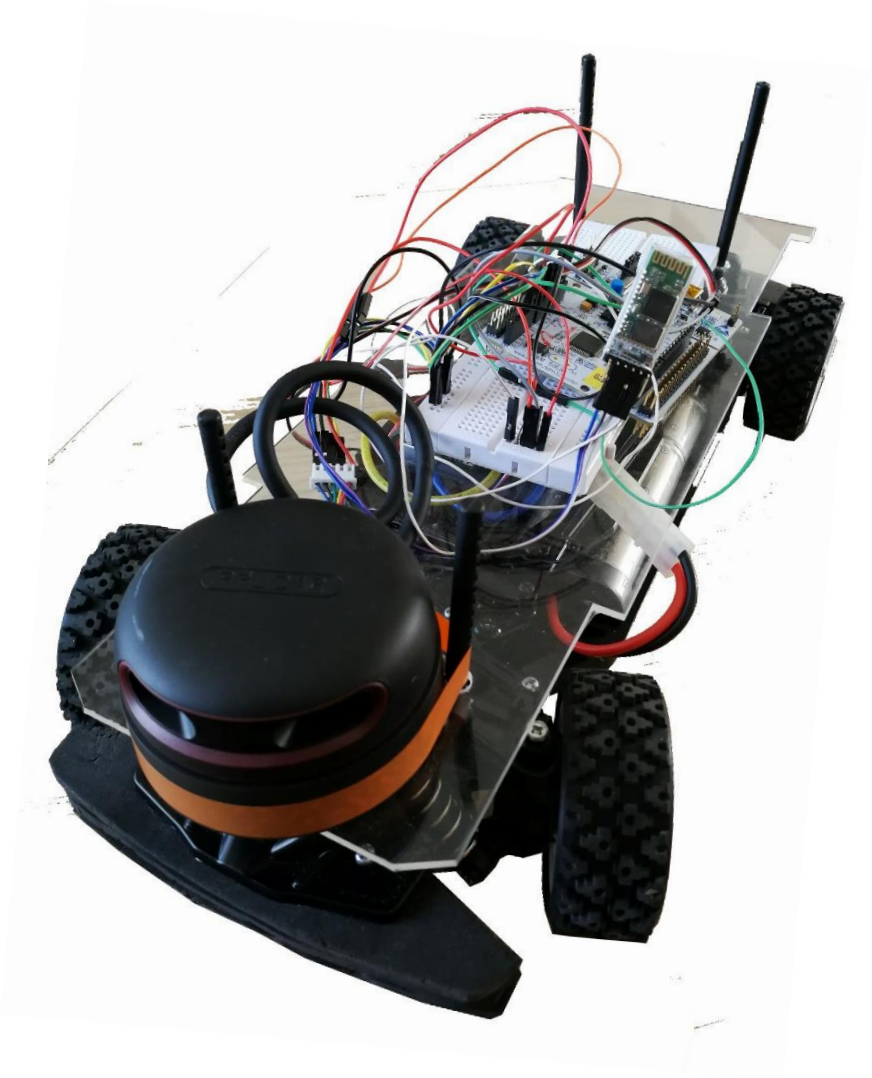
Autono'Car

PIMS 2019 - 2020

Rapport technique



Professeur encadrant : Julien VILLEMEJANE

Kévin FALQUE – Kassim HANANE – Zhe WANG – Mahawa Cissé – Alexis KEDZIA – Benoît PARIS – Hugo AFFATICATI – Thomas SERRE

I. Présentation

1. Objectifs
2. Matériels à disposition
3. Schéma bloc

II. Détection et communication

1. Bluetooth
2. Capteurs IR
3. Traitement d'images
4. LiDAR

III. Traitements des données

1. Capteurs
2. Traitement d'images
3. LiDAR

IV. Adaptation la trajectoire et la vitesse

1. Avec les capteurs IR
2. Approche proportionnelle avec le LiDAR
3. Champ de potentiels
4. Adaptation de la vitesse

V. Réalisation pratique et résultats

1. Circuits imprimés
2. Utilisation de plusieurs contrôleurs
3. Résultats avec les capteurs
4. Résultats avec la caméra
5. Résultats avec le LiDAR

VI. Simulations

1. Définition d'une zone de sécurité
2. Simulation d'un code qui combine l'adaptation de la trajectoire et de la zone de sécurité

I. Présentation du projet

Notre projet consiste à réaliser une voiture autonome en modèle réduit. Ce projet s'inscrit dans le cadre d'une compétition organisée par l'ENS Cachan. Chaque école doit monter une voiture miniature respectant les normes imposées par les jurys. Ensuite les participants doivent l'équiper des capteurs de leur choix et programmer une carte Nucléo afin que la voiture soit capable de se piloter toute seule, c'est-à-dire analyser son environnement afin de suivre la route et éviter les obstacles comme les autres véhicules. La compétition se soldera par une course entre tous les participants avec plusieurs épreuves : une épreuve de vitesse de la voiture seule sur un circuit vide puis rempli d'obstacles et dans un troisième temps une course entre la voiture au cours de laquelle les voitures devront donc être capables d'éviter de suivre un circuit le plus rapidement tout en évitant des obstacles fixes et mobiles.

1. Objectifs et performances souhaitées

Les principaux objectifs sont alors les suivants :

- Démarrer et arrêter la voiture à distance
- Se diriger sur un circuit
- Détecter les obstacles fixes et mobiles et les éviter
- Asservir en vitesse le véhicule autonome pour être le plus rapide possible tout en évitant les obstacles

On définit aussi les performances que l'on souhaite atteindre :

- **Rapidité :** La voiture doit aller le plus vite possible pour remporter la course !
- **Fiabilité :** La voiture ne doit pas s'approcher à moins de 10cm des obstacles pour ne pas risquer rentrer dedans.
- **Précision :** La voiture doit garder une trajectoire la plus cohérente possible et ne pas dévier d'un autre trop important lors du dépassement d'un obstacle : par rapport à l'angle d'entrée θ, angle entre le LiDAR et un obstacle, et une distance d (en cm) entre la voiture et l'obstacle la voiture ne doit pas tourner de plus de θ + (10/d) (où 10 correspond à la demi-largueur de la voiture).
- **Ergonomie :** L'interface Humain-Machine, permettant de transmettre les ordres pour démarrer et s'arrêter doit se faire à travers une application sur Smartphone et par transmission Bluetooth.

2. Matériel utilisé

Le challenge a imposé une partie du matériel pour que la compétition ne soit pas sur l'aspect mécanique mais bien sur l'aspect robotique et programmation. Ainsi, le châssis de la voiture ainsi que la batterie, le moteur à courant continu et le servomoteur sont imposés par le jury. On indique ces éléments sur la figure suivante.

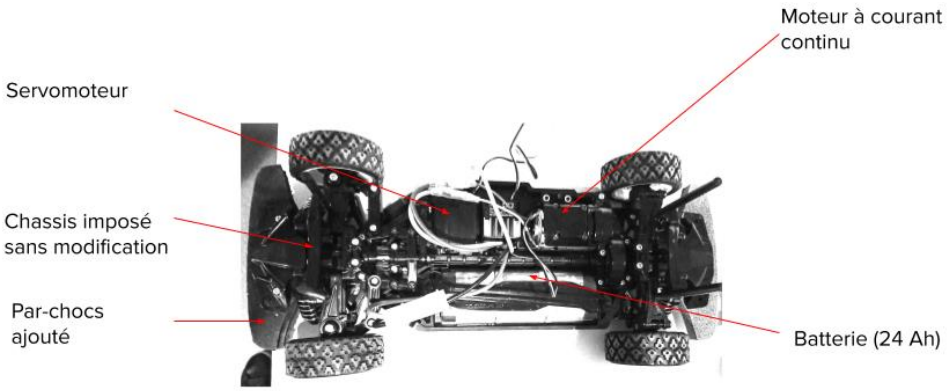


Figure 1 - Photo du chassis imposé

Le système de détection est en revanche entièrement à choisir et ainsi un enjeu majeur de ce challenge. Nous nous sommes alors attachés à étudier et caractériser du mieux possible plusieurs solutions pour essayer d'obtenir une proposition la plus optimale possible, éventuellement en combinant plusieurs dispositifs. Les trois systèmes de détection que nous avons testés sont les suivants :

- Capteurs à infrarouge Sharp GP2Y pour laquelle on a testé différentes plages de mesure
- LiDAR
- Détection des objets par traitement d'image à l'aide d'une caméra embarquée

3. Schéma bloc

On donne les schéma bloc suivant pour décrire le fonctionnement. La voiture fonctionne grâce à 4 blocs principaux :

- Un *moteur à courant continu* qui lui permet d'avancer et donc de faire tourner ses roues. Il prend en entrée une tension consigne (codée entre 1500 à l'arrêt, et 1700 pour rester dans les bonnes conditions d'utilisation pour la voiture) et en sortie sa vitesse de rotation qui est transmise aux roues.
- Un *servomoteur* pour se diriger et donc pour faire pivoter les roues de la voiture vers la direction souhaitée pour éviter les obstacles. Il prend en entrée une tension consigne (codée entre 1100 tout à gauche et 1500 tout à droite) et en sortie l'angle qui est appliqué aux roues
- Un *LiDAR* pour détecter les obstacles et déterminer les distances à ceux-ci avec les angles correspondants
- Une *carte Nucléo* pour calculer, à partir des informations données par le LiDAR, la direction que la voiture doit prendre pour éviter les obstacles et la vitesse à laquelle elle doit aller pour être la plus rapide possible dans la course tout en étant contrôlée pour éviter les obstacles correctement et en sécurité. Autrement dit, la carte Nucléo permet de calculer la vitesse à laquelle doit aller la voiture en fonction de la distance de celle-ci aux obstacles ainsi que l'angle que doit prendre la voiture par rapport à sa trajectoire actuelle pour éviter les obstacles. Elle prend donc en entrée les distances

aux obstacles données par le LiDAR avec leurs angles associés et donne en sortie les tensions consigne adaptées pour le servomoteur et le moteur à courant continu.

