



# Rapport de stage assistant ingénieur

ANTOINE HUOT-MARCHAND

ANNÉE UNIVERSITAIRE 2021-2022

# **Remerciements**

Tout d'abord, je tiens à remercier vivement, Mme Diane Lingrand pour la proposition de stage ainsi que pour son aide pendant toute la période du stage. De plus je souhaite remercier le centre de la MIA pour avoir mis à disposition le matériel nécessaire. Enfin je remercie mes autres collègues stagiaires et ingénieurs dont Erdal Toprak, Abdel-Qahar Traore et Li Yang pour leur aide et conseils.

## **Informations du rapport intermédiaire**

La nature du stage est un stage se déroulant en France, à Biot au sein de l'entreprise Polytech et plus précisément au près de l'équipe du laboratoire i3S. Le stage se déroule en présentiel du 18/07/2022 au 26/08/2022 et l'intitulé du sujet est « GUI pour la génération de trajectoire ».

# Résumé étendu du stage

Le but du projet est de développer une interface graphique permettant de générer des trajectoires par tracé manuel prenant en compte les variations de vitesse du tracé et les contraintes physiques des véhicules (par exemple : entraxe, angle de braquage ...). Il conviendra d'étudier différents types de sorties (commandes) selon les véhicules. On pourra envisager la création de scénarios à plusieurs véhicules.

Bien entendu, afin de vérifier le comportement de la voiture selon la trajectoire tracée, on utilisera un logiciel de simulation et de visualisation nommé RVIZ. Ce logiciel qui est une interface graphique du langage ROS connu pour la robotisation, permettra de se rendre compte du comportement de la voiture et d'ajuster les différents calculs et paramètres par la suite.

Afin de simuler le comportement de la voiture selon la trajectoire tracée en amont, on générera un fichier de commandes obtenu par des calculs mathématiques portant sur plusieurs modèles expérimentés au long du stage.

Ainsi à l'aide d'un suivi de trajectoire de plus en plus précis, cela servirait à entraîner des voitures miniaturisées sur un circuit prédéfini afin que celles-ci deviennent autonomes. Ce projet pourrait alors par la suite être expérimenté à plus grande échelle.

## **Extended summary of the internship**

The goal of the project is to build a graphic interface allowing to generate several trajectories by manual tracing which takes into account the variations of tracing speed and the physical constraints of the vehicles (for example: wheelbase, steering angle ...). It will be necessary to study different types of outputs (commands) according to the vehicles. We can consider the creation of scenarios with several vehicles in the future.

Of course, in order to check the behavior of the car according to the drawn trajectory, we will use a simulation and visualization software called RVIZ. This software which is a graphic interface for ROS language known for robotization process, will make it possible to realize the behavior of the car and adjust the various calculations and data in function of results afterwards.

In order to simulate the behavior of the car according to the trajectory traced upstream, a command file is generated obtained by mathematical calculations relating to several models tested throughout the course.

Thus, using increasingly precise trajectory tracking, this would serve to train miniaturized cars on a predefined circuit so that they become autonomous. This project could then subsequently be tested on a larger scale.

# Sommaire

I - Introduction et contexte du travail	6
II - Description du travail réalisé	7
II.1 - réalisation de l'interface graphique	7
II.2 - Installation et utilisation du logiciel RVIZ	9
II.3 - Étude de la trajectoire et génération du fichier de commandes	10
II.3.1 - Le format de commande	10
II.3.2 - Les modèles de trajectoire envisagés	12
II.3.2.1 - Le modèle 1 (modèle intuitif)	12
II.3.2.2 - Le modèle 2 (modèle intuitif corrigé)	14
II.3.2.3 - Le modèle 3 (modèle intuitif amélioré)	16
II.3.2.4 - Le modèle 4 (modèle bicyclette dit « Pure pursuit »)	17
II.3.2.5 - Le modèle final (modèle bicyclette simple)	20
III - Difficultés et problèmes rencontrés	23
III.1 - La différence du système de coordonnées	23
III.2 - Le calcul de la position initiale	24
III.3 - Intervalles et limites	25
III.4 - Le temps de commande	26
IV - Planning	27
V - Conclusion, apports de l'expérience et perspectives	28
VI - Bibliographie	29

# I - Introduction et contexte du travail

Tout d'abord, ce stage porte sur un projet mené par des membres de l'équipe i3S de Sophia Antipolis. Le projet global consiste à développer et étudier des modèles de voitures autonomes. En effet, des membres de l'équipe ainsi que d'autres stagiaires travaillent sur l'apprentissage et l'entraînement des voitures afin que celles-ci puissent se déplacer toutes seules sans ordres ni commandes à recevoir.

Pour mener à bien cette mission, un circuit pour des voitures miniatures robotisées est à disposition au centre de la MIA (Maison de l'intelligence artificielle) de Sophia Antipolis. Le but est donc d'apprendre dans un premier temps aux véhicules à prendre les bonnes trajectoires tout en évitant les divers obstacles et bordures présents sur le circuit. Cela permet donc d'étudier différents modèles à petite échelle.

Ainsi c'est dans cette optique que j'ai pu prendre part à une autre mission portant sur ce même projet durant mon stage. Celle ci consiste donc à réaliser une interface graphique permettant de dessiner des trajectoires pour que la voiture puisse se déplacer à l'aide d'un fichier de commandes généré par ce tracé. À l'origine les fichiers de commandes étaient créés à la main ce qui était plutôt fastidieux.

Le travail demandé se divise donc en plusieurs parties. En effet, il faut en premier lieu, créer une interface graphique permettant à l'utilisateur de dessiner des trajectoires sur un circuit prédéfini. De plus, il faut réussir à étudier les positions et autres variables du tracé afin de générer un fichier de commandes. Enfin il faut analyser les résultats de la simulation pour ajuster nos calculs afin de générer un fichier de commandes plus adapté à la trajectoire et au modèle de la voiture.

Selon le temps restant avant la fin du stage, il sera possible d'améliorer la précision du tracé et d'envisager un scénario à plusieurs véhicules tout en améliorant l'IHM (interface homme-machine) de l'application avec l'ajout de nouvelles fonctionnalités qui peuvent faciliter la tâche de l'utilisateur.

## **II - Description du travail réalisé**

### ***II.1 - réalisation de l'interface graphique***

Effectivement, lors de la première semaine du stage, je me suis concentré sur la réalisation de l'interface graphique qui permettra à l'utilisateur de tracer les différentes trajectoires souhaitées. Il a fallu tout d'abord choisir à partir de quelle technologie et langage travailler afin que cela soit le plus simple et portable possible. Le choix s'est donc porté sur le langage Python ainsi que sur la bibliothèque Qt permettant de générer des interfaces graphiques sur divers langages comme Python ou encore C++. Dans un premier temps, j'ai eu l'idée de travailler sur un langage web afin de réaliser l'interface à l'aide de HTML et de CSS mais cela pose plus de contraintes car les applications web sont plus difficiles à démarrer si celles-ci sont sur un port local et ne sont pas distribuées via un nom de domaine par exemple. De plus concernant les autres langages, Python semblait plus accessible et adéquate pour la taille du projet.

Après avoir choisi, la technologie utilisée, j'ai donc pu commencer à expérimenter le module PyQt (bibliothèque Qt au sein de Python). J'ai alors procédé par étapes car j'ai dû apprendre à utiliser ce module dont je ne m'étais jamais servi auparavant.

Tout d'abord, j'ai essayé de créer une interface basique permettant uniquement de tracer des trajectoires sur un fond blanc à l'aide de la souris. PyQt permet en effet de tracer des formes géométriques que ce soient des points, polygones, des lignes (etc...) mais cela se fait en connaissant les points constituant les figures à l'avance. Aucune fonction ne permet de tracer une courbe suivant le mouvement de la souris de façon prédéfinie. Ainsi en faisant des recherches et en lisant la documentation officielle de PyQt, j'ai pu savoir comment obtenir la position de la souris sur l'interface.

L'interface possède son propre système de coordonnées (pixels en 2D), avec le coin supérieur gauche qui correspond au pixel (0;0). Le système de coordonnées est tel que x est croissant dans le sens de la droite et y dans celui du bas.

