

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

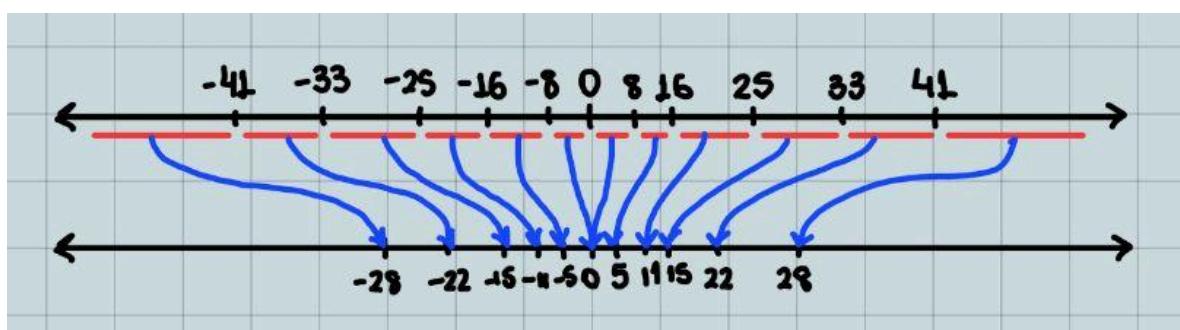
Faire un clone du repo dans <https://github.com/bebraga00/Webots>. Ici, vous trouverez les fichiers :

- 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 et 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).

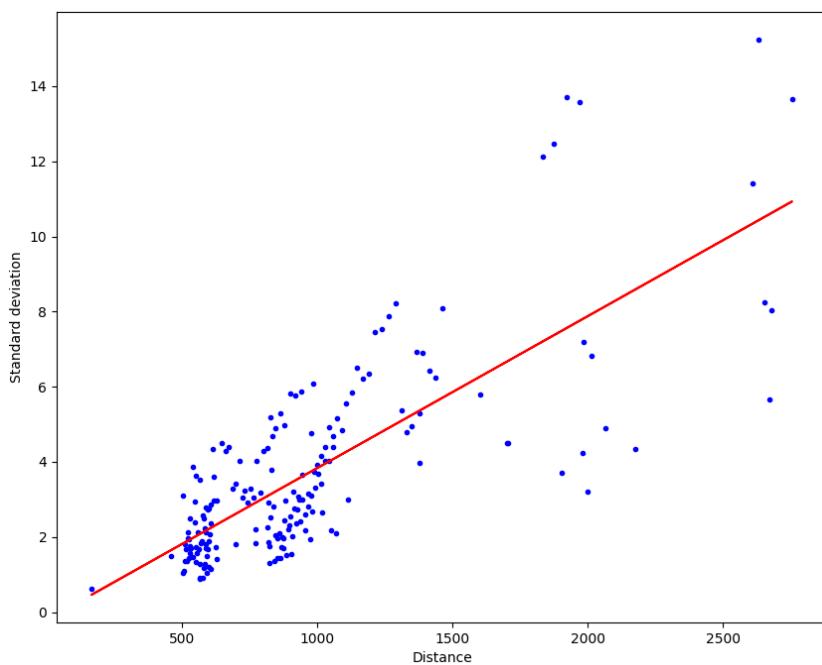
Bilan des simulations 2023-2024

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 lidar modélisé par Webots est parfait, alors on y ajoute du bruit pour le rapprocher au lidar réel. Il 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.

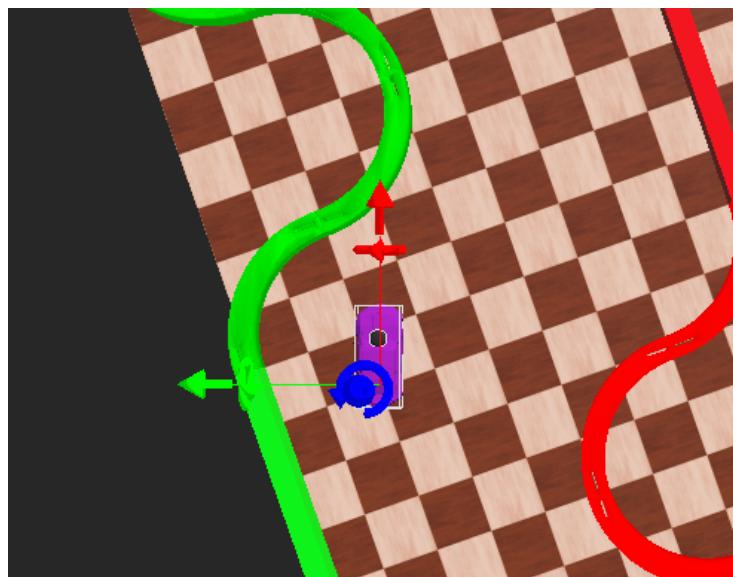


Les résultats montrent que $\sigma \approx \frac{4}{100} \mu$.

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 8° (donc l'angle de consigne serait 0°), alors la voiture entrerait dans un cycle d'avancer tout droit et de reculer tout droit.



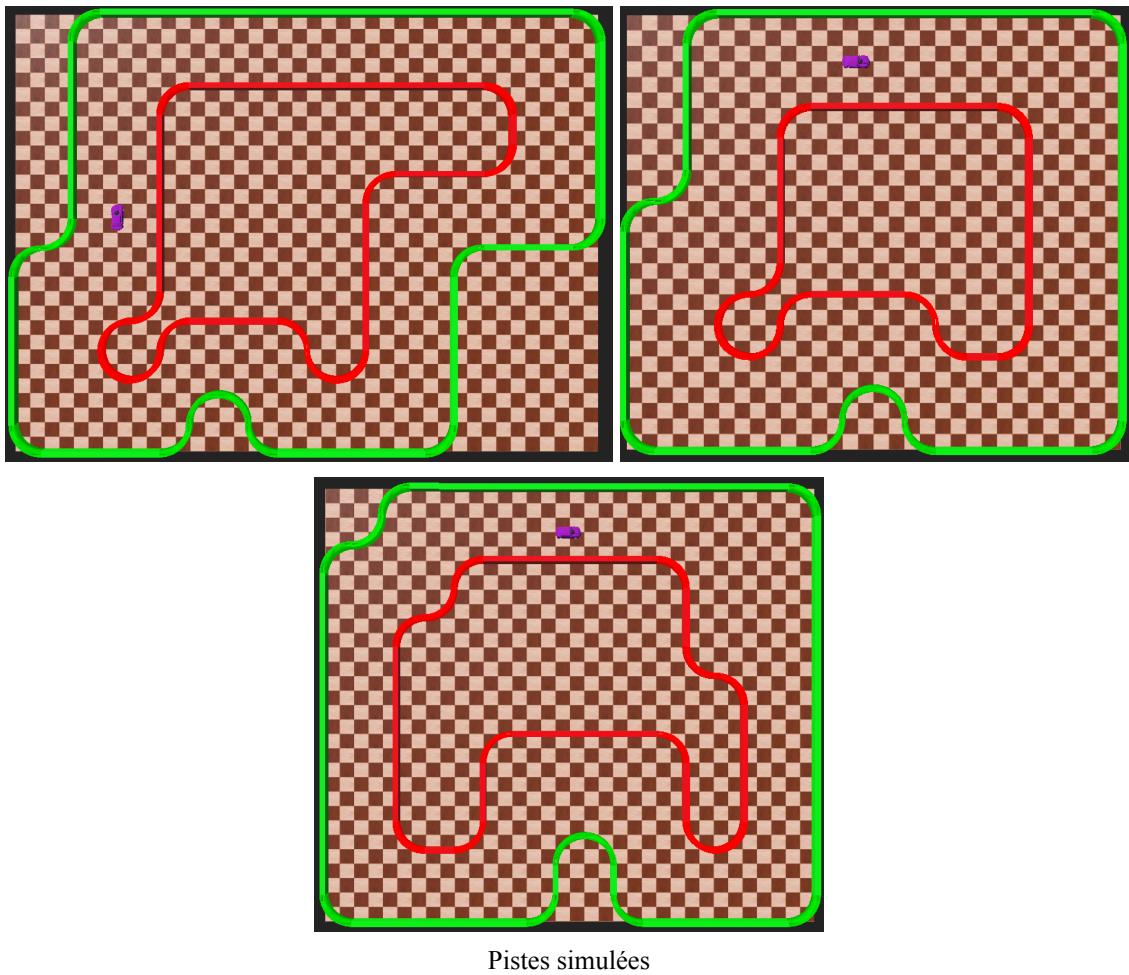
Situation exemple.

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 20° 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 :

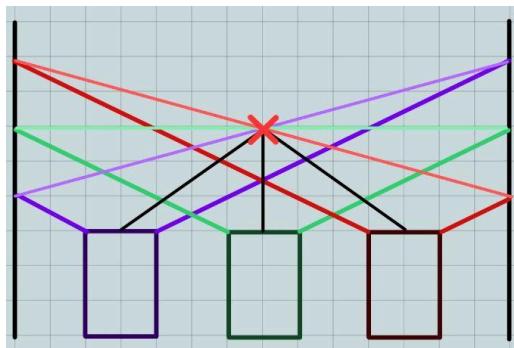




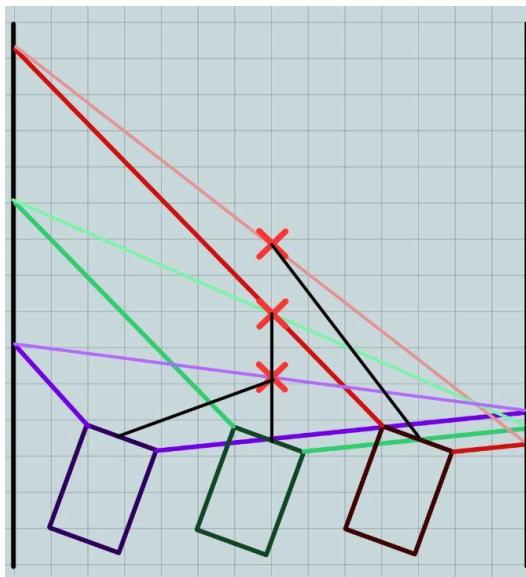
Positionnement de la voiture au milieu de la piste

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. À partir de ces points, on prend une moyenne sur un intervalle de quelques degrés pour éviter des mauvaises mesures (entre 65° et 70° , par exemple). 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.

CoVAPSY – Mise en œuvre du Simulateur Webots

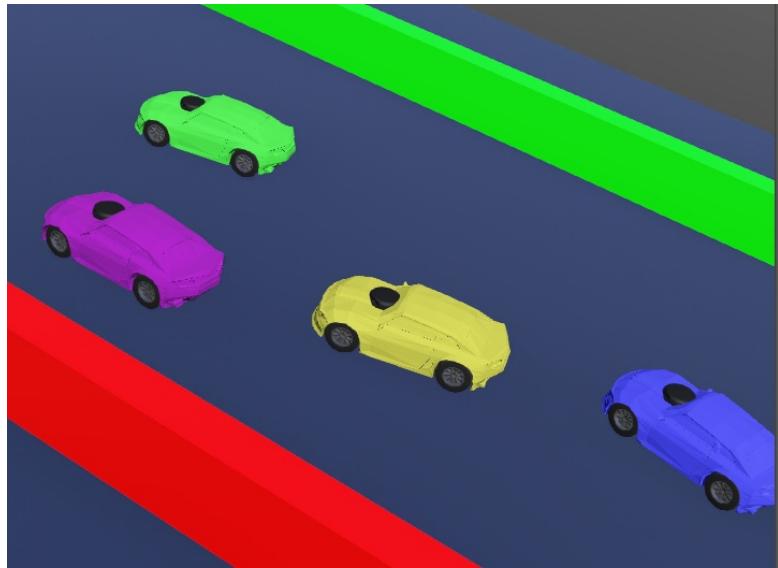


Figure 1: Course entre 4 voitures

Kévin Hoarau, kevin.hoarau@ens-paris-saclay.fr, élève en M2 FESup Intranet, ENS Paris Saclay.

