Développement d'un système de navigation sans carte et de contrôle d'un véhicule autonome

Elian Belmonte, *Étudiant, UTBM,* Benjamin Stach, *Étudiant, UTBM,* Grégori Mignerot, *Étudiant, UTBM,* Thomas Bédrine, *Étudiant, UTBM*

***Résumé*—Ces dernières années, la conduite autonome est l'un des champs les plus concurrentiels de la recherche et développement dans le secteur de l'automobile. À ce jour, la plupart des systèmes de navigation autonome se basent sur une carte à très haute résolution de l'environnement à naviguer, une approche efficace et simplifiant considérablement la navigation. Néanmoins, une telle carte est extrêmement difficile à tenir à jour en conditions réelles sur de larges superficies, et requiert un positionnement dans l'espace approximativement aussi précis que la carte elle-même, ce qui représente une contrainte matérielle et financière significative. Ce projet vise à proposer une approche de la navigation détachée du besoin d'une carte à haute résolution, aussi autonome que possible.**

***Index Terms*—autonomous vehicle, navigation, robotics**

I. Introduction

Au cours de ce projet, nous avons travaillé au développement des différents modules permettant la navigation totalement autonome d'un véhicule sans nécessiter de positionnement ni de cartographie haute résolution. Notre approche comprend trois modules principaux :

— Détection de la signalisation verticale (panneaux, feux) et prise de décisions à partir de ces informations

— Calcul de la trajectoire à suivre à partir des marquages au sol

— Contrôle du véhicule à partir d'une trajectoire dans l'espace

La section II définit les prérequis en terme de positionnement dans un repère en mouvement, utilisés par tout le reste du projet. La section III présente les méthodes utilisées pour la détection et la reconnaissance de la signalisation routière verticale. La section IV décrit l'algorithme de construction de la trajectoire à partir des marquages au sol. Enfin, la section V définit la méthode utilisée pour contrôler le véhicule à partir des informations ainsi obtenues. Enfin, les résultats obtenus, les améliorations et les moyens de compléter ces éléments seront discutés en VI

II. Prérequis à l'approche sans carte

Le système présenté par ce projet cherche à éviter toute dépendance à un positionnement à haute résolution, ce qui implique l'absence de repère inertiel fixe, ou du moins de repère fixe par rapport auquel on peut se localiser précisément. Cette nécessité impose des contraintes fortes sur toutes les opérations : les positions et vecteurs ne peuvent être exprimés que dans un repère local au véhicule, qui est constamment en mouvement.

Ce problème est résolu par deux éléments :

— Un service qui garde la trace de la vitesse linéaire et angulaire en provenance d'une IMU et peut en déduire les transformations du repère local entre différents points dans le temps

— La conservation rigoureuse du moment auquel chaque information a été prise, pour pouvoir la transformer correctement du moment où elle a été prise au moment où elle doit être exploitée

A. Intégration de l'orientation

Les données venant d'une centrale inertielle sont les suivantes :

— Vitesse angulaire ω en provenance du gyroscope

— Accélération linéaire $\mathbf{a}$ en provenance de l'accéléromètre

— Un magnétomètre peut éventuellement compléter ces données. Dans le cas présent, il ne sera pas exploité

L'objectif du service de positionnement est d'intégrer le mouvement du véhicule afin d'en déduire la rotation et la translation entre deux instants. Cependant, ces données sont exprimées dans le repère local du véhicule à chaque instant, ce qui rend difficile cette intégration.

Nous n'avons pas eu le temps d'ajouter cette procédure au projet. Cependant, celle-ci a été partiellement testée et fonctionnerait une fois corrigée et correctement ajustée au cas d'usage.

La difficulté principale est l'orientation du véhicule : à chaque instant, le véhicule a une orientation différente, ce qui change l'orientation du repère dans lequel sont exprimées toutes les données de mouvement. Ce problème peut être résolu avec une méthode d'intégration adéquate. Ici, nous avons choisi d'employer un filtre de Madgwick [1]. Celui-ci permet d'intégrer l'orientation du véhicule par rapport à un repère fixe à partir des données du gyroscope, de l'accéléromètre et éventuellement d'un magnétomètre, de façon plus spécifique et plus précise qu'un filtre de Kalman. Le repère de référence peut être un repère terrestre si un tel positionnement est disponible, mais dans notre cas cela peut être le repère local du véhicule à l'instant où le filtre est initialisé.

Le filtre de Madgwick permet ainsi d'obtenir une orientation par rapport à un repère de référence, ce qui permet d'effectuer la transformation des données inertielles dans un même repère. Étant des informations de vitesse et d'accélération, elles sont invariantes en translation.

B. Intégration du mouvement

L'état réellement implémenté dans le projet commence ici, en exploitant les données fournies par le simulateur qui sont déjà dans un repère fixe.

À ce stade, tous les points de données inertielles sont dans un même repère. Cela simplifie considérablement les calculs, en découplant la translation de la rotation. Il suffit en effet d'intégrer l'accélération ou la vitesse linéaire pour calculer la translation, et d'intégrer la vitesse angulaire pour la rotation.

Le but est alors d'itérer sur les points de données mesurés, et d'interpoler afin d'intégrer de la façon la plus précise possible.

1) Calcul de la translation

Pour calculer le vecteur de translation, on peut intégrer analytiquement le mouvement entre chaque paire de points de données. Soient $[0, t_1]$ l'intervalle de temps entre les points de données inertielles. Les calculs sont équivalents en commençant à un instant t_0, t_0 devra alors simplement être retranché de tous les points dans le temps. $\vec{v_0}$ et $\vec{v_1}$ sont les vecteurs vitesse linéaire mesurés aux instants 0 et t_1. On cherche à intégrer ce mouvement entre les instants t_i et t_f. On appelle $\vec{a}$ le vecteur accélération entre les instants 0 et t_1. Si on dispose des vitesses linéaires, celui-ci s'exprime par la pente de l'interpolation linéaire entre les deux vecteurs vitesse :

$$\vec{a} = \frac{\vec{v_1} - \vec{v_0}}{t_1 - 0}$$

On peut ensuite intégrer la vitesse linéaire $\vec{v}(t)$ à chaque instant :

$$\vec{v}(t) = \int_0^t \vec{a} dx = \vec{a}t + \vec{v_0}$$

Ce vecteur vitesse peut à nouveau être intégré sans valeur initiale pour donner la translation $\vec{T}$ entre les instants t_i et t_f :

$$\begin{aligned}
\vec{T}(t) &= \int_{t_i}^{t_f} (\vec{a}t + \vec{v_0}) dt \\
&= \left[\frac{1}{2}\vec{a}t^2 + \vec{v_0}t \right]_{t_i}^{t_f} \\
&= (\frac{1}{2}\vec{a}t_f^2 + \vec{v_0}t_f) - (\frac{1}{2}\vec{a}t_i^2 + \vec{v_0}t_i) \\
&= \frac{1}{2}\vec{a}(t_f^2 - t_i^2) + \vec{v_0}(t_f - t_i)
\end{aligned}$$

2) Calcul de la rotation

Les orientations et les rotations sont ici représentées par des quaternions.

La version avec IMU réel et filtre de Madgwick permet d'obtenir la rotation simplement à partir des orientations. On obtient les orientations q_i et q_f par la slerp (interpolation linéaire sphérique) des orientations situées autour des instants voulus, puis la rotation $R = q_f * q_i^{-1}$.

La suite de cette section décrit la version actuellement implémentée dans le projet, qui est l'intégration des vitesses angulaires déjà exprimées dans un repère fixe. Le problème est plus complexe que pour la translation. Comme démontré dans [2], la rotation peut s'exprimer par l'équation différentielle suivante :

$$\frac{d\mathbf{R}}{t}(t) = \frac{1}{2}\omega(t)\mathbf{R}(t)$$

Où $\mathbf{R}(t)$ est la rotation par rapport au repère de référence et $\omega(t)$ la vitesse angulaire à l'instant t. Si cette équation est relativement simple pour une vitesse angulaire constante, elle ne l'est pas pour notre cas où cette dernière évolue de manière imprévisible dans le temps. On passe donc par un développement de Taylor du premier degré pour l'intégrer numériquement :

$$\mathbf{R}(t + \delta t) = \mathbf{R}(t) + \frac{1}{2}\delta t\, \omega(t)\mathbf{R}(t)$$

Pour une précision correcte, il est nécessaire d'effectuer cette intégration numérique par très courts intervalles (un intervalle δt de l'ordre de la milliseconde semble donner des résultats satisfaisants). Tant que la vitesse angulaire reste relativement faible, un développement de Taylor de plus haut degré n'est pas nécessaire.

L'intégration numérique pouvant faire rapidement perdre son statut de quaternion unitaire à $\mathbf{R}$, il devrait être nécessaire de le normaliser régulièrement. Cependant, comme souligné par Michael Boyle [2] et confirmé par l'expérimentation, cette normalisation n'est pas nécessaire aux étapes intermédiaires, les résultats sont identiques en ne normalisant qu'à la toute fin.

III. Détection de la signalisation

A. Modèle de détection YOLO

You Only Look Once (YOLO) est un algorithme de détection d'objets en temps réel introduit en 2015 par *Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi* dans leur article « You Only Look Once : Unified, Real-Time Object Detection ».[3]

La détection d'objets regroupe diverses approches telles que R-CNN rapide, Retina-Net, et Single-Shot MultiBox Detector (SSD). Bien que ces approches aient résolu les problèmes de limitation de données et de modélisation dans la détection d'objets, elles ne sont pas en mesure de détecter des objets dans un seul cycle d'algorithme. L'algorithme YOLO a gagné en popularité en raison de ses performances supérieures sur les techniques de détection d'objets mentionnées précédemment. En effet, YOLO est plus rapide, tout en étant très précis. De plus, l'algorithme démontre d'excellentes capacités d'apprentissage qui lui permette d'apprendre les représentations des objets et de les appliquer dans la détection des objets.

Comment fonctionne l'algorithme YOLO ?

L'algorithme YOLO utilise des réseaux de neurones convolutifs (CNN) pour détecter des objets en temps réel. Comme son nom l'indique, l'algorithme ne nécessite qu'une seule propagation directe à travers un réseau neuronal pour détecter les objets. Cela signifie que la prédiction de l'image entière se fait en un seul cycle d'algorithme. Le CNN est

utilisé pour prédire simultanément diverses probabilités de classe et les boîtes englobantes.

L'algorithme YOLO vise à prédire une classe d'un objet et sa boîte englobante qui définit l'emplacement de l'objet sur l'image d'entrée. Il définit les boîtes englobantes en utilisant quatre valeurs :

— Centre de la boîte englobante (ou coin supérieur gauche) (b_x, b_y)

— Largeur de la boîte (b_w)

— Hauteur de la boîte (b_h)

En ajoutant à cela, les probabilités d'appartenance à chaque classe considérée $(C_1, C_2, ..., C_n)$ et la probabilité globale de la prédiction (ou confiance) (P_c), il est possible de localiser et de classifier un objet avec un vecteur caractéristique de la forme suivante :

$$v = (P_c, b_x, b_y, b_w, b_h, C_1, C_2, ..., C_n)$$

L'algorithme YOLO divise une image d'entrée en une grille $N \times N$ de plus petites images de même taille. Un vecteur caractéristique est associé à chacune des subdivisions de l'image. La couche de sortie d'un réseau de neurone YOLO prendra donc la forme d'un vecteur de dimension $nb_{hauteur} \times nb_{largeur} \times dim_v$.

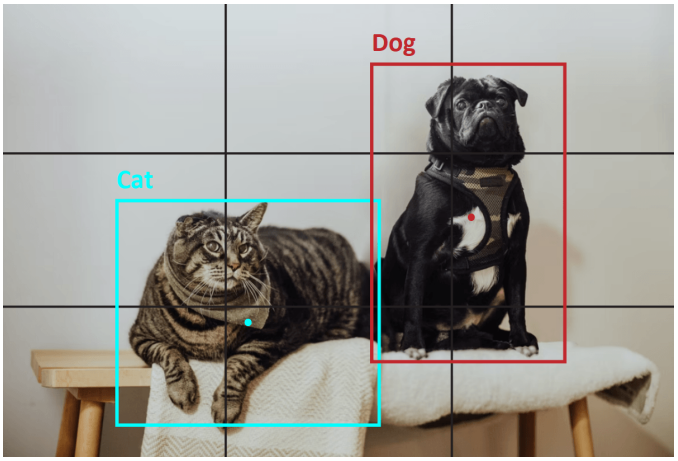


FIGURE 1 – Représentation d'un modèle de détection YOLO pour la reconnaissance des chats et chiens.

Dans l'exemple ci-dessus, on souhaite détecter des éléments parmi deux classes distinctes (C_1, C_2) et l'on utilise une grille de taille 3×3. La couche de sortie du modèle sera donc un vecteur de dimension $3 \times 3 \times 7$.