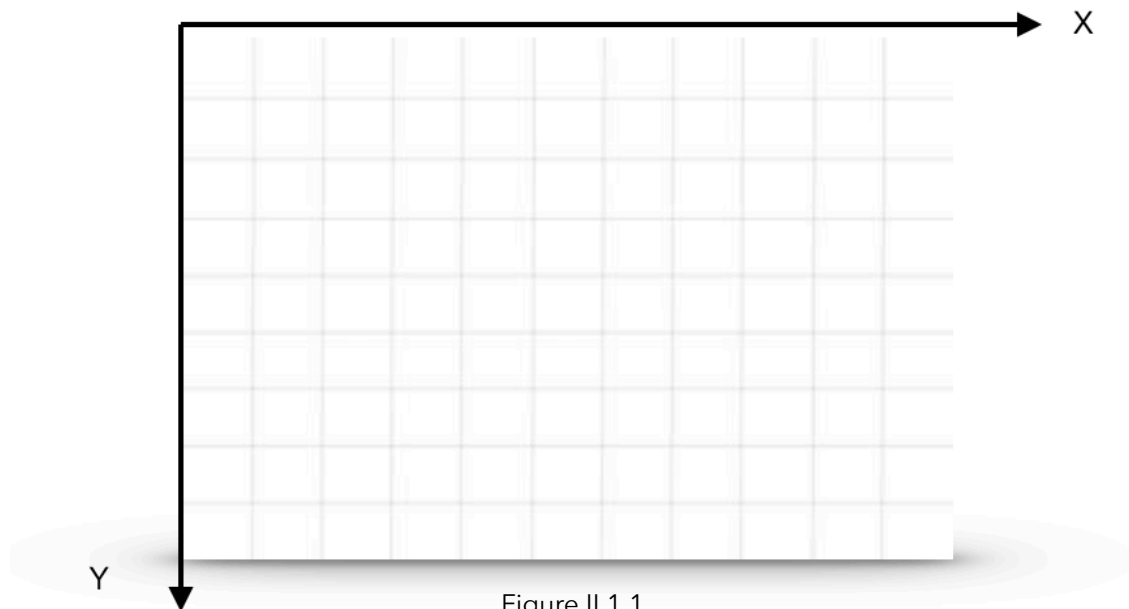


Figure II.1.1

Une fois que la position de la souris peut être obtenue assez facilement, il suffit donc pour tracer une courbe suivant les mouvements de la souris, de mémoriser tous les points par lesquels la souris passe et de tracer une ligne droite entre chacun de ses points deux à deux afin d'avoir une courbe continue et non pas un nuage de points. Bien sûr, j'ai mis en place un booléen qui s'assure que le tracé et la mémorisation des points survolés par la souris ne soient actifs que lorsque le clic de la souris est enfoncé.

Dans PyQt, des méthodes-événements sont déjà créées telles que ***mousePressEvent***, ***mouseMoveEvent***, ***mouseReleaseEvent*** ou encore ***paintEvent***. Il suffit donc de redéfinir ces méthodes afin de traduire ce que l'on souhaite faire pendant ces différents événements que sont le clic, le mouvement ou relâchement de la souris par exemple.

Ensuite, j'ai donc pu améliorer l'interface en insérant en fond, l'image du circuit sur lequel on veut faire le tracé et qui correspond d'ailleurs à une représentation du circuit réel sur lequel les voitures robotisées sont testées. Notre interface est donc un *Canvas* sur lequel on va ajouter des éléments à l'aide de widgets comme *pixmap* qui est un format de fichier graphique pouvant supporter des images comme c'est le cas ici.

Afin de faciliter et d'améliorer l'expérience et l'interaction de l'utilisateur avec l'interface j'ai décidé de mettre en place une palette de couleurs. Ainsi lorsque l'utilisateur clique sur la couleur voulue, la couleur du tracé est automatiquement modifiée.



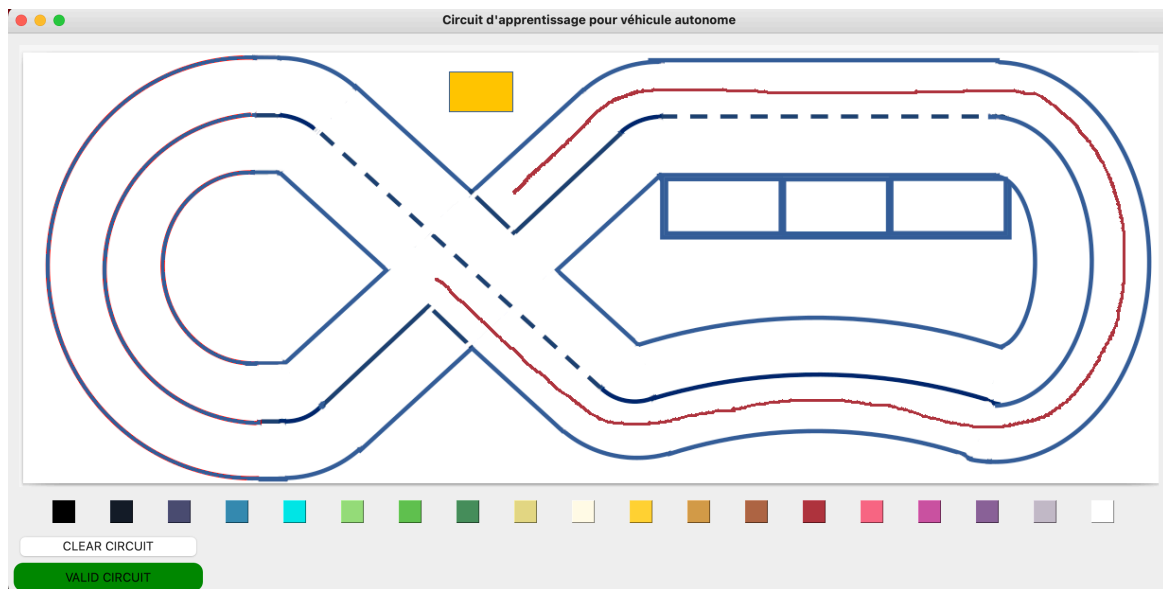


Figure II.1.2

Enfin deux boutons sont pour le moment à disposition de l'utilisateur. L'un permet de réinitialiser le tracé et donc d'effacer le tracé effectué auparavant (dans le cas d'une erreur par exemple). L'autre bouton permet quant à lui de valider la trajectoire dessinée sur l'interface et génère donc le fichier de commande dont on parlera ultérieurement dans ce rapport.

Les différents boutons et la palette de couleur implémentés sont eux aussi des widgets (*buttons*) dans le cas présent, que l'on va implémenter à l'intérieur de la fenêtre principale contenant le *Canvas*.

## II.2 - Installation et utilisation du logiciel RVIZ

Une fois que l'interface graphique minimale était au point, il était essentiel d'installer les outils nécessaires à la simulation et la visualisation du parcours des voitures et ce avant de réfléchir aux différents moyens de générer le fichier de commandes.

J'ai donc poursuivi mon travail en réalisant le tutoriel ROS et MuSHR qui explique comment créer ou obtenir l'environnement nécessaire à l'utilisation de RVIZ. Ce tutoriel n'est censé ne prendre que quelques minutes ou heures. Cependant j'ai mis plus de temps à le finir complètement à cause de plusieurs problèmes. Le logiciel RVIZ n'est disponible que sur Linux et ne possédant pas une machine Linux, il était alors impératif d'avoir un environnement virtuel. Or certains environnements virtuels ne sont pas directement compatibles avec la version de ROS proposée et conseillée dans le tutoriel. Après plusieurs heures, j'ai donc pu terminer le tutoriel.

Par conséquent, il me restait donc à changer certains paramètres tels que le circuit par défaut de RVIZ qu'il fallait remplacer par le nôtre ou encore la position initiale de la voiture, la résolution de la carte etc...


C'est à la suite de tous ces changements effectués, que j'ai pu alors me diriger vers l'étude et la génération du fichier de commandes permettant de simuler le comportement de la voiture dans RVIZ.