Anthony Juton, anthony.juton@ens-paris-saclay.fr, professeur agrégé de physique appliquée, ENS Paris Saclay.

Pour travailler sur les algorithmes en robotique, en s'affranchissant des problèmes matériels, il est très intéressant d'utiliser un simulateur, avant de travailler sur le robot réel. C'est d'autant plus vrai avec l'apprentissage automatique qui demande des milliers d'essais auxquels le robot physique ne survivrait pas.

Dans ce cadre, pour la course de voitures autonomes de Paris Saclay, plusieurs équipes ont choisi le simulateur Webots et y ont développé un modèle de la voiture proche de la voiture 1/10^{ème} utilisée pour la course, notamment pour faire de l'apprentissage par renforcement ([voir article associé](#)). Webots est un simulateur open-source de robotique populaire dans la recherche et l'enseignement. Il utilise la bibliothèque ODE (Open Dynamics Engine) pour détecter des collisions et simuler la dynamique des corps rigides et des fluides. Il est bien documenté, multiplateforme (Linux, Windows, MacOS), utilisable sur un PC non doté d'un processeur graphique performant et permet la programmation en C, en java ou en python directement depuis le logiciel (le plus simple) ou à partir d'un environnement tiers (le plus efficace pour le debug).

L'objectif de cet article est de guider le lecteur vers une course de voitures 1/10ème simulées. La programmation peut se faire en python ou en C. L'article se limite à un algorithme très simple, les étudiants ayant en charge de travailler sur des algorithmes plus performants.

Le code issu du simulateur peut ensuite être réutilisé dans la voiture 1/10^{ème} réelle ([voir article associé](#)).

Prérequis : Les bases de la programmation Python (connaissance des classes non nécessaire) ou en C suivant le langage retenu : manipulation des tableaux, utilisation de fonctions.

1. Installation de Webots

La version utilisée pour cet article est la R2023b. Il est recommandé de travailler sur cette version, le passage d'une version à l'autre de webots demandant une bonne connaissance de l'outil.

La dernière version de webots se télécharge à l'adresse : <https://www.cyberbotics.com/>

Les versions antérieures se téléchargent à l'adresse : <https://github.com/cyberbotics/webots/releases>

Sous Linux, mieux vaut ne pas utiliser l'installation par *snap*, celle-ci ayant un lien plus complexe avec les autres logiciels (notamment l'IDE python ou C) du PC.

En plus du logiciel, pour travailler sur les voitures autonomes 1/10^{ème}, il faut télécharger le projet de base *Simulateur_CoVAPSY_Webots2023b_Base.zip* à l'adresse suivante :

https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay/blob/main/Simulateur_Simulateur_CoVAPSY_Webots2023b_Base.zip

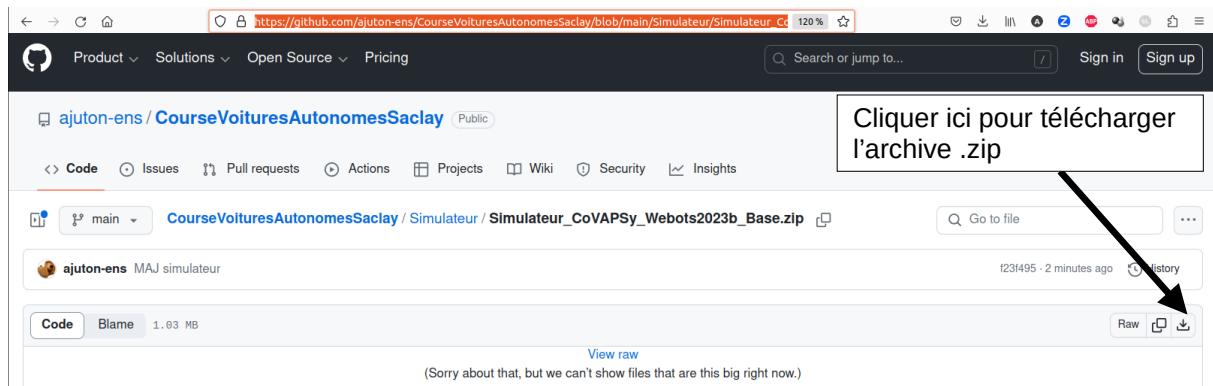


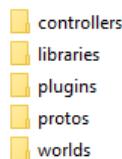
Figure 2: Téléchargement de l'archive contenant le projet webots de base

2. Prise en main de l'environnement Webots

Il est recommandé de suivre rapidement le tutorial <https://www.cyberbotics.com/doc/guide/getting-started-with-webots> pour prendre en main le logiciel, en particulier la section *The 3D window*.

2.1 Éléments d'un projet

Dans le dossier *Simulateur_CoVAPSY_Webots2023b_Base*, une fois décompressé, on trouve les éléments d'un projet webots.



Tous les projets Webots sont composés des dossiers montrés ci-dessus. Voici un détail de leur utilité :

- **controllers** : Dossier contenant les programmes qui contrôlent les robots présents dans le projet
- **libraries** : non utilisé ici

- **plugins** : non utilisé ici
- **protos** : Dossier contenant les fichiers des modèles des robots qui ne sont pas présents dans la base de données de Webots (dont la voiture TT-02)
- **worlds** : Dossier contenant tous les mondes créés pour le projet (la piste **Piste_2_voitures.wbt** notamment)

Lancer webots.

2.2 Interface Webots

Le logiciel se présente sous la forme de 4 panneaux :

- **L'arborescence des éléments** permet de configurer le monde (la piste ici) et les voitures. On peut y modifier les paramètres de la piste, y ajouter des voitures et y modifier leur couleur ou les paramètres du lidar.
- **L'environnement graphique** permet de visualiser l'évolution des voitures sur la piste et de modifier la position des voitures ou des éléments de la piste à la souris.
- **L'éditeur de texte** permet de modifier le code des contrôleurs des voitures, mais aussi des définitions des voitures (les PROTOs).
- **La console** affiche les avertissements et erreurs du logiciel (dont ceux liés à l'interprétation du code python ou à la compilation du code C) et les affichages (print en python, printf en C) lors de l'exécution du programme.

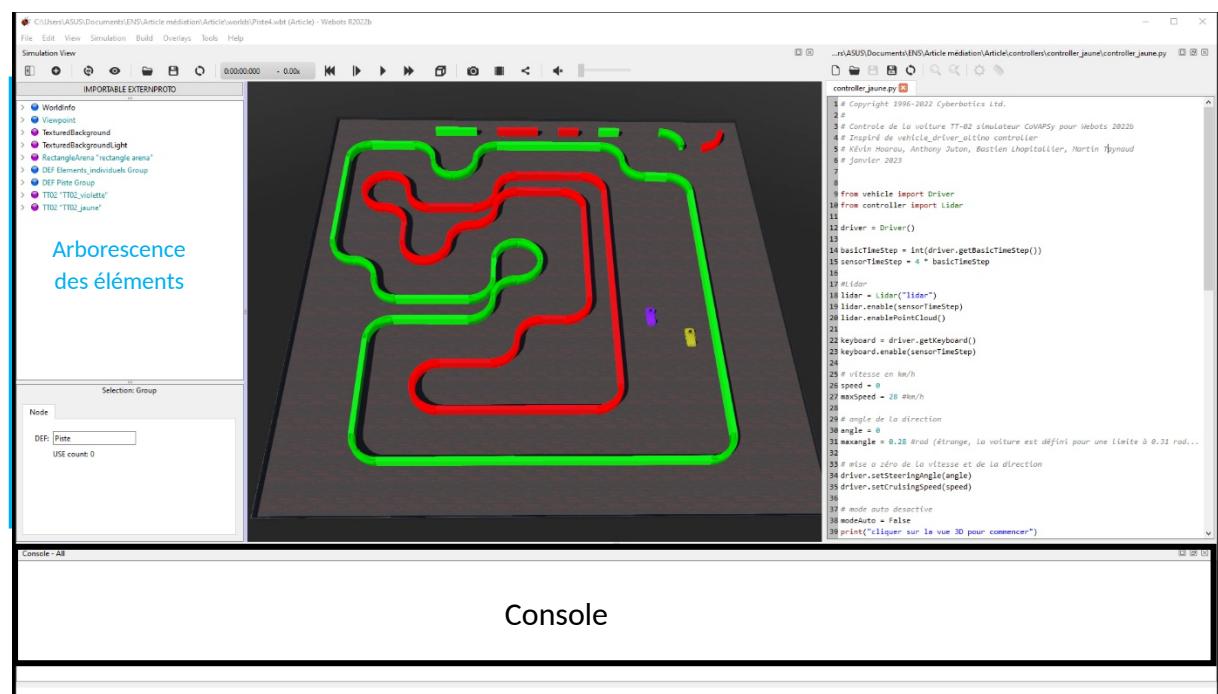


Figure 3: Les différents panneaux de Webots

- Aller dans l'onglet **File → Open World**
- Choisir le fichier **Piste_CoVAPSy_2023b.wbt** qui se trouve dans le dossier :
`/<chemin_vers_le_dossier>/Simulateur_CoVAPSy_Webots2023b_Base/worlds/`

2.3 Modification de la piste

Le World ouvert précédemment contient une piste déjà construite. Mais il est possible de modifier le tracé avec les blocs individuels présent en haut de la piste.



Dans l'arborescence, ces blocs se trouvent dans « **DEF Elements_individuals Group** » dans le dossier **children**. Il est possible de copier ces blocs et de les rajouter à la piste pour redéfinir le tracé.

Pour cela, il suffit de sélectionner le bloc voulu et de faire **Clic droit →Copy** (ou **Ctrl+C**). Il faut maintenant sélectionner dans l'arborescence « **DEF Piste Group** » dossier **children** pour coller le bloc avec **Clic droit + Paste** (ou **Ctrl+V**). Le bloc copié se trouve au même emplacement que le bloc d'origine.

Il y a 3 manières de déplacer un bloc :

- Sélectionner le bloc. Maintenir la touche **Shift**. Bouger la souris avec le **clic gauche** enfoncé pour translater le bloc et **clic droit** enfoncé pour la rotation
- Utiliser les flèches verte, rouge et bleu pour déplacer et faire tourner le bloc. Une métrique est disponible pour aider en précision
- Dans l'arborescence, dérouler le **Solid** correspondant.Modifier les champs **Translation** et **Rotation** pour modifier le positionnement du bloc. C'est la méthode la plus précise.

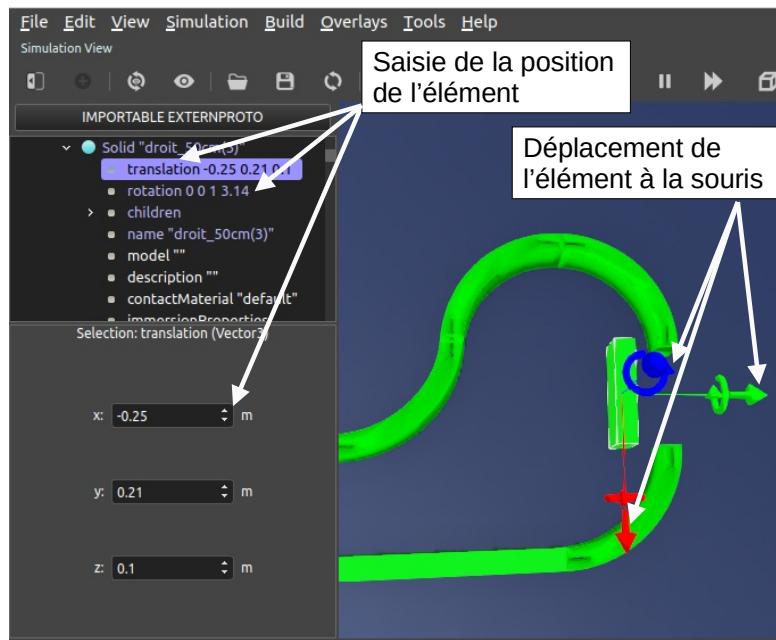


Figure 4: Modification de la piste

3. Programmation des voitures

Cette partie aborde le cœur du travail : la programmation des voitures dans le simulateur, d'abord en python, puis en C.