Dans le cas de notre véhicule autonome, nous utiliserons le modèle YOLOv4. Cette version de YOLO a une vitesse et une précision de détection d'objets supérieure à toutes ses versions précédentes et à d'autres détecteurs d'objets de l'état de l'art. YOLOv4 est spécialement conçu pour les systèmes de production et optimisé pour les calculs parallèles. L'ossature de l'architecture de YOLOv4 est CSPDarknet53, un réseau contenant 29 couches de convolution et environ 27,6 millions de paramètres.

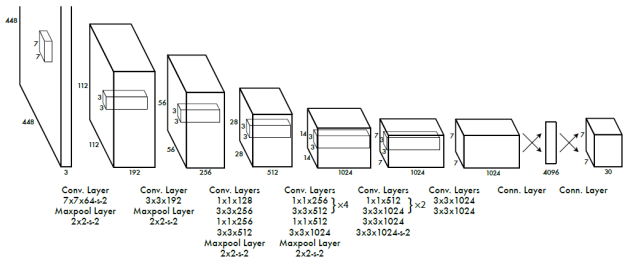


FIGURE 2 – Architecture du réseau de neurones YOLO basique.

B. *Détection des panneaux*

Dans le cadre de ce projet, nous avons réutilisé les modèles pré-entraînés proposés par *Alzaib Karovalia*[4]. Le problème de détection de la signalisation a été divisé en deux sous-problèmes, à savoir détecter et localiser les panneaux de signalisation dans l'image, puis classifier ces panneaux une fois isolés.

Dans un premier temps, un modèle du type YOLOv4-Tiny a été réentraîné sur la base d'un jeu de données labellisé et pré-traité pour le format YOLO. Cependant, ce jeu de données est issu du GTSRB [5] (German Traffic Sign Recognition Benchmark) qui est disponible en open-source. Le GTSRB contient 43 classes de panneaux de signalisation, divisés en 39 209 images de formation et 12 630 images de test. Les images présentent des conditions de luminosité variables. Le modèle YOLOv4-Tiny sert à localiser et isoler tous ces différents panneaux.

Dans un second temps, on utilisera un modèle de classification pré-entraîné sur la base du même jeu de données par *Junth Basneth*[6]. Sur une base de 43 classes de panneaux différents, le modèle atteint la performance de 99,22% sur l'ensemble de données GTSRB. Comparé à d'autres algorithmes, le modèle proposé basé sur LeNet-5 a une précision remarquable, fonctionne en temps réel, une forte capacité de généralisation et une grande efficacité d'entraînement.

Malgré le fait que les deux modèles soient entraînés à détecter et classifier des panneaux allemands, ils démontrent de très bonnes capacités à généraliser. Dans le cadre de notre projet, leur aptitude à détecter les panneaux français les plus courants (stop, cédez-le-passage, direction obligatoire, limitation de vitesse etc.) a été jugée suffisante. Cependant, afin d'être encore plus précis, il serait envisageable de réentraîner ces modèles sur la base d'un jeu de données de signalisation français.

C. *Détection des feux tricolores*

1) *Modèle YOLOv3*

En plus des panneaux de signalisation, on rencontre également des feux tricolores sur le parcours du véhicule.

L'objectif est de repérer le feu bien avant de l'atteindre, afin de prendre une décision et d'arrêter le véhicule linéairement. Toutefois, les modèles précédemment cités ont été entraînés pour des panneaux, mais pas pour des feux tricolores.

Un autre modèle dédié à cette tâche était donc nécessaire, et nous nous sommes tournés vers le noeud déjà existant pour l'édition précédente de l'UTAC. Le modèle a été entraîné avec YOLOv3 et non YOLOv4-Tiny, cependant les résultats attendus sont relativement proches considérant notre problème. Le script de détection peut considérer plusieurs frames pour avant de décider d'un résultat afin de vérifier la pertinence des détections et d'éliminer les faux positifs.

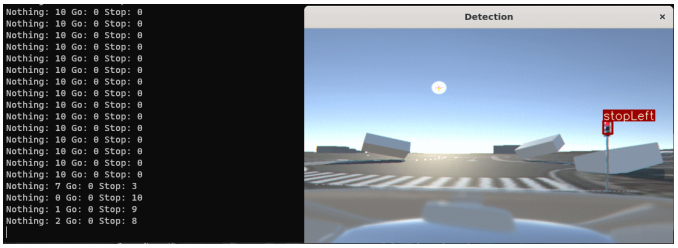


FIGURE 3 – Détection d'un feu rouge.

Toutefois, nos résultats ont été mitigés avec ce script, qui ne détecte généralement le feu que lorsque nous en sommes très proches - à la distance d'arrêt le plus souvent. Ce noeud a en effet été conçu pour une navigation avec carte, dans un cas où la position exacte des feux est connue et où la détection sert simplement à vérifier que le feu est bien présent.

Les résultats sont donc tout à fait justes, toutefois ils supposent un certain nombre d'hypothèses :

— le véhicule navigue avec une carte et connaît l'emplacement du feu

— le véhicule connaît déjà l'état du feu et s'arrêtera s'il est rouge

— le feu sera recherché et détecté à partir de la distance d'arrêt.

Dans ces conditions, il est complexe d'obtenir une détection qui réponde à nos besoins, puisque nous ne remplissons aucune des trois hypothèses. Il serait donc nécessaire de développer une première étape de détection du feu avant sa classification, ou un modèle combiné. Sur le moment, nous avons omis cette possibilité et avons choisi d'implémenter une solution de type« handcrafted ».

2) Transformée de Hough

Nous avons donc tenté de réaliser la détection des feux avec une méthode « handcrafted », en passant par la transformée de Hough. Une telle méthode computationnellement plus simple peut en effet paraître plus adaptée et plus performante qu'un réseau de neurones profond sur ce type de tâches.

On cherche donc à localiser des cercles rouges, jaunes ou verts, qui indiqueront la présence d'un feu tricolore. Cependant la détection par la transformée de Hough s'est avérée moins efficace que YOLOv3.

Le feu à détecter se trouve en effet le plus souvent en bordure de l'écran, or la caméra utilisée pour le véhicule est de type fisheye. Elle a donc tendance à déformer sensiblement les éléments proches des bords, comme c'est le cas pour les feux. Nous ne cherchons donc pas des cercles mais des ellipses en réalité, sans compter le fait que le feu n'est de toute façon pas vu de face mais incliné par rapport à la caméra.

FIGURE 4 – Feu tricolore en situation idéale (fronto-parallèle), et avec inclinaison et/ou distorsion. Ce deuxième cas présente des cercles fortement distordus donc inexploitables par la transformée de Hough

Les résultats avec l'une ou l'autre méthode ne sont malheureusement pas aussi concluants que pour les panneaux, aussi nous devons nous contenter d'une détection assez spécifique avec le YOLOv3. L'implémentation de cette détection est donc à améliorer pour l'instant, en contournant le problème des feux déformés. Typiquement, la méthode de transformation caméra inverse décrite en IV-A2 peut aussi permettre d'éliminer la distorsion de l'image. Des méthodes basées sur des taches de couleur ou d'autres canaux d'information comme la saturation seraient également plus adaptées que la transformée de Hough qui nécessite des cercles parfaits. Autrement, un modèle de localisation puis classification comme celui des panneaux de signalisation pourrait permettre l'utilisation du réseau de neurones existant dans le contexte sans carte.

D. Position de la signalisation

Afin que le noeud de contrôle puisse prendre des décisions adéquates, il est très important de détecter et reconnaître les panneaux de signalisation présents dans le champ de vision du véhicule. Cependant, il est également nécessaire pour le véhicule de connaître la position de ces panneaux dans l'espace. Nous nous servirons des points du LiDAR ainsi que de la localisation dans l'image des panneaux détectés par le modèle YOLO pour remonter à la position réelle de ces panneaux.

D'abord, la première étape consiste à projeter les points de l'espace dans l'image. Une partie du nuage de points

capté par le LiDAR pourra être élaguée afin de limiter les calculs inutiles. Par exemple, tous les points qui se situent derrière la caméra ne se trouvent pas dans son champ de vision et ne seront donc pas projetés dans l'image. La projection se fera après estimation des paramètres intrinsèques et extrinsèques (par rapport au repère LiDAR) de la caméra. Afin de projeter les points avec précision, il est nécessaire d'estimer les paramètres de distorsion de la caméra.

Ensuite, la seconde étape consiste à déterminer et regrouper les points qui se retrouvent projetés à l'intérieur des boîtes englobantes détectées par le modèle YOLO.

Enfin, il s'agira d'estimer la position réelle des panneaux en faisant la moyenne des points de l'espace projetés dans chaque boîte englobante. Il pourrait également être envisageable de prendre la position de la médiane des points afin d'éviter que d'autres éléments qui se situent dans la boîte englobante viennent perturber la mesure.

IV. Construction de la trajectoire

Afin de contrôler le véhicule en accord avec son environnement, l'algorithme présenté en section V a besoin qu'on lui fournisse une trajectoire à suivre. Pour cela, les marquages au sol sont exploités afin de détecter la voie dans laquelle circule le véhicule, la suivre, et naviguer dans les intersections.

Cet partie du système peut se décomposer en plusieurs éléments détaillés ci-après.

A. Prétraitement des images

1) Opérations de prétraitement

Pour rendre l'information exploitable par le reste du pipeline, on applique certaines opérations de prétraitement aux images.

Premièrement, on convertit l'image en niveaux de gris et on applique un léger flou gaussien afin de réduire le bruit des images.

Ensuite, on extrait une vue du dessus d'une région située devant le véhicule afin d'obtenir des positions directement dans l'espace 3D par la suite. La procédure de transformation inverse est décrite dans la section suivante IV-A2. La figure 5b montre le résultat de la transformation inverse.

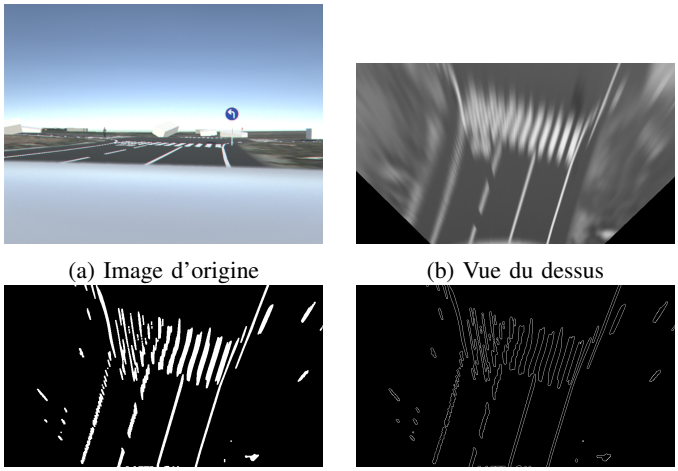
La vue du dessus est ensuite binarisée. Initialement, un seuillage par la méthode d'Otsu était employé. Cependant, et bien qu'elle donne des meilleurs résultats dans des bonnes conditions d'éclairage, cette technique s'est avérée trop sensible aux variations d'illumination. Un seuillage adaptatif basé sur un noyau gaussien permet de limiter ces problèmes. Néanmoins, ce type de seuillage peut fortement amplifier certain contours indésirables de l'image et perturber les résultats. Ce point reste à améliorer à la suite de ce projet.

Du fait de cette amplification des contours, les bordures noires dues à l'absence d'information sur la vue du dessus sont transformées en larges contours blancs. On applique un masque à l'image binarisée pour éliminer ces bordures. Le résultat après seuillage et masquage est présenté par la figure 5c.

Une légère opération d'ouverture est réalisée afin d'éliminer les plus petits éléments. Une ouverture puis une fermeture plus larges ont été testées précédemment, mais la finesse des marquages de voie ne permet pas de les utiliser sans perdre d'informations importantes.

Enfin, on effectue une détection des contours par l'algorithme de Canny sur la vue binarisée, comme montré par le résultat final sur la figure 5d.

La détection de contour pourrait être effectuée directement sans binarisation intermédiaire. Cependant, à cause de la qualité des images provenant de la simulation utilisée et du flou résultant de la transformation caméra inverse, les contours obtenus seraient épars et fortement bruités, même après fermeture, alors que les opérations suivantes ont besoin de courbes continues dans l'image.



(a) Image d'origine

(b) Vue du dessus

(c) Résultat du seuillage adaptatif

(d) Résultat de tous les prétraitements

FIGURE 5 – Visualisation des différentes étapes du prétraitement des images provenant de la caméra

2) Transformation caméra inverse

La caméra grand-angle exploitée par notre système emploie le modèle décrit par Christopher Mei dans [7]. Afin d'extraire efficacement la géométrie des voies, on applique la transformation inverse sur les images afin d'obtenir une vue orthogonale du sol à l'avant du véhicule, sur laquelle la conversion entre coordonnées pixéliques et coordonnées sur le plan 3D sera triviale.

