Rapport final - SY27

Adrien Hervé, Ahamed Mohamed, Anais Bourguignon,
Manon Desjardins, Samuel Mejia Vallecilla, Victor Billaud

Automne 2023

Sommaire

Introduction

Ce rapport d'études résume le travail effectué par l'équipe 1 lors du projet d'automne 2023 de l'UV SY27 : « Machines intelligentes ».

Il a été rédigé dans le but de rendre compte de l'approche que nous avons choisi d'adopter afin de répondre au mieux au sujet d'étude de cette année.

Il détaillera entres autres l'organisation générale du projet ; le sujet du projet, l'angle d'attaque que nous avons choisi et la décomposition de tâches que nous avons adoptée, ainsi chacune des parties dans leurs détails : l'intégration, la perception, la fusion de données, la correction de la carte et le contrôle du véhicule. Enfin, il sera question de conclure quant à l'état du projet et quant aux résultats obtenus.

1 Organisation du projet

Dans cette partie, il sera question de traiter du sujet du projet, l'approche que nous avons décidé d'adopter ainsi que la décomposition de tâches en découlant.

1.1 Le sujet du projet

L'objectif principal du projet est de faire naviguer de manière autonome un véhicule sur la piste *Séville*[1] à l'aide d'une localisation basée sur l'utilisation de panneaux en guise d'amers.

Pour se faire, deux cartes sont fournies : une carte contenant les coordonnées (x, y) des points de la piste et une carte qui contient les coordonnées et l'orientation (x, y, θ) des panneaux de la piste.

En revanche, la carte contenant les panneaux présente des erreurs et le but du système développé est de la corriger afin d'avoir une localisation la plus précise possible.

Techniquement, cela se découpe en deux phases : une phase d'apprentissage pendant laquelle la voiture est contrôlée « manuellement » (c'est-à-dire par le conducteur de sûreté) et une phase de conduite autonome pendant laquelle la localisation est estimée précise et la voiture peut naviguer sans aide extérieure. Cela se traduit en pratique par un changement de phase : le système doit savoir lorsque la localisation est assez précise et doit l'indiquer au conducteur de sûreté.

1.2 L'approche adoptée

Notre approche se schématise ainsi :

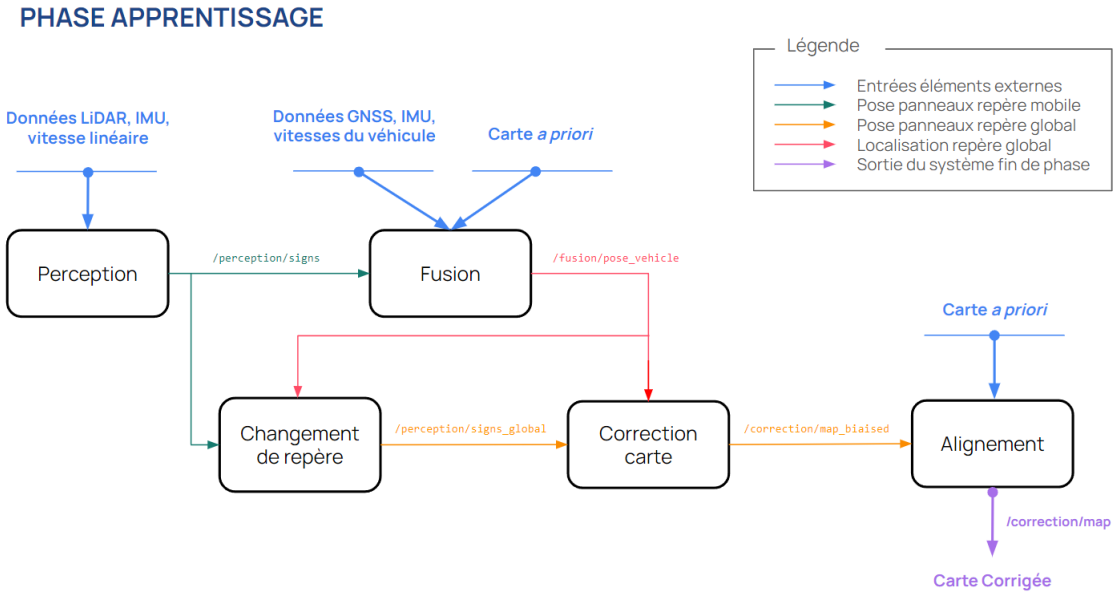

FIGURE 1 – Schéma de l'approche générale du projet en phase d'apprentissage

1. Piste appartenant au laboratoire Heudiasic.

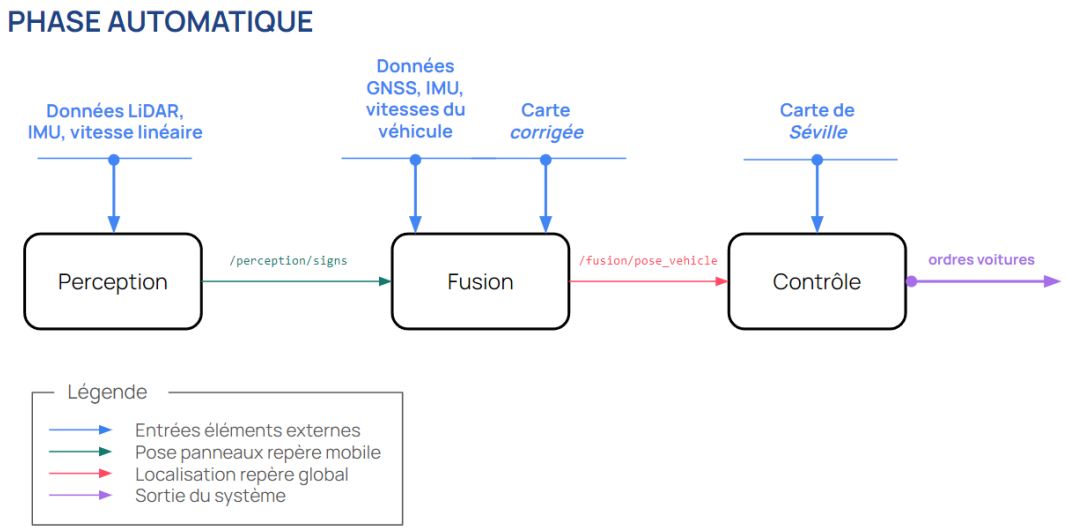

FIGURE 2 – Schéma de l'approche générale du projet en phase automatique

Notre approche se décompose en deux phases.

Lors de la première phase, la phase d'apprentissage, on a :

— Le noeud de perception qui détecte les panneaux dans le repère mobile du véhicule,
— Le noeud de fusion qui détermine une localisation via une fusion de données entre les panneaux détectées, la carte *a priori* et les données GNSS, IMU et les vitesses du véhicule.
— Le noeud de correction qui, à l'aide de la localisation *a priori* et de la carte donnée initialement, corrige via un alignement la carte des panneaux.

Lors de la seconde phase, la phase « automatique », on a :

— Le noeud de perception dont le fonctionnement reste le même.
— Le noeud de fusion qui détermine la localisation en utilisant cette fois-ci la carte corrigée lors de la phase d'apprentissage.
— Le noeud de contrôle qui, en comparant la position du véhicule dans le repère global et la carte de *Séville*, contrôle le véhicule sur la route.

La transition entre les deux phases est déterminée par la partie de correction.

1.3 La décomposition en tâches

Les tâches découlant de notre approche sont les suivantes :

— Tâche de perception : permet de détecter les panneaux via un seuillage sur l'intensité et un *clustering* sur les données du LiDAR.
— Tâche de fusion : permet de fusionner les données provenant de la perception et de la correction afin de déterminer la localisation du véhicule.

— Tâche de correction : permet de corriger la carte fournie à l'aide des données de perception et de la localisation déterminée par la fusion.
— Tâche de contrôle : permet de contrôler le véhicule longitudinalement et latéralement en fonction de la pose déterminée par la fusion.

Également, une tâche d'intégration est prévue dans laquelle toutes les tâches sont mis en commun et intégrées sur la voiture.

1.4 La répartition du travail

En pratique, concernant la répartition du travail, le projet a été organisé ainsi :

— Manon, Adrien et Samuel se sont occupés de la perception.
— Ahamed s'est occupé de la fusion, aidé par Adrien.
— Manon s'est occupée de la correction, aidé par Adrien.
— Victor et Anaïs se sont occupés du contrôle (Victor pour le contrôle latéral et l'intégration sous *ROS* et Anaïs pour le contrôle longitudinal).

2 Intégration (Adrien)

Le but de l'intégration est de coordonner les divers pôles du projet afin d'avancer le plus efficacement. Cela passe par la mise en place d'un *workflow* sous *Git* et la mise a disposition d'outils pour tester facilement (*launch files*, *mocks*). Il s'agit aussi d'aider les différents pôles quand cela est nécessaire.

2.1 Décomposition sous Git

Le projet est découpé en 5 *repositories*. Le premier, « main » est maintenu par l'intégrateur et contient l'intégralité du projet. C'est a partir du dossier main que le projet est compilé. Les 4 autres *repositories*, « controle », « correction », « fusion » et « perception » sont des *submodules* de « main » maintenus par les responsables de chaque pôle.

2.2 Architecture du projet ROS

Schématiquement, l'architecture du projet sous *ROS* est se décompose ainsi :

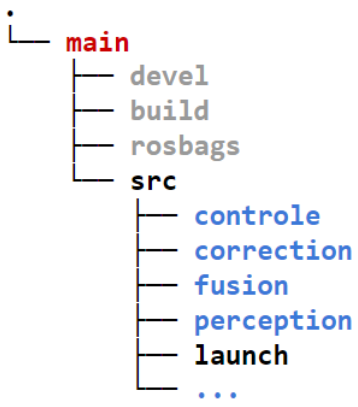

FIGURE 3 – Schéma de l'organisation *ROS* du projet

Avec en rouge le *repo* principal et en bleu les *repo* enfants du « main ». On remarque aussi les dossiers *devel* et *build*, essentiels à tout projet *ROS*. Le dossier *rosbags* nous a servit seulement en pratique afin de réaliser nos essais.

2.2.1 Organisation des *topics* et des messages personnalisés

Pour l'interfaçage des différents modules entre eux, nous avons fait le choix de noms de *topics* formatés de la manière suivante : *package, data_type*, avec *package* le nom du module émetteur des données et *data_type* le type des données qu'il traite. Les types de message sont définis par les modules qui les génèrent, et sont adaptés aux besoins de ceux qui les utilisent.

2.3 Organisation des dépendances

Pour faire fonctionner notre projet dans la voiture, de nombreuses dépendances ont été nécessaires. Les deux premières séances du projet on été dédiées à faire fonctionner un noeud fictif sur la voiture, en vérifiant la bonne acquisition des données des capteurs. Ainsi, nous avons ajouté plusieurs *packages* comme les drivers des différents capteurs, leur intégration dans la voiture ainsi que le driver de la voiture elle-même (pour lui envoyer des commandes).

3 Perception (Adrien, Manon & Samuel)

La partie de « perception » est la base sur laquelle repose le reste du projet. L'objectif est ici de transformer le nuage de points « brut » détecté par le capteur LiDAR en une liste de poses (coordonnées dans le plan du sol et orientation) des panneaux à chaque passage du LiDAR.

Dans cette section, nous traiterons de la décomposition sous *ROS* de cette partie puis des étapes et les méthodes utilisées lors du traitement de l'information.

3.1 Interfaçage ROS

Le schéma de la décomposition *ROS* de la partie est le suivant :

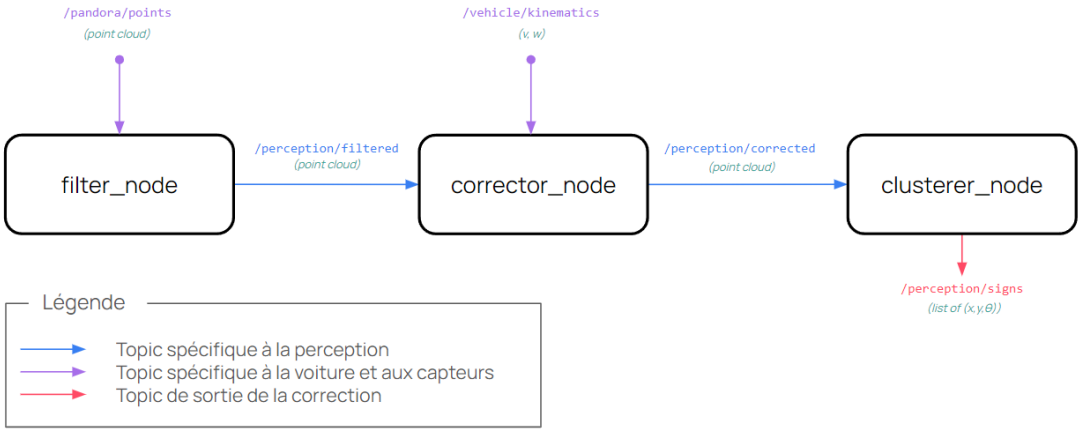

FIGURE 4 – Schéma de l'interfaçage *ROS* de la perception

Remarque : le *topic* `/vehicle/kinematics` n'est en réalité pas utilisé.

3.2 Filtrage par seuil d'intensité

La première étape du traitement des données LiDAR consiste à filtrer les points du nuage qui ne nous intéressent pas.

3.2.1 Principe général

En effet, nous avons choisi de ne prendre en compte que les panneaux de signalisation, qui sont les seuls objets que nous souhaitons détecter. Ces derniers sont particulièrement réfléchissants à la lumière incidente depuis n'importe quel angle, ce qui facilite leur seuillage. Ainsi, les détections du LiDAR correspondantes au panneaux vont toutes avoir une intensité lumineuse élevée.

3.2.2 Détermination du seuil en pratique

En pratique, grâce aux multiples enregistrements, nous avons pu déterminer empiriquement un seuil d'intensité fixe. Celui-ci permet de filtrer la majorité des points, tout en laissant intacts les points réfléchis par les panneaux.

3.3 Correction des distortions géométriques

La deuxième étape de la perception consiste à corriger les distortions géométriques présentes dans les mesures dues au mouvement de la voiture.