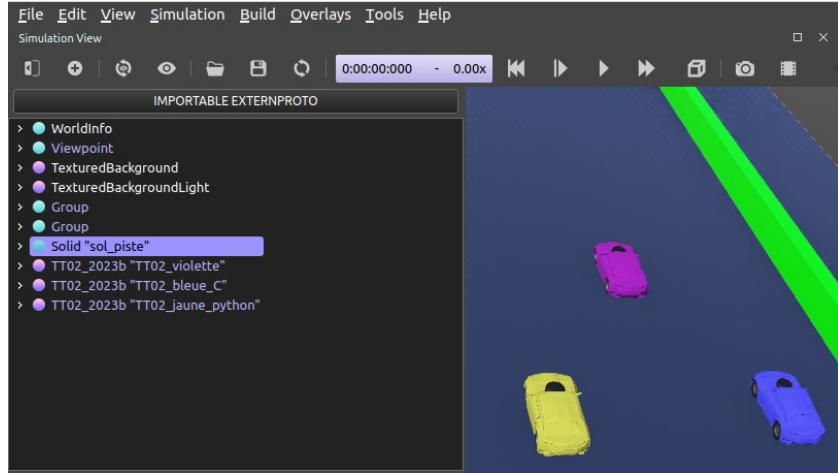


Figure 5: 3 voitures, pour la programmation en C et en python

La voiture violette est le *sparring partner*, elle n'est pas à programmer. La voiture jaune a une programme de base en python et la voiture bleue un programme de base en C (La vitesse est fixe, l'angle de braquage est déterminé avec les données captées par le Lidar au niveau des angles -60° et 60°). Il est facile de supprimer la voiture inutile dans l'arborescence des élément.

En C comme en python, 3 fonctions et un tableau sont utilisées pour contrôler la voiture :

- `set_vitesse_m_s()` : fonction qui permet de contrôler la voiture en indiquant sa vitesse en m/s
- `set_direction_degre()` :fonction qui permet de contrôler la voiture en indiquant sa direction en degré. Pour tourner à gauche, il faut indiquer un angle positif et un angle négatif pour tourner à droite.
- `recule()` : fonction qui permet à la voiture dans le simulateur de reculer à la demande.
- Les données du Lidar sont récupérées dans la variable `tableau_lidar_mm` (en python) ou `data_lidar_mm_main` (en C) qui est un tableau de 360 valeurs (1 par degrés) dans laquelle sont stockées les distances en millimètres. L'indice 0 correspond à l'avant de la voiture. Attention, les valeurs entre 100 et 260 ne sont pas significatives car elles correspondent à des angles auxquels le lidar est face à l'habitacle de la voiture.

Que ce soit en C ou en python, l'objectif est d'abord de dépasser la voiture violette (pas très compliqué), puis d'aller le plus vite possible. Pour cela, plusieurs pistes sont proposées ici :

- Il est possible de regrouper les rayons en secteur de 10° pour chercher le secteur dont le rayon le plus court est le plus long parmi les plus courts des autres secteurs.
- Il est possible d'adapter sa vitesse à la distance de l'obstacle devant.
- Il est intéressant de détecter un obstacle pour réussir à l'éviter. On peut pour cela ajouter des morceaux de bordure de piste au milieu de la piste.
- Il est possible de reculer quand on est dans un mur, en surveillant une valeur minimale des rayons du lidar à l'avant et sur les côtés. Pour déterminer les situations de quasi-collision, on peut se baser sur 3 valeurs du Lidar : la mesure à 0°, celle à -30° et celle à 30°. Si les distances captées par le Lidar sur ces angles spécifiques sont inférieures à un certain seuil, on considère que la voiture s'est crashée. On a alors 3 possibilités, qui demande un peu de travail car il n'est pas possible d'utiliser `time.sleep`, cette fonction bloquante mettant aussi en pause le moteur physique (l'utilisation de `time.time()` peut alors être intéressante) :

- Seuil franchi pour l'angle 0° (mur devant) : On replace les roues pour aller droit puis on recule jusqu'à ce que la valeur de lidar franchisse un seuil ou pendant 0,5s.
- Seuil franchi pour l'angle 30° (mur à gauche) : On tourne complètement à gauche puis on recule jusqu'à ce que la valeur de lidar franchisse un seuil ou pendant 0,5s.
- Seuil franchi pour l'angle -30° (mur à droite) : On tourne complètement à droite puis on recule jusqu'à ce que la valeur de lidar franchisse un seuil ou pendant 0,5s.
- Des méthodes avancées sont bien évidemment possible, en sortant du cadre du lycée, avec des trajectoires en forme de tentacules (<https://doi.org/10.1002/rob.20256>), ou avec de l'apprentissage par renforcement (ajouter lien vers article associé).

Aucune connaissance sur la bibliothèque du simulateur n'est requise. Webots supporte aussi le java. Pour aller plus loin, il est possible de se référer à la documentation de Webots :(<https://www.cyberbotics.com/doc/guide/tutorials>)

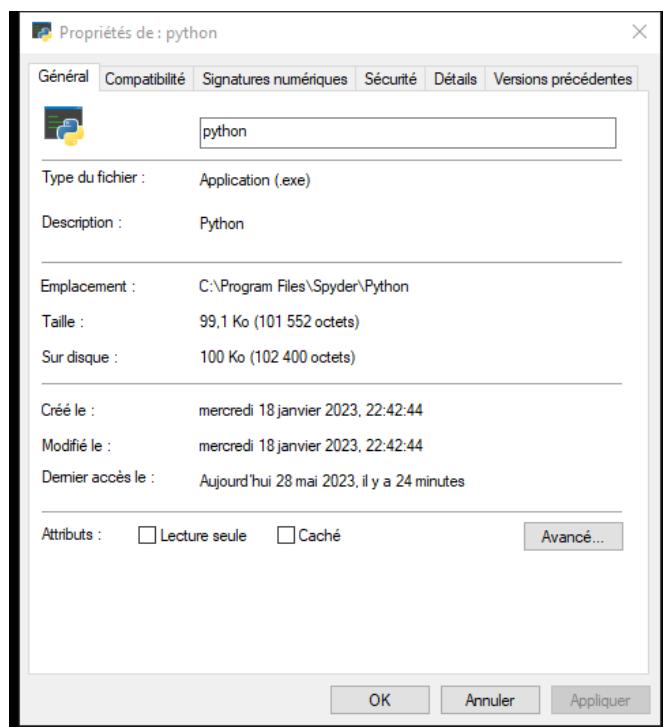
3.1 Programmation en python

1) Configuration

Linux ne demande pas de configuration particulière. Passer directement à « [Modifier un programme dans l'éditeur de texte du simulateur](#) »

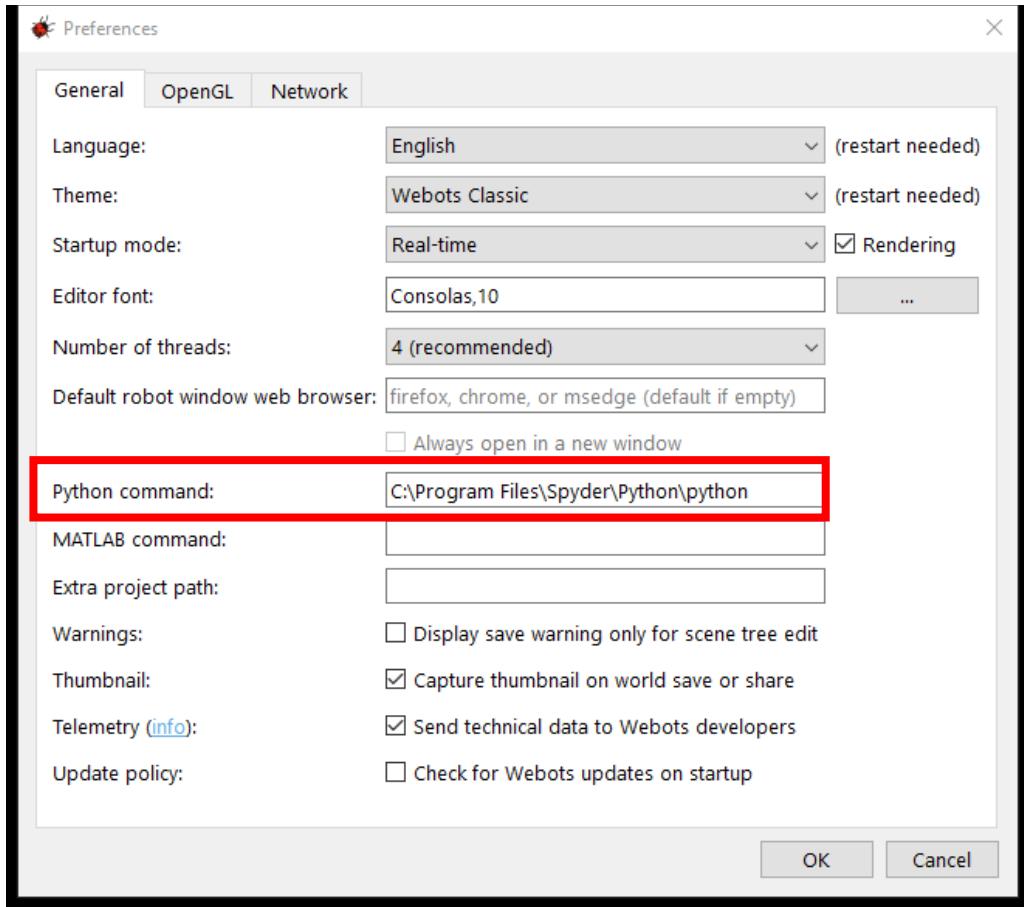
Sous windows, il faut indiquer l'interpréteur Python qui sera utilisé par le logiciel pour lancer les différents programmes.

- Chercher le chemin d'accès vers l'exécutable `python.exe`



- Dans Webots, cliquer sur **Tools→Préférences**

- Copier le Chemin d'accès de l'exécutable dans la case **Python Command**



2) Modifier un programme dans l'éditeur de texte du simulateur

Webots permet de modifier les programmes Python directement.

- Sélectionner une des voitures dans l'arborescence et dérouler son arbre
- Sélectionner la case **controller** puis plus bas **Edit**

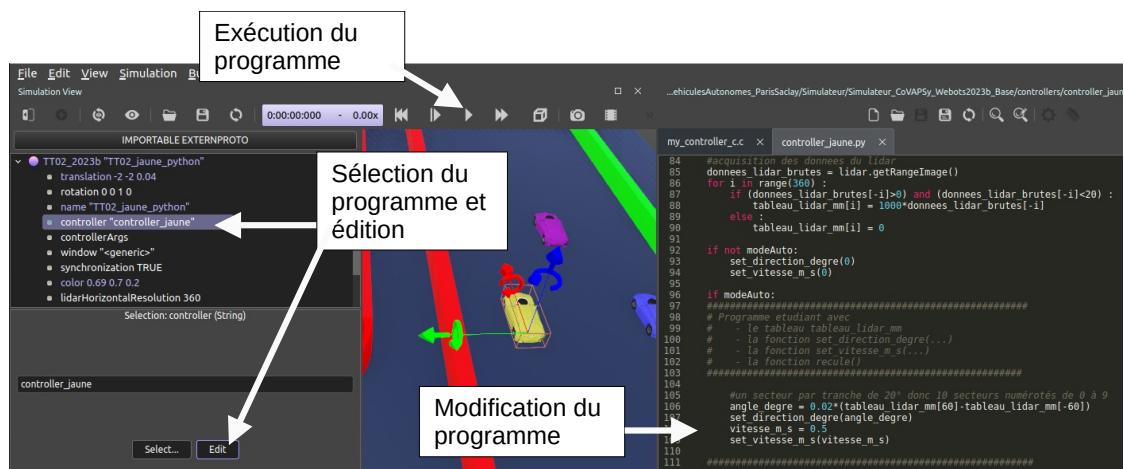


Figure 6: Modification du programme controller de la voiture jaune.