Avant de calculer ces tensions, un système de test booléen est ajouté pour assurer la sécurité de la voiture. En effet, si un obstacle est détecté dans une certaine zone qu'on définira plus tard dans ce rapport, la carte Nucléo impose l'arrêt à la voiture (pour l'instant) puis une marche arrière.

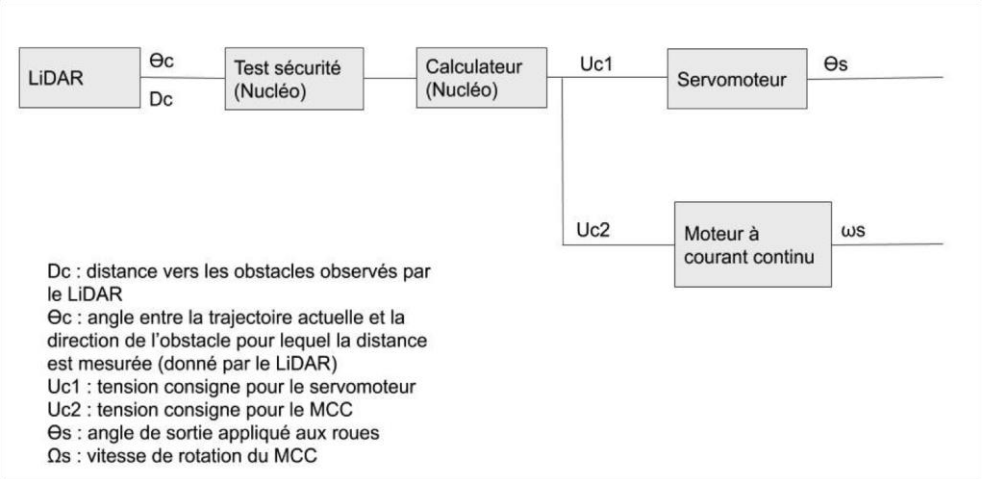


Figure 2 - Schéma bloc de la voiture

II. Détection et communication

1. Bluetooth

L'émetteur-récepteur Bluetooth utilisé est un HC-05. Il autorise une communication entre la voiture et l'utilisateur. Ainsi l'utilisateur envoie un "go" pour faire démarrer la voiture et un "stop" lorsqu'elle a terminé son parcours. Il émet et transmet des fréquences comprises entre 2402GHz et 2480GHz. Ce module Bluetooth est alimenté par une tension de 5V.
l'émetteur-récepteur Bluetooth est connecté aux entrées PC10 et PC11 de la carte nucléo qui gèrent l'émission et la réception de données par Bluetooth.

2. Capteurs IR

Les capteurs utilisés sont des capteurs longues distances SHARP GP2Y de portée maximale 150cm. Ils sont capables de mesurer des distances comprises entre 20cm et 150cm. Ces capteurs sont constitués d'un émetteur infrarouge (LED), d'un récepteur et d'un circuit de traitement de signal. Un rayonnement infrarouge collimaté est émis et va ensuite se réfléchir sur un objet. Le faisceau réfléchi va être détecté par le récepteur et la tension pourra être évalué grâce au circuit de traitement du signal. Le capteur nous renvoie une valeur de tension qui correspond à une valeur de distance. La tension est une fonction décroissante de la distance.

Les capteurs sont alimentés par une tension de 5V et sont reliés aux entrées analogiques de la carte nucléo (entrée A0 à A4).

3. Traitement d'images

Afin d'éviter les obstacles près de la voiture, notre projet a également essayé d'utiliser le Raspberry Pi pour détecter. Le principe de cette méthode est de prendre une photo à travers la caméra Raspberry Pi puis de traiter les informations contenues dans la photo.

Figure 3 - Raspberry Pi et appareil photo (http://liyao.me/raspberry-pi-wifi-camera/)

Dans la partie détection, nous écrivons du code en python dans le Raspberry Pi pour réaliser l'appel à la caméra. Nous pouvons non seulement l'utiliser pour prendre des photos, mais aussi pour filmer des vidéos. Dans notre projet, nous utilisons principalement la fonction de prendre des photos. Nous pouvons modifier le nombre de pixels de l'image, le contraste, la luminosité, etc.

Le plus grand avantage de cette méthode de détection est qu'elle permet à la voiture intelligente d'obtenir des images de l'environnement environnant, c'est-à-dire de la vision industrielle. Contrairement aux méthodes de télémétrie laser et d'induction radar, qui obtiennent la position et la distance des obstacles, Raspberry Pi nous permet d'obtenir toutes les informations d'apparence telles que la forme et la couleur de l'obstacle, bien que les informations de distance entre la voiture et les obstacles ne puissent pas être obtenues directement.

Avec le Raspberry Pi, les gens peuvent contrôler la voiture en fonction de la forme et de la couleur de l'objet détecté. Cette technologie a été mise en œuvre avec succès dans le tri des bandes transporteuses, qui peut également être appliqué dans les voitures intelligentes.

4. Le LiDAR

Le LIDAR est un appareil qui permet de "cartographier" l'environnement via des mesures de distance. Le principe est le même que sur la plupart des capteurs de distances. C'est à dire que le LIDAR comporte un émetteur qui envoie un rayonnement électromagnétique très directionnel et d'un capteur qui va détecter le "retour" du rayon qui aura été diffusé par l'objet

pointé. L'intérêt d'un LIDAR provient du fait qu'il combine l'action décrite plus haut de manière répétée avec une rotation de l'appareil lui permettant ainsi de voir à 360°.

Le LIDAR qui a été utilisé est le RPLIDAR A3. Il a une vitesse de rotation réglable et réalise 16000 mesures de distance par seconde. Ainsi en adaptant la vitesse de rotation on peut échantillonner l'espace de manière plus ou moins précise et le tout jusqu'à une distance de 25m.

III. Traitement des données

1. Capteurs IR

Le traitement des données fournis par les capteurs IR est non seulement plus simple à réaliser mais aussi plus rapide qu'avec les autres capteurs. Les capteurs sont disposés en arc de cercle à l'avant de la voiture et espacé de façon régulière chacun regardant dans une direction différente. Un des capteurs regarde dans la même direction que la voiture, deux autres à 30° et -30° et deux autres encore à 60° et -60°. On peut voir une représentation schématique de ce que voit les capteurs sur la figure ci-dessous.

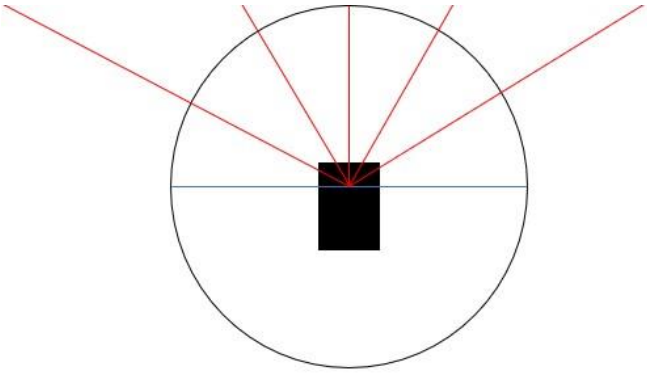


Figure 4 - Représentation schématique de la vision des capteurs

Les capteurs envoient une tension à la carte Nucléo via les entrées A0 à A4 qui fournit une information sur la distance séparant le capteur de l'obstacle le plus proche. La tension envoyée par le capteur est inversement proportionnelle à la distance qui la sépare de l'obstacle, autrement dit plus l'obstacle est proche plus la tension est élevée.

La carte Nucléo reçoit donc cinq valeurs de tension aux différentes entrées A0 à A4 correspondant chacune à un capteur de la voiture. Il faut alors réaliser un algorithme qui va imposer la direction, nous avons choisi de la diriger dans la direction du capteur qui indique la plus basse tension.