3.3.1 Principe général

En effet, comme vu en TD de SY27, la vitesse influe sur les données du LiDAR puisque le temps d'une révolution Δt, le véhicule est en mouvement, ce qui décale le point d'arrivée du point de départ.

En considérant un déplacement à vitesse linéaire v et vitesse angulaire ω pendant une durée Δt, soient x_s, y_s, et θ_s tels que

$$\begin{bmatrix} x_s \\ y_s \\ \theta_s \end{bmatrix} = \begin{bmatrix} \Delta t \cdot v \cdot cos(\Delta t \cdot \omega) \\ \Delta t \cdot v \cdot sin(\Delta t \cdot \omega) \\ \Delta t \cdot \omega \end{bmatrix}$$

Dans ce cas, les coordonnées corrigées d'un point (x, y, z) sont données par

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ z_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} cos(\theta_s) & sin(\theta_s) & 0 & x_s \\ -sin(\theta_s) & cos(\theta_s) & 0 & y_s \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ z_t \\ 1 \end{bmatrix}$$

3.3.2 Observations sur la correction

En pratique, cette correction n'a pas eu beaucoup d'impact car les vitesses du véhicules sont de l'ordre de $20km/h$ maximum, ce qui reste relativement faible. De plus, notre implémentation induisait un délai beaucoup trop important sur le traitement des données, rendant impossible de garantir d'avoir des résultats en temps réel.

Cette solution a donc été abandonnée mais le noeud continue d'être utilisé afin de faire le changement de repère de « pandora » à « base ».

3.4 Clustering et détermination de pose

Enfin, nous *clusterisons* les points restants de manière à différencier les différents panneaux visibles à un moment et à déterminer leur pose.

3.4.1 Principe général

Le *clustering* permet de créer des « groupements » de points et d'exclure les points solitaires. Une fois le nuage séparé, nous calculons une position moyenne des points de chaque groupement pour y localiser le panneau correspondant. Ensuite, pour déterminer son orientation, nous effectuons une analyse par composants principaux, *Principal Component Analysis* (PCA). En effet, ceci nous permet d'approcher le panneau par un plan, ce qui nous permet de savoir (à π près) quel est l'angle orthogonal au plan. Une simple vérification par la suite nous permet de déterminer lequel des angles dans l'intervalle $]-\pi, \pi]$ est celui qui nous intéresse.

3.4.2 Implémentation choisie

Le *clustering* choisi est basé sur une méthode de *K-D Tree* de la librairie *Point Cloud Library* (PCL), une méthode de découpage K-dimensionnel de l'espace. Dans notre cas, chaque point du nuage LiDAR contient 7 champs d'information, ce qui veut dire que nous pourrions utiliser jusqu'à 7 critères pour le *clustering*. Cependant, certaines des informations ne sont pas pertinentes pour notre objectif (telle que la nappe du LIDAR sur laquelle se trouve le point, par exemple). Ceci, combiné au fait d'avoir déjà filtré le nuage de points (ce qui créée des accumulations éparses de points), fait qu'un *clustering* euclidien simple est le plus adapté en pratique.

Les paramètres de ce *clustering* (distance maximale entre deux voisins, taille minimale et maximale d'un *cluster*) ont été déterminés empiriquement à l'aide de *rviz*.

Techniquement, il fonctionne ainsi :

— On crée d'abord un *K-D Tree* spécifique à la structure de points choisie. Ensuite, on attribue à ce *K-D Tree* le nuage de points que l'on veut *clusteriser* et on crée un conteneur pour les indices des points dans le nuage correspondants à chaque *cluster*.

— Puis, on crée un membre de la librairie PCL correspondant à un *clustering* euclidien, et on y indique les paramètres : la tolérance (distance maximale entre chaque point considérés faisant partie du même *cluster*), la taille minimale et la taille maximale d'un *cluster* pour qu'il soit considéré valide.

— Enfin, on indique à cet objet le nuage de points d'entrée, le *K-D Tree* associé, et on appelle une méthode qui réalise l'opération de *clustering*. Ceci remplit le conteneur d'indices, ce qui permet de parcourir le nuage de points en utilisant ces indices par la suite.

Une fois les *clusters* déterminés, nous calculons les coordonnées de chaque panneau avec une moyenne sur x et y des points appartenant à chacun des *clusters*.

Ensuite, nous déterminons l'orientation du panneau par rapport à la voiture, c'est-à-dire l'angle entre le vecteur normal au panneau (depuis son côté visible) et l'angle θ du véhicule.

Pour faire ceci, en profitant du fait que les panneaux sont des objets assez plats, nous pouvons appliquer une *PCA* aux points dans chaque *cluster*. Cette analyse permet d'extraire en particulier les vecteurs propres qui expliquent le mieux la forme du *cluster*

et leurs valeurs propres associés. Ainsi, les vecteurs qui définissent le plan approché ont les valeurs propres les plus élevées, donc nous retrouvons le vecteur orthogonal au plan en cherchant le vecteur associé à la valeur propre minimale.
Toutefois, rien ne nous assure que ce vecteur est orienté selon la face réfléchissante du panneau. Nous partons du principe que si le LIDAR peut apercevoir le panneau, ceci veut toujours dire que sa face avant est dans le champ du vision du LIDAR, donc l'angle recherché ne doit pas dépasser $\pi/2$ en valeur absolue. En outre, une simple vérification sur le signe de coordonnées permet de savoir si le panneau détecté est devant ou derrière la voiture. Par conséquent, nous pouvons déterminer si l'angle que nous cherchons est celui entre l'angle θ du véhicule et le vecteur normal au panneau calculé par PCA, ou celui entre l'angle θ du véhicule et l'opposé de ce vecteur.
Enfin, lorsque ces trois valeurs (x, y, θ) ont été calculées pour un *cluster*, elles sont mises dans une liste. Cette liste est publiée lorsque tous les *clusters* dans le nuage ont été parcourus.

3.5 Résultat global

Le résultat obtenu en sortie de perception est donné ci-dessous :

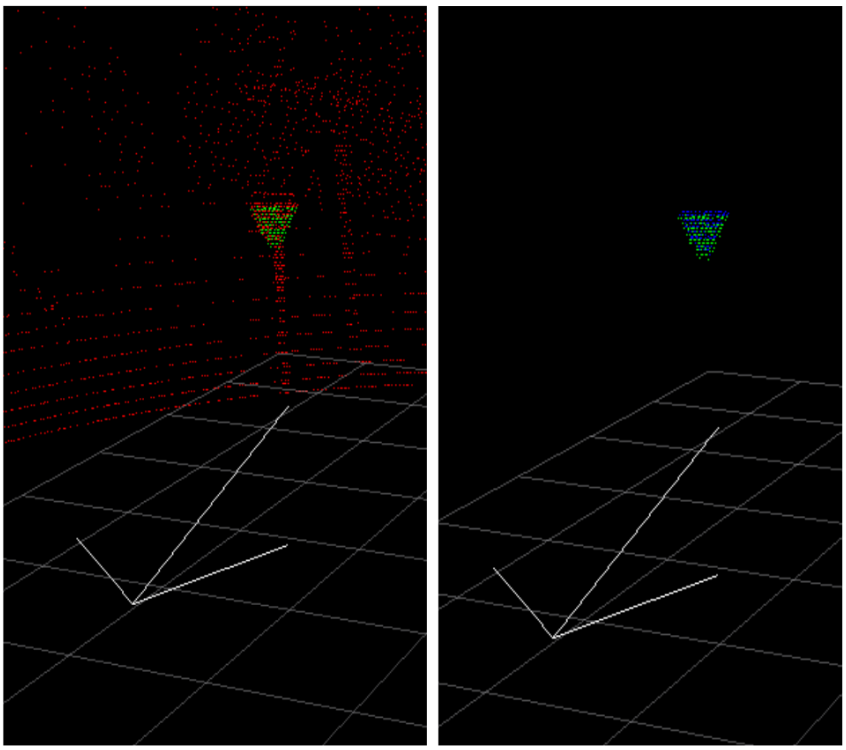

FIGURE 5 – Illustration de l'action de la perception

Avec en rouge le nuage de points non traités, en vert les points filtrés, en bleu les points corrigés et en blanc la position et l'orientation du *cluster* détecté.

3.5.1 Prise de recul quant aux résultats et améliorations possibles

Nous pourrions qualifier les résultats du module de perception comme fonctionnels, mais pas tout à fait satisfaisants. En effet, les données sont traitées suffisamment rapidement pour le fonctionnement en temps réel et la localisation des panneaux est précise et discriminatoire.
Cependant, nous avons abandonné la correction des distorsions géométriques à défaut de pouvoir l'implémenter assez efficacement, ce qui nuit à la qualité globale. De plus, l'orientation des panneaux est très sensible aux perturbations et peu fiable lorsque le LIDAR ne détecte pas assez de points car la PCA ne peut pas approcher assez bien un plan. Une première amélioration consisterait à déterminer la fiabilité de l'orientation du panneau selon les résultats de la PCA.

Une amélioration envisagée mais non implémentée serait de déterminer le seuil d'intensité par rapport aux premières valeurs envoyées par le LiDAR, comme par exemple un pourcentage des valeurs les plus intenses, lors du lancement du noeud. En effet, ceci permettrait de rendre la perception plus robuste, notamment face à des changements des conditions météorologiques.
En revanche, pour que cette approche soit fonctionnelle, il faudrait être certain que le système soit lancé dans un environnement qui contienne un nombre connu de panneaux. Cela étant plus complexe, nous avons finalement décidé de ne pas l'implémenter.

Enfin, il serait possible d'ajuster davantage la distance maximale dans laquelle on considère une détection valide en analysant statistiquement les détections effectuées. Ceci serait autant plus pertinent dans le cas où l'on aurait implémenté la métrique de fiabilité de l'orientation. Il est aussi dommage que nous n'ayons utilisé que le LIDAR en ayant accès à des caméras, dont une stéréoscopique.

4 Fusion de données (Ahamed & Adrien)

La fusion de données est une partie fondamentale puisque elle permet d'avoir la localisation du véhicule.

Afin d'avoir cette localisation, nous utilisons les données *GNSS* et les données disponibles par plusieurs autres capteurs. Dans le cadre de ce projet, ce sont des amers, dont les postions sont connues, qui nous permettront d'avoir une localisation plus précise (du moins suffisante pour une conduite autonome).

Ainsi, nous utilisons les données venant de *GNSS*, celles de l'*Inertial Measurement Unit* (IMU ou central d'inertie), les données cinématiques de la voiture, les coordonnées des panneaux observées (sortie de la perception) et leurs postions sur la carte. Toutes ces données sont ensuite fusionné par un filtre de *Kalman*. La sortie de ce filtre est une estimation de la position de la voiture.

4.1 Interfaçage ROS

Schéma de la décomposition ROS de la partie

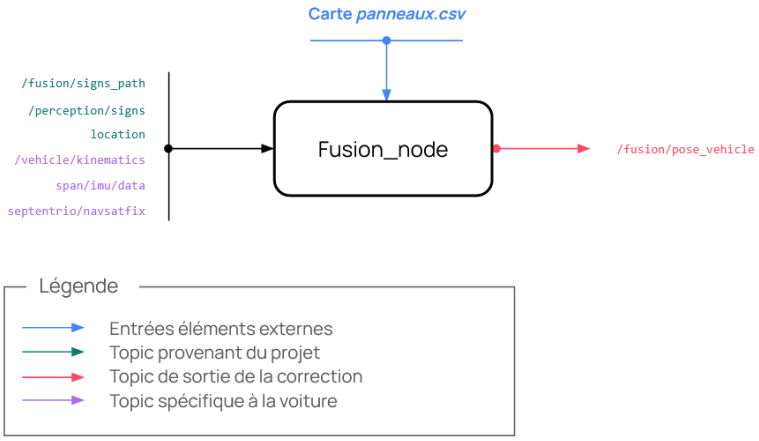

FIGURE 6 – Schéma de l'interfaçage *ROS* de la fusion

4.2 Estimation de d'état

Pour l'estimation de l'état nous utilisons un filtre de *Kalman*. L'état de notre voiture comprend la sa pose (x, y, θ) les données cinématiques de la voiture (vitesse linéaire v, vitesse angulaire ω, accélération linéaire a) et les biais *GNSS* (ex, ey) pour une meilleure estimation.

4.2.1 Filtre de *Kalman*

Pour estimer la pose de la voiture, nous utilisons la modèle « uni-cycle » ou « robot char » pour représenter la pose. Avec ce modèle on estime la position de la « base » située au milieu de l'essieu arrière de la voiture.

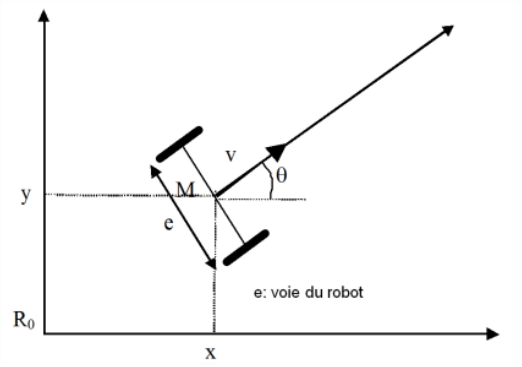

FIGURE 7 – Modèle uni-cycle de la voiture

Cet modèle a pour modèle d'évolution :

$$\dot{X} = \begin{cases} \dot{x} = v \cdot \cos\theta \\ \dot{y} = v \cdot \sin\theta \\ \dot{\theta} = \omega \\ \dot{v} = a \\ \dot{\omega} = 0 \\ \dot{a} = 0 \end{cases}$$

En temps discret le modèle d'évolution devient :

$$X_{k+1} = \begin{cases} x_{k+1} = x_k + dt \cdot v_k \cdot \cos\theta \\ y_{k+1} = y_k + dt \cdot v_k \cdot \sin\theta \\ \theta_{k+1} = \theta_k + \omega_k \cdot dt \\ v_{k+1} = v_k + dt \cdot a_k \\ \omega_{k+1} = \omega_k \\ a_{k+1} = a_k \end{cases}$$

Comme ce modèle est non-linéaire, nous devons le linéariser. Cela se traduit par le calcul de la jacobienne de la modèle de l'évolution dans chaque point. Nous trouvons la matrice ci-dessous après le calcul.

$$F = \begin{bmatrix} 1 & 0 & -dt \cdot v_k \cdot \sin\theta & dt \cdot \cos\theta & 0 & 0 \\ 0 & 1 & dt \cdot v_k \cdot \cos\theta & dt \cdot \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Observation de GNSS

La capteur *GNSS* fait partie des deux capteurs extéroceptifs de la voiture avec le *LIDAR*, les observations de ce capteur permet d'avoir une première estimation de la position de la voiture dans la repère *East, North, Up* (ENU).