- Le code de la voiture s'affiche dans l'éditeur de texte. Le programme python de base est donné en annexe. Il faut cliquer sur a dans la fenêtre de visualisation graphique pour lancer le mode automatique, une fois le programme en exécution.

Il est possible qu'il faille supprimer la voiture bleue programmée en C sous windows (le code devant être compilé pour windows, comme expliqué dans la partie suivante).

- Attention, webots n'enregistre pas automatiquement le code python avant l'exécution. Il faut donc enregistrer puis lancer l'exécution.

3.2 Programmation en langage C

De la même manière, il est possible de programmer la voiture bleue en C. Il faut juste penser à enregistrer et compiler avant de lancer l'exécution. Le programme C de base est donné en annexe.

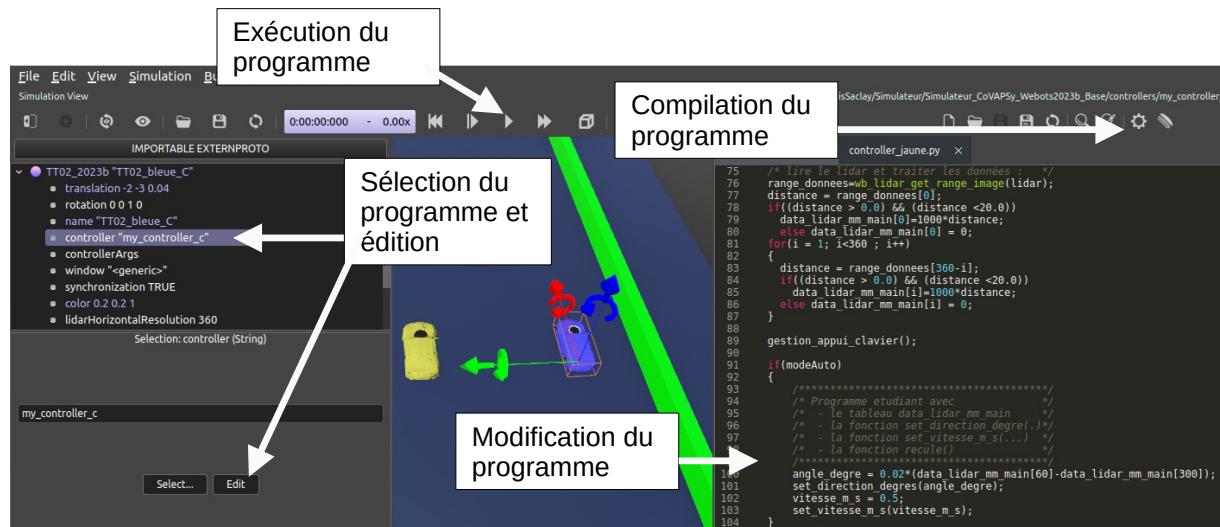


Figure 7: Modification du programme my_controller_c.c de la voiture bleue.

3.3 Programmation avec un environnement de développement tiers (utilisation avancée)

Pour programmer de manière complète, il est recommandé d'utiliser un environnement de développement (IDE) dédié comme *Spyder* ou *VSCode* par exemple pour python ou *Eclipse* pour le C, IDE qui permettent notamment l'usage du débogueur. Ceci ajoute cependant une couche de complexité. 2 méthodes sont possibles, une seule est exposée ici. Plus détails sont donnés dans les 2 sections correspondantes de la documentation de Webots :

- <https://cyberbotics.com/doc/guide/using-your-ide>
 - <https://cyberbotics.com/doc/guide/running-extern-robot-controllers>

L'utilisation d'un contrôleur externe, celle retenue ici, permet d'avoir un contrôleur tournant sur un IDE complet (avec débogueur), y compris sur une machine distante, ce qui peut être intéressant pour une course à plusieurs voitures, chaque étudiant se connectant sur une des voitures de la piste, son tour venu.

Pour pouvoir faire le lien entre ces IDE et Webots, il est nécessaire de rajouter 2 lignes de codes en début du programme.

- **Dans le programme python Spyder ou VSCode :** Rajouter les 2 lignes de codes suivantes indiquant le chemin vers les bibliothèques webots, en adaptant le nom du dossier au PC et à la version de python utilisée.

```
import sys
```

Pour Windows :

```
sys.path.append('C:/Program Files/Webots/lib/controller/python38/')
```

Pour Linux :

```
sys.path.append('/usr/local/webots/lib/controller/python38/')
```

Il faut également créer la variable d'environnement indiquant le dossier d'installation de webots :

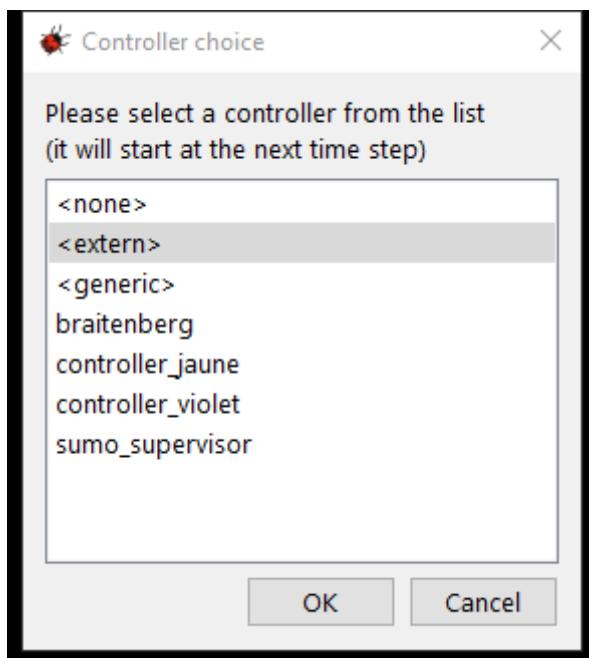
```
export WEBOTS_HOME=/usr/local/webots/
```

- **Dans Webots:**

- Sélectionner une voiture et dérouler l'arborescence

- Sélectionner **controller** puis **Select**

- Choisir **<extern>**



Il suffit par la suite de lancer la simulation puis d'exécuter le programme sur *Spyder* ou *VSCode*.

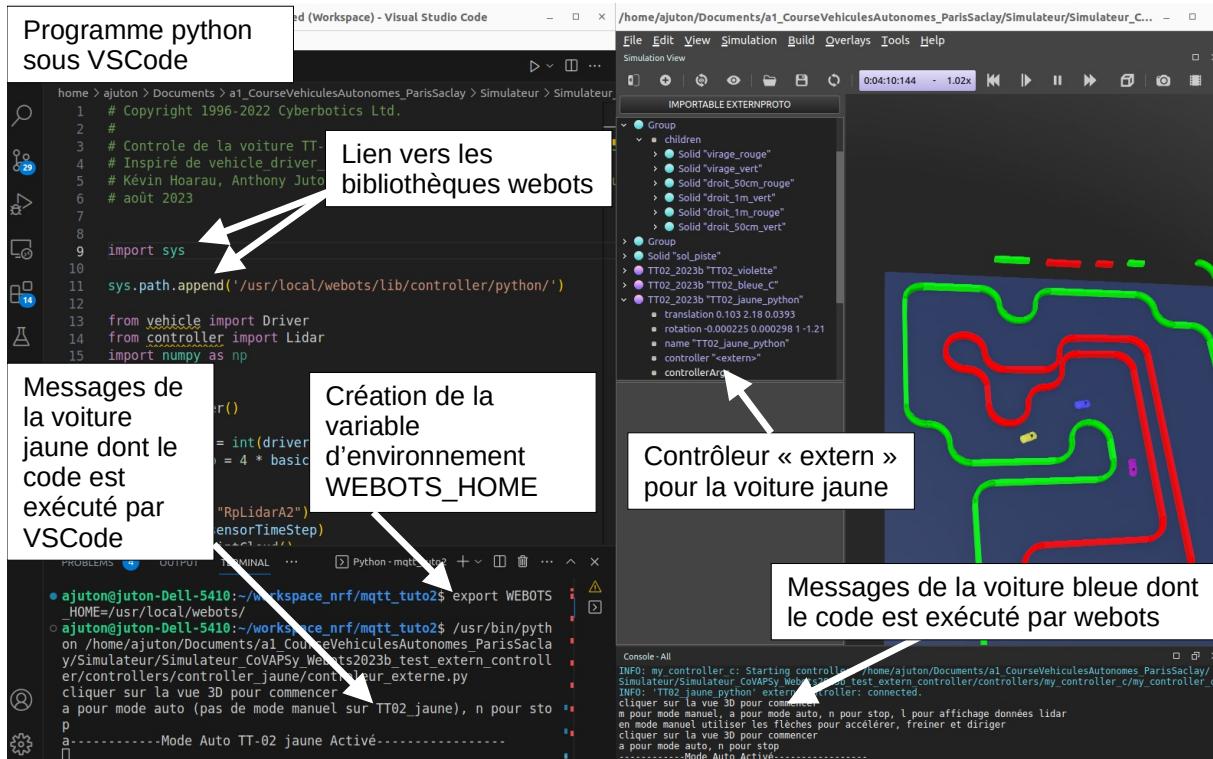


Figure 8: voiture jaune simulée sous webots avec un contrôleur exécuté sous VSCode

Pour exécuter du code depuis des PCs extérieurs, il faut utiliser l'exécutable webots, comme expliqué sur la documentation webots.

```
$WEBOTS_HOME/webots-controller --protocol=tcp --ip-address=X.X.X.X
$WEBOTS_HOME/projects/robots/softbank/nao/controllers/nao_demo/nao_demo
```

4. Course entre 4 voitures

Pour une course entre plusieurs voitures, il suffit d'ajouter d'autres voitures, de changer leur couleur et d'ajouter les contrôleurs.

4.1 Ajout du contrôleur

Les contrôleurs (hors contrôleurs externes) étant situé obligatoirement dans le dossier `controllers` du projet, il est nécessaire d'ajouter le nouveau contrôleur à cet emplacement, soit en copiant un dossier contrôleur envoyé par un étudiant, soit en copiant/collant un dossier contrôleur existant. Le nom du fichier doit correspondre au nom du dossier.

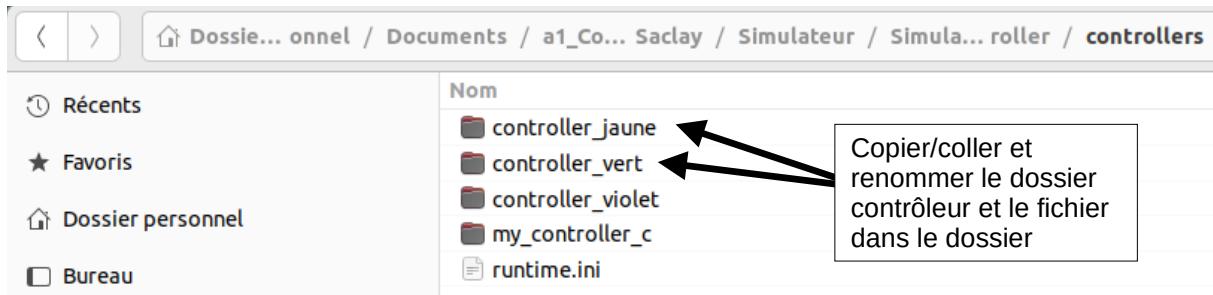


Figure 9: Ajout d'un contrôleur au projet pour une course à 4 voitures

4.2 Ajout de la voiture

Une fois le contrôleur ajouté, il est possible d'ajouter la voiture et de modifier ses paramètres.

