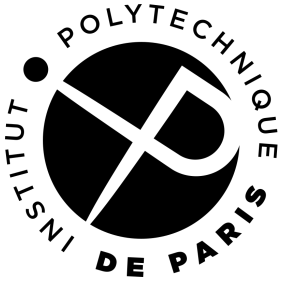


Pôle Simulation - Voitures Autonomes

Projet d'Ingénieur en Équipe - PIE

Avril 2024

Introduction

Les simulations ont été implémentées dans le logiciel Webots, selon ce qui avait été fait par les équipes des années précédentes et les directives des organisateurs de la CoVAPSY. La vaste documentation disponible sur cet outil a permis sa maîtrise efficace.

Initialement, les efforts ont été dirigés vers l'appropriation des codes des équipes des années précédentes. Ensuite, des modifications du code et des paramètres des composants ont été apportées pour rapprocher le comportement en simulation du comportement réel sur la piste. Enfin, des modifications du modèle de prise de décision d'angle ont été mises en œuvre pour explorer différentes stratégies de mouvement. Dans ce cas, on note la mise en œuvre d'un virage plus brusque en fonction de la distance de la voiture par rapport aux limites de la piste et un algorithme pour centraliser la voiture sur la piste. De plus, plusieurs nouvelles pistes ont été construites en simulation afin d'évaluer le comportement du modèle dans différentes conditions.

Pour les simulations futures, il est recommandé de poursuivre le travail de suivi de l'angle absolu de direction de la voiture à l'aide du gyroscope. Dans ce contexte, un outil de cartographie serait également souhaitable, en prolongement de ce qui a été fait avec l'algorithme SLAM cette année.

Appropriation de la simulation sous Webots

Installation

Suivre les instructions dans le lien : https://cyberbotics.com/doc/guide/installation-procedure.

Téléchargement des documents pertinents

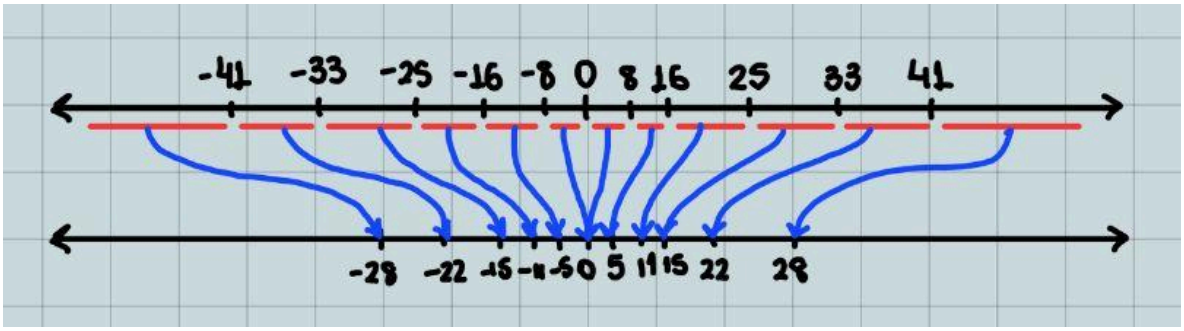
Faire un clone du repo dans https://github.com/bebraga00/Webots. Ici, vous trouverez les fichiers :

- Des contrôleurs avec l'extension .py (on a utilisé controller_jaune.py) ;
- Des prototypes des composants utilisés en simulation avec l'extension .proto ;
- Des pistes avec l'extension .wbt.

Un document diffusé par les organisateurs de la course avec un tutoriel plus complet est ajouté en annexe à ce document. D'autres tutoriels sont aussi disponibles sur le site de Webots : https://cyberbotics.com/doc/guide/tutorials.