Avec ce capteur, on observe la pose 2D de la voiture. Par conséquent, le modèle d'observation de cette observation est la position de la voiture.

$$g = \begin{bmatrix} x_k - ex_k \\ y_k - ey_k \end{bmatrix}$$

Ce modèle étant linéaire, sa matrice d'observation est la suivante :

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Observation IMU

L'IMU permet de calculer et estimer la vitesse linéaire et angulaire de la voiture. Nous utilisons les observations de la vitesse angulaire et la accélération linéaire telles que :

$$g = \begin{bmatrix} \omega_k \\ a_k \end{bmatrix}$$

Ce modèle étant linéaire, sa matrice d'observation est donc :

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Observation cinématiques de véhicule

Ce capteur est une redondance pour l'IMU, il s'agit d'un deuxième gyroscope. Contrairement à l'observation IMU où on estimait la vitesse linéaire à partir de l'accélération linéaire, ce capteur nous permet mesurer directement la vitesse linéaire. La modèle d'observation est donc :

$$g = \begin{bmatrix} \omega_k \\ a_k \end{bmatrix}$$

Ce modèle étant linéaire, sa matrice d'observation est :

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Observation LIDAR

Les observations du *LIDAR* jouent un rôle très important dans l'estimation de pose de la voiture. En effet, ce capteur extéroceptif est le seul à fournir des mesures très précises, ce qui nous permet d'estimer la position de la voiture de manière très

précise. Ici, il mesure les distances du véhicule par rapport aux panneaux détectés, ce qui se traduit dans la fusion de données par l'obtention d'une liste de coordonnées des panneaux dans le repère local par la partie de perception.

Pour un panneau le modèle d'observation est alors :

$$g = \begin{bmatrix} cos\theta \cdot (x_v - x_k) + sin\theta \cdot (y_v - y_k) \\ -sin\theta \cdot (x_v - x_k) + cos\theta \cdot (y_v - y_k) \end{bmatrix}$$

Ce modèle étant non-linéaire, sa matrice d'observation est la jacobienne suivante :

$$G = \begin{bmatrix} -\cos\theta & \sin\theta & -\sin\theta \cdot (x_v - x_k) - \cos\theta \cdot (y_v - y_k) & 0 & 0 & 0 & 0 & 0 \\ \sin\theta & -\cos\theta & -\cos\theta \cdot (x_v - x_k) - \sin\theta \cdot (x_v - x_k) & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.3 Différences entre phases

Comme vu plus haut, le fonctionnement de cette partie ne change pas durant les deux phases. Le seul changement dans le filtre est la carte des panneaux `panneaux.csv`.

Pendant la phase d'apprentissage, nous utilisons la carte « bruitée » (la carte *a priori*) seulement.

Lorsque la partie de correction a estimé avoir corrigé toutes les positions des panneaux dans la carte bruité, on passe en phase automatique, durant laquelle la nouvelle carte remplace la carte bruitée dans l'estimation de la position de la voiture.

4.4 Essai de l'estimation

Nous avons initialement testé notre filtre sans inclure les données des panneaux.

Au cours de cet essai, nous avons observé des erreurs moyennes de $-0.9139\ m$ avec un écart-type de $1.2167\ m$ sur l'axe x, et une moyenne de $0.1300\ m$ avec un écart-type de $1.0322\ m$ sur l'axe y.

Ces résultats soulignent l'impossibilité d'utiliser uniquement le *GNSS* comme capteur extéroceptif.

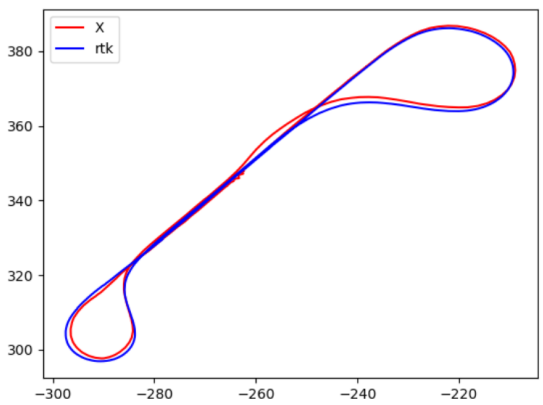

FIGURE 8 – Localisation sans LIDAR comparé avec la pose RTK

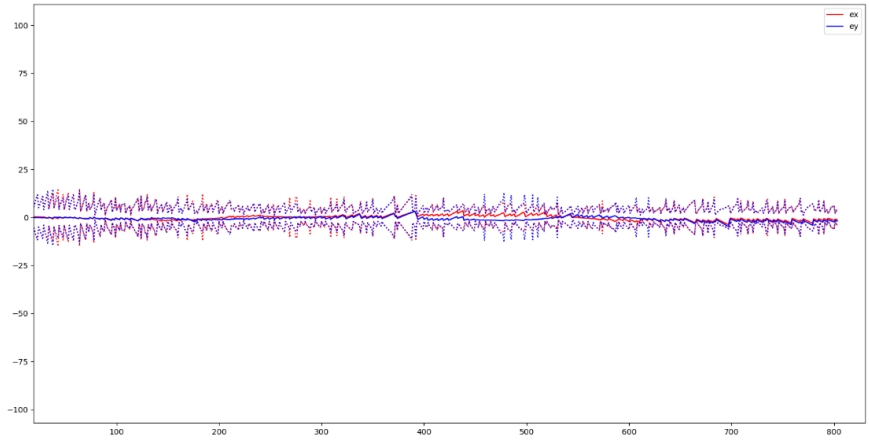

FIGURE 9 – Graphe d'erreur de pose

4.5 Problèmes des messages dans le passé

L'utilisation des observations des capteurs nécessite un traitement rapide après leur acquisition. Le capteur de cinématique de la voiture a une fréquence périodique de mesure de 100 Hz, et le LIDAR a une fréquence de mesure périodique de 10 Hz, mais il nécessite un temps de pré-traitement additionnel. Cela conduit souvent à des retards importants dans les observations LIDAR par rapport aux mesures de cinématique de la voiture, créant ainsi des situations où les messages du LIDAR appartiennent au passé.

Pour résoudre ce problème, nous avons implémenté l'algorithme appelé « Time-Machine ». Le fonctionnement de cet algorithme est expliqué dans l'algorithme 1. Dans la figure 10, nous avons illustré la façon dont l'algorithme fonctionne :

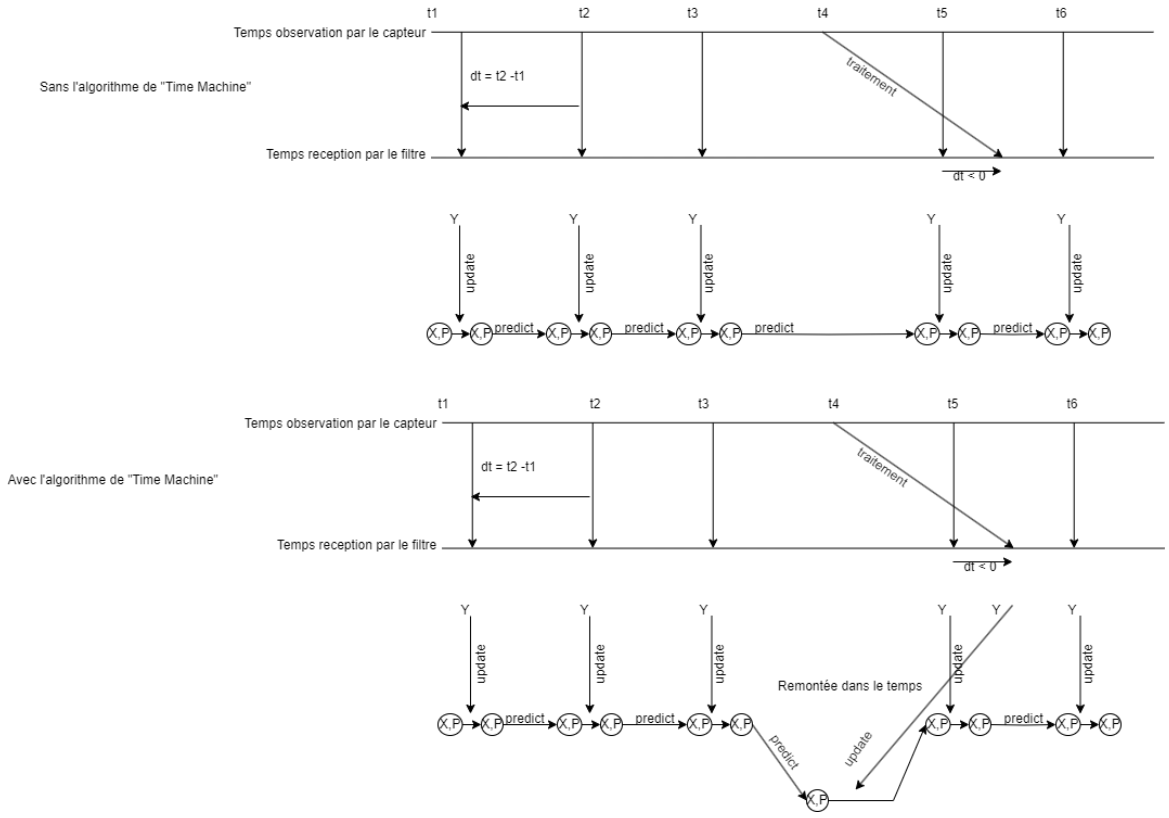

FIGURE 10 – Problème de message de passé et la fonctionnement de la solution de «Time-Machine»

Avec cet algorithme, nous pouvons prendre en compte les messages nécessitant un traitement supplémentaire avant leur utilisation dans notre filtre, améliorant ainsi l'estimation de la pose de la voiture.

Algorithm 1 Algorithme de Time-Machine

// Pour chaque observation reçue, calculer la différence de temps
$dt = t_{obs} - t_{prec}$
$X\_save \leftarrow X$ // Sauvegarder l'état actuel
$P\_save \leftarrow P$ // Sauvegarder la matrice de covariance actuelle
$T\_save \leftarrow T_{obs}$ // Sauvegarder le temps d'observation actuel
$dt \leftarrow T_{obs} - T_{prec}$ // Mettre à jour la différence de temps
$T_{prec} = T_{obs}$ // Mettre à jour le temps d'observation précédent
if $dt < 0$ **then** // Si la différence de temps est négative, l'observation est dans le passé
    $i \leftarrow \text{lowerbound}(t_{obs})$ // Trouver l'indice dans les temps sauvegardés
    $X \leftarrow X\_save[i]$ // Mettre à jour l'état
    $P \leftarrow P\_save[i]$ // Mettre à jour la matrice de covariance
    $T_{prec} \leftarrow T\_save[i]$ // Mettre à jour le temps d'observation précédent
    $dt \leftarrow t_{obs} - t_{prec}$ // Mettre à jour la différence de temps
    $X, P \leftarrow \text{predict}(dt)$ // Prédire l'état et la matrice de covariance
    $X, P \leftarrow \text{update}(Y_{obs}, X, P)$ // Mettre à jour l'état et la matrice de covariance
    while $i \neq \text{end}(T\_save)$ **do** // Pour chaque temps sauvegardé de i à la fin
        $dt \leftarrow t_{obs} - t_{prec}$ // Mettre à jour la différence de temps
        $X, P \leftarrow \text{predict}(dt)$ // Prédire l'état et la matrice de covariance
        $X, P \leftarrow \text{update}(Y\_save[i], X, P)$ // Mettre à jour l'état et la matrice de covariance
        $t_{prec} \leftarrow t_{obs}$ // Mettre à jour le temps d'observation précédent
        $i = i + 1$ // Passer à l'indice suivant
        $t_{obs} \leftarrow T\_save[i]$ // Mettre à jour le temps d'observation actuel
    end while
end if
// Si la différence de temps n'est pas négative, procéder à l'opération normale

L'algorithme baptisé «Time-Machine »a été conçu pour sauvegarder les fonctions de mise à jour des observations, permettant ainsi une réutilisation efficace pour n'importe quel état du système. Comme son nom l'indique, cet algorithme offre la possibilité de remonter dans le temps dans le cas où un message serait reçu avec un délai par rapport à l'observation précédemment traitée par le filtre temporel.

Lorsqu'une observation est reçue avec une différence de temps négative, indiquant un retard dans la réception du message, l'algorithme rétablit le filtre dans l'état auquel il aurait dû être au moment de la réception prévue de l'observation. Ensuite, il applique séquentiellement les fonctions de mise à jour, prenant ainsi en compte l'observation arrivée en retard. Cette approche assure une correction appropriée de l'état interne du système, maintenant sa cohérence même en présence d'observations décalées dans le temps.

4.6 Intégration de LIDAR et l'algorithme de Time-Machine

Nous avons intégré l'algorithme « Time-Machine » dans notre filtre pour prendre en compte des mesures LIDAR nécessitant un prétraitement, ce qui peut prendre un certain temps. Malheureusement, nous avons rencontré un *bug* dont nous n'avons pas pu identifier l'origine, et cela a rendu notre filtre instable en conduisant à une matrice de covariance négative. Malgré nos multiples tentatives de « débogage », nous n'avons pas réussi à localiser la source de ce problème, ce qui nous a empêchés d'intégrer la

partie de fusion de données pour la démonstration finale.

4.7 Prise de recul et améliorations éventuelles

Dans un premier temps, nous avons implémenté un filtre en python qui permettait d'estimer la pose à partir des données *GNSS*, IMU et la cinématique de la voiture sans le LIDAR. Le programme était fonctionnel ; la pose convergeait bien vers la pose du *GNSS*, mais nous nous sommes rendus compte que le *rospy* (module *ROS* de Python) ne nous permettait pas de traiter les messages en temps réel puisque nous avions des observations à haute fréquence.