## **II.3 - Étude de la trajectoire et génération du fichier de commandes**

### **II.3.1 - Le format de commande**

La majeure partie du travail consiste donc à étudier les données de la trajectoire tracée afin de générer un fichier de commande qui permettra à la voiture de suivre ce parcours de la manière la plus précise possible.

Pour ce faire, il est donc obligatoire de connaître le format du fichier de commandes demandé. Celui-ci est disponible dans le tutoriel présenté dans la section précédente et montre alors le format ci-contre :



```
1 4.950,1.770,2.40626010
2 0.36796739,0.01649648
3 0.47087684,0.00815393
4 0.49952477,-0.00252343
5 0.47423623,-0.01030159
6 0.49175705,0.01174289
7 0.39755503,0.13348988
8 0.36045111,0.33144389
9 0.41012193,0.27426611
```

Figure II.3.1

Ainsi ce format est représenté par une première ligne contenant 3 paramètres représentant la position initiale de la voiture dans le système de coordonnées de RVIZ :  $X, Y, \theta$

$\theta$  représente ici l'angle initial de la voiture par rapport à la ligne des abscisses, quant à  $X$  et  $Y$ , ils représentent la position initiale de la voiture sur le circuit. À savoir que la position initiale de la voiture est calculée en un point particulier (en rouge sur le schéma ci-dessous). Ce point correspond au capteur de la voiture sur RVIZ. C'est en ce point que se produit l'impact si la voiture touche une bordure et se bloque.

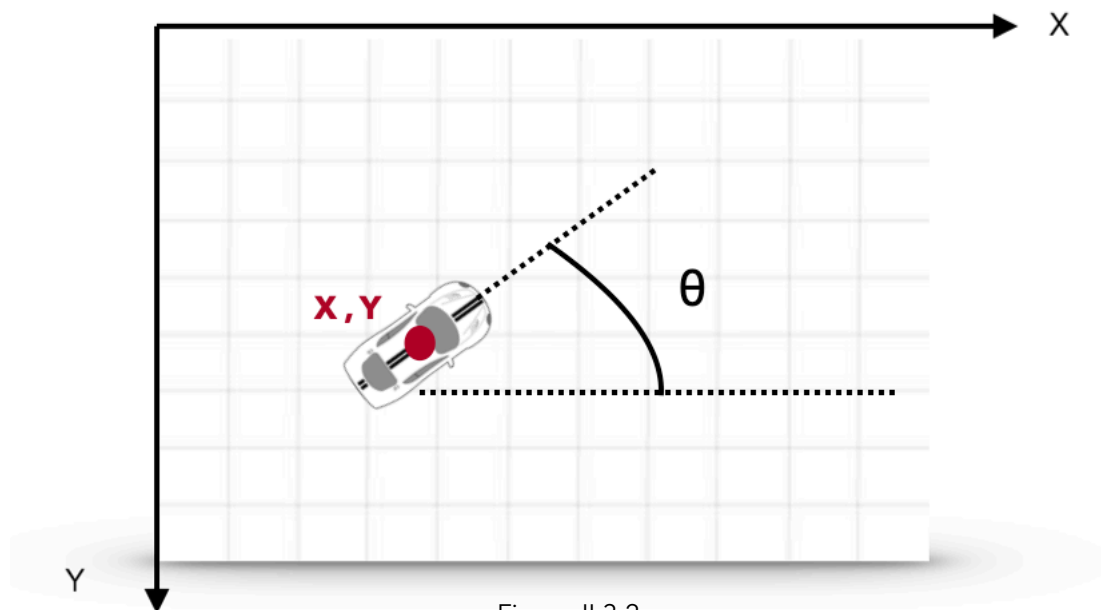


Figure II.3.2

De plus le fichier est composé d'autres lignes suivantes qui ont toutes le même format qui est :  $V, \delta$

$V$  représente la vitesse de la voiture et  $\delta$  l'angle des roues. Chacune de ces lignes s'exécute pendant un temps donné qui est par défaut de 1 seconde. Il est important de préciser que l'angle  $\delta$  est positif lorsque la voiture tourne à gauche et négatif lorsqu'elle tourne à droite.

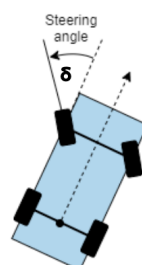


Figure II.3.3

## II.3.2 - Les modèles de trajectoire envisagés

### II.3.2.1 - Le modèle 1 (modèle intuitif)

Pour ce premier modèle, nous partons de rien et devons donc essayer de déterminer les différentes vitesses et angles à passer en commande. Pour ce modèle, nous ne nous préoccupons pas de la position et de l'angle de départ de la voiture que l'on fixe toujours au même endroit pour l'instant.

Ainsi nous avons besoin dans un premier temps de calculer la vitesse qui, on le rappelle doit être proportionnelle à la vitesse du tracé pour qu'on puisse ainsi faire ralentir ou accélérer la voiture à notre gré. Ma première approche a donc été de chronométrer le temps que l'utilisateur met à effectuer le tracé et ce pour plusieurs points de passage. Il faut d'abord se fixer des points de passage afin de récupérer le temps écoulé à chacun de ceux-ci. Ces points de passage ne doivent pas être ni trop éloignés ni trop rapprochés car une nouvelle commande s'effectue toutes les secondes et on parcourt tout de même une certaine distance pendant cette durée. L'idée est donc de définir un intervalle de points (environ 10 ou 15) et de mesurer le temps pour chacun de ses intervalles, c'est à dire du 1er au 15ème point, du 15ème au 30ème point et ainsi de suite.

Il nous faut aussi pour calculer la vitesse, mesurer la distance parcourue pendant cet intervalle de points. Le système de coordonnées étant en 2 dimensions, on mesure dans deux variables distinctes le déplacement en **X** noté **dx** et celui suivant l'axe **Y** noté **dy**. On obtient alors la distance **d** parcourue sur l'interface en utilisant le théorème de Pythagore. En effet, on a

$$d = \sqrt{dx^2 + dy^2}$$

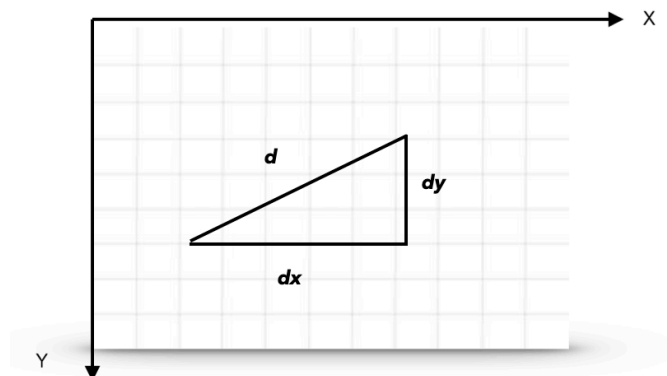


Figure II.3.4

Or pour obtenir la distance effectuée sur le circuit il faut réaliser une conversion pour passer de **pixels/s** à **m/s**. Pour cela il est nécessaire de calculer le ratio entre la taille du circuit sur l'interface graphique (en pixels) et celle du circuit réel (en mètres). Cependant la vitesse du tracé étant très rapide, il est encore nécessaire d'appliquer un coefficient pour régler ce problème et obtenir des vitesses atteignables par la voiture.

De plus, nous avons besoin de calculer l'angle à prendre à chaque commande pour que la voiture suive la trajectoire demandée. L'angle  $\delta$  demandé est celui des roues par rapport à l'axe de la voiture mais ne voyant pas comment calculer cet angle au début de mes recherches, j'ai pris en compte l'angle  $\alpha$  de la voiture qui est donc celui de la trajectoire. Il y a effectivement une différence car pour prendre un angle à  $90^\circ$ , la voiture ne tourne pas ses roues avec le même angle.

Afin d'obtenir cet angle, la méthode est donc la suivante, on calcule l'angle entre deux droites consécutives. Il faut donc au moins 3 points pour réaliser ce calcul. Le premier point est le point duquel provient la voiture, le 2ème celui sur lequel elle se trouve à l'instant présent et le dernier point et celui sur lequel elle doit se rendre.