Malgré la simplicité du programme, les capteurs renvoient des valeurs erronées de façon régulière et cela se voit sur la trajectoire de la voiture. En effet, il arrive que celle-ci heurte des obstacles et pour y remédier, nous avons décidé d'utiliser la moyenne des 10 dernières

valeurs renvoyées par chaque capteur. Pour que la voiture réagisse à chaque tour du programme et pas seulement tous les 10 tours, nous avons réalisé une moyenne glissante.

2. Traitement d'images

Comme mentionné précédemment, les données que nous obtenons sont une image avec un nombre connu de pixels, et le traitement que nous voulons faire est d'en extraire des informations utiles. Nous pouvons utiliser la bibliothèque opencv en python pour le traitement.

OpenCV peut être utilisé pour développer des programmes de traitement d'image en temps réel, de vision par ordinateur et de reconnaissance de formes. Il est largement utilisé dans la reconnaissance faciale, le suivi de mouvement, la reconnaissance d'objets et la robotique. Nous avons essayé la procédure de détection des contours dans le fichier de bibliothèque et avons constaté que les seuils sont différents et que les résultats obtenus par la détection sont également très différents, ce qui signifie que lors de l'écriture de code, vous devez choisir soigneusement le seuil approprié

Dans le code que nous écrivons, la première étape consiste à choisir de créer une bibliothèque de couleurs, de sélectionner la couleur que nous voulons identifier et de déterminer sa plage de valeurs RVB, comme critère de reconnaissance des couleurs.

La deuxième étape consiste à choisir un seuil approprié pour retirer les objets colorés de l'arrière-plan, c'est-à-dire que nous séparons une image avec plusieurs couleurs en plusieurs images avec une seule couleur.

La troisième étape compare les données de l'image réelle obtenue avec la bibliothèque de couleurs pour déterminer la couleur de l'obstacle.

La dernière étape consiste à envoyer la commande correspondante à la voiture en fonction de la couleur de l'obstacle.

Par exemple, dans notre expérience, nous voulons contrôler le sens de rotation de la voiture en fonction des différentes couleurs des murs des deux côtés de la voiture dans le sens de la marche. Le mur de gauche est vert et le droit est rouge. Par conséquent, nous calculons le nombre de pixels verts et rouges dans l'image obtenue par la caméra. Si le nombre de pixels verts est supérieur au nombre de pixels rouges, cela signifie que la voiture est plus proche du mur de gauche, et la voiture reçoit un virage à droite Sinon, tournez à gauche.

3. LiDAR

Contrairement aux capteurs IR qui étaient dans notre cas au nombre de 5, le nombre de données à traiter est nettement plus important dans le cas du LiDAR. En effet, comme vu précédemment, la fréquence de ce dernier est d'environ 16000 données par seconde. De ce fait, il est important d'optimiser la récupération de ces données afin de diriger la voiture en conséquence.

Nous avons premièrement utilisé la bibliothèque ‘rplidar’ sur mbed dans le but de recueillir les informations. Celle-ci nous met à disposition une batterie de fonctions permettant d'interagir avec le LiDAR. Nous avons ainsi utilisé les fonctions de bases fournies dans le but de construire notre code.

A partir de celles-ci, nous avons ensuite réfléchie à la manière la plus efficace de les utiliser dans le but d’optimiser l’interaction LiDAR-direction. Pour ce faire, nous nous sommes d’abord penchés sur la sélection des données. Dans la mesure où le but de la voiture est d’avancer, celle -ci ne nécessite pas une vision supérieure à 180°. Nous avons alors réduit la quantité de données de moitié en conservant que les angles compris entre -90° et 90°. Bien entendu, le LiDAR nous envoie des valeurs d’angle entre 0° et 360° et c’est pourquoi nous avons dans un second temps ‘normalisés’ ces angles afin de travailler en -90° et 90°.
Une fois que nous avons sélectionnés et normalisé ces données nous avons mis en place le système de distance maximale. De la même manière que pour les capteurs IR, nous réalisons une recherche de maximum sur un tour puis nous indiquons aux roues de ce diriger dans la direction en question. Cependant, dans le cas du LiDAR, cela signifie que le servomoteur reçoit une nouvelle direction environ toutes les ms ce qui produit des trajectoires trop instables. De plus, nous avons également remarqué que pour certains tours, il arrive que l’angle de distance maximale ne soit pas le bon. Afin de pallier ce problème, nous avons choisi de moyenner la valeur de l’angle correspondant à la distance maximale sur plusieurs tours afin de stabiliser la trajectoire.

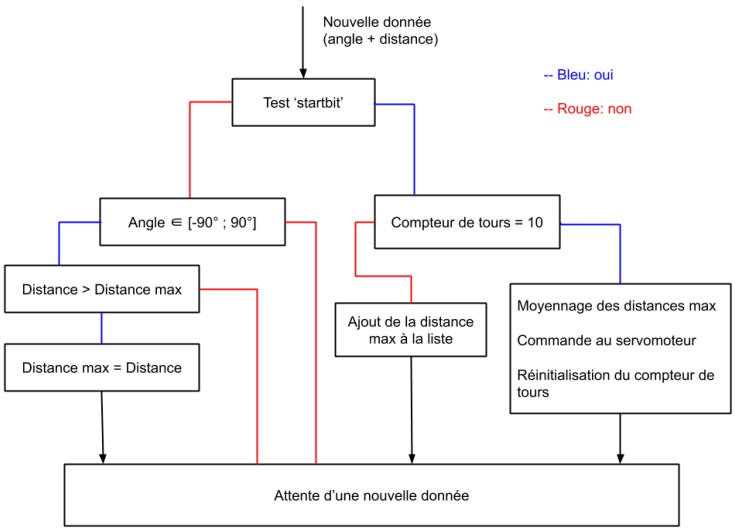


Figure 5 - Schéma de fonctionnement l’algorithme de traitement de données du lidar

Dans ce schéma, le test startbit correspond au test pour la réalisation d’un tour du LiDAR. En effet, si celui-ci repart à 0 alors un bit de départ en envoyé. On a également choisi de moyenner sur 10 tours dans cet exemple ce qui correspond à la valeur choisie en pratique.

IV. Adaptation de la trajectoire et de la vitesse

1. Capteur IR

Choisir la direction la plus dégagée pour diriger la voiture n'est pas suffisant pour éviter toutes les collisions. En effet, nous ne pouvons pas prendre en compte uniquement la direction où la distance entre le capteur et l'obstacle est la plus grande. Supposons qu'un obstacle apparaisse sur les côtés de la voiture mais que le capteur central ne le voit pas comme on peut le voir sur la figure ci-dessous.

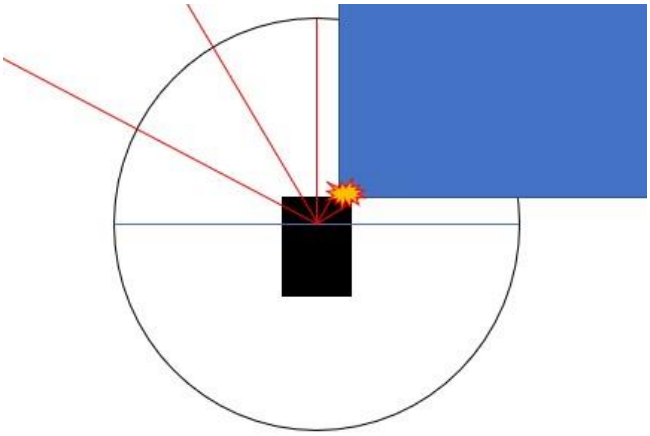


Figure 6 - Schéma de la vision des capteurs en présence d'un obstacle

Pour remédier à ce problème, nous avons modifié le programme afin de prendre en compte tous les capteurs. Pour choisir l'angle d'inclinaison des roues, nous réalisons une moyenne des angles de chaque capteur pondéré avec la distance les séparant de l'obstacle qu'ils détectent. On trouve ainsi l'angle de braquage des roues grâce à la formule suivante : $\theta_{final} = \frac{\Sigma(\theta_k d_k)}{\Sigma d_k}$ ou les d sont les distances indiquées par les capteurs et les Θ sont les angles correspondant aux capteurs en question. Cela permet d'éviter les collisions tout en gardant une trajectoire optimisée.

Malgré ce nouvel algorithme pour le choix de la direction, il faut aussi réguler la vitesse. En effet, lorsque la voiture roule à une vitesse trop élevée, il lui arrive de ne pas avoir le temps d'éviter certains obstacles. Nous avons donc rajouté des conditions sur la vitesse. Nous avons utilisé une condition sur la somme des distances indiquées par les capteurs de façons pondéré car les capteurs aux extrémités sont moins significatifs. Si les trois capteurs à l'avant de la voiture sont bien dégagés, celle-ci roulera à la vitesse maximale. Mais nous avons ajouté un terme dans la définition de la vitesse qui correspond aux tensions délivrées par les capteurs. Si celle-ci augmentent, alors la consigne de vitesse diminue. $Vitesse = V_{max}(1 - \frac{(\Sigma tension)}{3,3})$ Dans l'équation du programme nous avons rajouté des coefficients devant les valeurs de tension des capteurs aux extrémités. Une autre méthode a été envisagée pour gérer la vitesse avec les capteurs IR, elle consiste simplement à ralentir la voiture de façon proportionnelle à la distance la plus basse séparant un capteur de l'obstacle lorsque celle-ci passe sous un certain seuil. Cela permet à la voiture de manœuvrer avec plus de précision lorsqu'elle est à proximité d'un obstacle qu'elle n'aurait pas pu éviter à pleine vitesse.