De ce fait nous avons décidé d'implémenter le filtre en C++ afin qu'il puisse traiter les observations plus aisément.

En revanche, l'implémentation s'avéra être plus compliquée. A ce jour, nous avons réussi a implémenter un filtre en C++ mais ce dernier ne prend pas en compte les messages en retard. Ce dernier n'était non plus optimal puisque la plupart de temps les observations du LIDAR arrivaient en retard.

Avec du recul, nous pensons qu'il aurait été meilleur de commencer l'implémentation en C++ au lieu de Python car nous aurions pu trouver les problèmes très à l'avance et cela nous aurait possiblement permis de faire marcher la localisation.

5 Correction de la carte (Manon)

La partie de correction de la carte a pour objectif de générer une carte de panneaux « correcte » lors de la première phase dite d'apprentissage afin que la voiture puisse rouler en mode autonome lors de la seconde phase. La carte a donc besoin d'être précise et de présenter le plus de panneaux possible afin de pouvoir assurer une bonne localisation lors de la conduite autonome. Pour ce faire nous avons utilisé comme outil principal un filtre PHD (*Probability Hypothesis Density*).

Cette partie a été nommée au début du projet « correction de carte » car une carte *a priori* des panneaux nous est fourni à l'origine. Cette carte comporte un certain nombre de bons panneaux, mais d'autres ont une position et un angle bruité et certains sont manquant. Cependant, malgré cette appellation de « correction de carte », il s'agit plutôt d'une phase de cartographie puisqu'on ne s'appuie pas directement sur la carte *a priori* fournie.

5.1 Interfaçage ROS

Le schéma de la décomposition *ROS* de la partie est le suivant :

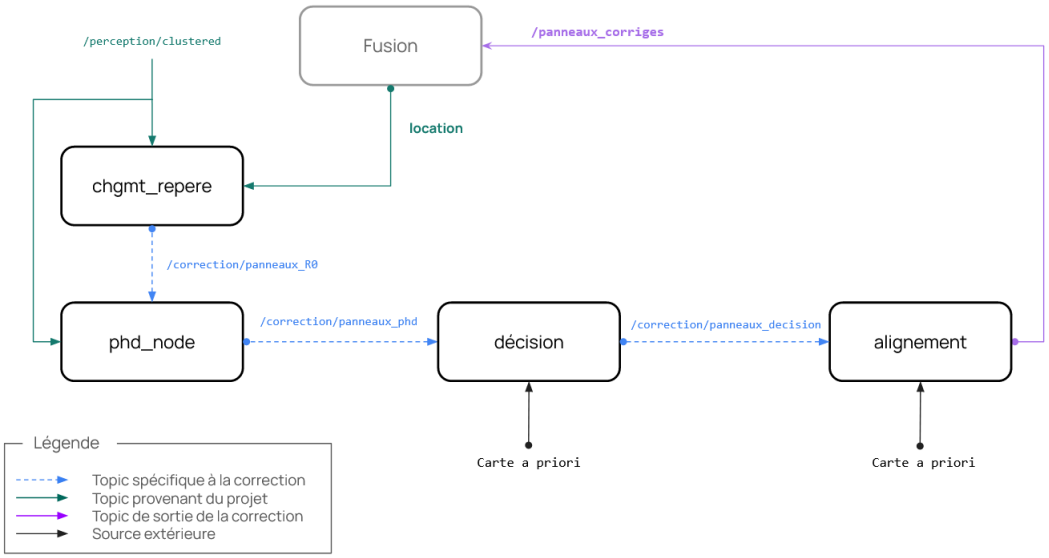

FIGURE 11 – Schéma de l'interfaçage *ROS* de la correction

La partie de correction de la carte comporte 4 noeuds :

— Changement de repère : il reçoit les poses des panneaux dans le repère mobile (venant de la partie de perception) pour les transformer dans le repère global grâce à la localisation estimée par le filtre de *Kalman*.
— Noeud PHD : il prend en entrée les poses des panneaux dans le repère global et donne la pose estimée des panneaux par le filtre PHD.
— Noeud de décision : ce noeud a pour objectif de calculer quand est ce que la carte des panneaux est assez fiable pour passer à la phase deux.
— Noeud d'alignement : il permet de réaligner la carte fournie par le noeud PHD avec la carte à priori afin de corriger un éventuel biais.

5.2 Probability Density Hypothesis Filter

Le filtre PHD (*Probability Hypothesis Density*) est un filtre à cibles mobiles multiples, fonctionnant pour un nombre de cibles inconnues, permettant d'estimer de manière récursive le nombre et l'état d'un ensemble de cibles (X_k) à partir d'un ensemble d'observations (Z_k). Ce filtre se base sur le fonctionnement d'un filtre *Bayésien* et sur la fonction PHD qui, intégrée sur son ensemble, donne le nombre d'éléments de l'ensemble.

Une approximation nécessaire à la réalisation des calculs du filtre est que ses éléments soient des gaussiennes [2]. Le filtre traite donc en réalité une mixture de gaussiennes, que ce soit en entrée ou en sortie.

5.2.1 Fonctionnement du filtre

Nous allons détailler de manière simplifiée les différentes étapes réalisées lors d'une itération du filtre.

Prédiction L'étape de prédiction va permettre pour chaque gaussienne de la mixture d'entrée d'évaluer leurs poses par rapport aux temps précédents, grâce au modèle d'évolution du système.

Update La phase d'*update* va permettre de prendre en compte les observations faites. Toutes les gaussiennes données en sortie de la prédiction sont appelées des hypothèses. Pour chaque hypothèse, on va multiplier son poids par un facteur $(1 - P_D)$ avec P_D la probabilité qu'un élément soit détecté.

Soit m_k le nombre d'observations et H_k le nombre d'hypothèses. Nous allons pour chaque observation calculer H_k nouvelles gaussiennes. Par définition, le poids d'une composantes calculées entre une observation et une hypothèse est d'autant plus élevé si l'observation coïncide avec l'hypothèse. Il est question de normaliser ensuite les poids pour prendre en compte d'éventuelles fausses détections et afin qu'une observation ne puisse contribuer qu'à un poids total de 1.

Pruning L'étape de *pruning* sert à enlever les composantes de poids trop faibles, jugée comme insignifiante.

Merging Le *merging* va permettre de fusionner les gaussiennes ayant une position et un angle proche. Cela va permettre, plutôt que d'avoir plusieurs composantes de petits poids très proches d'un point de vue spacial, de les réunir en une gaussienne avec une covariance nécessairement un peu plus grande.

Naissance La naissance permet d'ajouter une composantes par observations à la mixture.

2. Pour rappel, une gaussienne est composée d'un poids, une pose et une matrice de covariance.

5.2.2 Limites et ajustements

Le filtre PHD et les hypothèses faites sont en principe utilisés pour du *tracking* l'objet mobile qui sont sensés être visible quand ils existent. Or, le cadre de notre projet ne colle pas exactement à ce contexte, puisque nous utilisons des panneaux (entités statiques) avec une visibilité réduite (qui peuvent ne pas être visible). Cela nous a donc obligé à trouver des solutions pour adapter le filtre au cadre de notre étude.

Objets statiques Les panneaux étant statiques, la partie prédiction de la position de la phase prédiction n'est donc pas utile. Cependant, nous rajoutons tout de même les bruits de modèle à la matrice de covariance des gaussiennes.

Champs de visibilité Le moyen de perception utilisé étant le *LiDAR*, un panneau n'est visible que si on se situe devant lui. Ainsi, vu la configuration de la piste et les différentes positions auxquelles peuvent être placées les panneaux, ils sont visible seulement dans un champ réduit. La but a été de trouver un champs de visibilité qui correspondait au maximum à n'importe quelles configurations de panneaux autour de la piste.

Voici les éléments principaux pris en compte :

— La distance au panneau : si un panneau est trop près (<2 mètres) ou trop loin, il est considéré comme non visible.
— L'angle : suivant l'angle du panneau, il n'est visible que si on se trouve « assez » en fasse de lui.

La figure 12 donne un exemple de configuration possible. En rouge se trouve les panneaux non visible et en vert celui qui l'est avec le champ dans lequel il est visible.

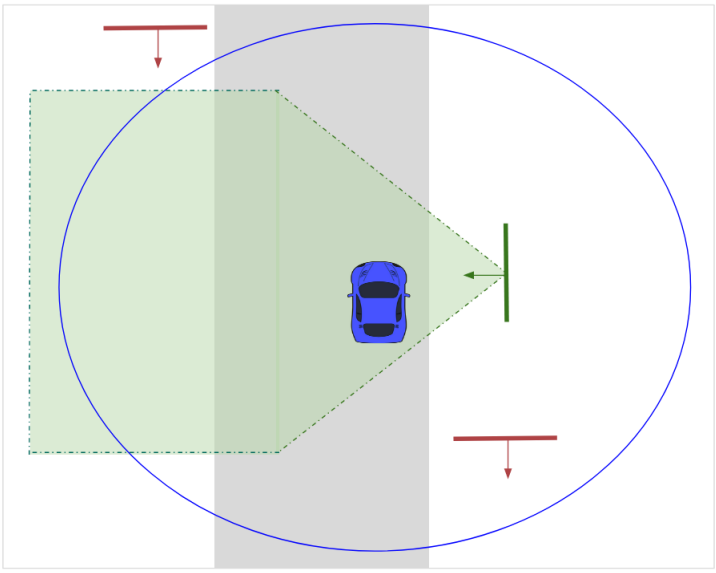

FIGURE 12 – Illustration mettant en valeur la visibilité d'un panneau selon son orientation et sa distance par rapport au véhicule

L'implémentation de la visibilité d'un panneau à pu servir dans le filtre PHD. Nous nous on sommes tout d'abord servi pour filtrer les observations. Nous ne prenons on compte que les observations si l'on est assez sûr que la voiture peut voir le panneau, cela permet de filtrer des observations de moins bonnes qualités. Cependant, le champ de visibilité définie reste assez large.

La deuxième utilisation se fait dans dans l'étape d'*update* du filtre, avec une zone de visibilité cette fois ci un peu plus réduite. La pose d'une gaussienne est considérée comme un panneau non visible, par conséquent, on met la probabilité de détection P_D à 1. Cela revient d'abord à dire lors de la multiplication du poids par le facteur $(1 - P_D)$ que le panneau ne peut pas être non-détecté puisqu'il ne peut pas être vu. Le seconde intervention apparaît lors du calcul d'une nouvelle composante gaussienne entre une observation et une hypothèse. La formule du poids de cette composante est la suivante :

$\tilde{w}_{k|k}^{i\times\mathcal{H}_k+h} \leftarrow P_D w_{k|k-1}^h \mathcal{N}(z_k^i; \hat{z}_{k|k-1}^h, S_k^h)$

$w_{k|k-1}^h$ correspond au poids de l'hypothèse, $\tilde{w}_{k|k}^{i\times\mathcal{H}_{k|k-1}+h}$ au poids de la nouvelle et z_k^i l'observation. On remarque que si cette hypothèse n'est pas visible, $P_D = 0$ et donc l'observation ne peut pas être associée à cette hypothèse.

Sensibilité aux détections manquées Le filtre PHD est un filtre d'ordre 1, c'est à dire qu'il ne garde en mémoire que l'espérance sur le nombre d'hypothèse et non l'incertitude. C'est pour cela que, dans un premier temps, il faut régler le champ de visibilité des panneaux présentés précédemment. Il faut qu'il soit suffisamment grand pour être sûr que l'on observe le panneau à un moment donné mais pas trop grand pour que l'on soit « sûr » de l'observer lorsque l'on est dans son champ de visibilité.

Cela n'évite malheureusement pas les détections qui, même sur une seul pas de temps, font chuter le poids à une valeur très basse même si le panneau a été vu 100 fois avant. Pour cela, nous avons décidé qu'avant de lancer une itération du filtre, on accumule les observations pendant n pas de temps (les tests ont été menés entre 3 et 5 suivant la vitesse) pour éviter les problèmes de détections manquées ponctuelles.

5.3 Noeud de décision

Le noeud de décision prend en entrée les poses des panneaux estimés par le filtre PHD. Il va permettre de décider quand la carte est suffisamment stable pour passer à la phase deux. Cette décision fait intervenir deux métriques : la distance parcourue par le véhicule depuis le lancement du noeud global de correction et le $GOSPA$.

Distance parcourue Au cours des tests et de manière intuitive, on peut facilement déduire qu'il faut au moins un tour de piste complet pour pouvoir espéré avoir pu voir tout les panneaux. On calcule donc la distance parcourue par la voiture et tant qu'elle est inférieure à un peu plus d'un tour de piste, la correction de carte continue.

GOSPA Le $GOSPA$ est une métrique permettant de calculer la distance ou plutôt à quel point deux ensembles finis d'éléments sont éloignés (différents). Elle se calcul en

faisant une association des points des deux ensembles. Puis, elle est calculé suivant la distance des points associés, le nombre de fausses détections et le nombre de détections manquées. On utilise plus exactement le $mean - GOSPA$, il s'agit simplement de le diviser par le nombre d'éléments détectés. Cela permet d'enlever le fait que plus il y a d'élément à comparer, plus il y a de sources d'erreurs.

Nous utilisons pour calculer le $mean - GOSPA$ la carte à priori et la carte du filtre PHD. Même si les données de la carte à priori peuvent être bruitées, il existe un certain nombre de panneaux bien placés. Le fait qu'il y ait des mauvais panneaux ou des panneaux manquants sur la carte à priori, cela n'empêche pas le $mean - GOSPA$ de diminuer au fur et à mesure que des nouveaux panneaux bien placés apparaissent dans la carte du PHD.

Après plus d'un tour de piste, si cette métrique est constante pendant un certain intervalle de temps, on considère que la carte n'évolue plus et donc que l'on peut mettre fin à la phase 1 du projet.

5.4 Alignement

Une fois qui la décision d'arrêter de modifier la carte du PHD a été prise, nous avions pour idée de procéder à un réajustement (alignement) de la carte générée sur la carte *a priori*. Cela a pour but de corrigé un potentiel biais qui peut notamment être induit par la localisation.