On utilisera ici les notations de l'article de Christopher Mei. On peut tout d'abord utiliser les formules présentées dans l'article pour projeter les points pixéliques sur la surface de la sphère unité, avec K la matrice de projection caméra et p un point de l'image en coordonnées pixéliques.

$$\Xi(\mathbf{m_u}) = \frac{\xi + \sqrt{1 + (1 - \xi^2)(x^2 + y^2)}}{x^2 + y^2 + 1}$$

$$\hbar^{-1}(\mathbf{m_u}) = \begin{pmatrix} \Xi(\mathbf{m_u})x \\ \Xi(\mathbf{m_u})y \\ \Xi(\mathbf{m_u}) - \xi \end{pmatrix}$$

$$\mathbf{X_s} = \hbar^{-1}(\mathbf{K}^{-1}\mathbf{p})$$

Sachant les coordonnées cartésiennes du point X_s à la surface d'une sphère unité de centre $(0, 0)$, on peut obtenir les coordonnées sphériques du point X_s :

$$\begin{cases} \rho_s = 1 \\ \theta = \arccos \dfrac{z_s}{\rho_s} \\ \phi = \arctan \dfrac{y_s}{x_s} \end{cases}$$

Pour ρ le rayon, θ la colatitude et ϕ la longitude.

On fait l'hypothèse que tous les points de l'image proviennent d'un même plan dans l'espace 3D. Par simplicité dans notre cas d'utilisation précis, on part du principe qu'il existe un repère cible T tels que tous les points de l'image proviennent du plan défini par l'équation $z_T = 0$ dans le repère T, et que l'on connaît la matrice ${}^{C}\mathbf{M_T}$ de transformation du repère caméra C vers le repère cible T. Dans cette situation, un point X_C exprimé dans le repère caméra est projeté dans le repère cible par l'équation suivante :

$$\mathbf{X_T} = {}^{C}\mathbf{M_T}\mathbf{X_C}$$

Sachant que $z_T = 0$, on peut extraire l'équation du plan cible exprimée dans le repère caméra de la ligne correspondante de la matrice ${}^{C}\mathbf{M_T}$:

$$z_T = {}^{C}M_{T_{3,1}}x_C + {}^{C}M_{T_{3,2}}y_C + {}^{C}M_{T_{3,3}}z_C + {}^{C}M_{T_{3,4}} = 0$$

On peut renommer ces paramètres respectivement a_C, b_C, c_C et d_C pour obtenir l'équation de plan suivante dans le repère caméra

$$a_C x_C + b_C y_C + c_C z_C + d_C = 0$$

Comme $\mathbf{X_s}$ est la projection centrale du point $\mathbf{X_C}$ sur la sphère unité de centre $(0, 0)$, la colatitude θ et la longitude ϕ de $\mathbf{X_C}$ en coordonnées sphériques sont égales à celles de $\mathbf{X_s}$. Seul reste à déterminer le rayon ρ_C associé à $\mathbf{X_C}$, que l'on peut obtenir par la conversion en coordonnées cartésiennes et l'équation du plan.

$$\begin{cases} x_C = \rho_C \sin(\theta)\cos(\phi) \\ y_C = \rho_C \sin(\theta)\sin(\phi) \\ z_C = \rho_C \cos(\theta) \\ ax_C + by_C + cz_C + d = 0 \end{cases}$$

La résolution de ce système d'équations linéaires simple donne les coordonnées de $\mathbf{X_C}$, qu'il suffit de projeter dans le repère cible grâce à ${}^{C}\mathbf{M_T}$, puis de le placer sur la vue du dessus en éliminant l'axe Z, nul par définition.

En pratique, pour des raisons de qualité et de performance, sur des images complètes, on peut passer par l'opération inverse, projeter la position de chaque pixel de la vue du dessus sur la vue caméra et récupérer ainsi sa valeur. Cependant, la méthode présentée précédemment est nécessaire pour projeter des points individuels

B. Détection de la voie

1) Détection par fenêtres sur l'axe Y

Les premières méthodes de détection de la voie à suivre se basaient sur la détection du marquage au sol par fenêtres, chaque fenêtre étant estimée selon l'orientation du marquage détecté dans la précédente.

La première technique employée se base uniquement sur l'axe Y, pointant vers l'avant du véhicule. L'image est d'abord transformée en vue du dessus et binarisée. Les fenêtres initiales sont placées selon des connaissances a priori sur la position attendue de la voie. La fenêtre suivante est contigüe à la précédente sur l'axe Y, et sa position sur l'axe X est déterminée par la position sur l'axe X du centroïde des pixels blancs situés dans la fenêtre. Finalement, une régression polynomiale est effectuée sur les centroïdes de chaque fenêtre pour obtenir une représentation d'un marquage de voie. En réalisant ces opérations sur les marquages gauche et droit, on peut en déduire la trajectoire à suivre comme le centre de la voie. La figure 6 montre ce processus de manière schématique, et la figure 7 montre une visualisation de cette méthode.

Malgré des résultats presque parfaits en bonnes conditions, cette méthode a des inconvénients problématiques dans ce cas d'usage :

- Le paramétrage de la largeur de fenêtre est problématique : une fenêtre trop étroite rend la méthode beaucoup trop sensible à la courbure des voies, mais une fenêtre trop large capture des pixels de bruit ou d'autres marquages qui perturbent fortement les résultats
- Cette méthode a besoin d'un marquage isolé, sans quoi les autres marquages présents perturbent fortement les résultats
- Du fait de la régression polynomiale par les moindres carrés, toute perturbation dans la liste des centroïdes peut créer de forts écarts dans la courbe finale
- Ce processus itératif doit être correctement initialisé, ce qui nécessite des hypothèses basées sur des connaissances a priori qui ne sont pas nécessairement valides

2) Régression par fenêtre

Ces problèmes ont mené au développement d'une généralisation de cette méthode. Plutôt que de simplement avancer sur l'axe Y en se repositionnant par rapport au dernier centroïde, il est possible de calculer l'orientation du marquage dans chaque fenêtre, et de placer la fenêtre suivante à une distance fixe de la précédente, dans la direction du marquage. Cette généralisation permet de suivre précisément la courbure, sans avoir besoin d'élargir la fenêtre, ce qui réduit également l'effet des marquages parasites.

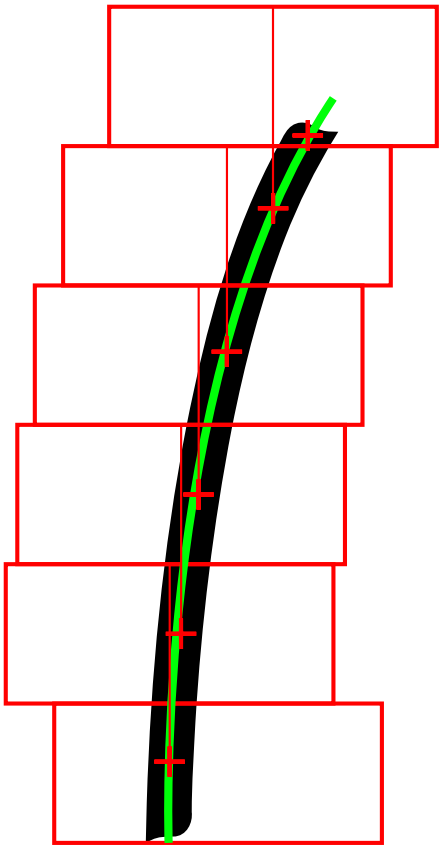


FIGURE 6 – Explication schématique de la première méthode de détection de voie. Les rectangles rouges sont les fenêtres successives, les croix rouges les centroïdes des pixels de chaque fenêtre, en vert le résultat de la régression polynomiale. Chaque fenêtre est placée après la précédente sur l'axe Y, et centrée au-dessus du centroïde de la fenêtre précédente sur l'axe X

L'image est d'abord transformée en vue du dessus, binarisée, puis une transformation *hit-or-miss* est appliquée afin de ne conserver que les contours en bas et à droite des marquages, ce qui permet de n'avoir qu'un seul contour par marquage et de clairement séparer les marquages proches.

Pour placer la fenêtre initiale, on utilise la transformée de Hough afin de détecter un segment de droite éligible comme un marquage de voie suffisamment fiable et suffisamment proche du véhicule. Cette méthode d'initialisation permet de s'affranchir partiellement des connaissances a priori nécessaires précédemment. Néanmoins, elle complique grandement l'initialisation pour plusieurs marquages, ce qui a mené à n'utiliser la présente méthode que sur un marquage unique, ici la ligne de rive.

Ensuite, dans chaque fenêtre, on réalise une régression linéaire sur les pixels de la fenêtre, pour obtenir le vecteur directeur et un point d'origine d'une droite paramétrant approximativement le marquage dans cette fenêtre. On utilise un algorithme de type RANSAC[8] afin de réduire l'influence des parasites sur la régression.

Pour positionner la fenêtre suivante, on projette orthogonalement le centre de la fenêtre sur la droite, puis on l'avance

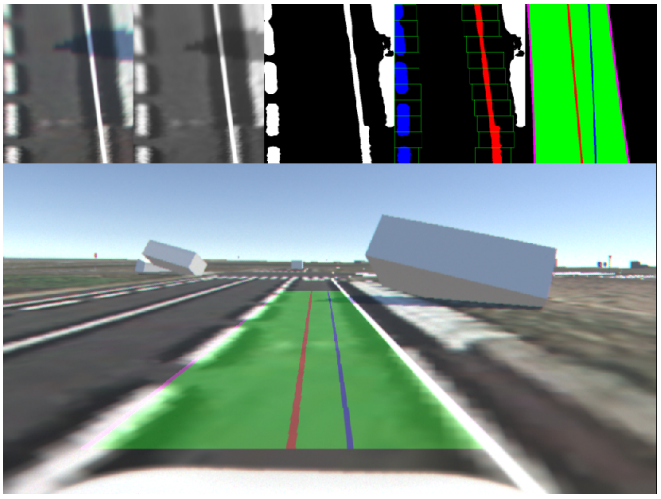


FIGURE 7 – Visualisation de la première méthode employée pour la détection de voie, par fenêtres sur l'axe Y

le long de cette droite d'une distance fixe, selon les équations (1) et (2). Pour $\mathbf{C_t}$ le centre de la t-ième fenêtre, $\vec{v}$ le vecteur directeur et $\mathbf{P}$ le point d'origine de la droite du marquage, et D la distance d'avancement, on obtient la projection du centre de la fenêtre sur la droite comme suit :

$$\mathbf{W_t} = \frac{(\mathbf{C_t} - \mathbf{P}) \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v} + \mathbf{P} \tag{1}$$

$$\mathbf{C_{t+1}} = \mathbf{W_t} + D\vec{v} \tag{2}$$

La suite des projections (W_t) suit alors le marquage à intervalles presque réguliers, ce qui permet d'en déduire une courbe paramétrique polynomiale de ce type :

$$\mathbf{M}(t) = \begin{pmatrix} a_x t^2 + b_x t + c_x \\ a_y t^2 + b_y t + c_y \end{pmatrix}$$

En prenant le paramètre t comme les indices de la suite des projections. Cette courbe peut être obtenue par une régression polynomiale des coordonnées des projections par rapport à t. Empiriquement, une régression polynomiale de degré 2 a donné les meilleurs résultats, par rapport à des polynômes de degré 3.

La figure 8 montre une visualisation de cette méthode de suivi de la ligne de rive.