2. Approche proportionnelle avec le LiDAR

Afin de diriger la voiture à l'aide des données du LiDAR, nous envoyons des commandes au servomoteur comme illustré précédemment. Pour ce faire, nous envoyons à travers une broche de la nucléo une valeur entre 1100 et 1500 correspondants respectivement à gauche et droite. Dans le cas du LiDAR, nous avons la possibilité d'observer entre -90° et 90° ce qui nous oblige ainsi à ajuster la commande à envoyer aux roues. Nous avons ainsi opté pour une approche proportionnelle : l'angle de braquage des roues est proportionnel à l'angle de la distance maximale. Dans un premier temps nous avons simplement résolu l'équation afin d'avoir un angle de braquage maximal pour $\pm 90°$. Cependant, en passant à la pratique, nous nous sommes aperçus que ce facteur n'était pas suffisant. En effet, celui-ci ne permettait pas de tourner assez vite compte tenu de la proximité des objets et du rapide débit de données du LiDAR. De ce fait, nous avons augmenté ce facteur jusqu'à obtenir des résultats satisfaisants. Nous sommes donc passé d'un facteur 2.2 à 5.

3. Champ de potentiel

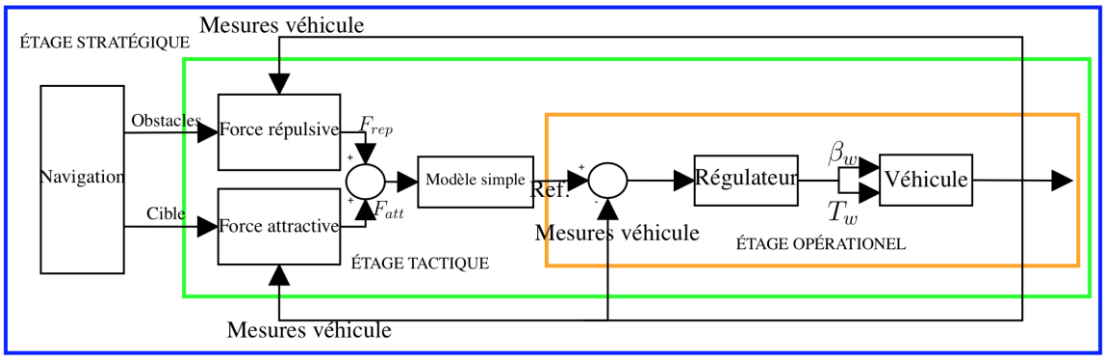


Figure 7 - Schéma de régulation à trois niveaux : stratégique (bleu), tactique (vert) et opérationnel (orange)

Une autre méthode pour diriger le véhicule autonome est d'utiliser le formalisme des champs de potentiels très largement employé en robotique mobile.
Le principe est assez simple, il s'agit d'affilier un champ de potentiel à l'environnement que perçoit la voiture à chaque instant. Pour faire simple, une zone à éviter (mur, obstacle, autre voiture…) correspond à un potentiel répulsif et à l'inverse une zone "d'espace libre" correspond à un potentiel attractif. Ensuite on applique un modèle simple de mécanique pour déduire la dynamique de la voiture en fonction de son environnement.

Pour l'illustrer comment l'implémenter en pratique on peut s'appuyer sur la Figure 7 [Planification de trajectoire par champs de potentiel fractionnaires appliquée au véhicule autonome" rédigé par Julien Moreau, Pierre Melchior, Stéphane Victor, Mathieu Moze, François Aioun, Franck Guillemard]

Cette approche propose une architecture à 3 étages

- l'étage stratégique : les données sont collectées et traitées à partir des capteurs de perception, c'est à dire le LIDAR dans notre cas. Cela permet de créer le champ de potentiel qui permettra à la voiture de se diriger.

- l'étage tactique : à partir des informations précédentes, une force résultante de la combinaison des champs de potentiel répulsif et attractif est générée, puis, par l'intermédiaire

d'un modèle simple (modèle masse ponctuelle), la position idéale, la vitesse et l'accélération sont générées en simulant le mouvement d'une particule chargée soumise à cette force générée ***F***=-***grad****(U)*.

- l'étage opérationel : la position idéale, la vitesse et l'accélération venant de l'étage précédent sont utilisées comme références pour une boucle de contrôle classique.

Dans notre cas on s'intéresse plus particulièrement aux deux premiers étages puisqu'aucune boucle de retour n'a été mise en place. On considère que la voiture peut suivre assez rigoureusement les consignes qui lui sont envoyées par l'étage tactique ou du moins avec une vitesse et une précision suffisante pour les attentes du concours.

En guise d'exemple on peut expliciter une construction de potentiel répulsif autour d'un obstacle. Premièrement on doit déterminer deux distances caractéristiques qu'on note r_{min} et r_{max} :

- r_{min} représente la distance minimale à laquelle le véhicule peut s'approcher de l'obstacle sans qu'il y ait collision, elle dépend donc de l'encombrement de la voiture et de la forme de l'obstacle.
- r_{max} représente la distance jusqu'à laquelle l'obstacle a un influence sur la voiture.

On peut alors comme le proposent [Julien Moreau, Pierre Melchior, Stéphane Victor, Mathieu Moze, François Aioun, Franck Guillemard] construire un potentiel ainsi :

$U(r) = (r^{n-2} - r_{max}^{n-2})/(r_{min}^{n-2} - {r_{max}}^{n-2})$ Où r est la distance entre le véhicule et l'obstacle considéré obtenue grâce au LIDAR et n représente l'ordre fractionnaire (voir Figure 2, cf [Julien Moreau, Pierre Melchior, Stéphane Victor, Mathieu Moze, François Aioun, Franck Guillemard])

En pratique nous avons considéré la méthode des potentiels un peu trop tard dans l'année et au vu de la situation de confinement nous avons préféré continuer le modèle plus simple proposé dans la suite qui était notamment plus accessible en termes de simulation.

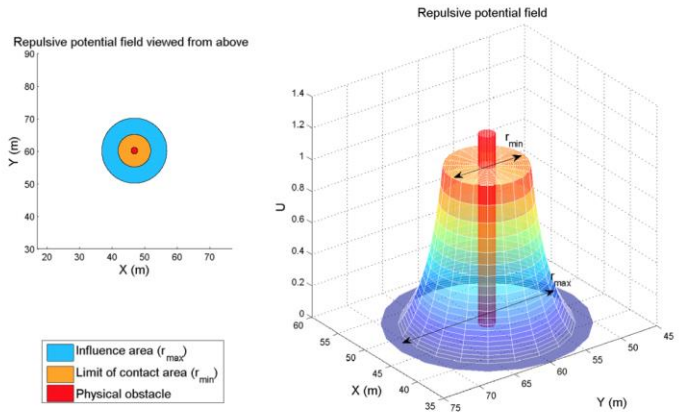


Figure 8 - Champ de potentiel répulsif construit autour d'un obstacle (cf [Julien Moreau, Pierre Melchior, Stéphane Victor, Mathieu Moze, François Aioun, Franck Guillemard])

4. Asservissement en vitesse

La première approche a été de travailler à vitesse constante et de simplement piloter la voiture en direction. Cette approche naïve a rapidement posé un problème puisqu'il y avait des situations où la voiture détectait l'obstacle trop tard et malgré un angle de braquage des roues maximal pour l'esquiver il y avait collision.
Il est donc nécessaire d'adapter la vitesse de la voiture pour qu'elle ralentisse si un obstacle est trop proche et qu'à l'inverse elle fonctionne à plein régime si elle le champ libre.
Tout comme pour la limite de distance avec le premier programme utilisant les capteurs, nous avons défini un seuil de distance à partir duquel la voiture ralentit de façon proportionnelle à la distance qui la sépare de l'obstacle le plus proche. Cela nous a permis de la rendre plus maniable et ainsi d'éviter les obstacles.

Enfin il faut traiter à part la situation où la voiture a percuté un mur ou une autre voiture et est bloquée à l'arrêt et lui imposant de passer en marche arrière et reculer suffisamment pour obtenir à nouveau un champ relativement libre devant elle.

V. Réalisation pratique et résultats

1. Circuits imprimés

Afin de s'affranchir des fils de connections, nous avons opté pour la réalisation d'un circuit imprimé pour la version avec les capteurs infrarouge et pour celle utilisant le LIDAR. Les schémas électriques ont été réalisé sur le logiciel Kicad.
Dans les deux schémas :

- le servomoteur est connecté à l'entrée PA15 (entrée de type Pwm)
- le moteur est connecté à l'entrée PB7 (entrée de type Pwm)
- le module bluetooth est connecté grâce aux entrées PC10 et PC11 (entrées qui gèrent l'émission et la réception de données bluetooth)

De plus, Le moteur alimente la carte nucléo via l'entrée Vin. Les autres composants sont ensuite alimentés grâce à leur connections à la patte 18 de la carte nucléo (tension de 5V). Et enfin toutes les masses sont reliées à la patte GND de la nucléo.