Pour cela nous utilisons un *Iterative Closest Point* (ICP). Il s'agit d'un algorithme utilisé dans le but de mettre en correspondance deux jeux de données, le plus souvent sous la forme de nuages de points, correspondant à deux vues partielles d'un même objet. Une vue étant constituée d'un ensemble de points, l'objectif est de minimiser itérativement la distance entre ces points.

Enfin, après toutes ces étapes, la cartes est envoyée à la localisation pour remplacer la carte *a priori*.

5.5 Résultats

Enfin, présentons les résultats obtenus.

5.5.1 Simulation : *Rosbag*

Les données qui vont être présentées ont été réalisées avec la localisation *RTK* à laquelle nous avons rajouté un bruit gaussien d'écart-type 0.75 mètres afin de visualiser le comportement du filtre avec une localisation moins précise (à défaut d'avoir celle de la partie de localisation).

Cartes La figure 13 montre le résultat brute en sortie du filtre PHD. En rouge la vérité terrain et en bleu notre estimation de la localisation des panneaux. On remarque qu'à certain deux endroit, dont un montré plus en détails figure 14, le filtre a prédit la présence de deux panneaux très rapprochés qui dans la réalité ne correspondent qu'à un seul et même panneau. Ceci est du en partie à cause de certaines observations qui

peuvent être assez éloignées de la réalité pour certains panneaux, particulièrement dans les virages ou le risque de distortion du nuage de point peu être plus important.

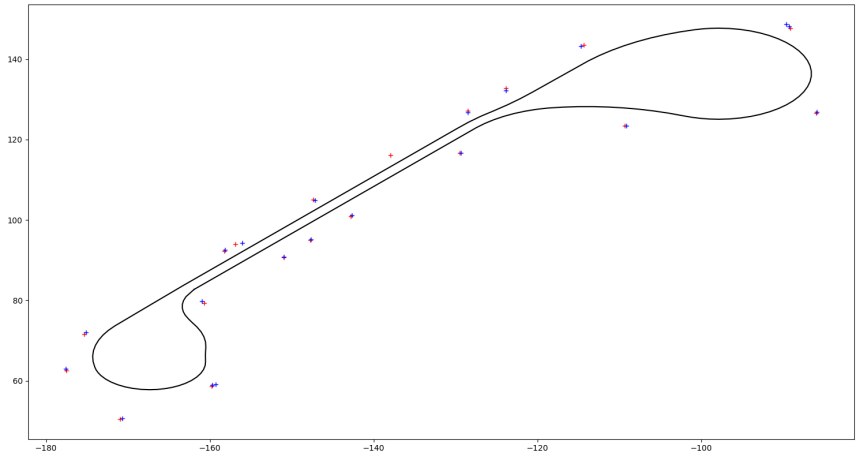

FIGURE 13 – Illustration mettant en valeur la carte corrigée - sans *cleaning*

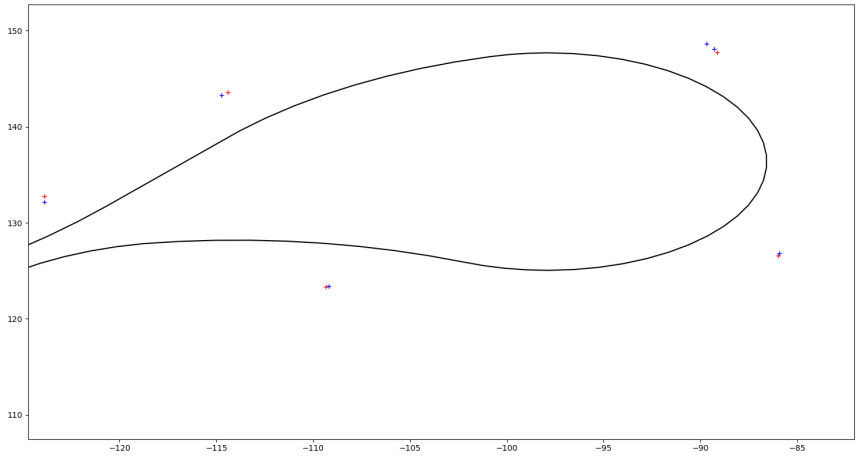

FIGURE 14 – Illustration mettant en valeur la carte corrigée - sans *cleaning* - *ZOOM*

Nous avons donc procédé à une étape de cleaning en sortie du filtre. Nous avons considéré que si des panneaux étaient trop proche et si l'un d'eux avait un poids significativement supérieur à l'autre, on ne gardait que celui avec le poids le plus grand. Le résultat finale se trouve figure 15.

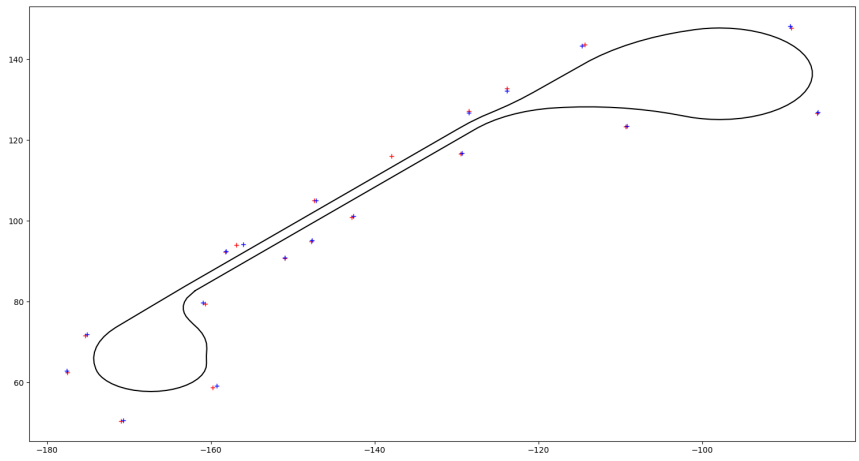

FIGURE 15 – Illustration mettant en valeur la carte corrigée - avec *cleaning*

GOSPA Voici figure 16 le résultat du calcul du mean-$GOSPA$ entre la carte à priori et notre carte estimée. Sachant que un tour de piste correspond à une abscisse de 250 environ, on voit bien qu'après un peu plus d'un tour de piste le mean-$GOSPA$converge vers une valeur proche de zéro.

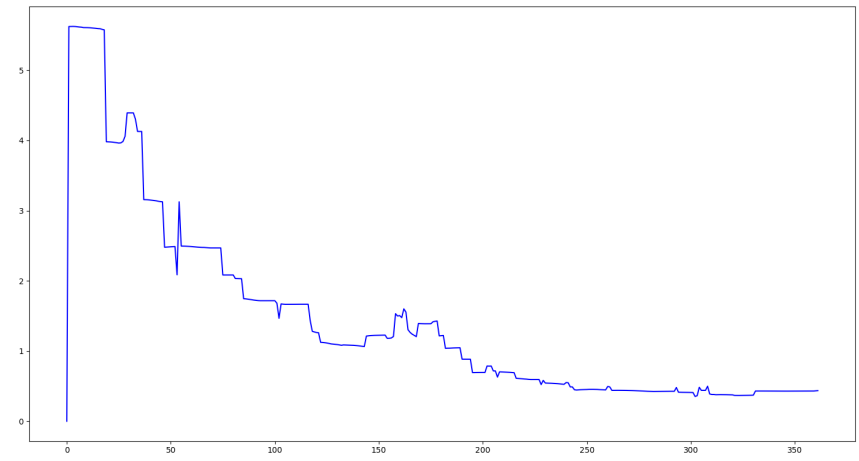

FIGURE 16 – mean-$GOSPA$ en fonction du pas de temps du bag

Alignement Afin de tester l'alignement, on induit un biais sur la localisation en x de 2 mètre. La carte ressemble alors à celle figure 17.

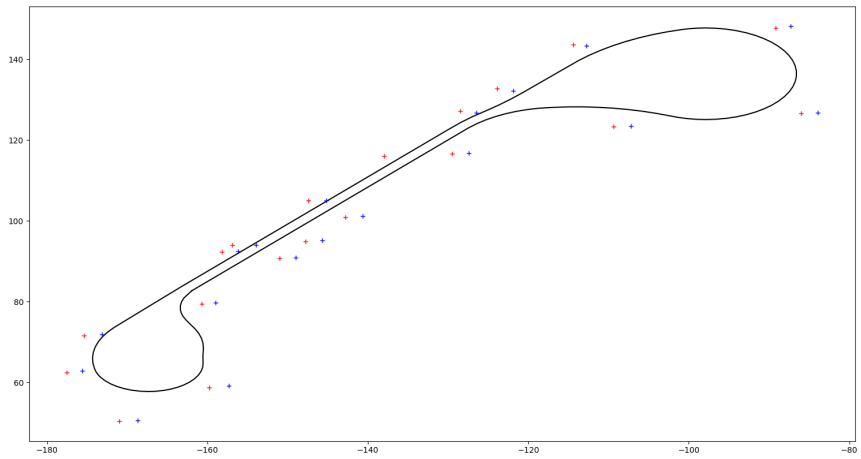

FIGURE 17 – Illustration mettant en valeur la carte corrigée - sans *alignement* et avec biais

En réalignant avec le module d'alignement qui réaligne à partir de la carte *a priori*, la carte en sortie est celle figure 18.

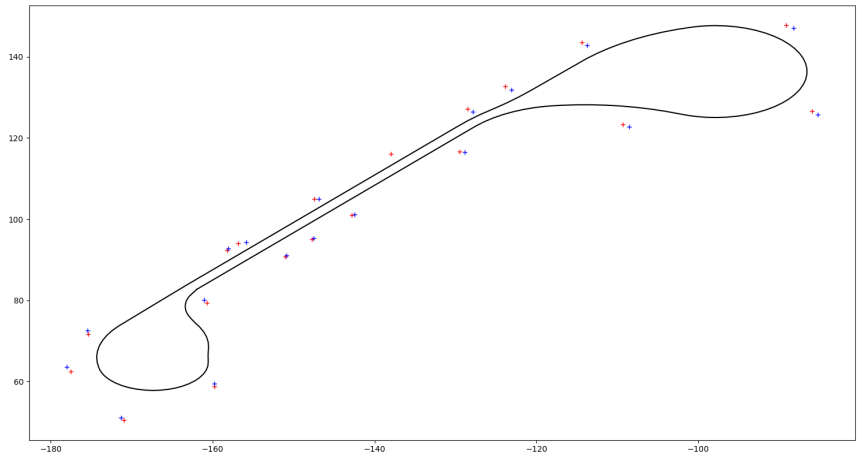

FIGURE 18 – Illustration mettant en valeur la carte corrigée - avec *alignement* et avec biais

la courbe du $mean - GOSPA$ (figure 19) a la même évolution que pour les données non-biaisées, sauf qu'elle converge vers une valeur au dessus de deux.

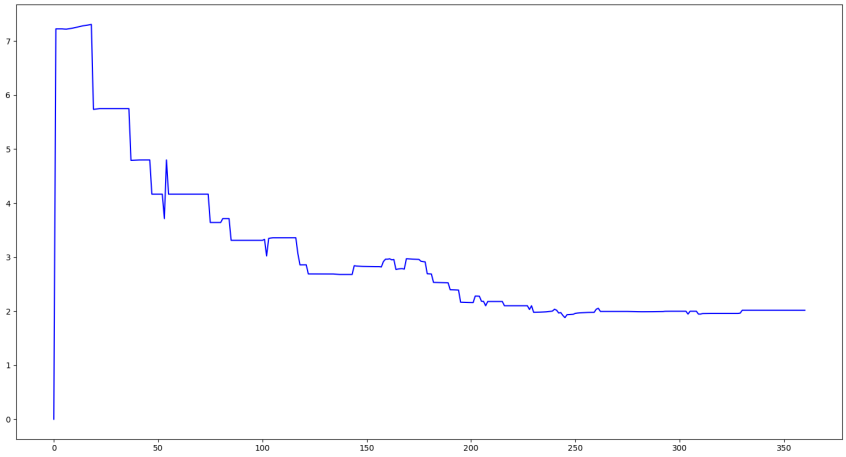

FIGURE 19 – Illustration mettant en valeur le mean-$GOSPA$ avec biais

Nous en concluons donc que même à partir d'une carte biaisée le ré-alignement est utile. Cependant, parce que nous utilisons la localisation au RTK qui n'est pas biaisée, ce module n'est pas utile voir dégrade les résultats si la carte corrigée par le PHD est déjà très précise.

5.5.2 Démonstration réelle

En démonstration réelle, nous avons pu observer les résultats obtenus figure 20. Sachant que la carte des vérités ne correspond pas exactement à la configuration de la démonstration.

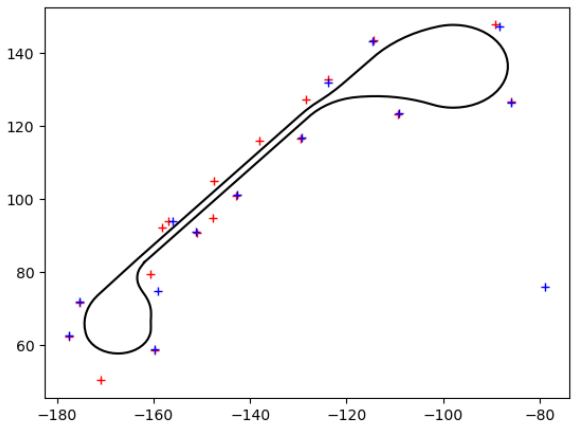

FIGURE 20 – Illustration mettat en valeur la carte provenant de la démonstration dans le véhicule

Qui se traduisent par le *mean-GOSPA* figure 21.