À partir de ce paramétrage de la ligne de rive, on peut construire une estimation de la trajectoire, sachant la largeur de la voie a priori. Pour cela, on peut estimer le point de la trajectoire correspondant à chaque point de la ligne de rive grâce au vecteur orthogonal à la courbe. Sachant que le vecteur tangent à la courbe est donné par les dérivées de la courbe paramétrique :

$$\vec{v}(t) = \frac{d\mathbf{M}(t)}{dt} = \begin{pmatrix} \frac{dM_x(t)}{dt} \\ \frac{dM_y(t)}{dt} \end{pmatrix}$$

On peut en déduire le vecteur orthogonal à la courbe, et ainsi les points correspondants sur la trajectoire. Pour C la

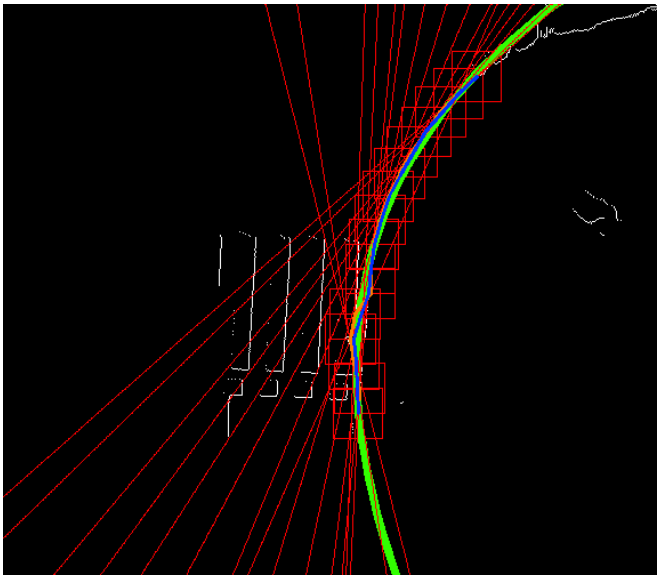


FIGURE 8 – Visualisation de la méthode de détection de marquage par régression par fenêtre. Les carrés rouges sont les fenêtres successives, en rouge les droites obtenues par régression linéaire sur chaque fenêtre, en bleu la suite des projections des centres de fenêtres sur les droites, en vert la courbe paramétrique obtenue à partir de ces derniers

courbe paramétrique représentant la trajectoire et L la largeur de la voie :

$$\mathbf{C}(t) = \mathbf{M}(t) + \frac{L}{2}\vec{v_\perp}(t) = \begin{pmatrix} M_x(t) - \frac{L}{2}\frac{dM_y(t)}{dt} \\ M_y(t) + \frac{L}{2}\frac{dM_x(t)}{dt} \end{pmatrix}$$

Cette généralisation de la technique précédente permet de résoudre certains de ses problèmes. En particulier, comme montré par la figure 8, elle permet de correctement suivre des marquages fortement courbés, même en présence d'autres marquages dans les environs, ici un passage piéton. Cette méthode semble adéquate pour la résolution du problème de la détection des voies. Cependant, malgré des résultats tout à fait corrects, cette méthode manque de fiabilité, ce qui a mené à son remplacement au cours du projet. En effet :

— Le problème de l'initialisation subsiste : à chaque image traitée, une mauvaise initialisation donnera toujours un résultat aberrant. Plus de travail serait nécessaire afin de trouver une méthode d'initialisation efficace et suffisamment fiable, sans besoin de connaissances a priori fortes pour ne pas pénaliser l'initialisation dans les fortes courbes. Une stabilisation par rapport à la trajectoire précédente serait envisageable, mais si le reste de l'algorithme donne des résultats anormaux, cela rendrait difficile de s'en détacher.

— De même, le paramétrage de la taille de la fenêtre reste un compromis difficile

— Cette méthode est extrêmement sensible à une régression de mauvaise qualité sur une fenêtre. Si la régression linéaire donne une droite dans la mauvaise direction, par exemple s'il y a trop de bruit, un marquage trop large ou mal centré, la fenêtre suivante sera placée dans la mauvaise direction et perdra le marquage, ce qui fait généralement totalement diverger le résultat.

— De même, les marquages parasites restent un problème sensible : un autre marquage trop proche du marquage de voie peut facilement capter l'attention du RANSAC, faisant partir la courbe sur le mauvais marquage.

— L'inconvénient de la régression polynomiale par les moindres carrés reste présent, et est amplifié par les facteurs précédents : souvent, les points extraits font des écarts, qui mènent à des polynômes totalement différents de la courbe attendue.

De manière générale, cette méthode est prometteuse, mais l'accumulation des facteurs pouvant la faire échouer à chaque étape du processus la rend moyennement fiable. Des tests ont montré que sur un trajet type dans le simulateur utilisé, en moyenne, la détection échouait totalement sur environ 33% des images (incapacité d'aller au bout du processus), et divergeait du résultat attendu pour 26% des images. Seules 41% des images donnaient une trajectoire satisfaisante. Bien que 33% de perte soit une proportion acceptable avec une fréquence suffisante, la quantité et la nature des résultats divergents rend très difficile leur séparation des bons résultats, ce qui rend difficile la stabilisation d'une trajectoire fiable.

3) Extraction de courbes discrètes

Les inconvénients mentionnés précédemment ont mené au développement d'une méthode d'extraction de la voie en cours radicalement différente de la précédente, qui résoudrait une majorité de ces problèmes. Les solutions employées pour cela sont :

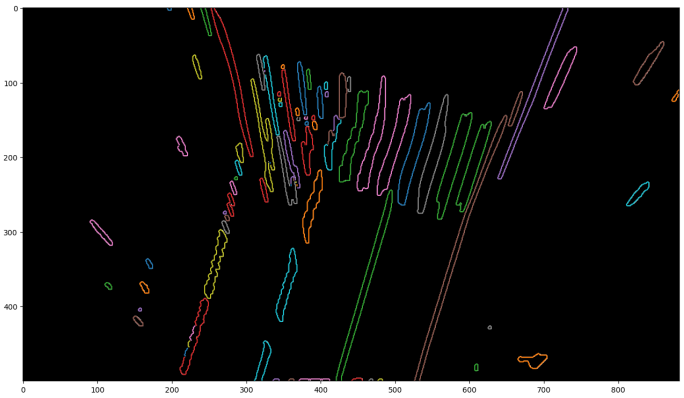
— Une représentation sous forme de courbes discrètes, une séquence de points dans le plan, afin de ne pas passer par une représentation paramétrique qui nécessite une régression par les moindres carrés, typiquement polynomiale, trop sensible même aux légers écarts

— Une approche globale, basée sur la génération puis le filtrage d'hypothèses, qui évite totalement le besoin d'initialiser un algorithme itératif puis de le garder sur la bonne voie avec pour seule base des informations locales.

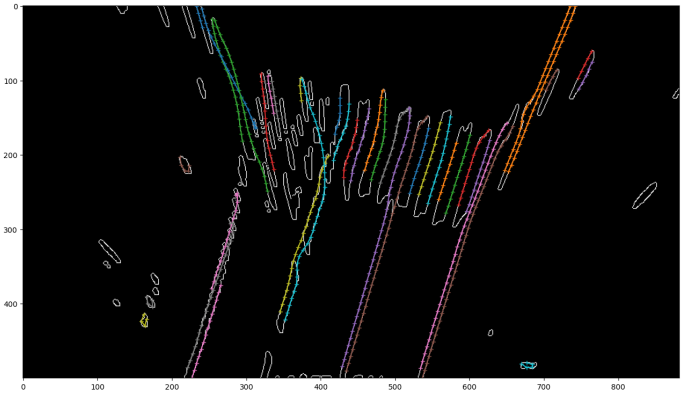
a) Détection de lignes

À ce stade, l'image a subi les prétraitements décrits par la section IV-A. L'essentiel de l'image contient ainsi seulement des contours d'un pixel de large, pour la plupart continus ou presque. Pour la suite des opérations, il est nécessaire d'extraire ces contours sous forme de courbes discrètes, soit une séquence de points ordonnés.

Un premier algorithme relativement simple extrait directement les contours sous forme de courbes discrètes, en prenant à chaque fois le premier pixel blanc encore non utilisé de l'image, et en extrayant de proche en proche la courbe auquel il appartient. Les surépaisseurs des contours sont gérées en prenant systématiquement le pixel 8-voisin le



(a) Visualisation de l'extraction des contours en courbes discrètes. Chaque couleur représente une courbe différente.



(b) Visualisation des courbes discrètes résultantes après filtrage. Ces courbes seront utilisées pour la sélection de la voie

FIGURE 9 – Visualisation de l'extraction et du filtrage des contours comme courbes discrètes

plus éloigné du précédent, et en supprimant tous les autres pixels adjacents. Ceci fait également que dans les diagonales, l'algorithme sélectionne prioritairement la diagonale plutôt que d'extraire un contour en créneau, ce qui donne des courbes plus lisses dès le départ.

Comme le montre la figure 9a, les courbes extraites ne sont pas encore exploitable pour détecter les marquages :

— Les courbes peuvent encore être en créneaux et ont un comportement local largement aléatoire

— Il peut manquer des points à certains contours, ce qui cause des discontinuités et éclate des contours en plusieurs courbes

— Les courbes extraites sont généralement des contours fermés, alors qu'une simple courbe permet une prise de décision plus simple

Pour cela, on applique des traitements supplémentaires à ces courbes avant de les exploiter.

Tout d'abord, les courbes sont ré-échantillonnées à un certain intervalle plus large que le pixel, puis lissées. Les tentatives initiales lissaient les courbes à l'aide d'un filtre gaussien sur les deux composantes X et Y. Malgré sa grande versatilité sur des simples valeurs, le filtre gaussien donne des résultats mitigés sur des informations géométriques. En effet, les courbes sont le plus souvent déformées et rétractées sur elles-mêmes.

La solution adoptée est un filtre de Savitzky-Golay [9] appliqué aux composantes en X et Y des courbes, qui correspond à une régression polynomiale par la méthode des moindres carrés sur une fenêtre autour de chaque point. Lorsque les points sont équidistants ou presque, ce filtre peut s'exprimer sous la forme d'un simple noyau de convolution. Ce filtre nous permet de réaliser approximativement le même type de lissage que la régression polynomiale utilisée précédemment, mais localement, donc tout en conservant le signal quel que soit sa forme, et d'une façon nettement plus performante. De surcroît, ce filtre conserve bien mieux la géométrie de la courbe que le filtre gaussien. Ici, le filtre correspondant à des polynômes du second degré s'est avéré tout à fait adapté.

Le filtre de Savitzky-Golay permet de conserver la géométrie de la courbe, mais rétracte tout de même les extrémités avec les méthodes de padding habituelles en bout de tableau. Cependant, comme il s'agit d'une courbe, il est très simple de l'étendre grâce aux vecteurs aux extrémités. Aux extrémités de la courbe, il suffit de générer des points grâce à ces vecteurs pour réaliser la convolution jusqu'au bout de la courbe sans la rétracter.

Ensuite, il faut réduire les contours fermés en courbes individuelles. Pour cela, on éclate les courbes en plusieurs parties, en coupant aux angles. On réalise cela en éliminant les parties qui ont une courbure supérieure à un certain seuil. La courbure est estimée grâce à l'angle entre le vecteur entrant et le vecteur sortant de chaque point, divisé par la longueur moyenne d'un segment de courbe, ce qui donne une courbure en radians par unité de longueur, plus facile à calculer et plus stable numériquement dans notre cas d'utilisation, qui correspond à l'inverse du rayon de courbure. L'angle relatif entre deux vecteurs s'exprime par la définition du produit scalaire, ce qui évite les effets de bord de la différence des arctangentes :

$$\begin{aligned} \vec{u} \cdot \vec{v} &= \|\vec{u}\| \times \|\vec{v}\| \times \cos(\theta) \\ \Leftrightarrow \theta &= \arccos \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|} \end{aligned} \tag{3}$$

D'où une courbure locale C_n :

$$C_n = \frac{2 \arccos \frac{\vec{v}_{n-1\to n} \cdot \vec{v}_{n\to n+1}}{\|\vec{v}_{n-1\to n}\| \times \|\vec{v}_{n\to n+1}\|}}{\|\vec{v}_{n-1\to n}\| + \|\vec{v}_{n\to n+1}\|} \tag{4}$$

Comme le comportement local des courbes reste relativement chaotique, un filtre gaussien est appliqué aux valeurs de courbure à chaque point de la courbe afin de les stabiliser avant le seuillage.

À ce stade, certaines courbes sont encore discontinues à cause de bruit dans l'image. Il est aussi intéressant de rejoindre des courbes sémantiquement identiques, comme les pointillés d'un marquage discontinu. On peut donc chercher à joindre les courbes qui peuvent l'être. Pour cela, pour chaque paire de courbe, on peut tout d'abord tenter de faire passer une droite par les points aux extrémités des deux courbes qui se font face.

On peut alors calculer une erreur RMS (Root Mean Squared Error), et joindre les paires avec une valeur d'erreur inférieure à un certain seuil.

Du fait de la nature géométrique du problème, une régression linéaire habituelle n'est pas satisfaisante pour la résolution de ce problème. En effet, les méthodes habituelles de régression linéaire sont adaptées pour le paramétrage d'une fonction du type $y = f(x)$. Les calculs d'erreur sont donc faits uniquement sur la valeur de la fonction donc la composante en Y, ce qui diminue à l'extrême la qualité du résultat à l'approche de la verticale, ou de l'horizontale dans le cas $x = f(y)$; alors que notre cas nécessite une distance orthogonale, sur les deux axes à la fois, et de pouvoir retrouver des droites de n'importe quelle orientation. Afin de pallier à ce problème, on passe plutôt par l'analyse en composantes principales (ACP). En effet, la composante principale est le vecteur directeur de la meilleure droite passant par les points. De plus, une fois les points transformés par l'ACP, leur distance orthogonale à cette droite est simplement l'écart à la moyenne sur la composante secondaire. Cette technique permet ainsi de résoudre le problème d'une manière extrêmement performante, sans résoudre le problème des moindres carrés directement.