Circuit imprimé pour la version avec les capteurs :
Dans le circuit avec les capteurs IR, les sorties des capteurs sont connectées à la carte nucléo grâce aux entrées A0 à A4.

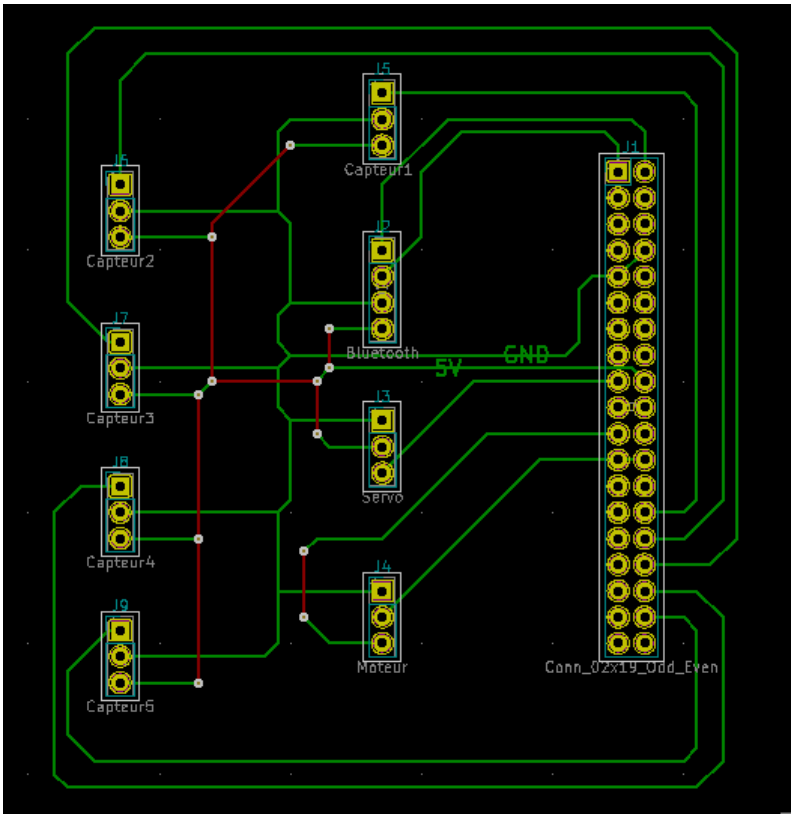


Figure 9 - Circuit imprimé pour la version avec les capteurs IR

Une fois la carte imprimée, on soude des fils de connections qui correspondent aux fils rouges sur le schéma. Ensuite on soude les fils des capteurs à l'emplacement qui leur est destiné sur le circuit imprimé puis on soude des supports destinés à accueillir le module bluetooth, le servomoteur, le moteur et la carte nucléo. Une fois les soudures réalisées, nous n'aurons plus qu'à insérer les composants dans leurs supports.

Circuit imprimé pour la version avec le LIDAR:

Dans le circuit avec le LIDAR, on soude les supports destinés à accueillir tous les composants (servomoteur, moteur, carte nucléo et LIDAR). On soude également les fils de connections qui correspondent aux fils rouges sur le schéma.

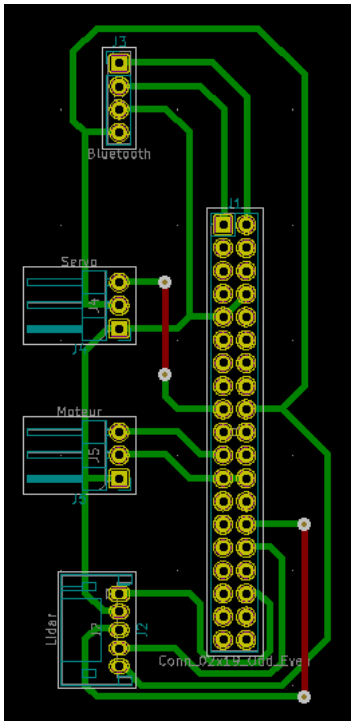


Figure 10 - Circuit imprimé pour la version avec LIDAR

2. Utilisation de deux contrôleurs

Nous avons vite réalisé à l'issu de tests que le contrôleur, qui est un élément électronique permettant d'adapter la tension délivrée par la batterie pour alimenter le MCC et le servomoteur, ajoutait un temps de réaction assez long, notamment au démarrage mais aussi lors du changement de vitesses. Nous avons donc décider de le remplacer par un autre qui nous paraissait mieux convenir. Toutefois, après démonter et remonter la voiture pour remplacer le contrôleur et effectué plusieurs tests, nous avons conclu que ce nouveau contrôleur n'était pas adapté au système. En effet, sa température augmentait considérablement ce qui indique qu'il n'était pas adapté aux tensions utilisées. De plus, s'il permettait effectivement d'obtenir des temps de réaction plus courts, il ne permettait pas d'atteindre une allure de la voiture aussi rapide. Nous avons donc décidé de revenir au contrôleur précédent en gardant l'idée qu'il fallait peut-être tester un troisième contrôleur mieux adapté aux tensions utilisées. Nous n'avons pas pu tester cette partie en raison du confinement.

3. Résultats avec les capteurs à infrarouge

Les capteurs à infrarouge sont les premiers que nous avons testés et déjà les résultats obtenus grâce à eux étaient assez satisfaisants. En effet, la voiture pouvait se diriger sur un circuit que nous avions fabriqué et éviter certains obstacles fixes. Il y avait toutefois certaines incohérences qui apparaissaient face à certains obstacles. On pense que cela peut être dû à un éventuel recouvrement des zones vues par chaque capteur qui pourrait interférer entre elles. Aussi, la répartition des 5 zones à l'avant doit être très précise et parfaitement symétrique par rapport à l'axe centrale pour pas que la détection soit meilleure d'un côté de l'autre et pour couvrir un maximum la zone totale avant de la voiture et éviter les angles morts. Ce genre de problèmes sont immédiatement résolus avec le LiDAR. Celui-ci permet aussi

d’observer les obstacles à une distance plus grande pour mieux les anticiper et notamment les repérer suffisamment tôt par rapport au temps de réponse total du système lorsqu’on augmente la vitesse. En effet, ces capteurs étaient moins performants lorsqu’on augmentait la vitesse.

4. Résultats avec la caméra

Malheureusement, en raison de l’influence de covid19, nos données expérimentales ont été conservées dans l’ordinateur de l’école. Ici je présenterai brièvement les résultats finaux que nous avons obtenus.

Nous avons établi des bibliothèques de couleurs pour comparer plusieurs couleurs courantes, y compris le rouge, le vert, le bleu, le noir, l'orange, le blanc et le gris, et avons raisonnablement modifié la gamme RVB correspondant à chaque bibliothèque de couleurs. Ensuite, nous sélectionnons des objets avec des couleurs spécifiques pour l'identification, et vérifions l'exactitude de la bibliothèque de couleurs établie ci-dessus.

Après cela, nous avons mené l'expérience susmentionnée pour changer la direction de la voiture en fonction de la couleur du mur. Nous avons traité l'image obtenue à partir de la caméra Raspberry Pi, analysé le nombre de pixels verts et rouges, puis envoyé un signal de virage à la voiture., Le signal est transmis par le Raspberry Pi à la carte nucléo qui contrôle la voiture.

Après des tests expérimentaux, la voiture a réussi à diriger selon les différentes couleurs des murs des deux côtés et a pu terminer le cercle sur la piste fixe.

Cependant, les inconvénients de ce système sont également évidents : en réalité, il y a très peu de cas où les couleurs des murs des deux côtés sont complètement différentes, de sorte que notre système de voiture intelligente n'utilisera finalement pas ce système. Si nous voulons ajouter la fonction de reconnaissance faciale de la voiture intelligente ou la fonction de juger du type d'obstacles dans les projets ultérieurs, nous pouvons continuer sur la base de cette partie de la recherche.

5. Résultats avec le LiDAR

En raison du confinement, nous n’avons pas eu l’occasion d’exploiter convenablement le traitement de données du LiDAR. En effet, nous sommes parvenus à faire rouler la voiture au cours des séances précédant le confinement mais nous nous sommes très vite aperçus que le LiDAR nécessitait grandement l’implémentation d’une zone de sécurité expliqué par la suite. A de nombreuse reprise, nous avons remarqué que la précision du LiDAR permettait de passer par des passages étroits qui ne prenaient pas en compte l’envergure de la voiture. Nous avons également essayé d’adapter la vitesse mais nous ne sommes pas parvenus à finir avant de quitter l’école. Ainsi, en ce qui concerne le LiDAR, nos résultats sont donc essentiellement représentés par les simulations.