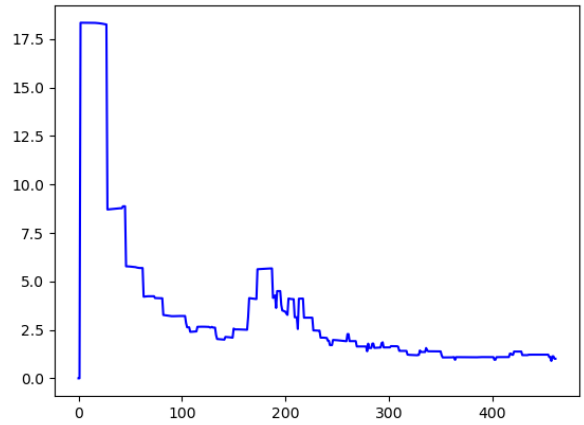

FIGURE 21 – Illustration mettant en valeur le *mean-GOSPA* provenant de la démonstration dans le véhicule

5.6 Prise de recul quant aux résultats et améliorations possibles

Les résultats obtenus sur les *bags* sont tout de même satisfaisants lorsqu'ils sont comparés aux premiers résultats obtenus. La localisation doit tout de même être assez fiable pour pouvoir réaliser une bonne cartographie.

Le plus gros point à améliorer reste le problème des détections manqués qui impactent considérablement le filtre. L'idée d'accumuler les observations améliorent les résultats mais cela reste une solutions qui n'est pas robuste. Une solution envisagée a été d'implémenter un filtre CPHD (*cardinalized probability hypohtesis density*). Il garde en mémoire l'incertitude sur le nombre d'élément. Ce filtre est donc beaucoup plus robuste, mais cela ne remplace pas non plus toute la partie de champ de visibilité mise en place. A cause au retard global pris par le groupe et du fait que nous n'étions vraiment pas optimiste d'arriver à coder ce filtre pour qu'il tourne en temps réel, son implémentation n'a pas pu être réalisée.

Avec notre propre localisation, nous avons aussi envisagé d'utiliser la covariance de la localisation et l'incertitude sur les panneaux en sortie de la perception pour les naissances des nouvelles gaussiennes du filtre.

6 Contrôle et commande du véhicule (Anaïs & Victor)

La partie du « contrôle » est une partie essentielle puisqu'elle permet le contrôle du véhicule. L'objectif est ici de transformer la pose reçue par la partie de fusion (en combinaison avec les autres parties) en un couple de flottant représentant l'ordre latéral et longitudinal que le véhicule doit adopter à un instant t.

Dans cette section, nous traiterons de la décomposition sous *ROS* de cette partie puis de son contenu : la mise en place de ses essais, le contrôle latéral et le contrôle longitudinal et les résultats obtenus.

6.1 Interfaçage ROS

Le schéma de la décomposition *ROS* de la partie est le suivant :

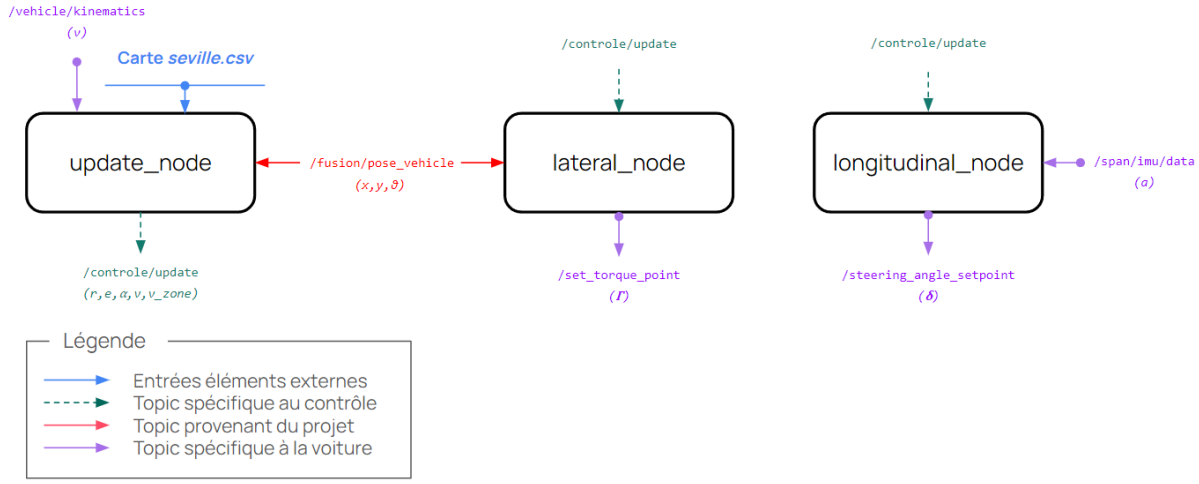

FIGURE 22 – Schéma de l'interfaçage *ROS* du contrôle

Notre interfaçage *ROS* est composé de trois noeuds :

- Le noeud `update_node` qui permet de récupérer la localisation de la partie de fusion et la *map-matcher* avec la carte de *Séville*. En sortie, il publie sur un *topic* le rayon de courbure r, l'écart latéral e avec la polyligne *matchée*, l'angle α entre le véhicule et cette polyligne, la vitesse longitudinal du véhicule v et la zone de vitesse v_zone dans laquelle le véhicule se trouve.
- Le noeud `lateral_node` qui permet de d'envoyer la commande d'angle au volant δ au véhicule. Pour cela, il récupère les informations nécessaires dans le *topic* de `controle_update`, construit la loi de commande latérale et la publie dans le *topic* de contrôle de la voiture.
- Le noeud `longitudinal_node` qui permet de réguler le véhicule en accélération. Pour cela, il récupère la vitesse comprise dans le `controle_update` et l'accélération du véhicule et publie le torque sur le *topic* de contrôle de la voiture.

De plus, il a été question de créer deux types de messages *ROS* personnalisés ainsi que des paramètres reconfigurables pour chacun des noeuds.

Les deux messages crées sont les suivant :

— un message `msg_control_pose` qui contient les données d'une pose x, y, θ,
— un message `msg_control_input` qui contient les données du *topic* d'*update*.

Les *dynamic reconfigure* permettent la reconfiguration de certain paramètres *ROS* et concernent les paramètres suivants : les valeurs des zones de vitesses du contrôle longitudinal et les gains de son contrôleur PID, les gains du contrôleur PD latéral et les constantes essentielles au véhicule ; le facteur de volant, le facteur liant l'accélération au couple, le facteur supplémentaire lors d'une décélération et les accélérations maximales et minimales tolérées.

6.2 Détails sur les essais mis en place

Avant toute chose, puisque les parties justifiant nos choix présenterons majoritairement des résultats issus de simulation, il nous faut détailler le contexte de simulation mis en place. Il sera question de détailler le contexte des essais réalisés en pratique à bord du véhicule.

6.2.1 Essais en simulation

La simulation mise en place nous a permis de reproduire des conditions au plus proche de la réalité et ainsi pouvoir tester nos approches même lorsque le véhicule n'était pas disponible ou hors des séances projet.

Techniquement, cette simulation est basée sur un *package* python fourni en TD de SY27 par M. Bonnifait. Ce *package* a ensuite été modifié de manière à satisfaire nos besoin. Ce *package* permet :

— de simuler l'acquisition de la localisation par l'ajout de bruit de mesure [3],
— de paramétrer une loi de commande longitudinale et une loi de commande latérale de manière décentralisée (via un autre fichier spécifiques à ces lois),
— de lancer une simulation [4] et de la visualiser (ou non),
— d'obtenir, à la fin de la simulation, les graphiques d'écart latéral, d'écart de vitesse, de commande latéral et de commande longitudinal.

De plus, afin de déterminer le paramétrage des différents correcteurs, nous avons créer une « batterie de test » qui lance des essais pour plusieurs valeurs de pôles.

6.2.2 Essais en pratique

Les essais « en pratique » constituent les essais réalisés à bord du véhicule. Durant ces derniers, nous utilisons les différents modules suivants :

— `span` : permet d'avoir les données de la span, notamment utile afin de récupérer l'accélération (topic `/span/data/imu`) et la pose x, y, θ du véhicule (topic `/span/EnuPoseCovs`.
— `controle_zoe` : permet d'avoir la possibilité de contrôler la Zoé via l'envoi de contrôle (*Send controls*).

3. En pratique, nous ajoutions un bruit inférieur au mètre.
4. Il est possible de paramétrer la simulation selon sa durée et la période d'échantillonage du calcul.

— `sensors` permet l'activation des différents capteurs à bord du véhicule.
— `roscore` : permet de lancer le noeud principal *ROS*.

En effet, nous avons majoritairement utilisé la pose de la *span* (*RTK*) bien qu'étant interdite le jour de la démonstration afin de pouvoir tester notre contrôle au plus tôt.

Il est à noter qu'en pratique, Adrien en a fait un *roslaunch* permettant l'automatisation du lancement de ces modules.

6.3 Contrôle latéral

Dans un premier temps, traitons du contrôle latéral. Dans cette partie, il sera question d'évoquer les lois que nous avons choisi de tester dans le cadre de ce projet, que ce soit en pratique ou en simulation, puis la loi que nous avons décidé d'adopter et son paramétrage.

6.3.1 Modélisation du système

Concernant la modélisation du système pour contrôler le véhicule latéralement, le modèle vélocipède à traction avant a été choisi.

$$\begin{cases} \dot{x} = v\cos\delta\cos\theta \\ \dot{y} = v\cos\delta\sin\theta \\ \dot{\theta} = \frac{v\sin\theta}{L} \end{cases} \qquad \text{tel que le vecteur d'état soit } X = [x, y, \theta]^T$$

Avec x et y les coordonnées cartésiennes du véhicule sur le repère global de la carte de *Séville*, θ son orientation, v la vitesse moyenne des roues avant, δ l'angle de la roue virtuelle avant par rapport à l'axe longitudinal et L l'entraxe du véhicule [5]

Remarque : en pratique, nous implémentons également une avance de phase afin d'anticiper les changements de segment en régulant l'écart latéral au niveau de l'avant du véhicule.

6.3.2 Lois de commandes testées

Dans le cadre de ce projet, nous avons choisi de tester trois lois de commande latérales : un correcteur PD avec prise en compte du rayon de courbure, une loi de commande *Stanley* ainsi qu'une loi de commande par vitesse latéral (*LatVel*).

Concernant la notation des équations données par la suite, on admet ed comme l'écart désiré, e comme l'écart vis-à-vis de la polyligne la plus proche, α comme l'angle entre le véhicule et la polyline la plus proche et θ comme l'angle de l'orientation du véhicule. Également, ayant choisi de faire apparaître les équations sous format « test », on aura Δ correspondant à un nombre très faible, sub_mod_pi2 une fonction permettant de retourner une différence comprise entre $-\pi$ et π entre deux angles et ρ le rayon de courbure [6].

5. En pratique, $L = 2588m$ d'après la fiche technique de la Renault Zoé.
6. Car utilisant le rayon de courbure dans le contrôle longitudinal et non la courbure, nous avons homogénéisé cela en tout point.

Correcteur PD avec rayon de courbure

La loi de commande d'un correcteur PD avec rayon de courbure est la suivante :

$$\delta = \frac{L}{v^2 + \Delta} \cdot kp(ed - e) + kd \cdot (\frac{L}{v + \Delta}) \cdot sub\_mod\_pi2(\alpha, \theta) + \frac{L}{\rho + \Delta}$$

Ici, l'ajout du rayon de courbure permet de prendre en compte l'état de la route et représente d'une certaine manière l'accélération latérale que le véhicule connaîtra et ainsi d'impacter la commande en conséquence.

Loi de commande *Stanley*

Nous avons choisi d'implémenter l'algorithme de contrôle utilisé par l'Université de Stanford à l'occasion du *DARPA* Challenge de 2005. La loi de commande utilisée a été la suivante :

$$\delta = \arctan(\frac{k_1}{v + \Delta}(ed - e)) - sub\_mod\_pi2(\theta, \alpha)$$

Loi de commande *LatVel*

La loi de commande par vitesse latérale, ou *Kinematic lateral speed control law* permet de contrôler la vitesse latérale $\dot{dr}$ qui elle-même est contrôlée par la vitesse angulaire de la voiture θ et l'angle au volent δ. Sa loi de commande est la suivante :

$$\delta = \arctan\left(L\left(-k1 \cdot \sin\theta_p - \frac{k1 \cdot k2 \cdot e}{v + \Delta} + \frac{c(s) \cdot \cos\theta_p}{1 - c(s) \cdot e}\right)\right)$$

Avec la courbure $c(s) = \frac{1}{\rho}$.

6.3.3 Résultats en simulation

Dans cette sous-partie, nous mettons en valeurs les résultats obtenus en simulation pour le contrôle latéral.

Correcteur PD avec rayon de courbure

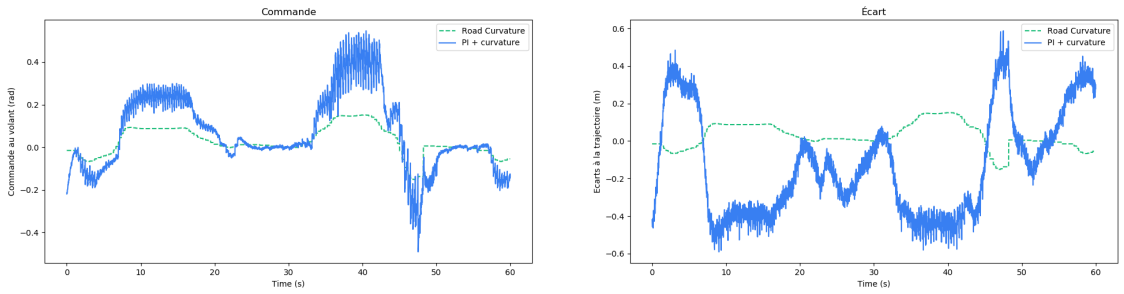

FIGURE 23 – Résultat simulation avec PD rayon de courbure ($kp = 2.0, kd = 1.0$)

Concernant le correcteur PD, nous remarquons une tendance générale à présenter une forte oscillation de la commande et un écart plutôt fort. Le meilleur résultat que nous avons obtenu est le suivant :

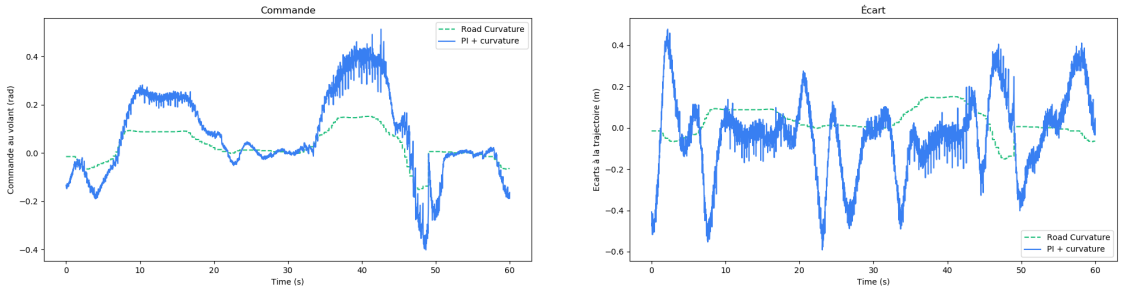

FIGURE 24 – Résultat simulation avec PD rayon de courbure ($kp = 2.0, kd = 0.1$)

Ici, l'écart latéral moyen est plus faible et la commande présente moins d'oscillation, ce qui se traduit par un comportement moins brusque mais plus « humain » en pratique.