Le code python utilise les composants **Gyro** (gyroscope qui mesure la vitesse angulaire de la voiture), **Lidar** (lidar qui mesure les distances pour chaque angle) et **Driver** (voiture qui reçoit l'angle et la vitesse souhaités), tous implémentés dans les bibliothèques de Webots (https://cyberbotics.com/doc/guide/sensors?version=R2022b). À chaque itération, on reçoit les données du lidar et du gyroscope et on prend la décision de vitesse et angle selon ces composants. On peut ajouter plusieurs voitures pour comparer des stratégies différentes.

La stratégie actuelle est plutôt empirique et basée sur la poursuite de la plus grande distance. Une interpolation linéaire des données du lidar est faite est on cherche l'angle avec la plus grande distance. Selon cet angle, on choisit un angle de consigne plus bas pour faire le rapport entre l'angle de distance maximale et l'angle que la voiture peut effectivement suivre vu ses contraintes mécaniques. Ce filtre évite aussi que la voiture se frappe contre les murs. Les valeurs ont été déterminées de façon empirique. La loi de vitesse suit une exponentielle selon l'angle de consigne, et une marche arrière est déclenchée si la distance en face est inférieure à 75 cm.



Relation entre l'angle idéal que la voiture suivrait (en haut) et l'angle de consigne (en bas).

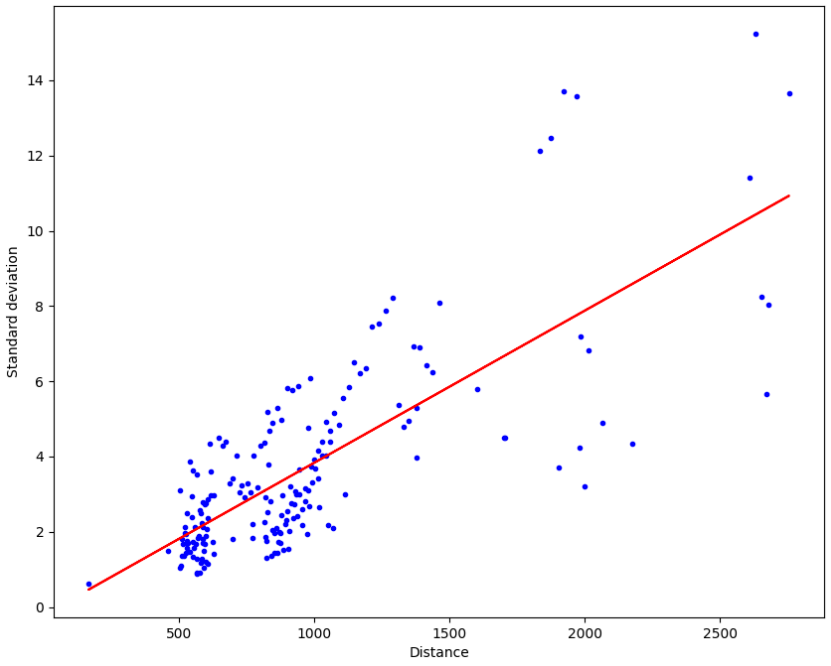
Bilan des simulations 2023-2024

Modélisation du lidar

Pour rapprocher le comportement en simulation au comportement dans la course, les paramètres suivants ont été adaptés du prototype standard donné par Webots. Ces valeurs ont été prises du *datasheet* du lidar utilisé, en annexe à ce document.

```
field SFInt32    lidarHorizontalResolution 1600
field SFFloat    lidarMinRange             0.1
field SFFloat    lidarMaxRange             12.0
field SFFloat    lidarNoise_OverMaxRange   0.00
field SFFloat    lidarResolution           0.01
field SFFloat    lidarDefaultFrequency     10
```

Le bruit est modélisé en temps réel dans le code du contrôleur selon une distribution gaussienne centrée en zéro et avec une variance mesurée. Cet experiment a été conduit avec plusieurs tournages du lidar en repos. Le code python est sur le repo de la section précédente. Le résultat est montré ci-dessous.