Attention, pour faire des modifications pérennes, il est préférable de se placer à t=0 (bouton « retour arrière » à gauche du bouton exécuter) et d'enregistrer ensuite les modifications avant de lancer l'exécution.

- Se placer à t=0
- Sélectionner la voiture jaune dans l'arborescence
- Faire **Clic droit**→**Copy** puis **Clic droit**→**Paste**

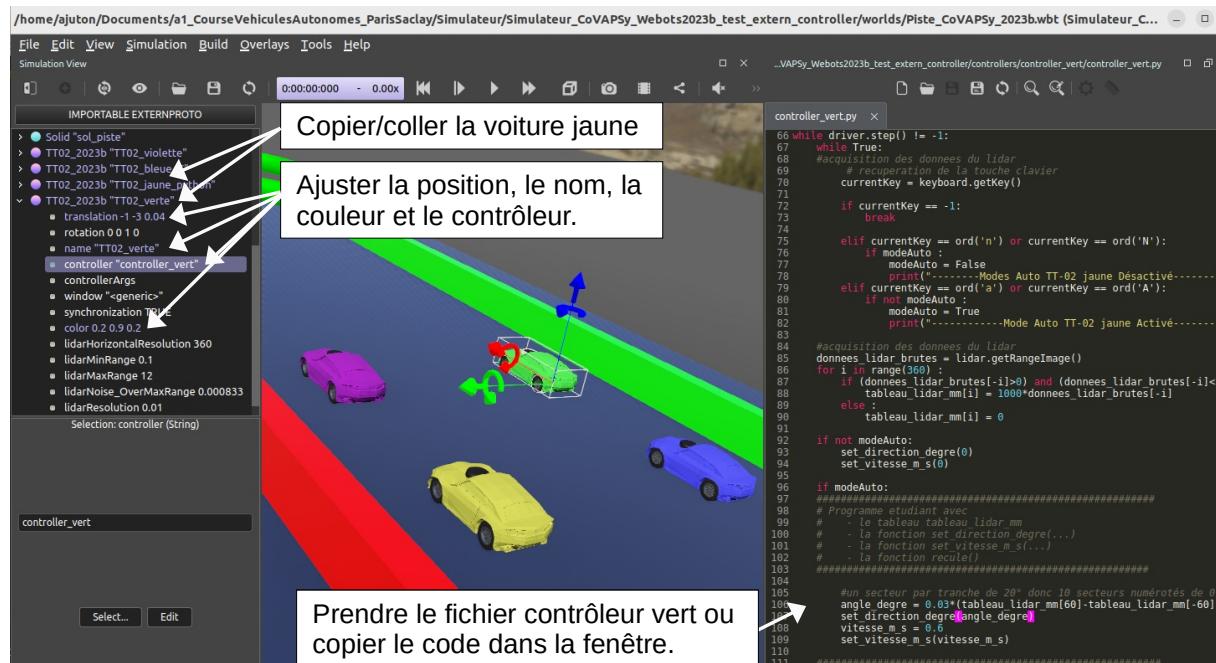


Figure 10: Ajout d'une voiture au projet

Attention, penser à sauvegarder la nouvelle configuration de départ : *File > Save World*

Ouvertures

Cet article amène à réaliser et tester des algorithmes permettant de faire concourir une ou plusieurs voitures de manière autonome sur le simulateur.

Côté informatique, la visualisation dynamique des données du lidar (graph en coordonnées cylindriques) et des actions de la voiture et leur archivage, comme la télémétrie pour les voitures réelles, peut être un axe de travail intéressant, dans l'objectif d'améliorer les performances de la voiture.

Sur Webots, l'ajout d'un robot « superviseur », ayant l'accès aux données des autres robots permet de faire un chronométrage individuel de chaque voiture :

<https://www.cyberbotics.com/doc/reference/supervisor#>!

Le travail suivant peut consister à transférer cet algorithme sur la voiture réelle, disposant des mêmes fonctions de contrôle de vitesse et de direction et du même tableau d'acquisition des données lidar. Le transfert de la simulation à la réalité est alors intéressant pour mettre en évidence les limites du simulateur (prise en compte imparfaite de la dynamique de la voiture, de sa direction notamment) et des contraintes de la réalité (nécessité de prendre en compte la tension batterie ou de faire un asservissement de vitesse, absence de mesure renvoyée par le lidar en cas de mauvaise réflexion...).

Annexe

Programme python (contrôleur de la voiture jaune)

```
# Copyright 1996-2022 Cyberbotics Ltd.
#
# Controle de la voiture TT-02 simulateur CoVAPSY pour Webots 2023b
# Inspiré de vehicle_driver_altino controller
# Kévin Hoarau, Anthony Juton, Bastien Lhopitallier, Martin Raynaud
# août 2023

from vehicle import Driver
from controller import Lidar
import time

driver = Driver()

basicTimeStep = int(driver.getBasicTimeStep())
sensorTimeStep = 4 * basicTimeStep

#Lidar
lidar = Lidar("RpLidarA2")
lidar.enable(sensorTimeStep)
lidar.enablePointCloud()

#clavier
keyboard = driver.getKeyboard()
keyboard.enable(sensorTimeStep)

# vitesse en km/h
speed = 0
maxSpeed = 28 #km/h

# angle de la direction
angle = 0
maxangle_degre = 16

# mise a zéro de la vitesse et de la direction
```

```

driver.setSteeringAngle(angle)
driver.setCruisingSpeed(speed)

tableau_lidar_mm=[0]*360

def set_vitesse_m_s(vitesse_m_s):
    speed = vitesse_m_s*3.6
    if speed > maxSpeed :
        speed = maxSpeed
    if speed < 0 :
        speed = 0
    driver.setCruisingSpeed(speed)

def set_direction_degre(angle_degre):
    if angle_degre > maxangle_degre:
        angle_degre = maxangle_degre
    elif angle_degre < -maxangle_degre:
        angle_degre = -maxangle_degre
    angle = -angle_degre * 3.14/180
    driver.setSteeringAngle(angle)

def recule(): #sur la voiture réelle, il y a un stop puis un recul pendant 1s.
    driver.setCruisingSpeed(-1)

# mode auto desactive
modeAuto = False
print("cliquer sur la vue 3D pour commencer")
print("a pour mode auto (pas de mode manuel sur TT02_jaune), n pour stop")

while driver.step() != -1:
    while True:
        #acquisition des donnees du lidar
        # recuperation de la touche clavier
        currentKey = keyboard.getKey()

        if currentKey == -1:
            break

        elif currentKey == ord('n') or currentKey == ord('N'):
            if modeAuto :
                modeAuto = False
                print("-----Modes Auto TT-02 jaune Désactivé-----")
        elif currentKey == ord('a') or currentKey == ord('A'):
            if not modeAuto :
                modeAuto = True
                print("-----Mode Auto TT-02 jaune Activé-----")

        #acquisition des donnees du lidar
        donnees_lidar_brutes = lidar.getRangeImage()
        for i in range(360) :
            if (donnees_lidar_brutes[-i]>0) and (donnees_lidar_brutes[-i]<20) :
                tableau_lidar_mm[i] = 1000*donnees_lidar_brutes[-i]
            else :
                tableau_lidar_mm[i] = 0

        if not modeAuto:
            set_direction_degre(0)
            set_vitesse_m_s(0)

```

```

if modeAuto:
#####
# Programme etudiant avec
#   - le tableau tableau_lidar_mm
#   - la fonction set_direction_degre(...)
#   - la fonction set_vitesse_m_s(...)
#   - la fonction recule()
#####

#un secteur par tranche de 20° donc 10 secteurs numérotés de 0 à 9
angle_degre = 0.02*(tableau_lidar_mm[60]-tableau_lidar_mm[-60])
set_direction_degre(angle_degre)
vitesse_m_s = 0.5
set_vitesse_m_s(vitesse_m_s)

#####

```

Programme C (contrôleur de la voiture bleue)

```

/*
 * File:           my_controller_c.c
 * Date:          23 mai 2023
 * Description:
 * Author:        Bruno Larnaudie, Anthony Juton
 * Modifications: 24 août 2023
 */

/*
 * You may need to add include files like <webots/distance_sensor.h> or
 * <webots/motor.h>, etc.
 */
#include <math.h>
#include <webots/robot.h>
#include <webots/vehicle/car.h>
#include <webots/vehicle/driver.h>
#include <webots/keyboard.h>
#include <stdio.h>
#include <webots/lidar.h>
/*
 * You may want to add macros here.
 */
#define TIME_STEP 32
#define SIZE_TABLEAU 200
#define MAX_SPEED 6.28 // Vitesse maximale des moteurs

const float* range_donnees;
//float range_donnees[SIZE_TABLEAU];
unsigned char gestion_appuie_clavier(void);
unsigned char modeAuto=0;

// prototype des fonctions
void affichage_consigne();
void set_direction_degres(float angle_degre);
void set_vitesse_m_s(float vitesse_m_s);
unsigned char gestion_appui_clavier(void);
void recule(void);

```

```

//vitesse en km/h
float speed = 0;
float maxSpeed = 28; //km/h

// angle max de la direction
float maxangle_degre = 16;

/* main loop
 * Perform simulation steps of TIME_STEP milliseconds
 * and leave the loop when the simulation is over
 */

int main(int argc, char **argv)
{
    unsigned int i;
    signed int data_lidar_mm_main[360];
    float angle_degre, vitesse_m_s;
    /* necessary to initialize webots stuff */
    //initialisation du conducteur de voiture
    wbu_driver_init();
    //enable keyboard
    wb_keyboard_enable(TIME_STEP);
    // enable lidar
    WbDeviceTag lidar = wb_robot_get_device("RpLidarA2");
    wb_lidar_enable(lidar,TIME_STEP);
    // affichage des points lidar sur la piste
    wb_lidar_enable_point_cloud(lidar);

    affichage_consigne();
    set_direction_degres(0);
    set_vitesse_m_s(0);
    while (wbu_driver_step() != -1)
    {
        float distance;
        /* lire le lidar et traiter les données : */
        range_donnees=wb_lidar_get_range_image(lidar);
        distance = range_donnees[0];
        if((distance > 0.0) && (distance <20.0))
            data_lidar_mm_main[0]=1000*distance;
        else data_lidar_mm_main[0] = 0;
        for(i = 1; i<360 ; i++)
        {
            distance = range_donnees[360-i];
            if((distance > 0.0) && (distance <20.0))
                data_lidar_mm_main[i]=1000*distance;
            else data_lidar_mm_main[i] = 0;
        }
        gestion_appui_clavier();
        if(modeAuto)
        {
            ///////////////////////////////
            /* Programme etudiant avec */
            /* - le tableau data_lidar_mm_main */
            /* - la fonction set_direction_degre(.)*/
            /* - la fonction set_vitesse_m_s(...) */
            /* - la fonction recule() */
            ///////////////////////////////
            angle_degre = 0.02*(data_lidar_mm_main[60]-
data_lidar_mm_main[300]); //distance à 60° - distance à -60°
            set_direction_degres(angle_degre);
            vitesse_m_s = 0.5;
        }
    }
}