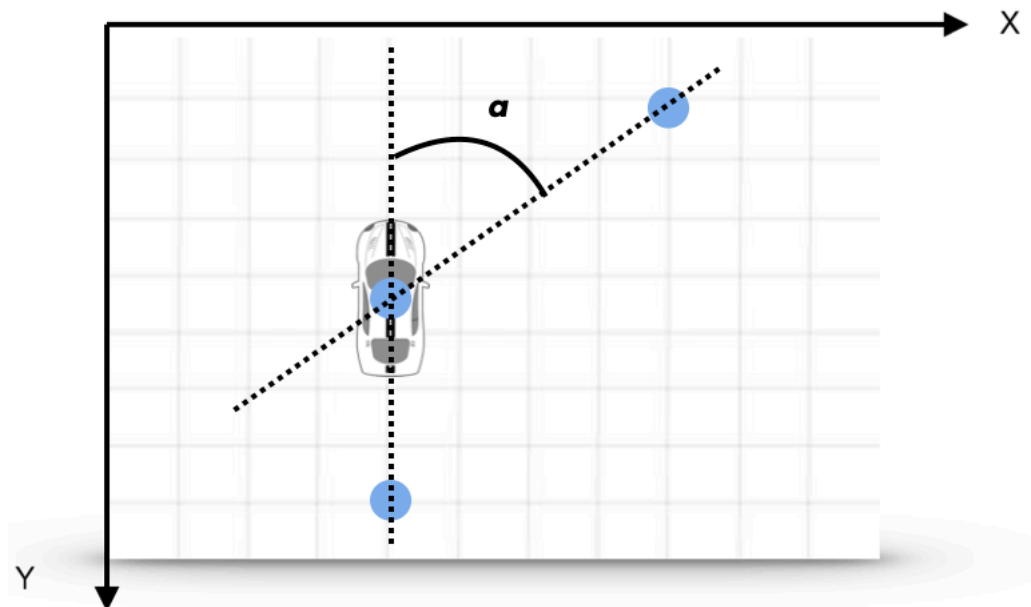


Figure II.3.5

Pour mesurer cet angle, il faut alors calculer le coefficient directeur de chacune de ses droites qu'on appelle  $m_1$  et  $m_2$  puis appliquer la formule suivante :  $\alpha = \tan^{-1} \left( \frac{m_1 - m_2}{1 + m_1 m_2} \right)$

Une fois ces deux paramètres obtenus, il faut vérifier que le suivi de la trajectoire par la voiture est conforme au tracé sur le logiciel de simulation. On remarque alors très vite que ce modèle n'est pas cohérent et pour cause, le calcul de la vitesse est fait sans se préoccuper précisément du temps de la commande. En effet, il est réalisé à partir de la distance parcourue certes, mais le temps de parcours est basé sur le temps du tracé. C'est pourquoi on ne calcule des vitesses qui ne sont pas faites pour un temps de parcours de 1 seconde mais pour un temps de parcours proportionnel à celui du tracé uniquement.

La simulation montre alors une voiture qui suit la tendance de la courbe comme les virages mais ne les effectue pas au bon moment. La voiture va donc soit trop vite, soit trop lentement selon le coefficient choisi et il est donc impossible d'adopter ce modèle pour la suite.

### **II.3.2.2 - Le modèle 2 (modèle intuitif corrigé)**

Ici, on reprend l'idée du modèle 1 sauf qu'on modifie la façon de calculer la vitesse de la voiture. Effectivement, on s'est rendu compte que la prise en compte du temps de parcours n'était pas la bonne. Le moyen le plus efficace est en fait de fixer le temps de parcours à 1 seconde pour chaque mesure puisque c'est le temps que met chaque commande à s'exécuter.

Ainsi le calcul de la vitesse correspond juste à la distance parcourue en pixels divisée par le ratio distance entre l'interface et le circuit réel. On réalise le calcul de l'angle de la même manière que pour le précédent modèle. La méthode est donc la suivante :

- ① On se fixe un intervalle ***i*** de points moyen (ni trop grand ni trop petit).
- ② À chaque fois que la souris a traqué autant de points que l'intervalle ***i***, on récupère la distance ***dk*** parcourue.
- ③ Cette distance ***dk*** devra être parcourue en 1 seconde (temps d'application de la commande) ce qui fait que la vitesse est autodéterminée par ce facteur.

④ Afin de déterminer l'angle à prendre pour la commande  $k$ , on regarde l'angle  $\alpha$  qui sépare la droite  $dk$  à parcourir et la droite  $dk-1$ .

⑤ Même si l'angle  $\alpha$  calculé n'est pas celui des roues, cela permet tout de même de faire des approximations.

Après avoir lancé la simulation à l'aide du fichier de commande généré, on remarque que la tendance de la trajectoire est globalement très bien reconstituée mais la voiture a soit un peu de retard ou soit un peu d'avance ce qui fait qu'à terme, la voiture dévie de la trajectoire et touche un obstacle ou une bordure.

Ce phénomène s'explique de plusieurs façons. Il est vrai que la vitesse calculée est une vitesse linéaire et donc dès lorsque l'angle  $\alpha$  à prendre est différent de  $0^\circ$ , la voiture ne parcourt pas exactement la distance prévue à l'origine car sa trajectoire est celle d'un arc de cercle plus ou moins arrondi selon la grandeur de l'angle. Ainsi plus l'angle est éloigné de  $0^\circ$ , plus la voiture dévie de sa trajectoire car elle n'atteint pas le point prévu à l'origine. De plus, en 1 seule seconde la voiture semble ne pas tourner assez vite pour prendre la trajectoire souhaitée.

J'ai donc, d'après mes différentes observations, réalisé quelques modifications au modèle. Afin de corriger les déviations de la trajectoire par la voiture, j'ai classé différents intervalles d'angles tels que les angles  $\alpha$  de 0 à 0.150 radians (catégorie A), les angles allant de 0.150 à 0.300 radians (catégorie B) et ceux qui sont supérieurs à 0.300 radians (catégorie C).

Pour les angles de la catégorie A, je fais en sorte que l'angle calculé s'ajoute à celui de la commande suivante. Pour les angles de catégorie B, j'augmente les angles à l'aide d'un coefficient car la voiture ne tourne souvent pas assez. Pour les angles de la catégorie C, je les limite à un angle maximum puisque l'angle des roues de la voiture ne peut pas dépasser 0.350 radians environ.

Ces différentes améliorations se ressentent mais sont trop instables et volatiles pour que le modèle soit adopté. En effet ces modifications pourraient devenir obsolètes dans le cas d'un changement de véhicule ou de circuit. Certaines portions du circuit s'adaptent d'ailleurs moins bien que d'autres à ces modifications qui ne sont qu'expérimentales.

### II.3.2.3 - Le modèle 3 (modèle intuitif amélioré)

Afin de rendre le modèle 3 moins volatile que le modèle précédent, j'ai abordé un autre point de vue quant à l'arrangement des commandes trop volatiles. Effectivement au lieu de me concentrer sur l'unique modification des angles, j'ai pris la décision de modifier les angles en fonction de la vitesse à laquelle la voiture est censée avancer. D'un point de vue physique, plus la vitesse de la voiture est élevée, plus celle-ci déviara vite avec un même angle donné. Il est donc nécessaire de trouver un juste milieu pour que la majorité des angles pris à la vitesse demandée puisse guider la voiture au bon point de destination. D'après les dernières observations, la voiture ne tournait pas assez vite dans certains virages. En analysant plus profondément le problème, j'ai donc pu remarquer que lorsque cela se produisait, la vitesse était relativement basse par rapport à d'autres sections de route. J'ai donc déterminé le seuil de vitesse à partir duquel les variations devenaient critiques pour un bon suivi de la trajectoire. Ce seuil a alors été observé à environ 0.3 m/s.