V. Simulations

1. Définition d’une zone de sécurité

a) Premières idées et constructions

Après avoir réalisé plusieurs tests avec le LiDAR le comportement de la voiture nous a amené à penser qu'il pourrait être utile de définir une zone de sécurité devant la voiture. Dans celle-ci, le traitement des données précédent ne s'appliquerait pas car il n'est alors plus performant pour éloigner suffisamment la voiture des obstacles. En effet, si un objet est détecté dans cette zone, cela signifierait que ce traitement n'a pas permis d'éviter cet obstacle et qu'il faut alors imposer à la voiture de tourner en fonction de l'angle auquel se situe l'objet dans la zone. Nous nous sommes alors basés sur deux paramètres pour déterminer cette zone, le premier étant la distance minimale de braquage de la voiture. En effet, si un objet est détecté à une distance inférieure à celle que la voiture requiert pour braquer ses roues au maximum, elle ne pourra pas éviter cet obstacle. On a réalisé l'expérience représentée sur le schéma suivant pour déterminer cette distance : on a implémenté un code à la voiture lui ordonnant de tourner toujours dans le même sens et en braquant ses roues au maximum et on a tracé le demi-cercle correspondant à la trajectoire de la voiture. On a ensuite mesuré la distance de la voiture à l'obstacle qui correspond donc à la distance de braquage recherchée.

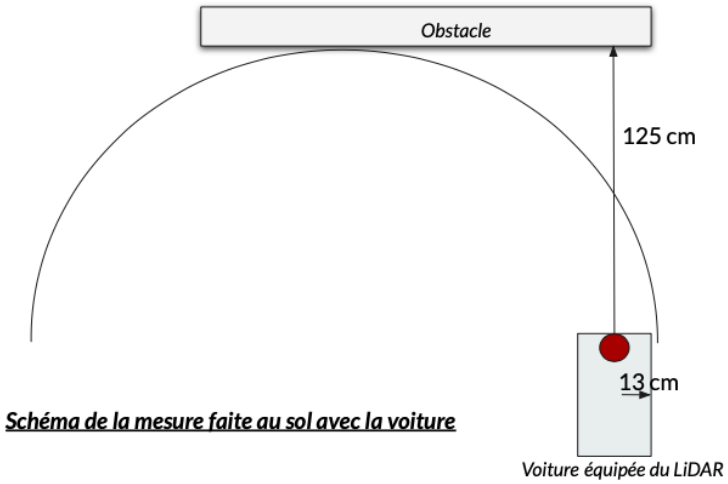
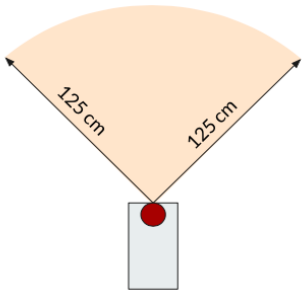


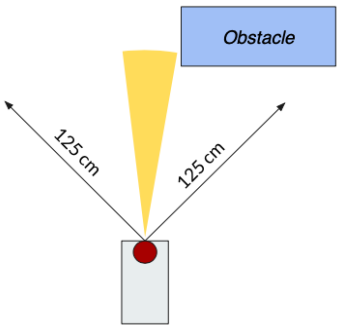
Schéma de la mesure faite au sol avec la voiture

On a estimé cette distance à environ 125cm, en rajoutant quelques centimètres de sécurité. Notre première idée, la plus naïve, a été d'établir un demi-cercle de sécurité de rayon 125cm de diamètre. Toutefois, nous avons vite réalisé qu'il est aberrant de prévoir 125cm de sécurité sur les côtés de la voiture. Nous avons alors pensé à prendre un demi-cercle sur une certaine plage d'angles devant la voiture uniquement pour réaliser une zone de la forme ci-contre.



Toutefois cette méthode tend toujours à éloigner la voiture de murs qu'elle longerait sans que cela soit nécessaire. De plus elle ne tient pas compte de l'envergure de la voiture. Or nous avons aussi remarqué lors de nos premiers tests que le LiDAR retient parfois une distance maximale qu'il choisit pour orienter la voiture alors que ce n'est pas une solution envisageable car l'envergure de la voiture est trop grande. C'est alors dans cette situation comme si le LiDAR seul pouvait accepter cette direction mais pas la

voiture en entière. Cette situation doit aussi être prise en compte par cette zone de sécurité. On prend pour l'exemple la situation schématisée ci-dessous.



Ici, on a réduit la largeur de la zone de sécurité imaginée précédemment pour que la voiture ne s'éloigne pas trop des murs. Toutefois cette zone n'est plus suffisante pour permettre à la voiture d'éviter l'obstacle comptes tenus de son envergure. En effet, on voit que si la voiture continue tout droit, ce qu'elle doit faire car elle suivra le traitement des données précédent car aucun obstacle est présent dans la zone de sécurité et que la distance maximale qu'elle voit est celle pour un angle de 0°, le coin droit de la voiture ne passera pas l'obstacle. Il faut donc imaginer une autre forme pour la zone de sécurité. C'est là qu'intervient le deuxième paramètre qui est la demi-largeur de la voiture que l'on prend égale à 13cm après mesures.

b) Recherche d'une loi entre les distances de sécurité voulues et l'angle

On imagine donc une forme de coma pour la zone, proche de la zone imaginée juste au-dessus, donc pas trop large pour que la voiture ne s'écarte pas trop des murs mais suffisamment large au bout pour prévenir l'exemple précédent de se produire. Il faut aussi vérifier que la voiture peut longer le mur donc que le LiDAR est bien au minimum à 13cm des murs. On a utilisé alors deux contraintes : à 0° on doit avoir une distance d = 125cm et à 90°, on doit avoir d = 13cm. On a ensuite cherché une loi reliant la distance d à l'angle theta puis tracer la courbe paramétrique x en fonction de y obtenue avec cette loi (puisqu'on a x = d*cos(theta) et y = d*sin(theta)). On a réalisé une première simulation en prenant une loi proportionnelle entre d et theta. Avec une telle loi, on obtient déjà une zone de sécurité à la forme de coma comme recherchée :

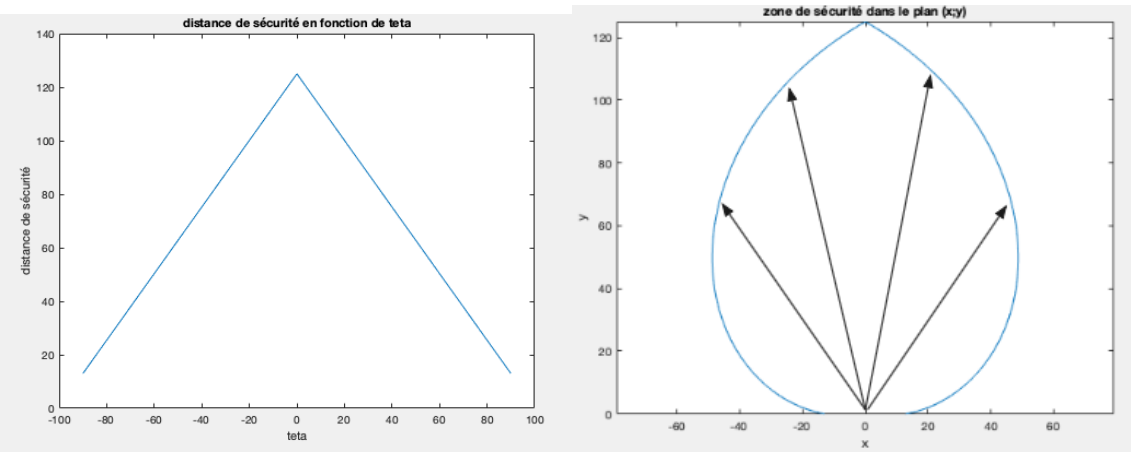


Figure 11 - Loi proportionnelle entre la distance et l'angle et sa représentation paramétrique

On a représenté sur le schéma les vecteurs de norme d et de direction theta. En raison du confinement, nous n'avons pas pu tester cette zone ni les suivantes donc nous baserons nos conclusions sur notre intuition. Cette zone qui a bien une forme de coma et respecte les deux contraintes imposées précédemment sur les distances, ne semble pas satisfaisante. En effet la partie bombée de la coma se situe vers les petits y et non les grands comme attendu. On s'attend donc à ce que cette zone ait pour conséquence que la voiture s'éloigne des murs plus que nécessaire mais qu'elle ne permette pas d'éviter des obstacles comme celui schématisé à la fin de la partie a) car elle ne semble pas assez large pour les grands y. On utilise alors une loi exponentielle pour que la distance de sécurité diminue plus rapidement avec l'angle. On obtient alors les résultats suivants :