```

```

        set_vitesse_m_s(vitesse_m_s);
    }
}
/* This is necessary to cleanup webots resources */
wbu_driver_cleanup();
return 0;
}

unsigned char gestion_appui_clavier(void)
{
    int key;
    key=wb_keyboard_get_key();
    switch(key)
    {
        case -1:
            break;

        case 'n':
        case 'N':
            if (modeAuto)
            {
                modeAuto = 0;
                printf("-----Mode Auto Désactivé-----");
            }
            break;

        case 'a':
        case 'A':
            if (!modeAuto)
            {
                modeAuto = 1;
                printf("-----Mode Auto Activé-----");
            }
            break;

        default:
            break;
    }
    return key;
}

void affichage_consigne()
{
    printf("cliquer sur la vue 3D pour commencer\n");
    printf("a pour mode auto, n pour stop\n");
}

void set_direction_degres(float angle_degre)
{
    float angle=0;
    if(angle_degre > maxangle_degre)
        angle_degre = maxangle_degre;
    else if(angle_degre < -maxangle_degre)
        angle_degre = -maxangle_degre;
    angle = -angle_degre * 3.14/180;
    wbu_driver_set_steering_angle(angle);
}

void set_vitesse_m_s(float vitesse_m_s){
    float speed;
    speed = vitesse_m_s*3.6;
}

```

```
if(speed > maxSpeed)
    speed = maxSpeed;
if(speed < 0)
    speed = 0;
wbu_driver_set_cruising_speed(speed);
}

void recule(void){
    wbu_driver_set_cruising_speed(-1);
}
```

RPLIDAR A2

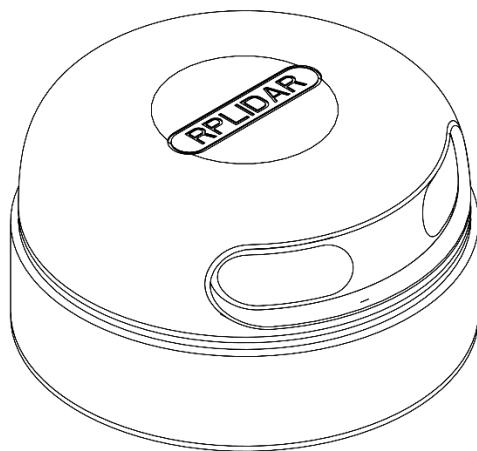
Low Cost 360 Degree Laser Range Scanner

Introduction and Datasheet

Model: A2M12

OPTMA

16K



CONTENTS	1
INTRODUCTION	3
SYSTEM CONNECTION	3
MECHANISM.....	4
SAFETY AND SCOPE.....	6
DATA OUTPUT.....	7
HIGH SPEED SAMPLING PROTOCOL AND COMPATIBILITY	7
APPLICATION SCENARIOS.....	8
SPECIFICATION	9
MEASUREMENT PERFORMANCE	9
LASER POWER SPECIFICATION.....	9
OPTICAL WINDOW.....	10
COORDINATE SYSTEM DEFINITION OF SCANNING DATA	10
COMMUNICATION INTERFACE.....	11
MISC.....	14
SELF-PROTECTION AND STATUS DETECTION	15
SDK AND SUPPORT	16
MECHANICAL DIMENSIONS	17
REVISION HISTORY	18
APPENDIX	19
IMAGE AND TABLE INDEX	19



The RPLIDAR A2M12 is the next generation low cost 360 degree 2D laser scanner (LIDAR) solution developed by SLAMTEC. It can take up to 16000 samples of laser ranging per second with high rotation speed. And equipped with SLAMTEC patented OPTMAG technology, it breakouts the life limitation of traditional LIDAR system so as to work stably for a long time.

The system can perform 2D 360-degree scan within a 12-meter range. The generated 2D point cloud data can be used in mapping, localization and object/environment modeling.

The typical scanning frequency of RPLIDAR A2M12 is 10Hz(600rpm), and the frequency can be freely adjusted within the 5-15Hz range according to the specific requirements. With the 10Hz scanning frequency, the sampling rate is 16kHz and the angular resolution is 0.225°.

Due to the improvements in SLAMTEC hardware operating performance and related algorithm, RPLIDAR A2M12 works well in all kinds of indoor environment and outdoor environment without direct sunlight. Meanwhile, before leaving the factory, every RPLIDAR A2M12 has passed the strict testing to ensure the laser output power meet the eye-safety standard of IEC-60825 Class 1.

System connection

The RPLIDAR A2M12 consists of a range scanner core and the mechanical powering part which makes the core rotate at a high speed. When it functions

normally, the scanner will rotate and scan clockwise. And users can get the range scan data via the communication interface of the RPLIDAR and control the start, stop and rotating speed of the rotate motor via PWM.

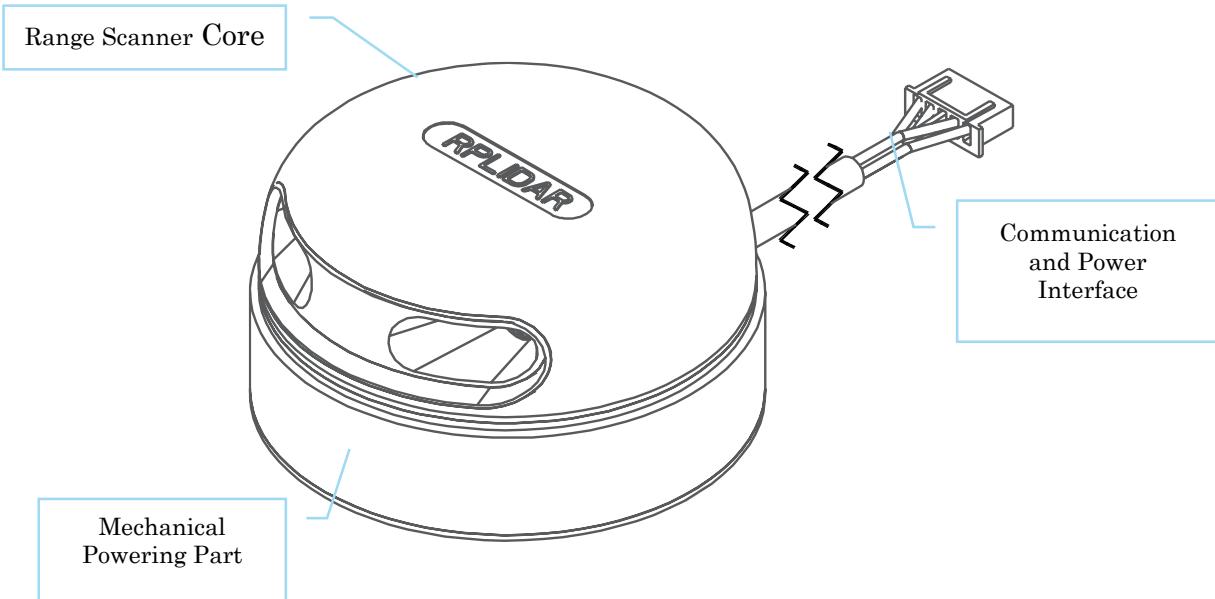


Figure 1-1 RPLIDAR System Composition

The RPLIDAR A2M12 comes with a rotation speed detection and adaptive system. The system will adjust the angular resolution automatically according to the actual rotating speed. And there is no need to provide complicated power system for RPLIDAR. In this way, the simple power supply schema saves the BOM cost. If the actual speed of the RPLIDAR is required, the host system can get the related data via communication interface.

The detailed specification about power and communication interface can be found in the following sections.

Mechanism

The RPLIDAR A2M12 is based on laser triangulation ranging principle and adopts the high-speed vision acquisition and processing hardware developed by SLAMTEC. The system ranges more than 16000 times per second.

During every ranging process, the RPLIDAR emits modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is then sampled by vision acquisition system in RPLIDAR and the DSP

embedded in RPLIDAR starts processing the sample data and outputs distance value and angle value between object and RPLIDAR via communication interface.

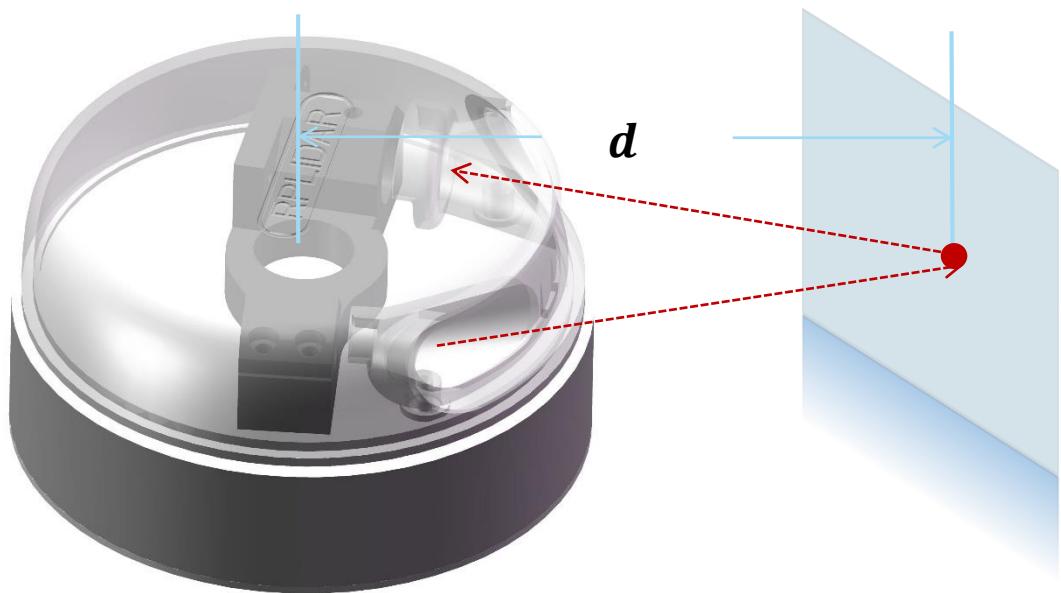
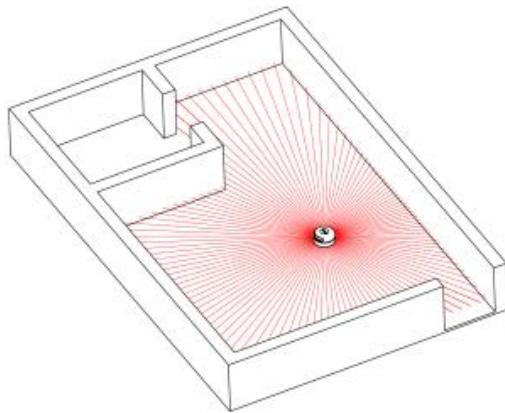


Figure 1-2 The RPLIDAR Working Schematic

When driven by the motor system, the range scanner core will rotate clockwise and perform the 360-degree scan for the current environment.



*Note : The LIDAR scan image is not directly relative to the environment showed here. Illustrative purpose only.

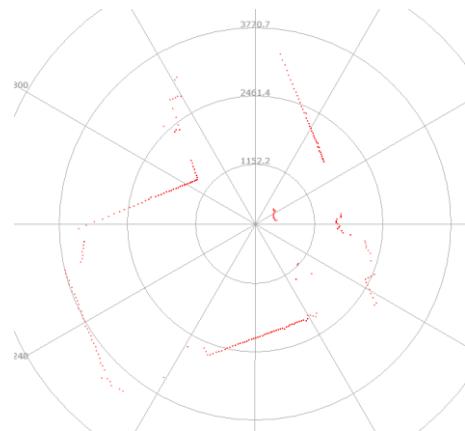


Figure 1-3 The Obtained Environment Map from RPLIDAR Scanning

Safety and Scope



Class I

The RPLIDAR A2M12 system uses a low power infrared laser as its light source, and drives it by using modulated pulse. The laser emits light in a very short time frame which can ensure its safety to human and pet, and it reaches Class I laser safety standard. Complies with 21 CFR 1040.10 and 1040.11 except for deviations pursuant to Laser Notice No. 50, dated June 24, 2007.

Caution: Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.