Néanmoins en dessous de ce seuil, il faut garder une certaine notion de proportionnalité puisque la voiture ne va pas se comporter de la même façon avec des angles trop éloignés. C'est pourquoi en dessous d'une vitesse de 0.3 m/s, j'ai appliqué à cette vitesse une rectification proportionnelle suivant la formule ci-contre :

$$v = v + \frac{|\sin \alpha|}{2.5}$$

Dans cette relation le sinus garantit la proportionnalité. C'est à dire que plus l'angle  $\alpha$  sera proche de  $90^\circ$ , plus la vitesse augmentera et plus l'angle  $\alpha$  sera proche de  $0^\circ$  plus la vitesse sera proche de celle calculée en amont. En effet si l'angle est de  $0^\circ$ , alors la trajectoire est une ligne droite et donc la distance parcourue calculée correspond exactement à celle effectuée par la voiture. La voiture doit cependant accélérer si la trajectoire forme un arc de cercle car la distance parcourue calculée sera un peu sous estimée vis à vis de l'angle  $\alpha$ . La constante 2.5 présente dans la formule se trouve expérimentalement de manière à obtenir un tracé convenable.



#### II.3.2.4 - Le modèle 4 (modèle bicyclette dit « Pure pursuit »)

Précédemment, nous avons mis en oeuvre différents modèles reposant tous sur des calculs expérimentaux. Cependant, il est essentiel de trouver un moyen de suivre la trajectoire de façon mathématiquement exacte afin que ce projet puisse être utilisé pour d'autres circuits, modèles de voitures avec des virages et des paramètres différents. En effet avec des modèles expérimentaux, la moindre fluctuation des paramètres peut remettre tout en cause ce qui n'est pas le cas autrement.

À l'aide d'une compréhension du projet plus étoffée qu'au début et après de nouvelles recherches, une nouvelle méthode peut être envisagée. Cette méthode repose sur le modèle bicyclette et plus précisément l'algorithme « Pure pursuit ».

Dans ce modèle, la voiture est représentée par une bicyclette. Effectivement le fichier de commande requiert l'angle des roues de la voiture. Or dans la réalité, lorsqu'une voiture est en mouvement et tourne, l'angle des roues n'est pas le même pour les roues gauches et droites. Cette modélisation permet donc de se ramener à un modèle à deux roues au lieu de quatre.

Voici une représentation schématique du modèle ci-dessous :

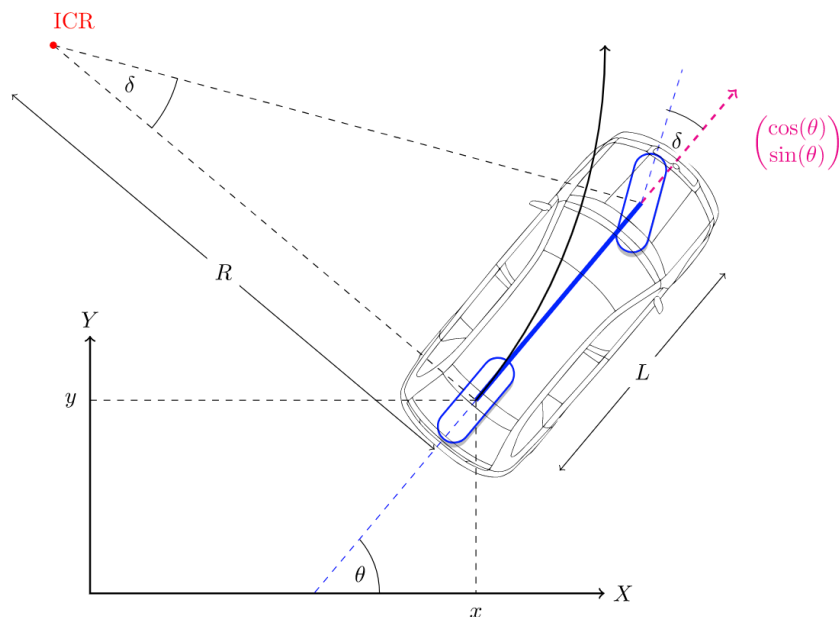


Figure II.3.6

Ici  $L$  représente l'entraxe ou empattement entre les roues et  $R$  correspond au rayon de braquage de la voiture par rapport au centre instantané de rotation.

Jusqu'à présent, le problème majeur rencontré était l'utilisation de l'angle  $\alpha$  de la trajectoire à la place de l'angle de braquage  $\delta$  qu'on ne savait pas obtenir. Avec ce nouveau modèle, il est maintenant possible de calculer l'angle  $\delta$ . De part la géométrie du modèle et de la situation, on remarque facilement que :

$$\tan \delta = \text{oppose/adjacent} \quad \text{soit} \quad \tan \delta = \frac{L}{R}$$

$L$  est un paramètre constant facile à obtenir et dépend uniquement du modèle de la voiture. Néanmoins  $R$  semble de premier abord difficile à calculer. On va donc essayer de contourner ce problème de manière géométrique. On peut facilement compléter les schémas précédents avec la figure ci-dessous :

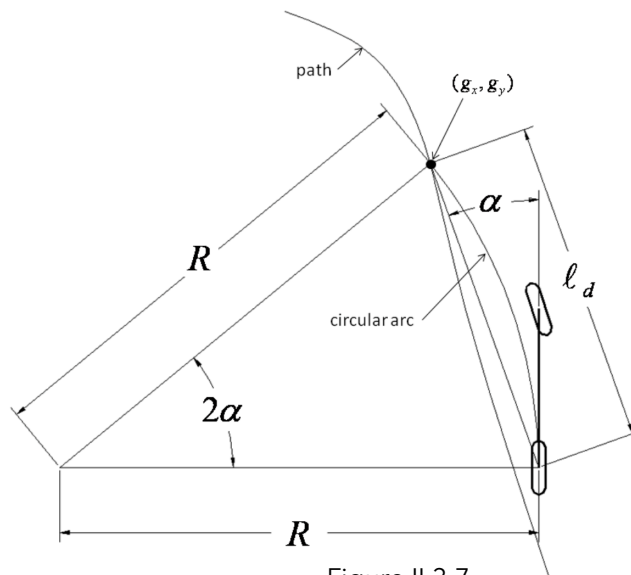


Figure II.3.7

Sur cette figure, on note  $(gx, gy)$ , le prochain point à atteindre. La distance  $ld$  représente la distance entre la roue arrière et le point de destination. Quant à  $\alpha$ , il représente dans cette figure l'angle entre l'axe du véhicule et le vecteur suivant  $ld$ . Attention, ici  $\alpha$  ne correspond pas exactement au même angle que dans les autres modèles car ici notre point de repère de la voiture se situe sur la roue arrière qui est différent du point de repère examiné en amont.

En appliquant la loi des sinus, on trouve alors :

$$\frac{\ell_d}{\sin(2\alpha)} = \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)}$$

$$\frac{\ell_d}{2\sin(\alpha)\cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

$$\frac{\ell_d}{\sin(\alpha)} = 2R$$

Une fois que l'on est maintenant capable de trouver la valeur du rayon  $R$ , on combine les deux équations, ce qui nous donne :

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

Nous avons mathématiquement trouvé la valeur de  $\delta$ , il reste maintenant à obtenir ces différents paramètres en analysant notre tracé.  $L$  étant déjà connu, il faut calculer  $ld$  et  $\alpha$ . Pour  $ld$ , il suffit de reprendre la méthode des anciens modèles mais il faut à chaque fois calculer la distance en ajoutant ce décalage qu'on appellera  $h$  entre notre point de repère de la voiture équivalent à la position du capteur/barycentre et celui de la roue arrière supposée dans le modèle bicyclette. De la même façon on obtient  $\alpha$  en calculant l'angle entre les deux droites. La différence avec les méthodes précédentes se situe dans le fait que la droite entre le barycentre et le point à atteindre et celle entre la roue arrière et le point à atteindre n'engendrent pas le même angle. L'angle étant maintenant obtenu, nous gardons le même modèle vitesse qu'auparavant.