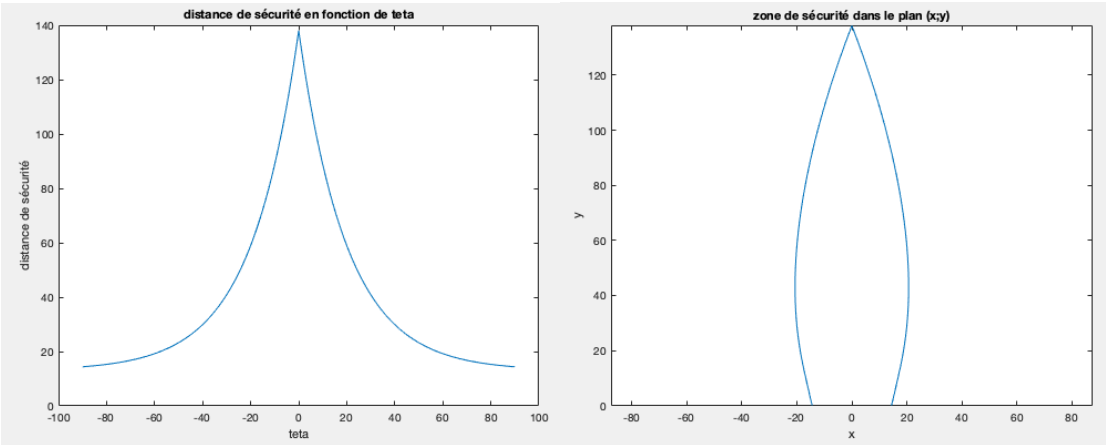


Figure 12 - Loi exponentielle entre la distance et l'angle et sa représentation paramétrique

Cette zone est plus satisfaisante que celle obtenue avec une loi proportionnelle car elle est beaucoup moins bombée donc la voiture ne devrait pas s'éloigner des murs plus que nécessaire. Toutefois une telle loi entraîne une zone très étroite pour les grands y ce qui n'est toujours pas satisfaisant pour des obstacles tels que celui de la fin de la partie a). On pense alors à prendre une loi gaussienne pour garder une assez grande distance aux petits angles mais permettre une décroissance rapide ensuite et éviter le côté trop bombé aux petits y. Les résultats obtenus sont les suivants :

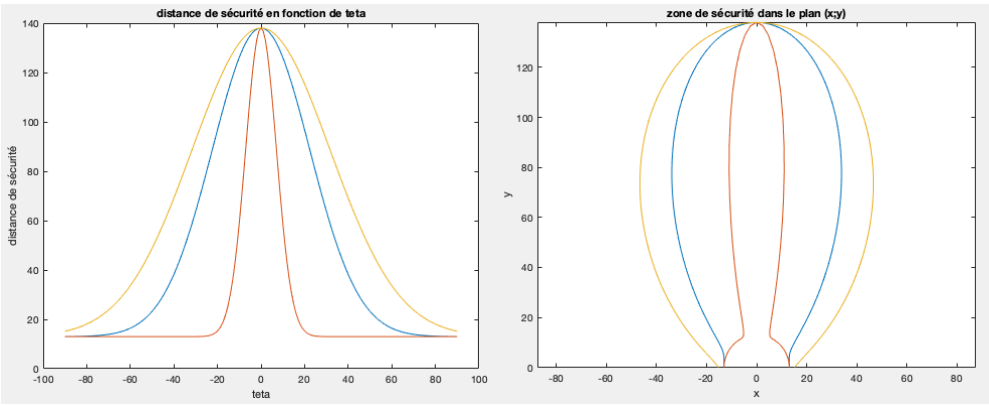


Figure 13 - Loi gaussienne entre la distance et l'angle et sa représentation paramétrique

Comme prévu, cette loi permet effectivement d'avoir une zone de sécurité qui semble répondre à nos conditions. Il faudrait maintenant pouvoir la tester pour ajuster au moins l'écart-type de la gaussienne et donc la largeur de la zone.

2. Simulation d'un code qui combine l'adaptation de la trajectoire et de la zone de sécurité

Dans la mesure où nous n'avons pas eu la possibilité de tester la théorie que nous avons mis en place, nous nous sommes penché sur sa simulation à l'aide de MATLAB. Pour ce faire, nous avons représenté le circuit à l'aide d'une matrice sur fond noir avec des bords blancs. Ainsi, pour ajouter des obstacles nous avons simplement ajouté des traits blancs modélisant ceux-ci. Dès lors, le principe de fonctionnement de la simulation est de chercher le point blanc le plus loin d'une position donnée. Voici donc les fonctions que nous avons créé et utilisé :

- **build_circuit**: Le but de cette fonction est construire le circuit en plaçant les obstacles à la place souhaité donnée en entrée de la fonction.
- **distance_max**: à partir d'une position donnée en entrée (colonne, ligne) (représentant des coordonnées (x,y) en repère cartésien) on trouve la position du point blanc le plus loin de la position initiale.
- **next_position**: à partir d'une position initiale et de la position du pixel blanc le plus loin, cette fonction détermine le prochain pixel sur lequel se placer.
- **next_position_zone:** réalise la même fonction que celle précédente en prenant cette fois ci en compte la zone de sécurité.
- **test** : à partir d'une position donnée, on test s'il y a des pixels blancs présents dans la zone de sécurité.

Nous avons enfin assemblé le tout et tracé la trajectoire en choisissant un nombre de points arbitraire permettant d'obtenir une trajectoire exploitable (environ 80 pour une matrice de taille 100). Nous obtenons ainsi les résultats présents sur la figure ci-dessus.

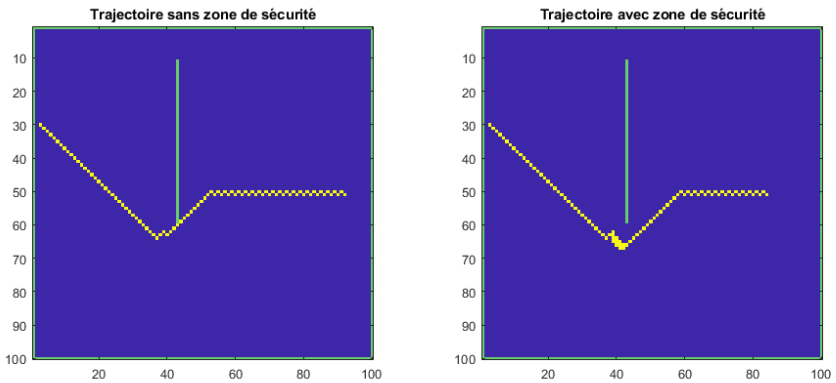


Figure 14 - Modélisation des trajectoires sur MATLAB avec et sans la zone de sécurité

On peut donc voir sur la figure ci-dessus que dans les deux cas, la trajectoire contourne bien l'obstacle (ici trait vert) en suivant a priori la distance maximale. En revanche, dans le cas où

on utilise la zone de sécurité, on peut bien voir que la trajectoire contourne le bord de l'obstacle au lieu de le frôler comme dans le cas sans zone. Ceci modélise donc bien nous souhait quant à l'implémentation d'une zone de sécurité dans notre trajectoire. En effet, ce contour de l'obstacle prouve bien de la prise en compte de l'envergure de la voiture. Sans cette prise en compte, le LiDAR indique de frôler l'obstacle car cela correspond bien à la distance maximale. Néanmoins, ceci ne permet pas le passage de la voiture mais seulement du LiDAR.
On peut également relever l'oscillation présente après le passage de l'obstacle sur la ligne 50. Cela correspond à l'alternance de distance maximale entre le coin en haut à droit et le coin en bas à droite. Nous avons pu étudier ce phénomène en pratique puisque la voiture ne roule jamais strictement en ligne droite mais oscille entre les différentes distances maximales. C'est également ce qui nous a poussé à moyenner les distances maximales obtenues avec le LiDAR afin de réduire cette oscillation et fluidifier la trajectoire.

Conclusion

Ce projet PIMS nous a permis d'aborder les thématiques des véhicules autonomes en plus d'apprendre à utiliser des détecteurs optiques tels que des capteurs à infrarouge et un LiDAR. Il nous à apporter des compétences en électronique, en programmation mais aussi un petit peu en mécanique puisque nous avons dû monter la voiture entièrement. Dans tous les cas nous avons été capable de remplir au moins les contraintes sur la trajectoire : notre voiture est capable de démarrer, de s'arrêter et d'éviter les obstacles fixes, au moins avec les capteurs IR. Ces performances ont été en partie atteintes avec le LiDAR bien que nous nous sommes confrontés à la difficulté de devoir prendre en compte l'envergure de la voiture. Cela nous a amené à concevoir des solutions sous forme de zones en partie parce que l'utilisation de fonctions trigonométriques n'était pas satisfaisante avec MBED. La dernière partie de notre travail a donc été sur la recherche de solutions plus complexes, avec l'élaboration de zones de sécurité mais aussi en se basant sur des recherches documentaires pour optimiser la trajectoire en prenant une approche du problème différente (champ de potentiel). Nous n'avons malheureusement jamais pu tester ces solutions en raison du confinement et avons dû nous contenter à des simulations sur Matlab.