La courbure des voies peut parfois rendre une jonction par une droite peu pertinente. On peut pour cela ajuster un arc de cercle sur ces mêmes points aux extrémités des courbes. La régression circulaire est normalement un problème non linéaire, qui nécessite des algorithmes itératifs pour résoudre les moindres carrés, ce qui aurait un poids rédhibitoire sur les performances étant donné que cela devrait être réalisé pour chaque paire de courbe de chaque image reçue.

Pour éviter ce problème, on utilise la méthode développée par Kåsa [10] en 1976, qui rend le problème linéaire et solvable de manière nettement plus performante. Les résultats sont différents d'une régression non linéaire, mais pour de simples arcs de cercles comme ici, cette méthode est tout à fait satisfaisante.

Si l'erreur RMS est inférieure à un certain seuil pour l'une ou l'autre de ces méthodes, les courbes sont jointes par une interpolation appropriée.

4) Sélection de la voie

À ce stade, on a généré un certain nombre de courbes discrètes pouvant être des contours de marquage au sol. Il reste ensuite à déterminer lesquelles correspondent à la voie à suivre. Pour cela, plusieurs solutions ont été envisagées :

— Sélectionner naïvement les courbes situées immédiatement à gauche et à droite du véhicule. Cette solution est satisfaisante en ligne droite quand les marquages sont nets, mais perd vite de son efficacité avec la courbure ou l'éloignement des marquages détectés

— Sélectionner la ligne de rive par des connaissances a priori, comme la position à droite du véhicule et sa continuité. Cette solution est plus stable quand le marquage droit est bien marqué, mais prive de l'information du marquage de gauche

— Calculer un score à partir de diverses informations sur chaque courbe, comme sa longueur, sa position, son angle et son éloignement. Ces valeurs ayant des grandeurs et des magnitudes très hétérogènes, il est difficile de les combiner de façon utile et cohérente.

De plus, ces solutions ne donnent aucun moyen sensé d'estimer la fiabilité de cette étape de détection et de sélection.

Finalement, la solution retenue est un système en logique floue, reprenant l'idée de la troisième solution avec un outil mathématique adapté.

a) Sélection et calcul des variables d'entrée

L'idée initiale était d'attribuer des variables à chaque courbe et de sélectionner les deux meilleurs candidats. cependant, cela néglige l'objectif fondamental qui est de sélectionner une *voie*, et non de simple marquages. Dans ce but précis, on peut calculer les variables pour chaque *paire* de courbes, et y adjoindre des informations supplémentaires bien plus discriminantes, comme le parallélisme et la largeur de la voie.

On calcule ainsi 5 variables d'entrée pour chaque paire de courbes :

— L'éloignement du marquage par rapport au véhicule. La transformation caméra inverse rendant les parties de l'image les plus éloignées de plus en plus floues, une courbe plus éloignée a d'autant plus de chances de correspondre à du bruit

— La distance relative à la position attendue du marquage. On calcule la distance de la courbe aux positions estimées des marquages gauche et droit, en sélectionnant la distance minimum parmi les deux combinaisons possibles des deux courbes. Les détails de ce calcul sont détaillés plus loin.

— La longueur de la courbe. Cette variable est peu discriminante mais permet de départager des courbes proches en terme de score, en sélectionnant la plus longue, qui a, en général, la meilleure qualité et le plus d'information.

— La distance relative des deux courbes. Pour cela, on réalise la projection orthogonale des points de l'une des deux courbes sur l'autre, pour récupérer la distance orthogonale à chaque point. On prend ensuite la moyenne de ces distances, à partir de laquelle on calcule une erreur à minimiser en prenant la différence avec la largeur attendue de la voie.

— Le parallélisme des deux courbes. Celui-ci est également calculé grâce à la projection orthogonale des points d'une courbe sur l'autre, cette fois en calculant l'angle entre les vecteurs correspondants sur les deux courbes. La variable résultante est la moyenne de ces angles relatifs, qu'il faut également minimiser

Pour les 3 premières variables, on doit combiner les valeurs correspondant aux deux courbes, typiquement par une moyenne arithmétique. Pour la distance aux marquages attendu, on calcule la distance maximum entre les deux courbes pour chaque combinaison (distance gauche pour la courbe 1, distance droite pour la courbe 2, et inversement), puis le minimum entre les résultats des deux combinaisons.

La distance relative aux marquages attendus présente quelques détails importants. Cette information est critique à la bonne sélection de la voie. En effet, les autres critères permettent de sélectionner les *voies*, mais ne permettent pas d'identifier la voie *en cours*. Ce paramètre doit donc avoir la meilleure estimation possible afin d'éviter les faux positifs et faux négatifs.

Initialement, les marquages attendus étaient simplement vus comme des droites allant droit vers l'avant du véhicule, située une demi-largeur de voie à gauche et à droite du véhicule. On calcule ensuite la distance du premier point de chaque courbe avec ces deux coordonnées sur l'axe X pour obtenir la variable. Cette solution est satisfaisante quand on dispose d'informations visuelles du sol suffisamment proches du véhicule, et quand la courbure est suffisamment faible, car le premier point de la courbe ne s'éloigne pas encore trop de ces positions. Cependant, avec des informations plus éloignées et en courbe, la position réelle des marquages peut s'éloigner considérablement de ces estimations naïves.

Pour cela, on généralise l'expression des marquages attendus. On commence par estimer l'angle des marquages grâce aux angles des segments de courbes situés dans une fenêtre devant le véhicule, afin de définir l'angle des marquages attendus. Comme beaucoup de courbes correspondent à du bruit ou des marquages transversaux, une simple moyenne sur ces angles serait fortement perturbée. Comme en général, les marquages de voies sont tous approximativement parallèles, ils auront tous un angle très proche, ce qui rend un calcul de densité très efficace, tel que montré par la figure 10. On peut ainsi récupérer l'angle des marquages attendus comme l'angle de densité maximale, ici calculée avec un noyau d'Epanechnikov.

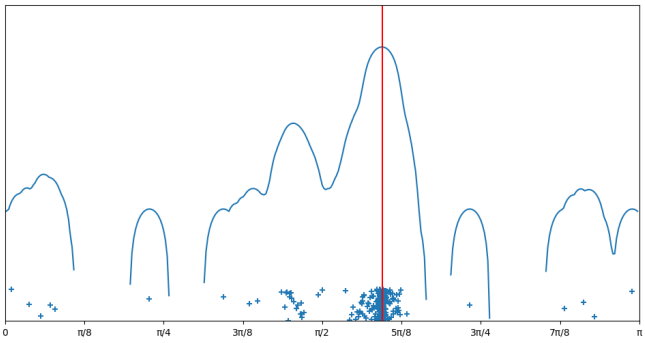


FIGURE 10 – Représentation de la densité des angles des segments de courbes présentées par la figure 9b sur une fenêtre proche du véhicule. Les points en bas du graphique représentent les angles de chaque segment de courbe (dispersés en hauteur simplement pour une meilleure visibilité), la courbe représente la densité calculée à l'aide d'un noyau d'Epanechnikov

On sait désormais qu'à la distance moyenne de la fenêtre utilisée pour extraire les angles, le marquage a approximativement l'angle de densité maximale que l'on notera α par rapport au véhicule. On peut donc estimer les marquages attendus comme des droites ayant cet angle α par rapport au véhicule, mais il faut également situer ces marquages dans le plan. Il serait possible de placer de représenter ces marquages avec l'angle α et des points d'origine fixes à gauche et à droite du véhicule, mais cela ne représenterait pas la géométrie la plus probable de la voie. Cette dernière serait plutôt une courbe, qui va tout droit au niveau du véhicule et tourne jusqu'à attendre l'angle α au niveau de la fenêtre, comme présenté par la figure 11.

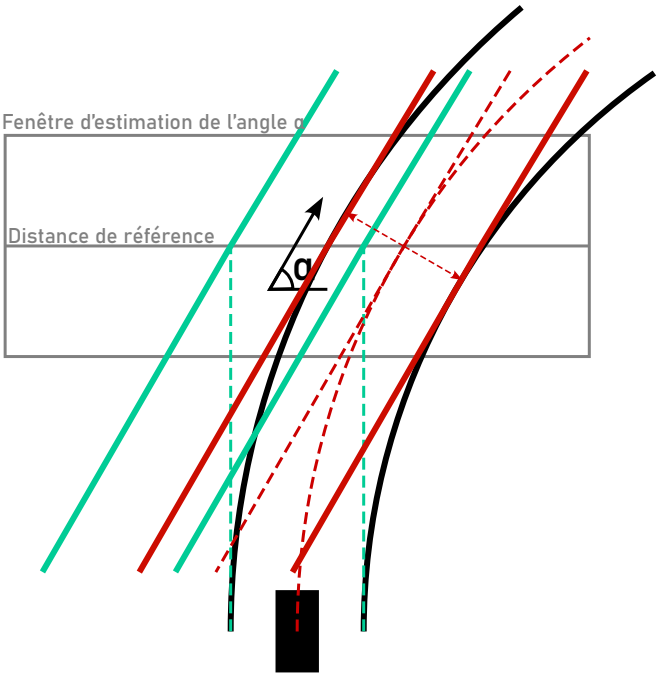


FIGURE 11 – Explication schématique de la construction du marquage attendu. En noir, la forme du marquage sur laquelle on base l'estimation. En vert pointillés, la construction de l'estimation naïve, et en vert, marquage attendu par l'approche naïve. Le vert pointillé donne aussi l'approche naïve sans angle. En rouge pointillé, la construction des marquages attendus en tenant compte de la courbure, et en rouge, les marquages attendus ainsi paramétrés

Pour estimer le marquage attendu dans cette situation, on a besoin de paramétrer un arc de cercle C, passant par l'origine du repère du véhicule, tel que son vecteur tangent est confondu avec l'axe Y au niveau du véhicule, et a l'angle α quand $y = \mu_d$, avec μ_d la distance de référence au milieu de la fenêtre d'estimation. Soient l'angle α l'angle tangent au cercle quand $y = \mu_d$, θ l'angle radial autour du centre du cercle et ρ le rayon du cercle, alors $\theta = \alpha + \frac{\pi}{2}$. Sachant que $x(\theta) = c_x + \rho\cos(\theta)$ et $y(\theta) = c_y + \rho\sin(\theta)$, on peut exprimer les contraintes du problème. Comme l'arc de cercle passe par l'origine du repère, pour une courbure vers la droite, le centre du cercle est à droite du véhicule et au même niveau sur l'axe Y, d'où :

$$\begin{aligned} x(\pi) = 0 &\Leftrightarrow c_x + \rho\cos(\pi) = 0 \\ &\Leftrightarrow c_x - \rho = 0 \\ &\Leftrightarrow c_x = \rho \end{aligned}$$

$$c_y = 0$$

$$\begin{aligned} y(\theta) = y\left(\alpha + \frac{\pi}{2}\right) &= \mu_d \\ \Leftrightarrow c_y + \rho\sin\left(\alpha + \frac{\pi}{2}\right) &= \mu_d \\ \Leftrightarrow \rho\cos(\alpha) &= \mu_d \\ \rho &= \frac{\mu_d}{\cos(\alpha)} \end{aligned}$$

On connaît ainsi le centre et le rayon de l'arc de cercle. Pour une courbure vers la gauche, il suffit de placer le centre du cercle à gauche du véhicule et d'utiliser $\theta = \alpha - \frac{\pi}{2}$.

À partir de là, on peut trouver les coordonnées du point $M(x(\theta), y(\theta))$. $y(\theta) = \mu_d$, et $x(\theta)$ découle directement :

$$\begin{aligned} x(\theta) &= c_x + \rho\cos(\theta) \\ &= \rho(1 + \cos(\theta)) \\ &= \rho\left(1 + \cos\left(\alpha + \frac{\pi}{2}\right)\right) \\ &= \rho(1 - \sin(\alpha)) \end{aligned}$$

De même, une courbure à droite donne $x(\theta) = \rho(\sin(\alpha) - 1)$

Ensuite, on peut estimer les deux droites correspondantes aux marquages attendus par des droites parallèles à la tangente au cercle en θ, à une distance orthogonale d'une demi-largeur de voie de celle-ci, comme montré par le schéma 11.