Après cette phase de calcul, vient la phase d'analyse des résultats liés à la simulation. Malheureusement, ils ne sont pas ceux espérés. En effet la voiture ne se comporte pas du tout comme prévu. Elle suit bien l'allure de la courbe mais ne tourne pas aux bons instants avec une vitesse adéquate et se retrouve systématiquement confrontée aux bordures du circuit. Cela s'explique par deux problèmes majeurs. Le modèle vitesse comporte encore quelques

incertitudes et n'est plus en adéquation avec la mesure de l'angle associé. Deuxièmement, le modèle « pure pursuit » est dans la réalité utilisé quelque soit la vitesse puisqu'elle sera répercutée sur la distance **ld** et donc prise en compte par défaut mais dans notre système la vitesse semble avoir plus de répercussions. Ainsi, de part les incertitudes de mesure des angles et des distances, le modèle ne fonctionne pas de manière attendue.

### II.3.2.5 - Le modèle final (modèle bicyclette simple)

Le dernier modèle retenu découle en fait du modèle précédent. En effet nous avons vu que  $\tan \delta = \frac{L}{R}$ . La difficulté principale reposait sur le fait de calculer **R** sans trop d'incertitudes.

Mais après mûre réflexion, je me suis rendu compte qu'il existait un second moyen de trouver la valeur du rayon de braquage **R**. Nous savons que le rayon **R** constitue le rayon entre le centre instantané de rotation et la position de la voiture. Ce rayon est en fait le rayon du cercle (cercle osculateur noté **Co**) épousant au mieux la courbe reflétant la trajectoire tracée. C'est à dire que dans des intervalles de points réduit, la trajectoire tracée peut s'assimiler à une courbe, plus souvent appelée courbure. La particularité de cette courbure est qu'elle est tangente en tout point aux différents cercles osculateurs décrivant la trajectoire.

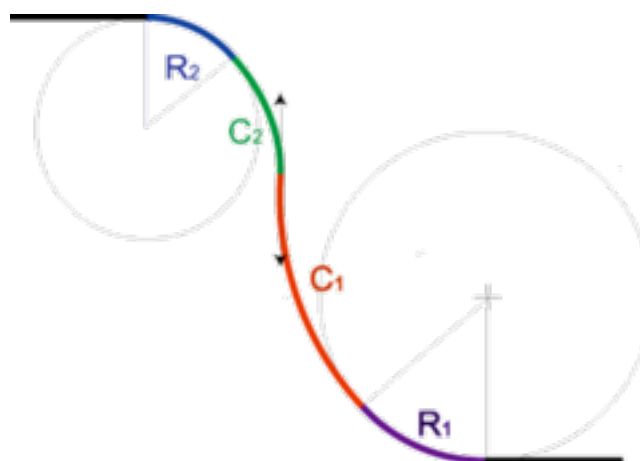


Figure II.3.8

Ainsi, il faudrait réussir à trouver ces cercles de façon à obtenir **R** et ainsi obtenir un angle de braquage **δ** précis. Aucune fonction connue ne permet facilement de tracer les cercles osculateurs d'une courbe. De plus trouver et tracer les cercles à la main serait fastidieux, peu précis et incorrect. Néanmoins ce problème m'a rappelé une notion étudiée en classe préparatoire. En effet, ayant étudié les courbes paramétrées, j'ai pu retrouver certaines formules très utiles :

Le rayon de courbure **R** est égal à l'inverse de la courbure **c** en un point : **R=1/c**

On sait aussi que :

$$\frac{d\vec{T}}{ds} = c\vec{N}.$$

De plus, avec d'autres recherches, on a :

$$1. \text{ Speed} = \frac{ds}{dt} = |\mathbf{v}(t)| = \sqrt{(x')^2 + (y')^2}.$$

$$2. \mathbf{v} = \frac{ds}{dt}\mathbf{T}, \quad \mathbf{T} = \frac{\mathbf{v}}{ds/dt}$$

$$3. \mathbf{a}(t) = \frac{d^2s}{dt^2}\mathbf{T} + \kappa \left(\frac{ds}{dt}\right)^2 \mathbf{N} = \frac{d^2s}{dt^2}\mathbf{T} + \frac{v^2}{R}\mathbf{N}$$

$$4. \kappa = \frac{d\mathbf{T}}{ds} = \frac{|\mathbf{a} \times \mathbf{v}|}{|\mathbf{v}|^3}.$$

$$4a. \text{ For plane curves } \mathbf{r}(t) = x(t)\hat{\mathbf{i}} + y(t)\hat{\mathbf{j}}: \quad \kappa = \frac{|x''y' - x'y''|}{((x')^2 + (y')^2)^{3/2}}.$$

En résumé, nous savons que **R=1/c** et que  $c = \frac{|x''y' - x'y''|}{((x')^2 + (y')^2)^{3/2}}$

Sachant que le couple **(x,y)** représente la position supposée de la voiture en chaque point et donc la position d'un point sur la courbe, on peut trouver facilement **c** et en déduire **R**.

$x'$  et  $x''$  respectivement  $y'$  et  $y''$  peuvent se calculer à l'aide de gradients. En python, on peut utiliser la méthode **gradient** de la bibliothèque numpy. Cette méthode permet de retourner un tableau de gradients de la même taille que le tableau passé en paramètres. Ainsi

en faisant  **$x\_array = np.gradient(x\_array)$**  , on obtient le tableau des dérivées premières de tous les x présents dans le tableau donné. On peut alors réaliser la même méthode sur le tableau des dérivées premières afin d'obtenir celui des dérivées secondes.

Ainsi après avoir mémorisé tous les couples  **$(x,y)$**  des points de la trajectoire tracée et calculé leurs dérivées premières et secondes, on peut appliquer les formules suivantes :

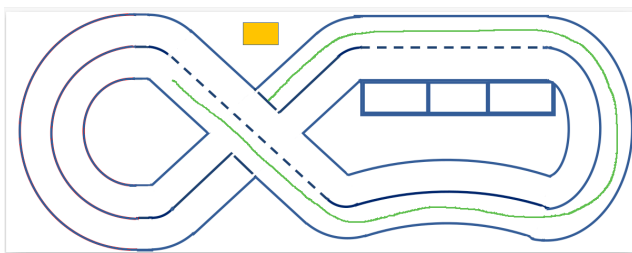
$$\textcircled{1} \quad v = \sqrt{(x')^2 + (y')^2}$$

$$\textcircled{2} \quad c = \frac{|x''y' - x'y''|}{((x')^2 + (y')^2)^{\frac{3}{2}}} = c = \frac{|x''y' - x'y''|}{v^3}$$

$$\textcircled{3} \quad \delta = \tan^{-1}(L/R) = \tan^{-1}(L * c)$$

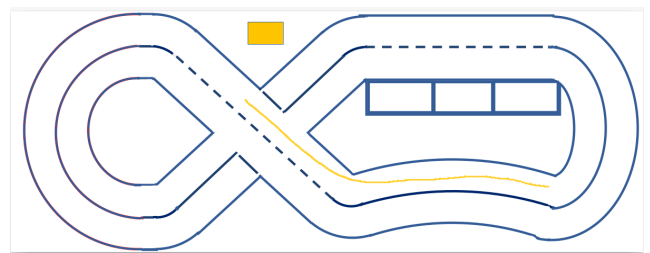
Nous avons donc bien calculé  **$\delta$**  et  **$v$**  de façon exacte. De plus  **$\delta$**  dépend bien de la vitesse  **$v$**  comme on le souhaite car la courbure  **$c$**  est elle aussi proportionnelle à la vitesse.

Les résultats et observations sont donc plutôt concluants puisque la voiture suit assez bien la trajectoire tracée même si ce n'est pas parfait. Cela s'explique notamment avec la mesure de l'entraxe qui est incertaine puis à cause de l'intervalle de points à laquelle on effectue les mesures. Il faut donc jouer entre ces deux paramètres afin d'avoir la meilleure précision possible. Ci-dessous, sont représentées plusieurs trajectoires tracées via l'interface ainsi que des liens vidéos permettant de voir les résultats simulés associés.



[video-trajectoire-1.mov](#)

Figure II.3.9



[video-trajectoire-2.mov](#)

Figure II.3.10

## III - Difficultés et problèmes rencontrés

### III.1 - La différence du système de coordonnées

Même si certains problèmes ont été résolus, ils ont été source de réflexion comme c'est le cas pour les systèmes de coordonnées. En effet, l'interface graphique est réalisée avec PyQt et possède son propre système de coordonnées (déjà expliqué dans la section II.1). Cependant, le logiciel RVIZ possède lui un autre système de coordonnées qui lui est propre et qui diffère du précédent. Il est donc impératif de réaliser les bons ajustements et conversions afin que les commandes données à la voiture soient efficaces.