Loi de commande *Stanley*

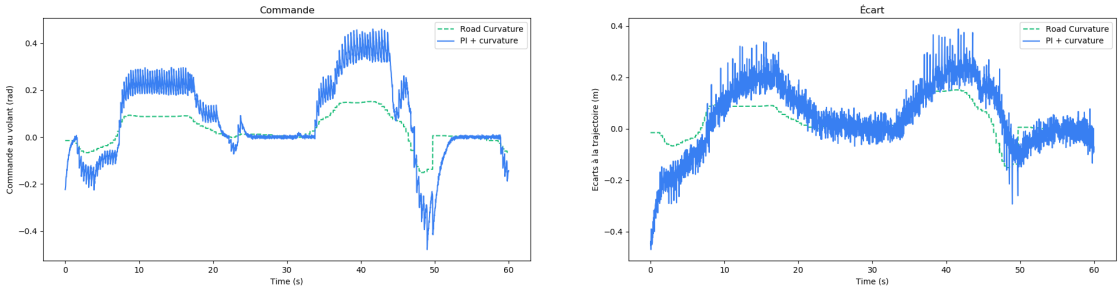

FIGURE 25 – Résultat simulation avec Stanley ($k = 0.5$)

Concernant la loi de commande de *Stanley*, nous remarquons une tendance à présenter un écart plus faible en moyenne, mais une forte oscillation de la commande. En changeant le paramétrage, on peut obtenir :

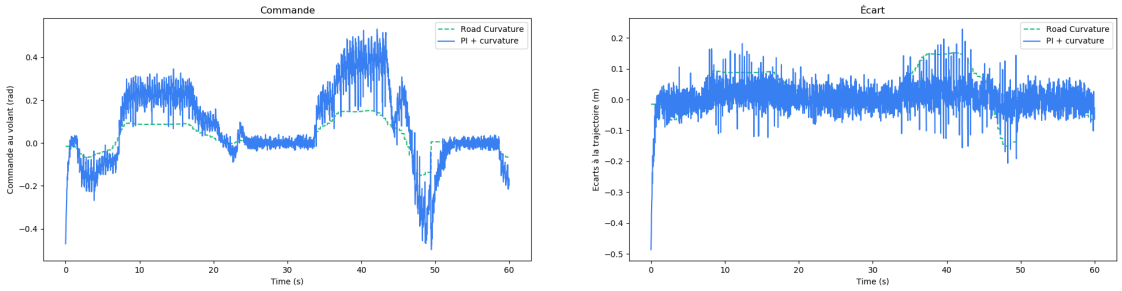

FIGURE 26 – Résultat simulation avec Stanley ($k = 3.0$)

Ce qui se traduit par un écart latéral moyen très faible, mais une commande beaucoup trop brusque. Nous ne sommes pas parvenu à obtenir de meilleurs

résultats pour la loi de commande de Stanley.

Loi de commande *Latvel*

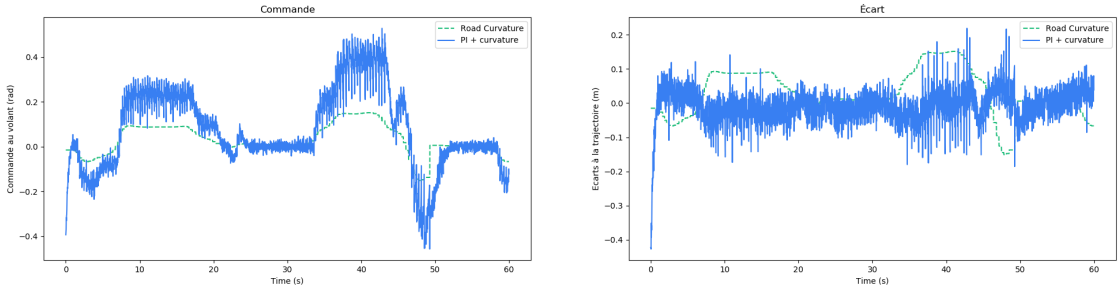

FIGURE 27 – Résultat simulation avec LatVel ($k1 = 0.3, k2 = 10$)

La loi de commande *LatVel* présente elle aussi un écart très faible, mais une forte oscillation de la commande. En changeant le paramétrage, on peut obtenir :

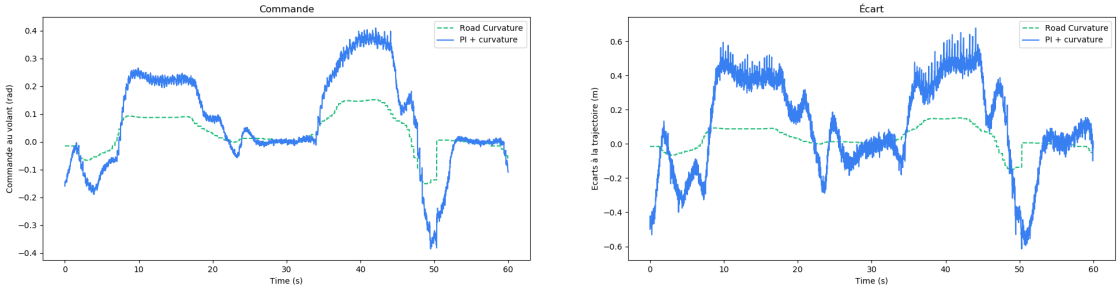

FIGURE 28 – Résultat simulation avec LatVel ($k1 = 0.1, k2 = 10$)

Ce qui se traduit par un écart latéral moyen plus élevé, mais une commande beaucoup moins oscillatoire.

Ainsi, nous sommes parvenu à la conclusion d'utiliser une loi de commande latéral par correcteur PI prenant en compte le rayon de courbure puisqu'elle présente une commande peu oscillatoire et un écart moyen plus faible que la loi de commande *LatVel*.

6.4 Contrôle longitudinal

Dans un second temps, traitons du contrôle longitudinal. Dans cette partie, il sera question d'évoquer la manière dont nous avons géré les vitesses à adopter sur *Séville*, la loi de commande que nous avons implémenté afin de les atteindre ainsi que son paramétrage et ses résultats.

6.4.1 Modélisation du système

Nous choisissons le modèle cinématique suivant de la voiture :

$\begin{cases} \dot{v} = a \end{cases}$ tel que a soit l'accélération du véhicule

6.4.2 Approche adoptée : zones de vitesses

Avant de parler de la loi de commande, il nous faut traiter de la manière dont nous avons déterminer les vitesses à adopter sur la piste.

L'approche que nous avons choisi se base sur la valeur du rayon de courbure et sur la définition de « zone de vitesse » selon cette valeur [7]

Pour chaque point x, y du chemin *Séville*, son rayon de courbure est défini par la formule suivante :

$$\rho(x, y) = \frac{(\dot{x}^2 + \dot{y}^2)^{3/2}}{\dot{x}\ddot{y} - \dot{y}\ddot{x}}$$

tel que (x, y) soient les coordonnées des points du chemin et $\dot{x}, \ddot{x}, \dot{y}, \ddot{y}$ les dérivées premières et secondes des coordonnées.

N'ayant pas accès à l'équation analytique du chemin de la route de *Séville*, nous avons, grâce aux points de la courbe et leurs variations, calculé des estimations des dérivées premières et secondes pour chaque point du chemin. Ainsi, nous avons obtenu pour chaque point x, y un rayon de courbure ρ.

En pratique, nous avons décidé de déterminer quatre zones de vitesses : 1, 2, 3 et 4 [8] dont leur vitesse est paramétrables sous *ROS* et dont leur attribution dépend d'un seuil sur le rayon de courbure. La carte obtenue est alors la suivante :

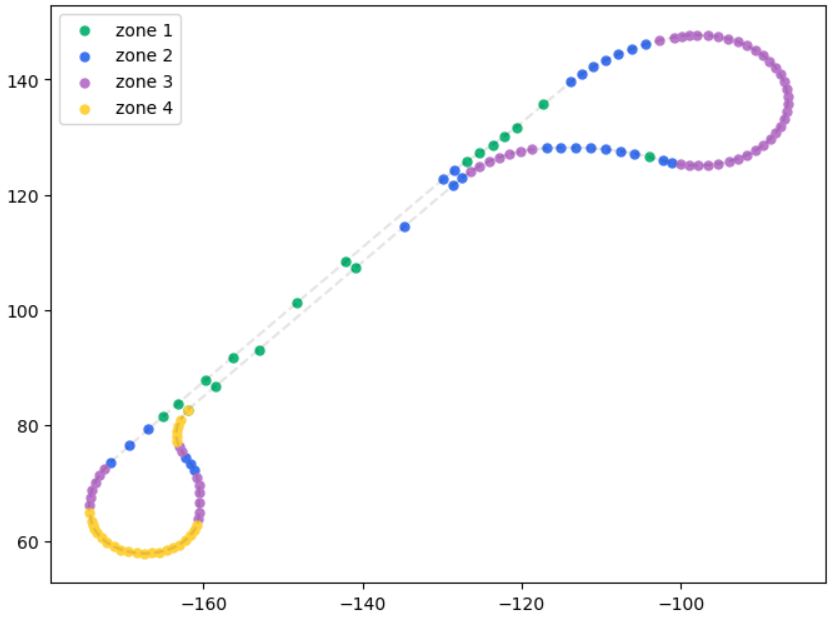

FIGURE 29 – Schéma des zones de vitesses de *Séville*

7. Cette valeur est limité en pratique puisque en ligne droite, elle est sensé être infinie.
8. 4 étant la zone de vitesse la plus faible et 1 la plus forte.

On remarque que cette décomposition de zone présente plusieurs erreurs. En effet, les des changements de courbures (le passage d'une courbure droite à gauche par exemple), le seuillage détecte une « ligne droite » et on passe alors en zone 1 après une zone 4, comme lors du petit rond point au Sud de la piste. Également, on remarque quelques points solitaires au milieu de point d'une autre zone.

Afin de corriger ces erreurs, nous avons décidé d'appliquer un filtrage « Zero-phase digital filtering » (ou *filtfilt*) en Python, ce qui permet de filtrer les zones de vitesses et de limiter les points solitaires.

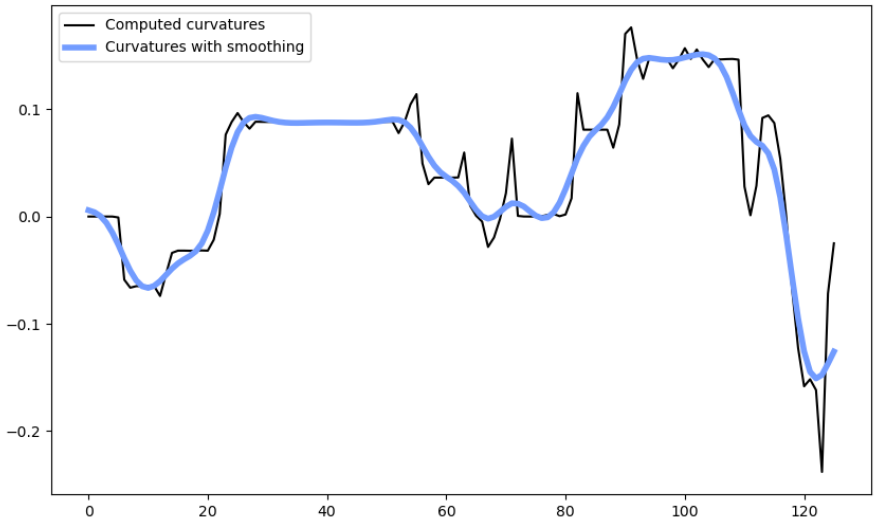

FIGURE 30 – Graphique du rayon de courbure de la piste filtrée et non

Ce qui donne la discrétisation en zones de vitesse sur la piste suivante :

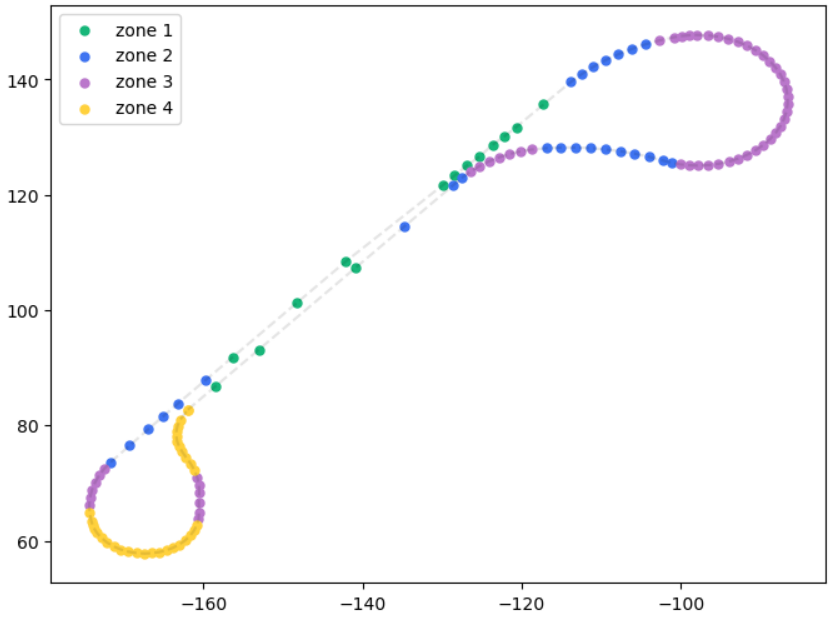

FIGURE 31 – Schéma des zones de vitesses filtrées de *Séville*

Ainsi, nous avons éliminé la majorité des problèmes et obtenons une délimitation en zones de vitesses convenable. La conduite ici représentée n'est pas très « humaine » (par exemple, nous n'accélérons pas en sortie de virage comme il est coutumier de le faire) mais, d'une certaine manière, cela se fait via la loi de commande qui permet d'atteindre ces vitesses.