On peut s'interroger sur la pertinence d'utiliser des droites pour le marquage attendu plutôt que les arcs de cercle. Il ne faut pas oublier que ce passage par les arcs de cercle est simplement un moyen d'estimer la position des marquages attendus de façon plus cohérente avec la réalité. La courbure réelle de la voie est bien plus changeante et correspond rarement à un tel arc de cercle — en réalité, l'angle vient souvent d'une courbe serrée sur laquelle le véhicule se situe, mais qui redevient droite plus loin vers l'avant, là où on dispose de la vue du dessus. Les points de référence estimés par les arcs de cercles sont alors équivalents, mais pas les arcs de cercle eux-mêmes. De plus, le début des courbes candidates tombe souvent dans la fenêtre d'estimation, où un tel arc de cercle s'écarterait peu de la tangente. Les droites permettent donc d'estimer de façon satisfaisante dans plus de situations probables.

b) Règles d'inférence et variable de sortie

Pour le système en logique floue lui-même, on utilise une simple inférence par la méthode des hauteurs pondérées, qui s'est montrée tout à fait satisfaisante malgré sa simplicité.

On fuzzifie chaque variable d'entrée en 3 sous-ensembles flous par variable (bon, médiocre et mauvais), par des fonctions d'appartenance triangulaires et trapézoïdales entre des centres prédéfinis, qui elles aussi se sont montrées satisfaisantes malgré leur simplicité.

En sortie, on utilise les hauteurs pondérées pour 5 sous-ensembles de sortie, correspondant à des valeurs de la variable de sortie. Cette dernière est un score entre 0 et 1, 1 correspondant à une paire de courbes certaine d'être la voie en cours. Ce score permettra par la suite de réaliser des pondérations dans la composition de la trajectoire en fonction de la fiabilité de chaque courbe.

Avec 3 ensembles pour chacune des 5 variables, on aurait une base de $3^5 = 243$ règles d'inférences, qui seraient extrêmement fastidieuses à paramétrer. Pour éviter cela, on part d'un nombre entier, auquel on applique des malus fixes pour chaque sous-ensemble de chaque variable d'entrée. On obtient ainsi un indice que l'on fait correspondre au sous-ensemble flou de sortie. Ceci permet de paramétrer la base de règles par seulement 15 paramètres indépendants entre eux et tout à fait intuitifs, beaucoup plus facile à estimer et à maintenir, qui permettent de déduire la totalité de la base de règles. Ce système est de surcroît facilement extensible si on souhaite changer les variables à utiliser.

On obtient ainsi une paire de courbes correspondant à la voie, avec les marquages gauche et droit, à défaut un seul des deux marquages, ou parfois aucune donnée viable si même la sélection de marquage seul n'est pas satisfaisante.

Par rapport aux méthodes utilisées précédemment, cette solution permet en plus d'exploiter les deux marquages s'ils sont présents, avec des informations a priori nécessaires mais relativement limitées et génériques, et réussit à éliminer totalement la plupart des données qui mèneraient à des résultats aberrants.

c) Exploitation des résultats

Ce système en logique floue nous permet de calculer un score de fiabilité entre 0 et 1 d'une façon mathématiquement cohérente à partir de données hétérogènes. Grâce à ce score, on peut sélectionner la paire de courbes la plus susceptible de constituer la voie à suivre. Afin d'assurer la correspondance, on n'accepte ce score maximum que s'il est supérieur à un certain seuil prédéterminé.

Dans de nombreuses situations, le marquage de gauche peut être absent ou indétecté. C'est pourquoi, si on ne trouve pas de paire de courbes, on utilise ce même système en logique floue sur les courbes individuelles, sans tenir compte des variables traitant des paires, pour estimer les scores individuels de chaque courbe, et ainsi être capable de détecter un marquage seul de façon relativement fiable, comme la ligne de rive sans marquage de gauche.

De plus, même avec une voie complète, on calcule les scores des marquages individuels afin de pondérer la composition de la trajectoire pour chaque marquage individuellement par la suite.

On obtient ainsi une voie complète soit les deux marquages de gauche et de droite, à défaut un seul des deux marquages, ou éventuellement aucune donnée si même la sélection de courbe individuelle n'a donné aucun résultat viable.

Par rapport aux méthodes utilisées précédemment, cette solution permet en plus d'exploiter les deux marquages s'ils sont présents, avec des hypothèses a priori nécessaires mais relativement limitées et génériques, et réussit à éliminer totalement la plupart des données qui mèneraient à des résultats aberrants tout en conservant un maximum d'information.

5) Détection des autres marquages

Une tentative de détecter les autres marquages et les lignes pointillées a été effectué, notamment pour essayer de détecter les intersections sur la base des passages piétons. Cette tentative n'a mené à aucune exploitation concrète, mais l'extension de la méthode utilisée pourrait permettre de compléter la procédure de détection des voies susmentionnées. Cela dit, la détection des passages piétons selon la méthode ci-après donne des résultats et permet d'éliminer les passages piétons avant la détection de la voie, ce qui permet de réduire le bruit à proximité de tels marquages.

Pour la détection de passages piétons, on se base sur les courbes discrètes extraites des contours, avant tout autre traitement. L'idée générale est d'ajuster des rectangles sur ces contours, vérifier s'ils y correspondent bien, puis grouper les rangées de rectangles similaires, qui sont pratiquement toujours des passages piétons, et qui n'interfèrent de toute manière pas avec la détection des voies.

Pour l'ajustement d'un rectangle à un contour peut se faire par la méthode des moindres carrés, en partant du minimum des distances entre le point et chacun des côtés du rectangle, soit une procédure approximativement équivalente à 4 régressions linéaires. Cependant, cette formulation est lourde et relativement complexe. Pour éviter ce problème, on réalise une ACP sur les points du contour. Les points étant alors tournés par rapport à l'angle principal du contour, les côtés du rectangle sont de simples droites verticales et horizontales par rapport aux vecteurs propres. Il suffit ainsi de diviser ces axes en une grille, de réaliser un histogramme le long des deux axes, et de placer les côtés du rectangle là où il y a le plus de points sur chacun des axes. On calcule ensuite une erreur RMS pour estimer si le contour correspond bien au rectangle, et on peut récupérer les paramètres du rectangle dans l'espace d'origine par la transformation inverse à l'ACP.

Un passage piéton est une rangée de rectangles ayant approximativement la même taille, le même angle, et disposés avec leur plus grand côté face à face, donc ayant leur composante secondaire de l'ACP alignée. On peut donc réaliser un clustering par l'algorithme DBSCAN [11] sur 4 paramètres convenablement normalisés :

— Longueur et largeur du rectangle

— Angle principal

— Projection du centre du rectangle sur la composante principale, qui doit être similaire entre tous les rectangles du groupe

Un tel clustering permet d'identifier des groupes de rectangles correspondant à un passage piéton.

On pourrait également effectuer un tel clustering avec la projection sur la composante secondaire afin de repérer des rectangles continus dans la longueur, qui correspondraient à des marquages de voie discontinus.

C. Estimation de la trajectoire

À ce stade, les marquages de voie ont été détectés sur l'image, et il reste à en déduire une trajectoire pour le véhicule, toujours sous forme d'une courbe discrète dans le plan de la route. Cette partie est séparée en deux étapes :

— Estimation d'une trajectoire pour l'image en cours à partir des marquages détectés

— Composition de la trajectoire finale à partir d'un historique des dernières trajectoires locales

Par ailleurs, le système en logique floue donne un score entre 0 et 1 à chaque courbe, très pertinent pour le filtrage et la pondération de ces deux étapes. Celui-ci permet de calculer un score par marquage individuel, cependant il est plus intéressant d'avoir un score par point de chaque courbe, afin d'appliquer ces pondérations pour chaque point indépendamment sur plusieurs compositions d'affilée.

Pour cela, en supposant que plus les points sont éloignés, moins il sont fiables car plus l'image est floue et l'information est parcellaire, on applique simplement une décroissance exponentielle sur le score du marquage en fonction de la distance $D(t)$ le long de la courbe discrète $C(t)$, pour t l'index d'un point de la courbe, S le score global de la courbe et $S(t)$ le score du point t :

$$D(t) = \sum_{i=0}^{t-1} \|C(i+1) - C(i)\|$$

$$S(t) = S \cdot max(0, 1 - e^{\frac{D(t)-R}{a}})$$

R est une portée au-delà de laquelle les scores vaudront toujours 0, et a est un paramètre de vitesse de diminution des scores avec la distance.

1) Composition de courbes discrètes

La composante principale de ces deux étapes est un algorithme permettant de composer une courbe discrète à partir des informations données par plusieurs courbes discrètes. Cet algorithme est itératif et calcule la moyenne des points de chaque courbe à intervalles réguliers.

Soient $C_0, ..., C_i, ..., C_n$ n courbes discrètes à combiner, et C_f la courbe résultante à construire. T est l'incrément de distance à chaque itération.

On initialise le premier point de C_f avec une valeur bien choisie, typiquement un point situé quelques mètres droit devant le véhicule. Commencer à $(0, 0)$ peut mener à des décalages latéraux très proches du véhicule dans la trajectoire finale qui produiraient des à-coups dans le contrôle.

À chaque étape $t \geq 1$, on réalise les étapes suivantes :

— Trouver les points d'intersection de la droite perpendiculaire au dernier segment de C_f avec chacune des courbes C_i, en étendant éventuellement la courbe vers le véhicule mais pas au loin.

— Calculer les points situés à une distance T du point d'intersection le long de chacune des courbes C_i. On appellera ces points $P_i(t)$. On peut éventuellement éliminer les points ayant un score trop bas.

— Calculer le centroïde de ces points par une simple moyenne arithmétique des points $P_i(t)$ pondérée par les scores associés à chacun.

— Calculer les distances des points $P_i(t)$ au centroïde.

— Réaliser un filtrage par l'écart interquartile sur les distances au centroïde. Avec $Q_1(t)$ et $Q_3(t)$ les premier et troisième quartiles de la série des distances au centroïde, on élimine tous les points $P_i(t)$ dont la distance est supérieure à $Q_1 + k(Q_3 - Q_1)$. Empiriquement, la valeur du facteur k a été déterminée à 1,5.

— On calcule le nouveau centroïde pondéré des points restants

— Ce centroïde est ajouté au bout de la courbe combinée C_f

— On y associe un score par une expression similaire à une formule de combinaison d'incertitudes explicitée par l'équation (5), où $S_i(t)$ est le score associé au point $P_i(t)$ et m le nombre de points $P_i(t)$ restants après le filtrage.

La figure 12 offre une visualisation d'une itération de cet algorithme.

Pour le calcul du deuxième point de C_f, il n'y a pas encore de segment sur lequel baser la droite perpendiculaire de la première étape. Pour récupérer les points $P_i(t)$ à cette étape, on peut passer par une projection orthogonale du dernier point de C_f sur les courbes $C_i(t)$. La projection orthogonale était auparavant utilisée tout le long de la courbe, mais donne des résultats moins stables et présente beaucoup plus de risque de convergence.

$$S_f(t) = 1 - \frac{\sqrt{\sum_{i=0}^{m}(1 - S_i(t))^2}}{m} \tag{5}$$

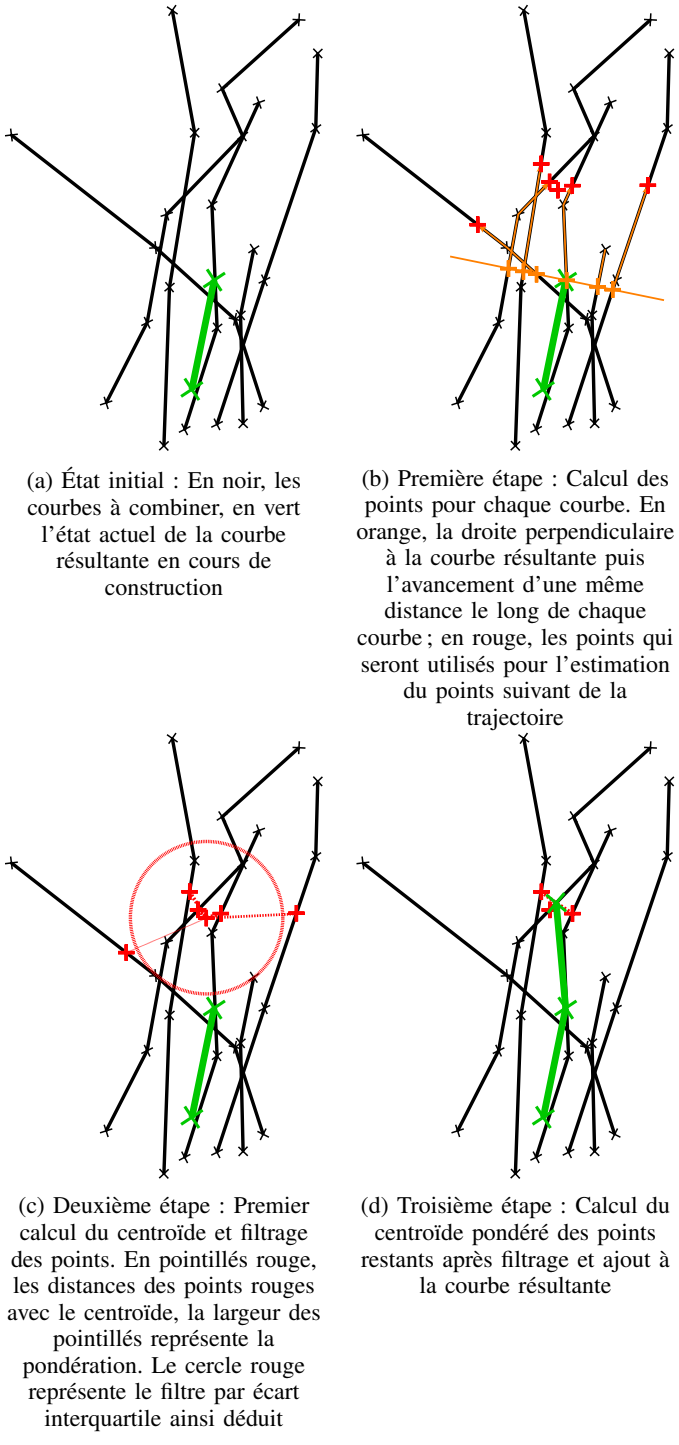
On arrête l'itération lorsque le calcul des points $P_i(t)$ ne donne plus aucun point valide (la courbe C_f est arrivée plus loin que toutes les courbes de départ, ou le filtrage a éliminé tous les points)

a) Convergence de l'algorithme

Lorsque des courbes de départ sont mal définies, ou bifurquent vers des directions opposées, il arrive que le centroïde des points $P_i(t)$ converge, ce qui crée une boucle infinie. Pour éviter cela, on doit mettre en place au moins deux conditions de sortie supplémentaires pour deux types de convergence :