Le système de coordonnées de RVIZ est un repère orthogonal basique (voir graphique ci-dessous). Ainsi pour que les positions ( $X_{in}, Y_{in}$ ) de l'interface correspondent à ( $X_{rviz}, Y_{rviz}$ ), il faut spécifié que pour un même point dans les deux systèmes de coordonnées on a :

$$X_{in} = X_{rviz}$$

$$Y_{rviz} = Y_{max} - Y_{in} \text{ avec } Y_{max} \text{ la largeur du circuit.}$$

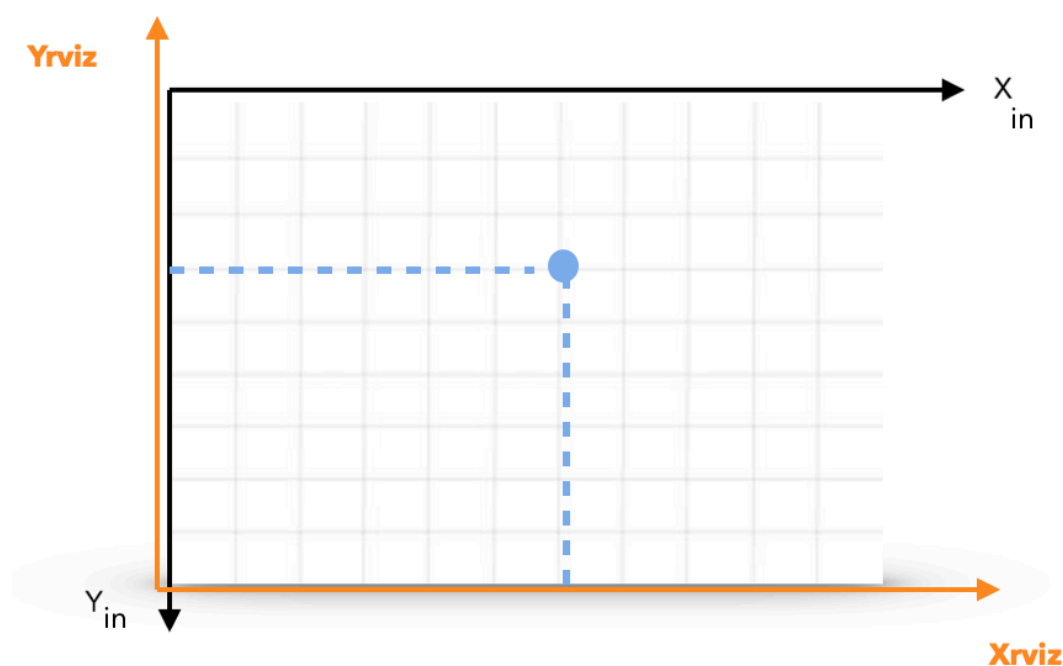


Figure III.1

Malgré ce problème résolu, il reste à appliquer la bonne résolution au circuit dans RVIZ pour que celui-ci corresponde à la taille réelle du circuit. On estime que 100 pixels équivaut à 1 mètre réel mais il y a de l'incertitude.

### **III.2 - Le calcul de la position initiale**

Pour rappel, les modèles expliqués en section II, ont tous été réalisés avec la même position et angle de départ. Après avoir mis en place le dernier modèle avec succès, je me suis donc demandé comment rendre ses paramètres interchangeables. Avec la résolution du problème précédemment évoqué (section III.1), on peut maintenant estimer la position de départ à l'aide de la position du début du dessin dans l'interface graphique. Ainsi peu importe où le dessin commence on recalcule  $X_{initial}$  et  $Y_{initial}$ .

Mais l'angle initial pose une difficulté supplémentaire. Effectivement, l'angle  $\theta$  correspond à l'orientation de la voiture selon l'axe X. Pour connaître la valeur de cet angle, on procède de la même façon que pour calculer  $\alpha$  (section II.3.2). Cette fois-ci la première droite sera l'axe des abscisses et la seconde l'axe de la voiture. Il faut ensuite calculer le coefficient directeur de cette droite qui normalement correspond à l'équation ci-contre sur un repère orthonormé basique :

$$coeff = \frac{y_B - y_A}{x_B - x_A}$$

En sachant que l'axe des ordonnées est dans l'interface inversé, il convient alors de dire que :

$$coeff = \frac{y_A - y_B}{x_B - x_A}$$

Pour avoir un coefficient plus précis, on calcule la moyenne des coefficients concernant les 3 premiers points. Ensuite pour trouver l'angle  $\theta$ , il faut réaliser l'opération suivante : Si le décalage  $dx$  correspondant à  $(x_B - x_A)$  est positif, alors  $\theta = \tan^{-1}(coeff)$  sinon  $\theta = \tan^{-1}(coeff) + \pi$

Cela se comprend à l'aide de la figure ci-dessous :



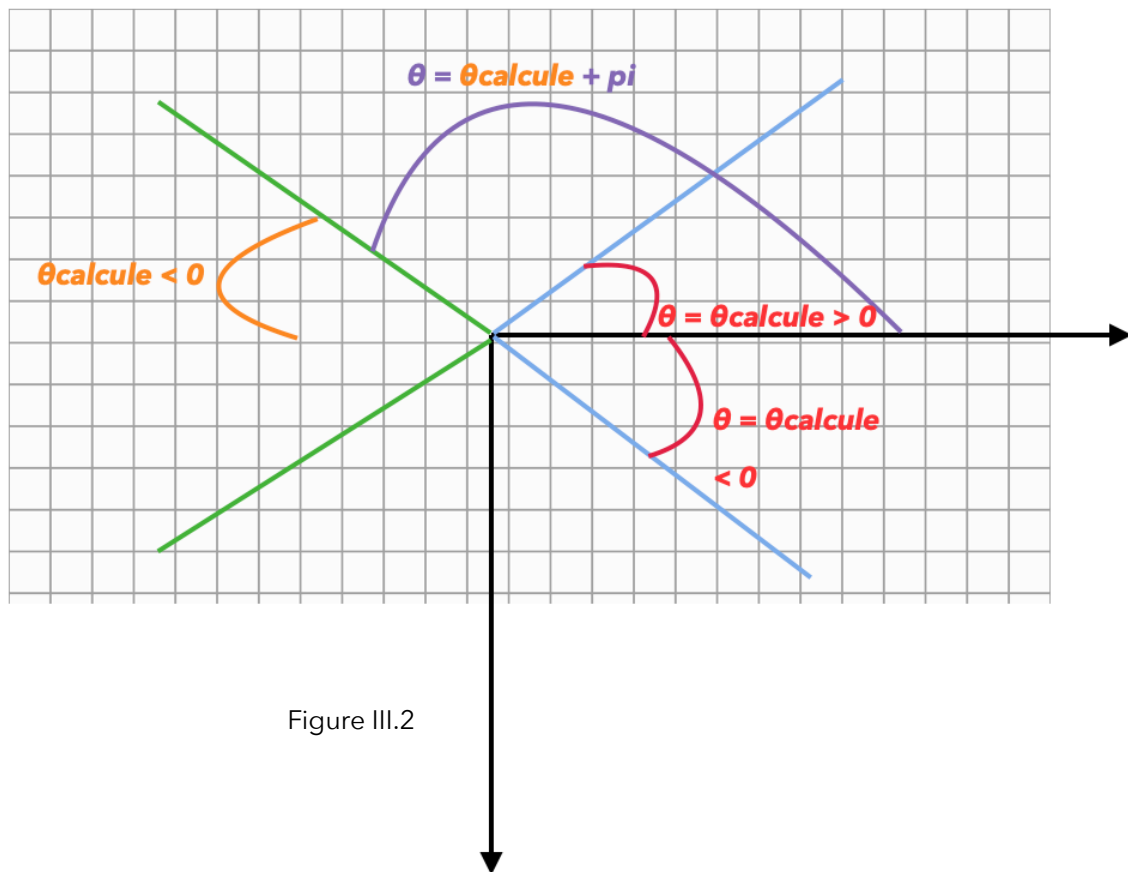


Figure III.2

### III.3 - Intervalles et limites

Finalement, la dernière difficulté rencontrée repose sur les intervalles et les limitations de la voiture. En effet, la voiture possède des caractéristiques qui lui sont propres comme l'intervalle de vitesse pouvant être pris [**Vmin**, **Vmax**] puis l'intervalle correspondant à l'envergure angulaire des roues [**δmin**, **δmax**].

