . . . . | 19 |