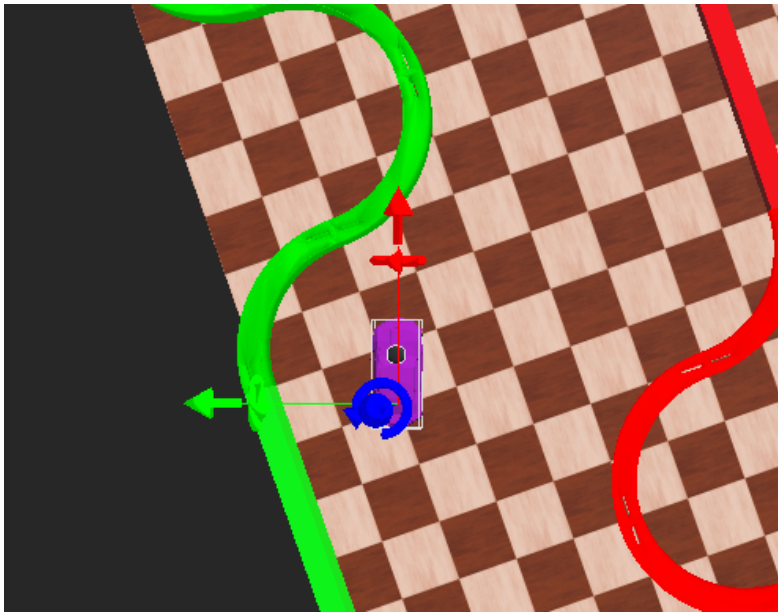


Les résultats montrent que $\sigma \approx \frac{4}{100}\mu$.

Pour rapprocher la simulation au comportement réel, il faut explorer les paramètres de simulation du prototype de la voiture. **Je suggère le travail conjoint entre un élève en informatique et un élève en mécanique pour en conduire des mesures**.

Raffinement modèle empirique

Lors d'une simulation avec une nouvelle piste, on a remarqué que la voiture se bloquait si l'angle de consigne n'était pas suffisant pour éviter un mur. Dans ces cas, l'angle de distance maximale était inférieur à 8º (donc l'angle de consigne serait 0º), alors la voiture entrerait dans un cycle d'avancer tout droit et de reculer tout droit.