The modulated laser can effectively avoid the interference from ambient light and sunlight during ranging scanning process, which makes RPLIDAR work excellent in all kinds of indoor environment and outdoor environment without direct sunlight.

Data Output

During the working process, the RPLIDAR will output the sampling data via the communication interface. And each sample point data contains the information in the following table. If you need detailed data format and communication protocol, please contact SLAMTEC.

Data Type	Unit	Description
Distance	mm	Current measured distance value between the rotating core of the RPLIDAR and the sampling point
Heading	degree	Current heading angle of the measurement
Start Flag	(Bool)	Flag of a new scan
Checksum		The Checksum of RPLIDAR return data

Figure 1-4 The RPLIDAR Sample Point Data Information

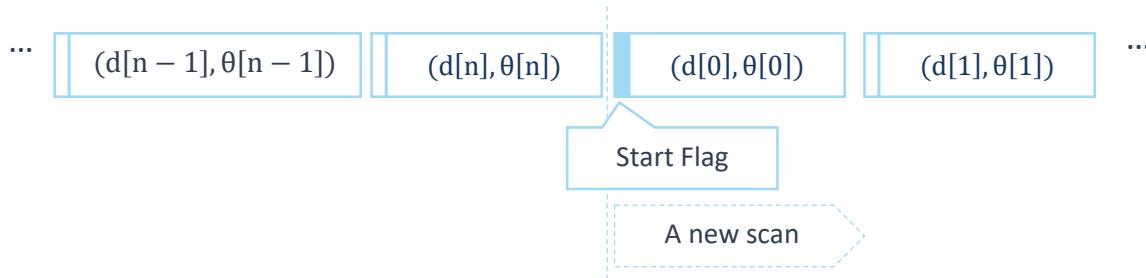


Figure 1-5 The RPLIDAR Sample Point Data Frames

The RPLIDAR outputs sampling data continuously and it contains the sample point data frames in the above figure. Host systems can configure output format and stop RPLIDAR by sending stop command. For detailed operations please contact SLAMTEC.

High Speed Sampling Protocol and Compatibility

The RPLIDAR A2M12 adopts the newly extended high speed sampling protocol for outputting the 16000 times per second laser range scan data. Users are

required to update the matched SDK or modify the original driver and use the new protocol for the 16000 times per second mode of RPLIDAR A2M12. Please check the related protocol documents for details.

RPLIDAR A2M12 is compatible with all communication protocols used by previous RPLIDAR products. Users can directly replace the previous model of RPLIDAR and use it directly on the original system.

Application Scenarios

The RPLIDAR can be used in the following application scenarios:

General robot navigation and localization

Environment scanning and 3D re-modeling

Service robot or industrial robot working for long hours

Home service /cleaning robot navigation and localization

General simultaneous localization and mapping (SLAM)

Smart toy's localization and obstacle avoidance

Measurement Performance

- For Model A2M12 Only

Item	Enhanced Mode
Application Scenarios	Extreme performance Ideal for indoor environments with maximum ranging distance and sampling frequency.
Operating Range	White object: 12 meters Black object: 10 meters
Minimum Operating ranging	0.2m
Sample Rate	16 kHz
Scan Rate	Typical value: 10 Hz (adjustable between 5 Hz-15 Hz)
Angular Resolution	0.225°
Scan Field Flatness	±1.5
Communication Interface	TTL UART
Communication Speed	256000 bps
Compatibility	Support former SDK protocols

Figure 2-1 RPLIDAR Performance

Note: the triangulation range system resolution changes along with distance.

Laser Power Specification

- For Model A2M12 Only

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	-	10	12	Peak power
Pulse length	Microsecond(us)	60	87	90	-
Laser Safety Class	-	-	IEC-60825 Class 1	-	-

Figure 2-2 RPLIDAR Optical Specification

Note: the laser power listed above is the peak power and the actual average power is much lower than the value.

Optical Window

To make the RPLIDAR A2M12 working normally, please ensure proper space to be left for its emitting and receiving laser lights when designing the host system. The obscuring of the host system for the ranging window will impact the performance and resolution of RPLIDAR A2M12. If you need cover the RPLIDAR A2M12 with translucent materials or have other special needs, please contact SLAMTEC about the feasibility.

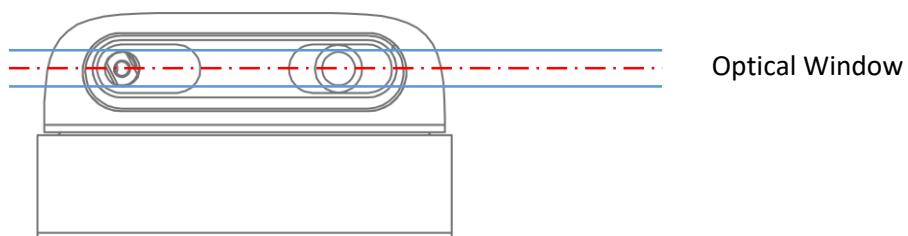


Figure 2-3 RPLIDAR Optical Window

You can check the Mechanical Dimensions chapter for detailed window dimensions.

Coordinate System Definition of Scanning Data

The RPLIDAR A2M12 adopts coordinate system of the left hand. The dead ahead of the sensors is the x axis of the coordinate system; the origin is the rotating center of the range scanner core. The rotation angle increases as rotating clockwise. The detailed definition is shown in the following figure:

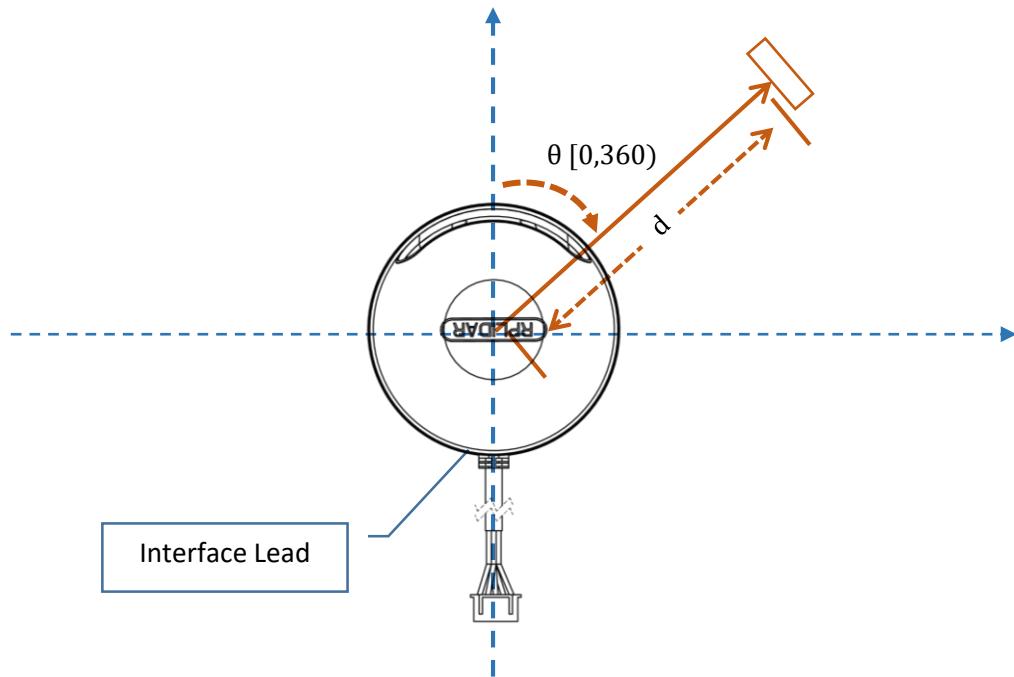


Figure 2-4 RPLIDAR Scanning Data Coordinate System Definition

Communication interface

The RPLIDAR A2M12 uses separate 5V DC power for powering the range scanner core and the motor system. And the standard RPLIDAR A2M12 uses XH2.54-5P male socket. Detailed interface definition is shown in the following figure:

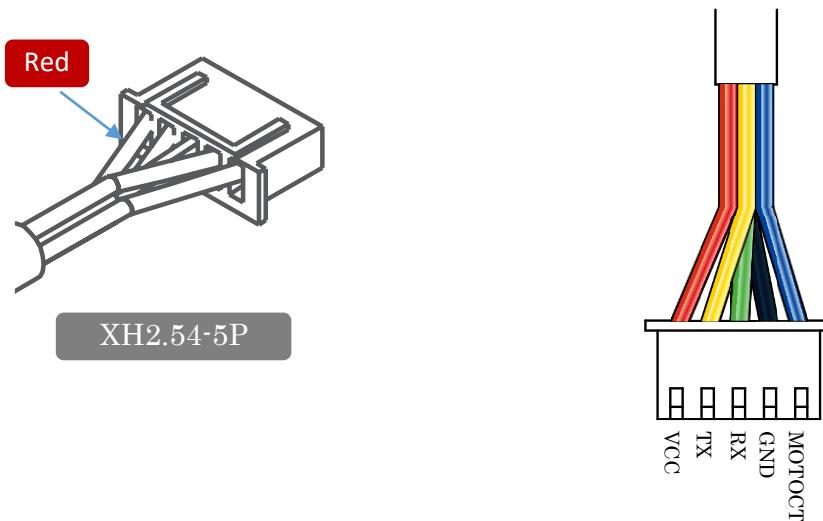


Figure 2-5 RPLIDAR Power Interface Definition

Color	Signal Name	Type	Description	Min	Typical	Max
Red	VCC	Power	Total Power	4.9V	5V	5.2V
Yellow	TX	Output	Serial port output of the scanner core	0V	3.3V	3.5V
Green	RX	Input	Serial port input of the scanner core	0V	3.3V	3.5V
Black	GND	Power	GND	0V	0V	0V
Blue	MOTOCCTL	Input	Scan motor /PWM Control Signal (active high, internal pull down)	0V	3.3V	5V

Figure 2-6 RPLIDAR External Interface Signal Definition

Power Supply Interface

RPLIDAR A2M12 takes the only external power to power the range scanner core and the motor system which make the core rotate. To make the RPLIDAR A2M12 work normally, the host system needs to ensure the output of the power and meet its requirements of the power supply ripple.

- For Model A2M12 Only

Item	Unit	Min	Typical	Max	Remark
Power Voltage	V	4.9	5	5.2	If the voltage exceeds the max value, it may damage the core
Power Ripple	Voltage mV	-	20	50	High ripple may cause the core working failure.
Inrush Current	mA			2500*	
System Current	Start mA	-	-	1500	The system startup requires relatively higher current.
Power Current	mA	TBD	200	220	5V Power, power off
		TBD	450	600	5V Power, power on

Figure 2-7 RPLIDAR Power Supply Specification

Note: When the lidar is connected to the power supply, there is a process of charging the input capacitor. The maximum transient current of charging can

reach 2500mA. After stable operation, the working current does not exceed 600mA.

Data communication interface

The RPLIDAR A2M12 takes the 3.3V-TTL serial port (UART) as the communication interface. The table below shows the transmission speed and the protocol standard.

Item	Unit	Min	Typical	Max	Comments
Band rate	bps	-	256000	-	-
Working mode	-	-	8N1	-	8n1
Output high voltage	Volt (V)	2.9	-	3.5	Logic High
Output low voltage	Volt (V)	-	-	0.4	Logic Low
Input high voltage	Volt (V)	1.6*	-	3.5	Logic High
Input low voltage	Volt (V)	-0.3	-	0.4	Logic Low

Figure 2-8 RPLIDAR Serial Port Interface Specifications

Note: the RX input signal of A2M12 is current control type. In order to ensure the reliable signal identification inside the system, the actual control node voltage of this pin will not be lower than 1.6v.

Scanner Motor Control

The RPLIDAR A2M12 is embedded with a motor driver which has speed tuning feature. Users can control the start, the stop and the rotating speed for the motor via MOTOCTL in the interface. MOTOCTL can be supplied using PWM signal with special frequency and duty cycle, and in this mode, the rotating speed is decided by the duty cycle of the input MOTOCTL PWM Signal.