Lors des premiers modèles n'ayant pas fonctionné, ces limitations étaient prises en compte directement mais dans notre modèle final, les angles et vitesses calculés repose sur la courbure (c'est à dire la trajectoire tracée). Ainsi si la courbure nécessite un angle ou une vitesse en dehors de l'intervalle de limitation, il est difficile de modifier ce calcul. Parce que si la courbure engendre un **δ > δmax**, faire en sorte que l'angle **δ = δmax** reviendrait à modifier l'allure de la courbe et donc d'autres paramètres tels que **R**. Ce serait donc toute la trajectoire qui serait modifiée ainsi que la position des points suivants sans qu'on ne puisse avoir la certitude que la voiture va rester sur la route. Ce problème est aussi vrai pour les vitesses **v** calculées.

La solution que j'ai envisagée consiste donc à choisir un intervalle de points de mesure tel que tous les angles et vitesses calculés à l'aide de la courbure se trouvent dans l'intervalle de limitation. En effet, si l'intervalle de points est trop petit (disons 5 points), alors le calcul de la distance et de l'angle tous les 5 points devra être effectuée en 1 seule seconde (temps d'exécution de la commande). La voiture risque alors d'être trop lente. Si l'intervalle est trop grand (disons 50 points), la voiture sera trop rapide à certains endroits et les angles pourraient être trop forts si le circuit comprend beaucoup de virages serrés. C'est pourquoi, il est recommandé d'utiliser un intervalle moyen de 25 ou 30 points. Bien entendu, les points ne sont pas équidistants (cela n'aurait pas beaucoup d'intérêt). Leur position dépend donc de la vitesse de déplacement de la souris. Plus le déplacement est rapide, plus les points seront éloignés car la souris trace des points à intervalle de temps régulier.

Néanmoins, malgré le succès de la simulation sur RVIZ, le suivi de la trajectoire ne fonctionne pas de manière satisfaisante sur le circuit réel. Cela est dû à la différence qu'il y a entre le modèle de voiture simulé sur RVIZ et le modèle réel. Les dimensions ne sont pas exactement similaires et les intervalles de limitation ne sont pas non plus les mêmes. L'angle des roues sur la voiture simulée n'implique pas un angle similaire sur le modèle réel. Un mauvais vissage et serrage des roues peut par exemple être à l'origine de ce décalage.

### **III.4 - Le temps de commande**

On l'a vu précédemment, le temps d'exécution des commandes est fixé à 1 seconde par défaut. Néanmoins, il est possible de le modifier à partir du fichier **path\_publisher.py** de RVIZ. Après différents essais avec un temps d'exécution plus ou moins élevé, j'en ai conclu que cela n'avait que très peu d'impact positif. Le temps fixé à 1 seconde me paraît donc être le meilleur compromis. De plus cela facilite les calculs.

## IV - Planning

	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6
Réalisation de l'interface graphique avec pyQT						
Prise en main du logiciel RVIZ et de ROS						
Essais de plusieurs modèles pour le suivi de trajectoire						
Amélioration du modèle retenu						
Documentation et publication du code						
Amélioration des fonctionnalités de l'interface graphique et de l'IHM						
Tests sur le circuit en condition réelle						

## **V - Conclusion, apports de l'expérience et perspectives**

Malgré sa courte durée, ce stage m'a appris beaucoup de choses. Le fait de travailler sur un réel projet comportant plusieurs acteurs a été très enrichissant. De plus, la tâche qui m'a été confiée fût intéressante puisqu'elle est assez inédite. En effet, aucun ou très peu d'autres projets de la sorte ont été publiés ces dernières années. Le fait que ce projet pourrait dans l'avenir servir à beaucoup de monde m'a donc rendu très motivé et impliqué.

Par ailleurs, ce projet m'a permis d'apprendre à utiliser le module PyQt et découvrir de nouveaux aspects de Python. Tout comme, il m'a initié à ROS et à la robotique. Souhaitant poursuivre en spécialité IHM, le fait de travailler sur une interface graphique a aussi été bénéfique pour ma part. Ainsi en explorant de multiples domaines et en étant confronté à de nombreuses difficultés pendant ce stage, j'ai pu enrichir mes connaissances informatiques.

Finalement, le stage s'est plutôt bien déroulé et l'objectif du projet a été partiellement atteint avec le succès du suivi de la trajectoire sur la simulation dans RVIZ. Des améliorations sont encore possibles vis à vis du projet, ce qui en fait la richesse. À ce jour, il me reste encore une semaine de stage pendant laquelle les missions sont de documenter le travail dont le code écrit jusqu'à présent, d'améliorer les fonctionnalités de l'interface avec l'ajout d'un bouton « éditer » qui permettrait de modifier une trajectoire déjà tracée. Cela permettrait encore une fois de réduire les imprécisions.

Les perspectives de ce projet sont assez riches puisqu'on pourrait imaginer un parcours du circuit avec plusieurs véhicules à la fois. Ce serait un nouveau défi intéressant à relever. Ce projet est disponible sur mon GitHub via : <https://github.com/ANTOINE-HM/GUI-autonomous-car/>

## VI - Bibliographie

*Python GUIs - Qpainter & Bitmap graphics par Martin Fitzpatrick (mis à jour le 11/08/2022)*

<https://www.pythonguis.com/tutorials/bitmap-graphics/>

*Tutoriel - intro to ROS and MuSHR « Fundamental Robot Operating System (ROS) concepts using MuSHR » par Matthew Rockett*

<https://mushr.io/tutorials/intro-to-ros/>

*Tutoriel - Quickstart « Run the MuSHR platform on your machine! » par Matt Schmittle*

<https://mushr.io/tutorials/quickstart/>

*La courbe de courbure dans numpy (mis à jour le 9/04/2018)*

<https://stackoverflow.com/questions/28269379/curve-curvature-in-numpy#28270382> &

<https://askcodez.com/la-courbe-de-courbure-dans-numpy.html>

*« Automatic Steering Methods for Autonomous Automobile Path Tracking »*

*Par Jarrod M. Snider (Février 2009)*

[https://www.ri.cmu.edu/pub\\_files/2009/2/](https://www.ri.cmu.edu/pub_files/2009/2/)

[Automatic\\_Steering\\_Methods\\_for\\_Autonomous\\_Automobile\\_Path\\_Tracking.pdf](#)

*« Understanding Geometric Path Tracking Algorithms – Stanley Controller »*

*Par Sachin Kundu (19/072020)*

[https://medium.com/roboquest/understanding-geometric-path-tracking-algorithms-](https://medium.com/roboquest/understanding-geometric-path-tracking-algorithms-stanley-controller-25da17bcc219)

[stanley-controller-25da17bcc219](#)

*« Pure Pursuit - Algorithms for automated driving » par Mario Theers and Mankaran Singh*

<https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/PurePursuit.html>

*« Erasing a pen on canvas » par eyllanesc (28/11/2018)*

<https://stackoverflow.com/questions/53515295/erasing-pen-on-a-canvas>

« Introduction aux interfaces graphiques en Python avec Qt 5 et PyQt5 » par David

Cassagne (mis à jour le 5/08/2022)

<https://courspython.com/interfaces.html>

« Fiche explicative de la leçon : Angle entre deux droites dans le plan cartésien »

<https://www.nagwa.com/fr/explainers/407162748438/>

[#:~:text=Pour%C3%A9terminer%20l'angle%20aigu%2C%20F0%9D%9B%BC%20%2C%20entre%20deux%20droites,coefficients%20directeurs%20des%20deux%20droites](#)