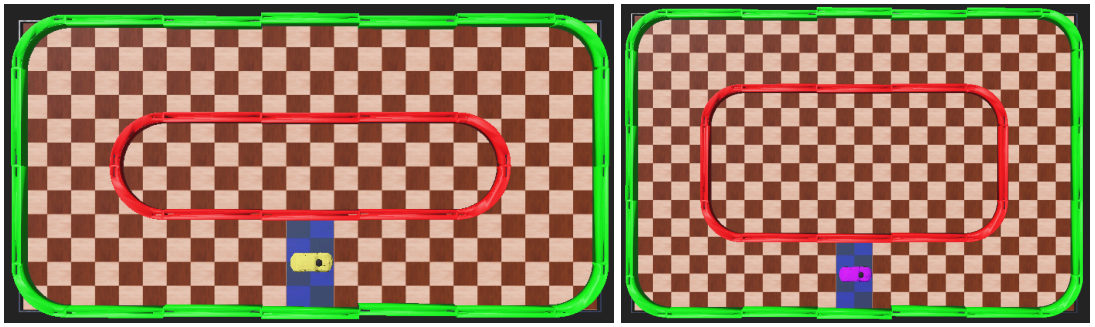


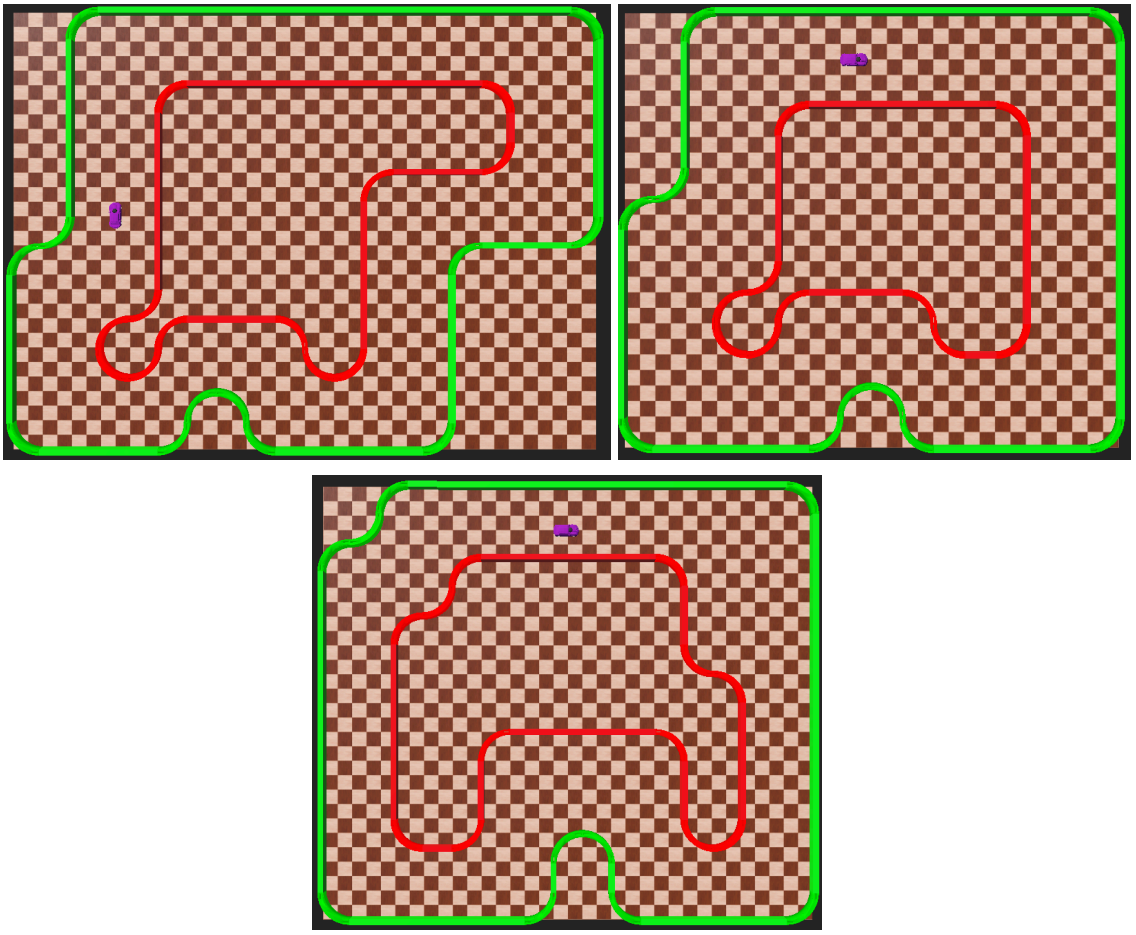
Situation exemple.

Ainsi, pour éviter ce problème, un mode pour éviter les obstacles proches de la voiture a été créé. Si la distance en face est entre 75 et 125 cm, on fait une moyenne de la distance en 20º à gauche et à droite. On compare ces résultats et on décide s'il faut faire un tournage plus brusque vers une de ces directions.

Nouvelles pistes

Des nouvelles pistes ont été construites sur Webots selon les essais et la course à l'ENS. Elles sont montrées ci-dessous :