— Pour éviter la convergence directe (les centroïdes convergent vers un point unique), on sort de la boucle si la distance entre deux points successifs de la courbe résultante C_f est inférieure à un certain seuil, typiquement une fraction de T

— Pour éviter la convergence oscillante (les centroïdes convergent vers une oscillation entre plusieurs points distincts, ce qui peut arriver quand plusieurs courbes prennent des directions opposées), on sort de la boucle quand l'angle entre les deux derniers segments de la courbe résultante C_f est plus aigu qu'un certain seuil.



(a) État initial : En noir, les courbes à combiner, en vert l'état actuel de la courbe résultante en cours de construction

(b) Première étape : Calcul des points pour chaque courbe. En orange, la droite perpendiculaire à la courbe résultante puis l'avancement d'une même distance le long de chaque courbe ; en rouge, les points qui seront utilisés pour l'estimation du points suivant de la trajectoire

(c) Deuxième étape : Premier calcul du centroïde et filtrage des points. En pointillés rouge, les distances des points rouges avec le centroïde, la largeur des pointillés représente la pondération. Le cercle rouge représente le filtre par écart interquartile ainsi déduit

(d) Troisième étape : Calcul du centroïde pondéré des points restants après filtrage et ajout à la courbe résultante

FIGURE 12 – Visualisation de l'algorithme de composition de courbes discrètes

2) Estimation locale de la trajectoire

Les marquages ont été estimés sous forme d'une ou deux courbes discrètes. La première étape est de créer une estimation de la trajectoire avec les informations d'une seule image. Pour cela, on crée une première estimation du centre de la voie à partir de chacun des deux marquages, que l'on combine par l'algorithme décrit en IV-C1.

Pour estimer le centre de la voie à partir d'un des mar-

quages, on doit décaler la courbe d'une demi-largeur de voie. Il existe plusieurs techniques simples pour effectuer une telle opération :

— Utiliser l'algorithme IV-C1 directement sur les marquages pourrait fonctionner, mais la pondération par les scores fait que la courbe résultante serait attirée par le marquage le plus certain, et non le centre réel de la voie

— Décaler les points par un vecteur fixe, typiquement le long de l'axe X. Cette méthode ne fonctionne pas du tout en courbe du fait du rétrécissement que cela engendrerait

— Décaler les points de la courbe par le vecteur orthogonal à la courbe en chaque point. Cette méthode est la meilleure, car les points de la courbe résultante sont bien toujours à la même distance des points d'origine. Cependant, la courbure peut créer des intersections dans la courbe résultante, en particulier quand il y a des écarts dans le marquage détecté, ce qui fait échouer l'algorithme de composition de courbes.

Pour décaler la courbe, on applique ainsi cette dernière méthode : chaque point de la courbe est décalé le long du vecteur orthogonal à la courbe en ce point d'une demi-largeur de voie. On effectue ensuite une passe de suppression des auto-intersections dans la courbe, où on coupe simplement les sections de la courbe situées dans une intersection, comme montré par la figure 13. Après avoir effectué ces opérations sur chacun des marquages, l'algorithme IV-C1 est appliqué dessus pour obtenir l'estimation de trajectoire.

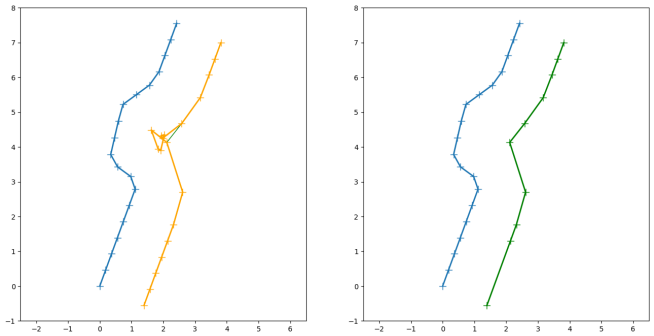


FIGURE 13 – Représentation des résultats de l'algorithme de décalage de courbe discrète. En bleu, la courbe d'origine, en orange la courbe après décalage par les vecteurs orthogonaux, en vert la courbe décalée après suppression des intersections

a) Détail de la suppression des auto-intersections

L'algorithme de suppression des auto-intersections développé crée un masque des points de la courbe. On vérifie pour chaque paire de segments de la courbe si ceux-ci s'intersectent. Pour A et B les extrémités du premier segment AB et C et D les extrémités du second segment CD :

$$\Delta A = \det \begin{pmatrix} C_x - A_x & D_x - A_x \\ C_y - A_y & D_y - A_y \end{pmatrix}$$

$$\Delta B = \det \begin{pmatrix} C_x - B_x & D_x - B_x \\ C_y - B_y & D_y - B_y \end{pmatrix}$$

$$\Delta C = \det \begin{pmatrix} A_x - C_x & B_x - C_x \\ A_y - C_y & B_y - C_y \end{pmatrix}$$

$$\Delta D = \det \begin{pmatrix} A_x - D_x & B_x - D_x \\ A_y - D_y & B_y - D_y \end{pmatrix}$$

Si ΔA et ΔB sont de signes différents et ΔC et ΔD sont de signes différents, alors les segments AB et CD s'intersectent.

Si les segments AB et CD s'intersectent, alors le masque est mis à zéro entre les points B et C inclus. Au final, tous les points situés entre des segments qui s'intersectent seront supprimés, coupant ainsi toutes les sections de la courbe qui s'intersectent comme montré par la figure 13.

3) Combinaison de l'historique des trajectoires

Afin de mieux stabiliser la trajectoire et d'éviter d'envoyer des trajectoires aberrantes à l'algorithme de contrôle du véhicule, on la stabilise en la combinant avec un historique des dernières trajectoires locales.

On transforme alors les dernières trajectoires vers le repère local grâce au service décrit en II, afin que toutes ces trajectoires soient dans un même repère. Elles sont ensuite combinées par le même algorithme IV-C1 en une trajectoire unique qui est alors envoyée au contrôle si elle est valide et satisfaisante. Afin d'éviter des à-coups dans le trajet du véhicule quand la trajectoire présente des angles trop aigus, on peut supprimer les points correspondants pour adoucir les angles, puis lisser cette trajectoire par un filtre de Savitzky-Golay avec une large fenêtre.

D. Navigation en intersection

1) Détection des intersections

Le problème de la détection et de la navigation des intersections est encore largement irrésolu et nécessitera une attention particulière à l'avenir. En l'état, aucune solution satisfaisante n'a été trouvée pour détecter les intersections de manière fiable de manière autonome avec le matériel à notre disposition. En particulier, le champ de vision disponible est trop étroit pour voir sur les côtés du véhicule, ce qui rend une identification des intersections suffisamment discriminante presque impossible. Dans cette situation, les solutions suivantes ont été tentées sans suite :

— Détecter une soudaine augmentation de la courbure du marquage de droite. Cette solution s'est avérée extrêmement peu discriminante au vu des changements de courbure existants hors intersection, menant à beaucoup de faux positifs.

— Se baser sur les autres types de signalisation comme les panneaux trouvés habituellement devant les intersections, les passages piétons, etc. Cette méthode peut fonctionner, quoique de manière imprécise, dans le contexte particulier du simulateur à notre disposition, mais serait peu généralisable.

Globalement, le repérage et la navigation fiables d'une intersection semblent difficiles sans pouvoir distinguer les autres voies sortant de l'intersection sur les côtés sur l'image. Il serait intéressant d'aborder le problème avec un plus grand angle de vue ou plusieurs caméras permettant de voir sur les côtés du véhicule.

En l'état, un remplacement pouvant fonctionner sur la base d'un positionnement et d'une cartographie grand public a été mis en place afin de permettre de tester le reste du système, mais n'a jamais été prévu comme une solution définitive.

2) Navigation des intersections

Ce même manque de visibilité rend aussi difficile la construction d'une trajectoire à travers une intersection. Nous avons choisi de nous focaliser sur des carrefours à angle droit, bien que certaines des solutions présentées soient extensibles à d'autres situations. Pour pallier au manque d'informations disponibles dans l'intersection même, on précalcule une trajectoire à l'entrée de l'intersection, et on arrête la détection de voie jusqu'à arriver à une certaine distance de l'entrée de l'intersection. On réactive alors le suivi de voie, qui tente de se raccrocher à la voie d'arrivée puis de la suivre et de reprendre le fonctionnement de croisière. La détection de voie doit être arrêtée dans l'intersection car elle pourrait reprendre le dessus au mauvais moment : typiquement, dans un virage à gauche, le véhicule doit d'abord avancer dans l'intersection, assez loin pour pouvoir détecter la voie d'en face qui n'est pourtant pas la voie d'arrivée voulue.

a) Suivi de voie

La procédure de construction de trajectoire présentée plus haut est faite à la base pour suivre une voie. Cependant, sa capacité à fonctionner sur la base d'un seul marquage lui permet également de suivre la ligne de rive dans une intersection, et donc d'assurer les virages à droite. Cependant, sa propension à perdre de vue les marquages à la fois fortement courbés et très proches du véhicule a mené au développement d'une trajectoire précalculée même pour les virages à droite.

b) Traversée rectiligne d'intersection

Pour traverser une intersection tout droit, la trajectoire est simplement une ligne droite dans la continuité du déplacement du véhicule, jusqu'à rattraper la voie d'arrivée.

c) Virage à droite

Pour le virage à droite, on exploite la courbure de la voie pour construire une trajectoire en arc de cercle. Pour cela, on utilise l'historique des trajectoires, en calculant la courbure de densité maximale parmi les segments de courbe situés devant le véhicule, par la même formule (4) puis par un calcul de densité avec un noyau d'Epanechnikov. Celle-ci étant exprimée en rad/m, son inverse donne le rayon de courbure. Il suffit alors de construire une trajectoire en arc de cercle de ce rayon vers la droite.

d) Virage à gauche

Le virage à gauche est un problème particulièrement complexe dans notre cas, car nous n'avons généralement aucune information sur la gauche de la route. Il est donc pratiquement impossible de résoudre cela de manière fiable. Afin d'avoir tout de même une solution, on se base sur la même technique que le virage à droite. En supposant que le virage à gauche aura la même courbure que le virage à droite, on utilise la même méthode, en plaçant l'arc de cercle une largeur de voie plus loin pour espérer se raccrocher à la bonne voie à l'arrivée, et en le faisant tourner à gauche. Cette méthode s'est avérée moyennement fiable, et ne fonctionne pas dans les intersections sans possibilité de tourner à droite.

e) Résultats de la navigation en intersection

Les résultats de ces méthodes de navigation en intersection sont globalement peu probants et peu généralisables. Là où un conducteur humain tournerait la tête pour étendre son champ de vision, il serait nécessaire d'ajouter des caméras pour voir sur les côtés du véhicule. Il serait éventuellement possible de jouer sur l'intensité des points renvoyés par le LiDAR, mais cette méthode semble lourde et peu fiable, surtout à grande distance. Le problème des intersections reste donc largement irrésolu et nécessiterait de plus amples investigations, sur le plan algorithmique autant que matériel.