Annexe : Code le plus récent pour le traitement de données du LiDAR

```
//////////////////////////////
///    Code Autono'Car    ///
//////////////////////////////////
//  RP Lidar - A2-M8 - Slamtec //
////////////////////////////////////////
/// Institut d'optique Graduate School ///
////////////////////////////////////////
///////    PIMS 2019 - 2020  //////////////
//////////////////////////////////////////////////////////////////////////
// Kévin Falque - Alexis Kedzia - Kassim Hanane - Hugo Affaticati //////////////
// Zhe Wang - Benoit Paris- Cissé Mahawa - Aurélie Lebrun - Thomas Serre /////
//////////////////////////////////////////////////////////////////////////



//////////////////////////////
////// Bibliothèques ///////
 ////////////////////////

#include "mbed.h"
#include "rplidar.h"

//////////////////
///// Brochage ////
//////////////////

Serial moduleBT(PC_10,PC_11);  // connection émetteur/récepteur bluetooth

DigitalOut my_led(LED1);
Serial my_pc(USBTX,USBRX);// connection émetteur/récepteur pc
RPLidar my_lidar;        // Définition du LiDAR
PwmOut my_lidar_pwm(A3); // Brcohe de connexion
PwmOut servo(PA_15);    //Broche de connection du servomoteur
PwmOut moteur(PB_7);     //Broche de connection du motor driver contrôlant la vitesse du moteur

BufferedSerial my_rp(A0, A1);       // connection émetteur/récepteur LiDAR (TX/RX)
rplidar_response_device_info_t lidar_info;


////////////////////////////////////
////////// VARIABLES //////////////
////////////////////////////////////


// DIRECTION //
```

```
double sumDist;
double sumAngle;
const  int taille_liste = 10;          // Nombre de tours sur lequel on moyenne la distance max
double maxDistance, angleAtMaxDist;     // Distance max et angle associé à chaque tour
int compteur=0;
double mAng;                            // Moyenne de l'angle de la distance max après 'taille_liste'
tour
double mDist;                           // Moyenne de la distance max de la distance max après
'taille_liste' tour
double zin[taille_liste][2];           // Liste stockant les distances max des 'taille_liste' tours
double angle_roues;                     // Commande envoyée au servomoteur
const double k = 6;
int s;
char data = 's';                       // consigne bluetooth


// CORRECTION //
////////////////////////////////////////////////////////////////////////
//  Variables pour l'optimisation de la tejectoire que nous n'avons ///
//  pas eu le temps de traiter en raison du confinement             ///
////////////////////////////////////////////////////////////////////////
double angleD;
double correction;
const double d90 = 1250;
const double d0 = 130;
const double largeur = 18 ;



////////////////////////////////////
/////////// FONCTIONS //////////////
////////////////////////////////////


// FONCTION NORMALISATION ANGLES //
////////////////////////////////////////////////////////////////////////////////
// Fonction qui passe les angles entre 270° et 360° en angles entre -90° et 0°//
////////////////////////////////////////////////////////////////////////////////

double normalisation(double angle){
   if ( angle > 270){
      angle = angle - 360 ;
      }
   return angle ;
   }


// FONCTION TEST DISTANCE //
```

```
////////////////////////////////////////////////////////////////////////////
// Fonction qui test la zone de sécurité: non testé en raison du confinement //
////////////////////////////////////////////////////////////////////////////

double test_distance(double angle, double distance){
    if ( angleD < 0 ){
        if ( distance < (angle*(d0 - d90)/90) + d0 ){
            return 1 ;
            }
        else{
            return 0;
            }
        }
    else{
        if ( distance < (angle*(d90 - d0)/90) + d0 ){
            return 1 ;
            }
        else{
            return 0;
            }
        }
    }

////////////////////////////////////

////////////////////////////////////
/////////////// MAIN /////////////////
////////////////////////////////////

int main() {
    // Initialisation moteur et servomoteur
    moduleBT.baud(9600);         //vitesse de transmission du capteur BT à la carte
    servo.period_ms(20);         //période de réception du servomoteur
    servo.pulsewidth_us(1300);   //première commande du servomoteur -> tout droit
    moteur.period_ms(20);        //période de réception du moteur
    moteur.pulsewidth_us(1500);  // 1500 -> point mort
    wait(2);                     // 2s imposée par la voiture
    int alp = 0;                 //constante qui prend la valeur 1 quand la voiture est en marche
    float vit;                   //vitesse du moteur

   // VITESSE DE ROTATION LIDAR
    my_lidar_pwm.period_us(40);
    my_lidar_pwm.pulsewidth_us(20);

    // TRANSMISSION PC
    my_pc.baud(115200);

    // INITIALISATION LiDAR //
```

```
   my_lidar.begin(my_rp);
   my_lidar.setAngle(0,360);

   my_led = 1;

   // Initialisation du régulateur //

   for(vit = 1500; vit < 1510; vit+=1){
      moteur.pulsewidth_us(vit);
      wait(0.1);
   }
   vit = 1500;
   moteur.pulsewidth_us(vit);

   wait(1);
   alp = 0;
   my_led = alp;

  ////////////////////////////////

   while(1) {
      my_led = alp;
   // CONNECTION BLUETOOTH : ALLUMER ET DEMARRER LA VOITURE
      if (moduleBT.readable()) {                    // car version avec fonction d'interruption non
fonctionnelle
         data = moduleBT.getc();                  // récupérer un caractère sur la carte venant du
capteur BT
         if (data == 'g'){                    // g pour go -> démarrer la voiture
            wait(1);
            moteur.pulsewidth_us(1600);        // 1600 -> vitesse de démarrage (la plus lente)
            alp=1;                           // stockage de l'état marche ou arrêt de la voiture
            data = '0';
         }
         if (data == 's') {                   // s pour stop -> arrêter la voiture
            moteur.pulsewidth_us(1500);
            alp=0;
            data = '0';
         }
      }


    //RECUPERATION DONNEES LIDAR ET CALCUL DE LA MOYENNE
      if (IS_OK(my_lidar.waitPoint())) {                    // ATTENTE D'UNE DONNEE
         float distance = my_lidar.getCurrentPoint().distance; // RECUPERATION DE LA
DISTANCE
         float angle = my_lidar.getCurrentPoint().angle;       // RECUPERATION DE L'ANGLE

         if (my_lidar.getCurrentPoint().startBit) {    // CHECK D'UN TOUR DU LIDAR : if(1)
```

```
        if (compteur==taille_liste){            // VIDE LA LISTE : if(2)
          sumDist=1;                          //
          sumAngle=1;                          //
          compteur=0;                          //
          int s = 0;                         //

          while(s<taille_liste+1){     // MOYENNAGE SUR taille_liste VALEURS
            sumDist += zin[s][1];     //
            sumAngle += zin[s][0];    //
            s++;                       //
            }                        //
          mAng=(double)sumAngle/taille_liste;   // CALCUL DE L'ANGLE MOYEN
          mDist=(double)sumDist/taille_liste;       // CALCUL DE LA DISTANCE MAX
MOYENNE

          moduleBT.printf("angle = %lf et distance = %lf \r\n", mAng , mDist); // AFFICHE
LES VALEURS MOYENNES SUR LE TELEPHONE

        //CONSIGNE SERVOMOTEUR
        angle_roues = 1300 + k*angleAtMaxDist;      // APPROCHE DE DIRECTION
PROPORTIONNELLE
        servo.pulsewidth_us(angle_roues);
        }

        compteur++;                       // REMPLISSAGE DE LA LISTE TEMPORAIRE
        zin[compteur][0]=angleAtMaxDist;    //
        zin[compteur][1]=maxDistance;       //


        // REINITIALISATION ANGLE ET DISTANCE
        maxDistance = 0;
        angleAtMaxDist = 0;
        }
        else {                                  // SI ON EST PAS AU POINT DE DEPART (IF(1)):
RECHERCHE DU MAX SUR LE TOUR EN QUESTION
                                    //
        if ((angle < 90) or (angle >270)){   // ON GARDE QUE LES ANGLES DE DEVANT


          if ( (distance > 0) &&  (distance > maxDistance)) {       // RECHERCHE DE LA
DISTANCE MAX
            maxDistance = distance;                          //
            angle = normalisation(angle);                      // ANGLES ENTRE 270 ET 360
DEVIENNENT ENTRE 0 ET -90
            angleAtMaxDist = angle;
          }
        }
```

```
          }
        } else {
            if (IS_OK(my_lidar.getDeviceInfo(lidar_info, 100))) {   // SI ON NE RECOIT PAS DE
DONEES DU LIDAR
                my_lidar.startScan();                           //
                wait(1);                                    //
                my_pc.printf("START SCAN\r\n");                  //
            }                                               //
        }


    //CONSIGNE MOTEUR
    if(alp==1){
       vit = 1600;
       moteur.pulsewidth_us(vit);
       }

    // ASSERVISSEMENT DE LA VITESSE
    // Nous n'avons pas eu la possibilité d'ajouter cette partie pour le LiDAR //
    // en raison de problèmes rencontrés durant le confinement et non résolus  //

    }
}
```