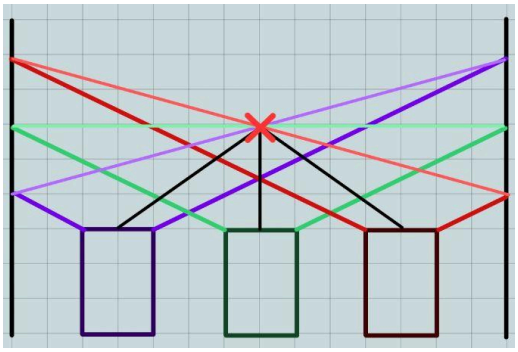


Pistes simulées

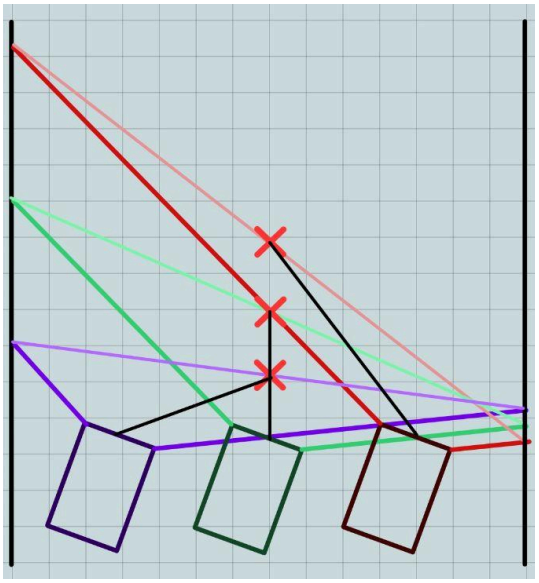
Positionnement de la voiture au milieu de la piste

Pendant les essais à l'ENS, on a remarqué que la voiture bleue ne suivait pas une trajectoire fluide pendant les parties droites de la piste, mais plutôt qu'elle empruntait un chemin sinueux et peu efficace. Ainsi, un software de positionnement de la voiture au milieu de la piste était souhaitable.

La stratégie choisie a été d'observer les mesures extrêmes du lidar, c'est-à-dire, celles le plus à gauche et le plus à droite et prendre la distance moyenne dans un intervalle de quelques degrés. Ces valeurs sont convertis en coordonnées x et y, le point moyen est calculé, et l'angle central est obtenu avec un arc tangent. Voici quelques exemples de cette stratégie :



Positionnement au milieu de la piste avec la voiture droite.



Positionnement au milieu de la piste avec la voiture inclinée.

L'angle de consigne serait donc une moyenne pondérée entre l'angle central et l'angle avec distance maximale **(les poids optimales pour cette moyenne sont à régler)**.

Cependant, on n'a pas eu beaucoup d'occasions de tester ce code sur la voiture réelle étant donné la proximité de la date de la course et la priorisation de la sûreté de fonctionnement par rapport aux nouvelles stratégies de mouvement à ce moment-là. **Il faut en tester plus**.

Suivi de l'angle absolu de direction de la voiture

On a remarqué, lors des essais et de la course avec la grande piste à l'ENS, que des fois la voiture se trouvait perpendiculaire au mur (après être frappée par une autre voiture, par exemple) et, après le déclenchement de la marche arrière, elle ne savait pas la bonne direction à suivre vu que le modèle utilisé ne cherche que l'angle avec la plus grande distance. Ainsi, il

est souhaitable que la voiture sache la direction à laquelle elle pointe pour ne pas tourner vers la direction contraire.

Pour cela, les arduinos disposent des gyroscopes, qui mesurent la vitesse angulaire à chaque itération. Ainsi, pour obtenir l'angle actuel il faut intégrer cette vitesse dès le début de la simulation. La mesure du gyroscope est multipliée à chaque fois par le temps entre les itérations pour obtenir la variation de l'angle. Voici le code utilisé :

```
previous_time = time.time()

while driver.step() != -1:
    current_time = time.time()
    time_diff = current_time - previous_time
    previous_time = current_time

    donnees_gyro = gyro.getValues()
    if(not np.isnan(donnees_gyro[2])):
        absolute_angle = (absolute_angle +
                          np.rad2deg(donnees_gyro[2]) * time_diff) % 360
```

Une stratégie pour utiliser cette information doit être conçue. Ce qu'on à imaginé serait de faire une moyenne mobile de l'angle pendant un intervalle de temps à déterminer (la voiture sait qu'elle va vers le nord, par exemple) et, après un changement brusque de l'angle et le déclenchement de la marche arrière, elle se positionne vers cet angle enregistré.