V. Contrôle du véhicule

A. Interprétation des trajectoires

Comment contrôler la commande de direction du véhicule afin de suivre une trajectoire donnée sans osciller le long de cette dernière ?

1) Algorithme Pure Pursuit : Théorie

Pure Pursuit [12] est un algorithme de suivi de trajectoire. Il calcule la commande de vitesse angulaire qui va déplacer un système (ici un véhicule autonome) de sa position actuelle pour atteindre un point d'anticipation placé devant lui. La vitesse linéaire est supposée constante, On peut donc modifier la vitesse du véhicule à tout moment. L'algorithme déplace ensuite le point d'anticipation sur la trajectoire en fonction de la position actuelle du véhicule jusqu'au dernier point de la trajectoire. On peut donc considérer que le véhicule poursuit constamment un point situé devant lui comme, pour imager, un chat poursuivrait une souris. La propriété de distance de vision (ou distance d'anticipation) détermine la distance à laquelle le point d'anticipation est placé.

La distance de vision est le paramètre principal du contrôleur. Cette propriété indique à quelle distance de sa position actuelle le véhicule doit regarder pour calculer les commandes de vitesse angulaire.

L'effet de la modification de ce paramètre peut changer la façon dont le véhicule va suivre la trajectoire. Il y a deux objectifs principaux lorsqu'on implémente un algorithme Pure Pursuit : retrouver la trajectoire et maintenir cette dernière. Afin de retrouver rapidement la trajectoire entre les points de passage, une petite distance d'anticipation va permettre au

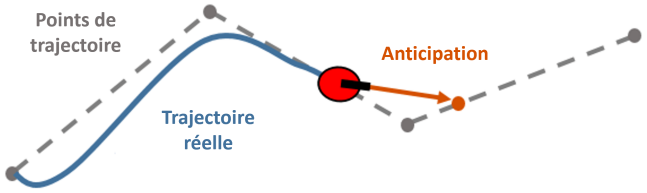


FIGURE 14 – Algorithme Pure Pursuit. Le cercle rouge correspond au véhicule et devant, le point d'anticipation. Notons que la trajectoire réelle ne correspond pas à la ligne directe entre les points de cheminement.

véhicule de se déplacer rapidement vers la trajectoire. Cependant, cela peut amener le véhicule à dépasser la trajectoire et osciller le long de la trajectoire souhaitée. Afin de réduire les oscillations le long de la trajectoire, une distance d'anticipation plus importante peut être choisie, mais elle peut entraîner des courbures plus importantes à proximité des virages ce qui n'est pas non plus souhaitable.

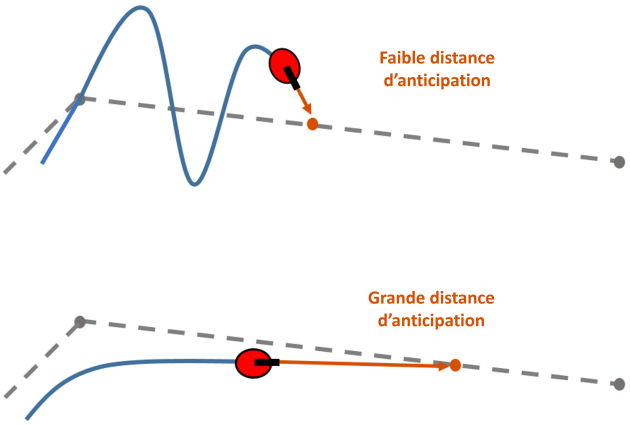


FIGURE 15 – Différents comportements du véhicule en fonction de la distance d'anticipation.

Le but est donc de trouver le bon compromis face à ces deux derniers points et ainsi de trouver la distance d'anticipation optimale. Des vitesses linéaires et angulaires différentes d'un problème à un autre affecteront également le comportement du véhicule et devront être prises en compte pour le paramétrage du contrôleur de suivi de trajectoire.

2) Algorithme Pure Pursuit : Implémentation

Comme le repère est local et en mouvement, et comme la trajectoire à suivre évolue constamment, nous avons du adapter le schéma de l'algorithme Pure Pursuit de base. La démarche utilisée pour implémenter notre contrôleur est décrite dans cette section.

La première étape est l'initialisation du contrôleur. Lorsque le contrôleur reçoit une nouvelle trajectoire, ce dernier vérifie dans un premier temps qu'il dispose d'une vitesse linéaire et d'une vitesse angulaire afin d'initialiser l'état du véhicule par la suite. Les points de la trajectoire reçue sont d'abord transformés vers un timestamp cible correspondant au timestamp actuel du contrôleur par le service décrit en section II, ce qui permet au contrôleur de prendre en compte la différence temporelle entre le moment où la trajectoire a été détectée et le moment où elle est traitée par le contrôleur. Une fois la matrice de transformation obtenue et appliquée aux points 2D de la trajectoire convertis en coordonnées homogènes, l'ancienne trajectoire connue est alors remplacée par celle reçue. Ensuite, l'état du véhicule est réinitialisé : sa position dans le repère 2D local est fixée à $x = 0$ et $y = 0$, sa vitesse linéaire ainsi que sa vitesse angulaire (autour de l'axe z dans l'espace 3D) sont initialisées. Enfin, le premier point d'anticipation du véhicule est recherché sur la trajectoire.

La deuxième étape applique l'algorithme Pure Pursuit afin de déterminer la commande de vitesse angulaire du véhicule. En même temps, la commande de vitesse linéaire est calculée en utilisant un gain permettant d'atteindre une vitesse cible. Ensuite, soit une nouvelle trajectoire est reçue par le contrôleur, au quel cas ce dernier retourne à la première étape d'initialisation, soit l'état du véhicule est mis à jour et un nouveau point d'anticipation est recherché en prenant en compte la vitesse linéaire et la distance d'anticipation choisie. Pour mettre à jour l'état, comme à l'initialisation, les données réelles de vitesse sont utilisées et non des vitesses approximées qui engendreraient des erreurs de trajectoire dès que les mises à jour s'enchaîneraient. Pour obtenir la position du véhicule, les vitesses linéaires, angulaire ainsi que la fréquence de rafraîchissement du contrôleur sont utilisées. La fréquence de rafraîchissement a été fixée à 10Hz dans le cadre de notre projet.

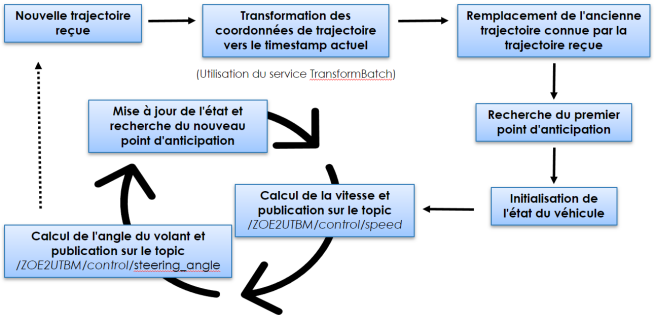


FIGURE 16 – Algorithme Pure Pursuit avec utilisation d'un repère local basé sur le véhicule et une trajectoire dynamique.

B. Interprétation de la signalisation verticale détectée

Dans le cadre de ce projet, nous avons sélectionnés seulement quelques panneaux pour lesquels le contrôleur prend des décisions. On divise ces panneaux en deux classes.

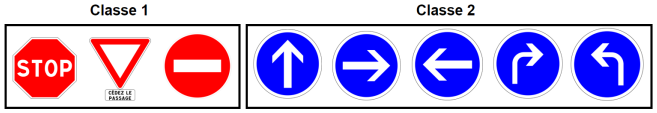


FIGURE 17 – La première classe correspond aux panneaux nécessitant l'arrêt du véhicule. La seconde classe correspond aux panneaux qui imposent une direction à la voiture.

Dans le cas des panneaux de la première classe, étant donné que la distance du panneau détectée est transmise au contrôleur, ce dernier va faire en sorte que le véhicule freine linéairement jusqu'au panneau et attendra ensuite une action du conducteur pour repartir. Pour le freinage linéaire, le contrôleur va tout d'abord calculer le nombre de valeurs qu'il faudra transmettre à la commande de vitesse en fonction de la distance du panneau, de la vitesse du véhicule et de la fréquence de rafraîchissement du contrôleur. Ensuite, les vitesses calculées sont transmises jusqu'à l'arrêt du véhicule.

Soit n le nombre de valeurs à transmettre jusqu'à l'arrêt complet du véhicule et $v(t)$ la vitesse (en m/s) à transmettre à l'instant t avec $t \in [1, n]$. Soit d la distance (en m) et f la fréquence de rafraîchissement du contrôleur (en Hz). Soit v_0 la vitesse du véhicule lorsque le panneau est détecté.

$$n = \frac{d}{v_0} f$$

$$v(t) = \frac{v_0}{n}(n - t)$$

Dans le cas des panneaux de la seconde classe, le contrôleur va transmettre au noeud qui gère les trajectoires la direction imposée par le panneau détecté afin de recevoir la bonne trajectoire à la prochaine intersection.

VI. Conclusion

Ainsi, ce projet a mené au développement de l'essentiel des modules nécessaires à la circulation autonome d'un véhicule. La détection de la signalisation et du marquage au sol, la construction d'une trajectoire et le contrôle du véhicule sont fonctionnels, la communication entre eux également.

La détection de la signalisation verticale est réalisée par un réseau de neurones profond de type YOLOv4-Tiny, puis la classification par une réseau LeNet-5, qui permettent des performances élevées et une grande précision. Leur distance est ensuite estimée grâce au LiDAR.

La construction de trajectoire se base sur la détection des marquages au sol comme des courbes discrètes, classifiés par un système en logique floue puis combinés sur plusieurs images pour stabiliser la trajectoire obtenue.

Enfin, le contrôle du véhicule est assuré par un algorithme de type Pure Pursuit adapté à un repère local en mouvement et à une trajectoire régulièrement mise à jour.

Certains points sont toutefois irrésolus, en particulier la navigation en intersection. La stabilisation de la trajectoire reste également à améliorer. Les méthodes développées au cours de ce projet donnent des résultats satisfaisants en environnement simulés sans obstacles ni autres usagers, dont un suivi de voie et une réaction à la signalisation verticale très efficaces. Dès lors, ces techniques pourraient être complétées et améliorées pour s'adapter à un environnement réel plus complexe. Par exemple, la détection de voie pourrait passer par le deep learning pour mieux s'adapter aux occultations. De plus, ces méthodes pourraient être avantageusement complétées par un positionnement et une cartographie grand public, qui permettraient de mieux anticiper la structure de l'environnement tout en conservant un comportement local purement autonome, résolvant ainsi à la fois les problèmes matériels et financiers du positionnement haute résolution et le manque d'information général lié à l'utilisation d'informations strictement locales dans un environnement aussi complexe.

Références

[1] S. O. H. Madgwick, Apr 2010. [Online]. Available : https://x-io.co.uk/downloads/madgwick_internal_report.pdf

[2] M. Boyle, "The integration of angular velocity," *Advances in Applied Clifford Algebras*, vol. 27, no. 3, pp. 2345–2374, may 2017. [Online]. Available : https://doi.org/10.1007%2Fs00006-017-0793-z

[3] R. G. A. F. Joseph Redmon, Santosh Divvala, "You only look once : Unified, real-time object detection," 2016.

[4] A. Karovalia, "Traffic-signs-detection-tensorflow-yolov3-yolov4," 2021. [Online]. Available : https://github.com/Alzaib/Traffic-Signs-Detection-Tensorflow-YOLOv3-YOLOv4

[5] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer : Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, no. 0, pp. –, 2012. [Online]. Available : http://www.sciencedirect.com/science/article/pii/S0893608012000457

[6] J. Basnet, "Traffic-sign-classification-using-convnets4," 2020. [Online]. Available : https://github.com/junthbasnet/Traffic-Sign-Classification-using-ConvNets

[7] C. Mei, "Projection model," 2006. [Online]. Available : http://www-sop.inria.fr/icare/personnel/Christopher.Mei/articles/projection_model.pdf

[8] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, no. 6, p. 381–395, 1981.

[9] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical Chemistry*, vol. 36, no. 8, p. 1627–1639, 1964.

[10] I. Kasa, "A circle fitting procedure and its error analysis," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-25, no. 1, p. 8–14, 1976.

[11] J. S. M. Ester, H. P. Kriegel and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." Portland, OR : AIII Press, 1996, pp. 226–231.

[12] M.-W. Park, S.-W. Lee, and W.-Y. Han, "Development of lateral control system for autonomous vehicle based on adaptive pure pursuit algorithm," pp. 1443–1447, 2014.