Remarque : pour certains détails, il a été question de corriger la carte « à la main », par exemple pour la sortie du grand rond point, afin d'obtenir une carte convenable minimisant les potentielles situations dangereuses.

6.4.3 Loi de commande longitudinale

La loi de commande longitudinale utilisée pour atteindre la vitesse souhaitée est un correcteur PID dont l'équation est donné ci-contre :

$$a = kp \cdot \epsilon + ki \cdot \int_0^t \epsilon + kd \cdot \dot{\epsilon}$$

avec ϵ l'erreur sur la vitesse.

Ainsi, il nous a suffit de paramétrer, en pratique, les trois gains kp, ki et kd afin d'obtenir un résultat « humanisant » le comportement longitudinal du véhicule.

6.4.4 Résultats en simulation

En simulation, le contrôle longitudinal a majoritairement deux comportements :

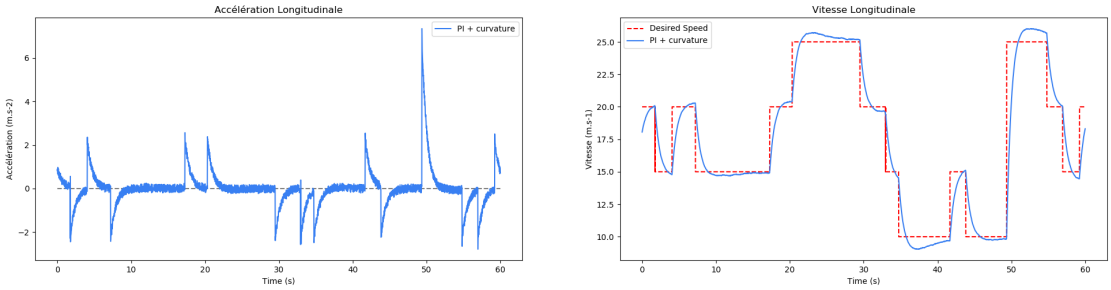

FIGURE 32 – Résultat simulation avec PID ($kp = 1.8, ki = 0.4, kd = 0.1$)

Un comportement premier durant lequel la consigne en vitesse est plus « douce » et ne « broute » [9] pas mais durant lequel on observe des dépassements de vitesses et une faible réponse aux vitesses souhaitées.

9. Cela s'observe lors des plateaux de vitesses, ce qui est plus présent dans la seconde simulation.

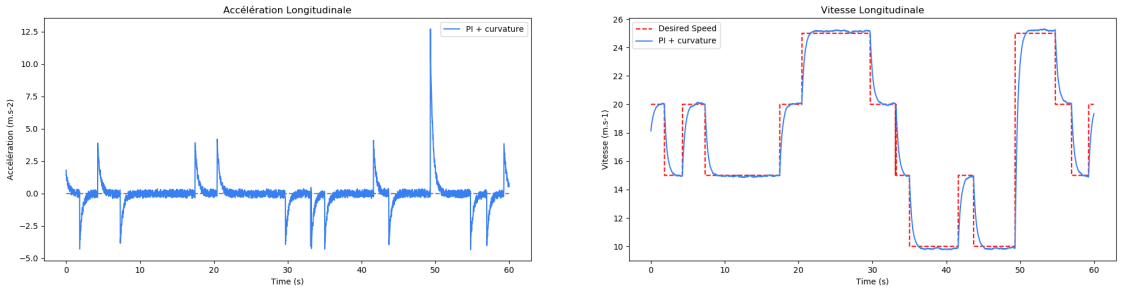

FIGURE 33 – Résultat simulation avec PID ($kp = 3.0, ki = 0.2, kd = 0.1$)

Durant ce second comportement, on converge plutôt vite aux vitesses souhaitées, mais les pics d'accélération sont plus élevés se traduisant pas un comportement plus « brusque » mais une tendance à « brouter » plus importante puisque le système cherche à rester à la même vitesse.

6.5 Résultats finaux observables

Voici, à titre d'illustration, les résultats que nous avons obtenus en pratique dans le véhicule après *tuning* des gains [10] déterminés en simulation :

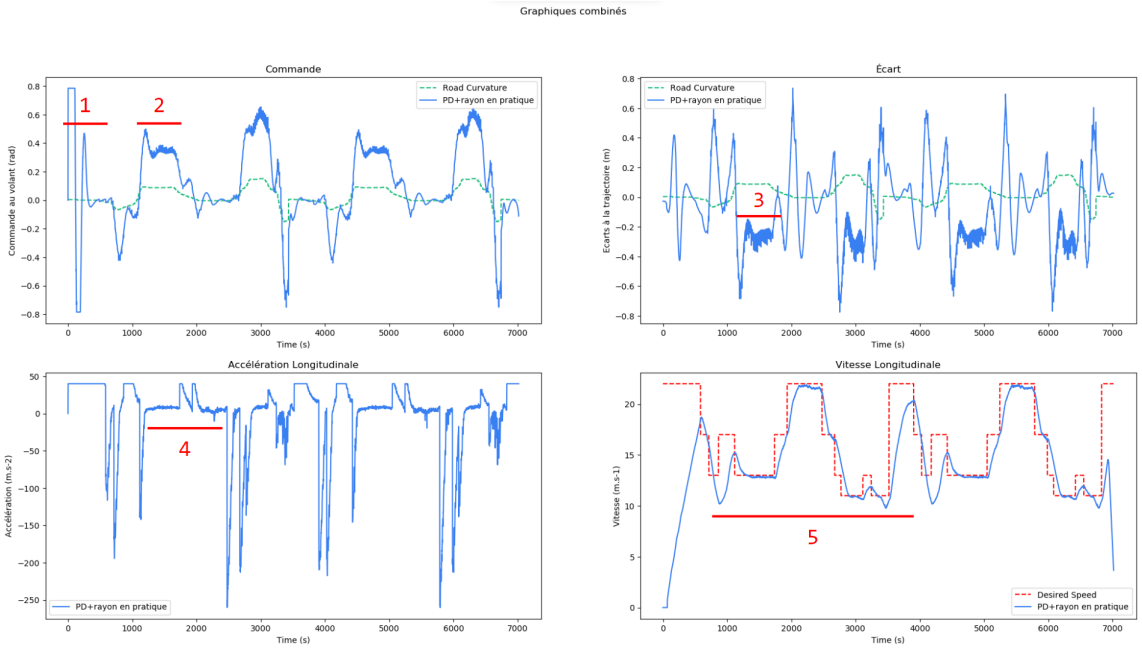

FIGURE 34 – Essai pratique sur *Séville* dans le véhicule

1. On observe durant cette phase que le système est en phase de convergence : on réalise deux oscillations latérales (manifestées par la saturation de la commande) et on cherche à converger vers la vitesse longitudinale souhaitée (illustrée par la saturation de l'accélération et l'augmentation croissante de la vitesse de à $0m/s$ à environ $17km/h$).

10. Avec pour gains latéral : $kp = 2.0 et kd = 0.1$ et pour gains longitudinal : $kp = 2.0, ki = 0.4, kd = 0.1$.

2. Ici sont observées des oscillations de la commande mais, ces dernières étant plutôt faibles, cela ne se ressent pas dans la voiture.
3. De la même manière, lors des virages longs, c'est-à-dire ceux des deux rond-points extérieurs, l'écart présente plusieurs oscillations.
4. Ici, l'accélération mesurée correspond bien à l'accélération envoyée à la voiture et cet écart entre accélération et décélération s'explique par le fait qu'il faille ajouter un facteur supplémentaire lors d'une décélération afin de pouvoir ralentir le véhicule convenablement. En pratique, notre facteur de décélération supplémentaire était de 5.
5. Est illustré ici le fait que notre correcteur PID soit réglé de manière à obtenir une vitesse plutôt lisse. En effet, cela permet de pallier à la discrétisation de la vitesse de consigne de seulement quatre valeurs et aux changements brusques de zones.

Remarque : Par simplicité d'affichage, l'angle au volant (appelé commande sur le graphique) à été divisé par le coefficient multiplicateur du volant. De la même manière, les vitesses ont été mises sous le format km/h bien qu'étant uniquement traitées en m/s dans notre programme.

6.5.1 Prise de recul quant aux résultats et améliorations possibles

Au final, d'après les essais que nous avons pu effectuer, que ce soit en simulation ou en pratique, nous trouvons le contrôle plutôt réussi. Le contrôle latéral adopte un comportement lissé durant les tournants et très peu brusque. Le contrôle longitudinal permet d'atteindre les vitesses souhaitées sans donner de coup d'accélération trop élevé, ni de coup de frein trop forts.

Néanmoins, il serait possible d'améliorer le contrôle longitudinal car la discrétisation en seulement quatre zones adoptées présente des failles. En effet, il est possible d'avoir une transition d'une zone de vitesse 1 à une zone 3 sans passer par la seconde phase, et simplement car elle ne propose qu'un faible choix de vitesse disponible. En pratique, cela est compensé par le réglage du PID, comme vu plus haut, mais il aurait pu être possible de créer un programme qui aurait pu déterminer les vitesses à adopter à chaque point de la piste en fonction des actions futures que le véhicule devra subir (non pas voir un point en avant mais bien plusieurs et pouvoir réguler sa vitesse en conséquence. De même, les oscillations au démarrage de la commande latérale, à très basse vitesse, peuvent être également un inconvénient de confort.

Conclusion

Conclusion personnelle de Manon :

Ce projet a été une vraie source d'intérêt pour moi. La partie dont j'étais en charge était très challengeante car l'outil utilisé (le filtre PHD, au cas où vous auriez encore un doute) n'est pas simple à appréhender. Aujourd'hui, nous obtenons des résultats concluants même si j'aurais souhaité apporter quelques petites améliorations. Ce projet a été frustrant sur beaucoup d'aspects, mais je le considère globalement comme une réussite pour ma part.

Conclusion personnelle de Ahamed :

J'ai adoré participer à ce projet et me charger de la partie de fusion de données. Ce semestre, j'ai eu l'occasion de faire SY28 également et cela m'a apporté une double approche aux filtrages et à la fusion de données. Comme la majorité d'entre nous, je suis déçu du problème que nous avons eu dans la fusion et j'ai été particulièrement frustré par la transition entre le filtre de Kalman python vers C++. Néanmoins, je suis content de l'effort de groupe et du projet dans son ensemble.

Conclusion personnelle de Samuel :

Ce projet est sans aucun doute le plus intéressant et le plus difficile auquel j'ai été confronté dans mon cursus. Je suis ravi d'avoir été dans cette équipe, et je suis très reconnaissant envers toute l'aide que j'ai reçue autant de la part de mes camarades comme de l'équipe enseignante. Je regrette ne pas avoir été en mesure de fournir un résultat fonctionnel par moi-même malgré cette aide.

Conclusion personnelle de Adrien :

Ce projet est sans doute l'un des plus prenant et stimulant auquel j'ai été confronté. Il m'a permis de bien mieux comprendre les contraintes de la navigation autonome, ainsi que la nécessite d'une optimisation pour le temps réel. La fin du projet cependant est pour moi décevante, dans la mesure où deux implémentations différentes du filtre de *Kalman* en C++ ont eu des *bugs* que nous n'avons pas pu résoudre à temps. Ne voulant pas rester sur cette sensation d'inachevé, je continuerai à chercher la source des problèmes sur ces programmes les prochains jours.

Conclusion personnelle de Victor :

À titre personnel, j'avais de mauvais souvenirs de mon apprentissage du contrôle en SY15, c'est pourquoi j'ai souhaité repartir sur de nouvelles bases en SY27 et changer mon *a priori* sur ce domaine. J'ai trouvé le contrôle très intéressant à implémenter sous *ROS* et les essais en voiture très pédagogique ; on y voit vraiment les conséquences de nos paramétrages. Je suis satisfait de ce que nous avons pu produire sur cette partie et souhaitais remercier Anaïs qui m'a aidé tout au long du semestre (surtout sur le contrôle longitudinal) et a su m'éclairer sur les aspects théoriques que je ne

comprenais pas. Ce projet est, à mes yeux, réussi.

Conclusion personnelle de Anaïs :

J'ai été contente d'avoir pu participer et aider au projet en tant qu'« externe » à SY27, il s'agissait d'une bonne occasion de mettre en pratique un contrôle autonome de véhicule à taille réelle et d'être confrontée aux difficultés de mise en pratique en voiture. Après réflexion, je regrette de ne pas avoir combiné mes cours de *Master* (notamment ARS5) avec ce projet. Il aurait été possible de mettre en place une commande latérale ou longitudinale dynamique en modélisant des lois de commande non linéaires, mais je reste contente du résultat final ! Merci de m'avoir comptée dans l'équipe, même si j'aurais aussi préféré gagner des crédits.

Bibliographie

[1] Salvador DOMINGUEZ, Alan ALI, Gaetan GARCIA and Philippe MARTINET. *Comparison of lateral controllers for autonomous vehicle : experimental results*. 19th International Conference on Intelligent Transportation Systems (ITSC), 2016.

[2] Qiang HUA, Baoshan PENG, Xiaolin MOU, Ouwen ZHANG, Tao HE, Li XIA, Heyan LI *Model Prediction Control Path Tracking Algorithm Based on Adaptive Stanley*

[3] Yaoting CHEN and Yanping ZHENG *A Review of Autonomous Vehicle Path Tracking Algorithm Research*

[4] Meng WANG, Juexuan CHEN, Hanwen YANG, Xin WU and Longjiao YE *Path Tracking Method Based on Model Predictive Control and Genetic Algorithm for Autonomous Vehicle*

[5] John Mullane, Ba-Ngu Vo, Martin Adams, Ba-Tuong Vo, *Random Finite Sets for Robot Mapping & SLAM, New Concepts in Autonomous Robotic Map Representations, 2011, Volume 72*