The following table describes the requirement for the input PWM signal of MOTOCTL:

Item	Unit	Min	Typical	Max	Comments
------	------	-----	---------	-----	----------

High voltage	level V	3.0V	3.3V	5V	-
PWM frequency	Hz	24,500	25,000	25,500	Square Signal
Duty cycle range	-	0%	60%*	100%	Typical value is the duty cycle of high pulse width when the scanner frequency is at 10Hz

Figure 2-9 RPLIDAR Specification for PWM Signal of MOTOCTL

Note: the typical value is tested when the scanner rotating frequency is 10Hz. With the same rotating speed, the PWM duty cycle of every RILIDAR A2M12 may vary slightly. If a precise rotating speed is required, users can perform a closed-loop control.

If the host system only need to control the start and stop of the motor, please use the direct current signal in high level and low level to drive MOTOCTL. Under this condition, when the MOTOCTL is the low level signal, the RPLIDAR A2M12 will stop rotating and scanning; when the MOTOCTL is the high level signal, the RPLIDAR A2M12 will rotated at the highest speed.

MISC

- For Model A2M12 Only

Item	Unit	Min	Typical	Max	Comments
Weight	Gram (g)	TBD	190	TBD	
Temperature range	Degree Celsius (°C)	0	20	40	

Figure 2-10 RPLIDAR MISC Specification

To ensure the laser of RPLIDAR always working in the safety range (<3mW) and avoid any other damage caused by device, the RPLIDAR comes with laser power detection and sensor healthy check feature. It will shut down the laser and stop working automatically when any of the following errors has been detected.

Laser transmit power exceeds limited value

Laser cannot power on normally

Scan speed of Laser scanner system is unstable

Scan speed of Laser scanner system is too slow

Laser signal sensor works abnormally

The host systems can check the status of the RPLIDAR via the communication interface and restart the RPLIDAR to try to recover work from error.

To facilitate the usage of RPLIDAR A3 in the product development and speed up the development cycle for users, SLAMTEC has provided the **Framegrabber** plugin in RoboStudio for testing and debugging as well as the SDK available under Windows, x86 Linux and Arm Linux. Please contact SLAMTEC for detail information.

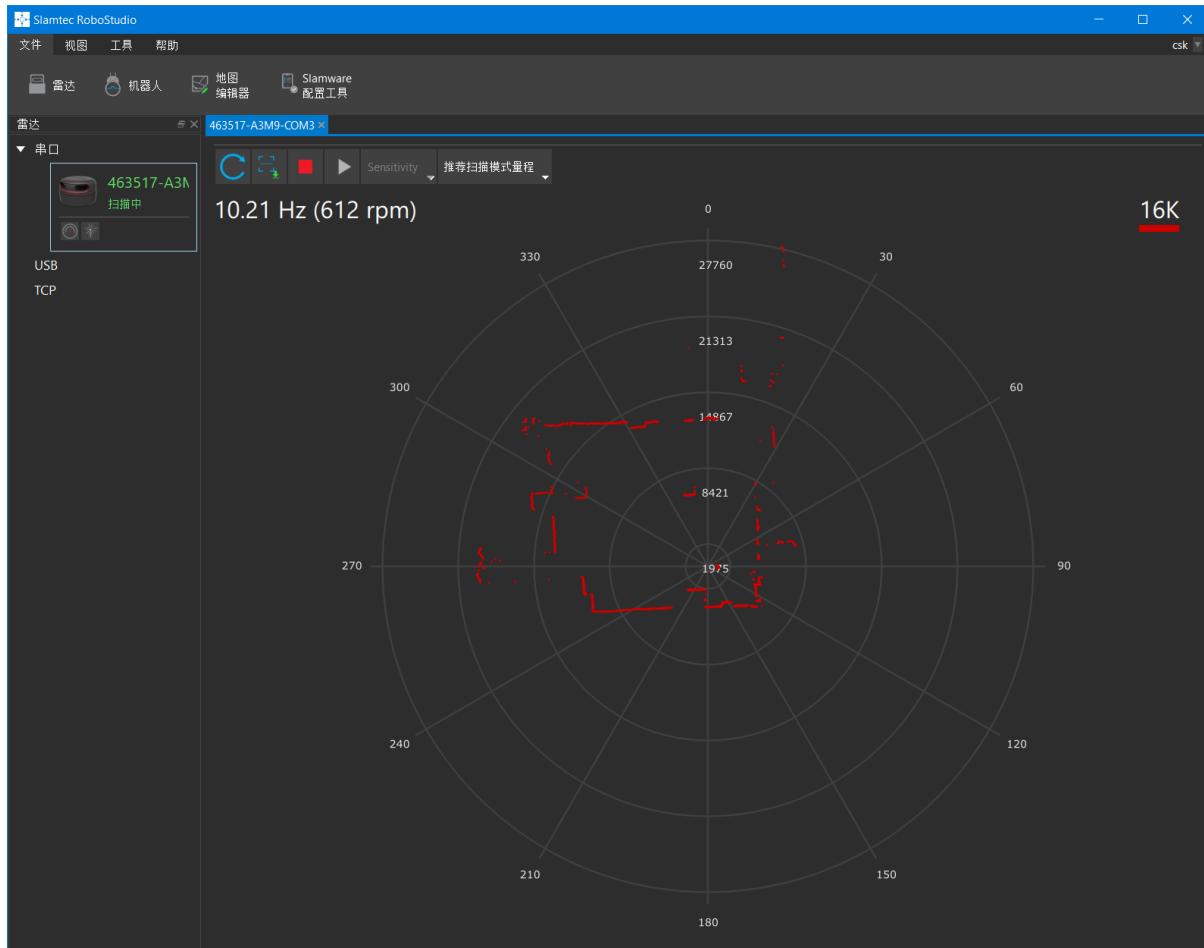


Figure 4-1 the Framegrabber Plugin in RoboStudio

The mechanical dimensions of the RPLIDAR A2M12 are shown as below:

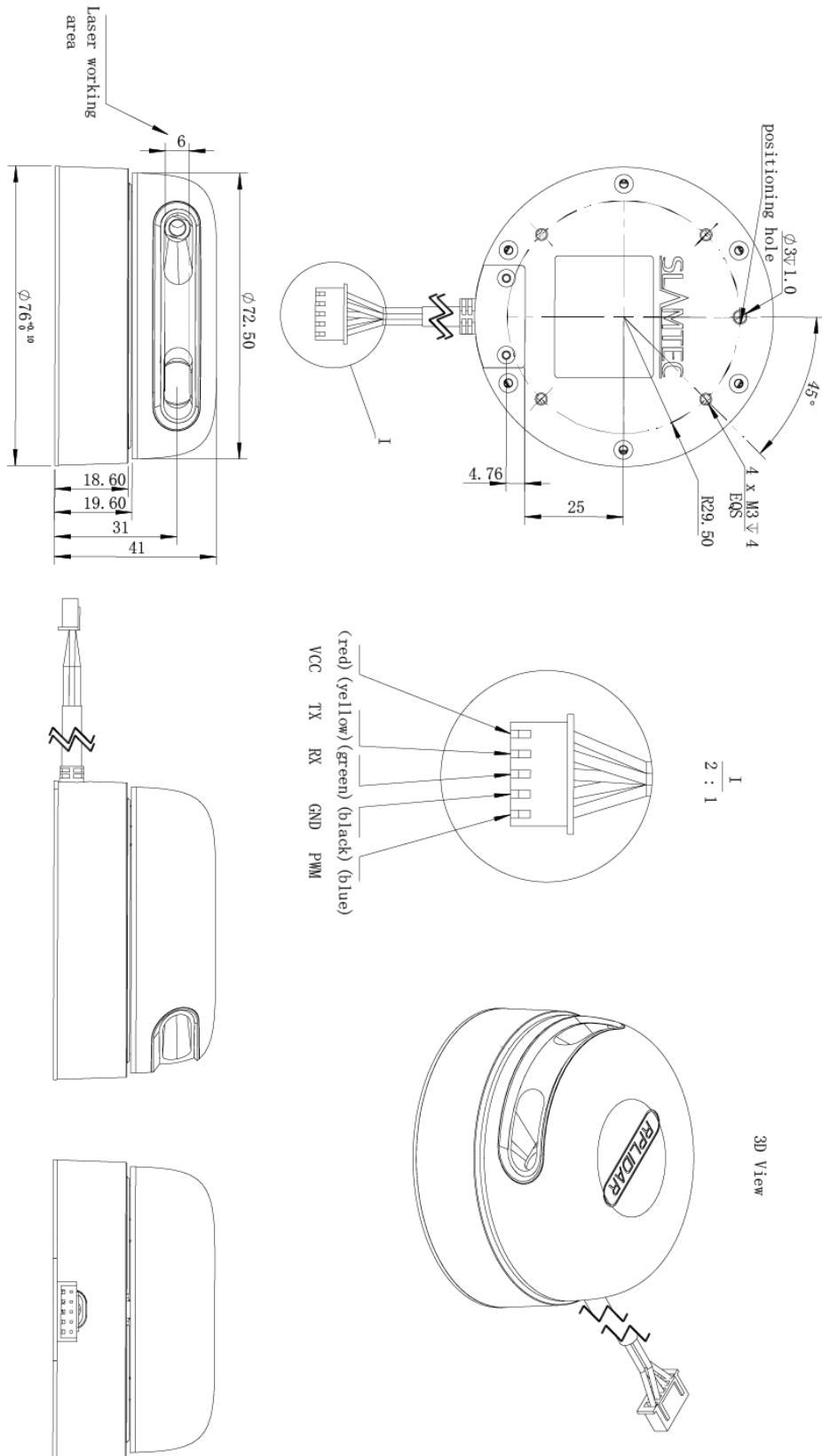


Figure 5-1 RPLIDAR Mechanical Dimensions

Note: the 4 M3 screws in the bottom should be no longer than 4mm, or the internal module would be damaged.

Revision history

Image and Table Index

FIGURE 1-1 RPLIDAR SYSTEM COMPOSITION	4
FIGURE 1-2 THE RPLIDAR WORKING SCHEMATIC.....	5
FIGURE 1-3 THE OBTAINED ENVIRONMENT MAP FROM RPLIDAR SCANNING	6
FIGURE 1-4 THE RPLIDAR SAMPLE POINT DATA INFORMATION	7
FIGURE 1-5 THE RPLIDAR SAMPLE POINT DATA FRAMES.....	7
FIGURE 2-1 RPLIDAR PERFORMANCE	9
FIGURE 2-2 RPLIDAR OPTICAL SPECIFICATION.....	9
FIGURE 2-3 RPLIDAR OPTICAL WINDOW.....	10
FIGURE 2-4 RPLIDAR SCANNING DATA COORDINATE SYSTEM DEFINITION.....	11
FIGURE 2-5 RPLIDAR POWER INTERFACE DEFINITION.....	11
FIGURE 2-6 RPLIDAR EXTERNAL INTERFACE SIGNAL DEFINITION	12
FIGURE 2-7 RPLIDAR POWER SUPPLY SPECIFICATION.....	12
FIGURE 2-8 RPLIDAR SERIAL PORT INTERFACE SPECIFICATIONS	13
FIGURE 2-9 RPLIDA SPECIFICATION FOR PWM SIGNAL OF MOTOCTL.....	14
FIGURE 2-10 RPLIDAR MISC SPECIFICATION	14
FIGURE 3-1 THE LIDARS PLUGIN IN ROBOSTUDIO	16
FIGURE 4-1 RPLIDAR MECHANICAL DIMENSIONS.....	17

Manufacturer: SHANGHAI SLAMTEC CO., LTD.

Address: D-501 Shengyin Tower, 666 Shengxia Rd., Shanghai, China

Made in China