CoVAPSY – Mise en œuvre du Simulateur Webots

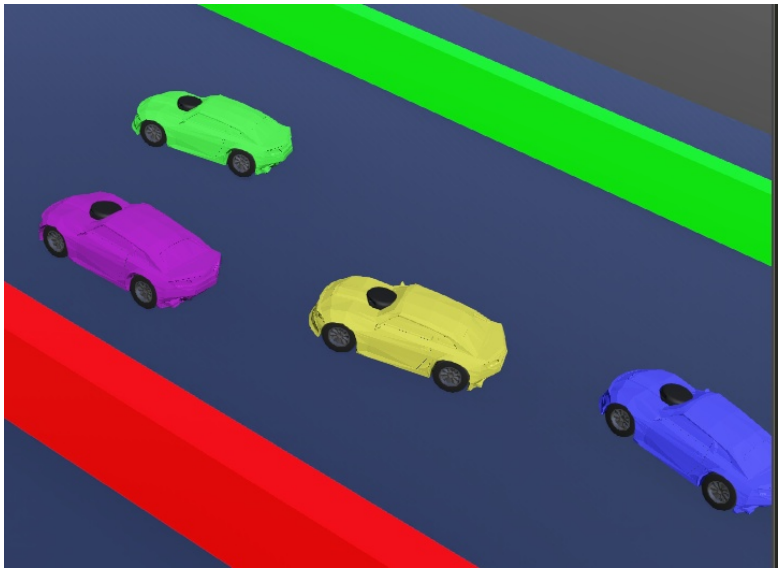


Figure 1: Course entre 4 voitures

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 1/10ème 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 1/10ème 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 1/10ème, 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

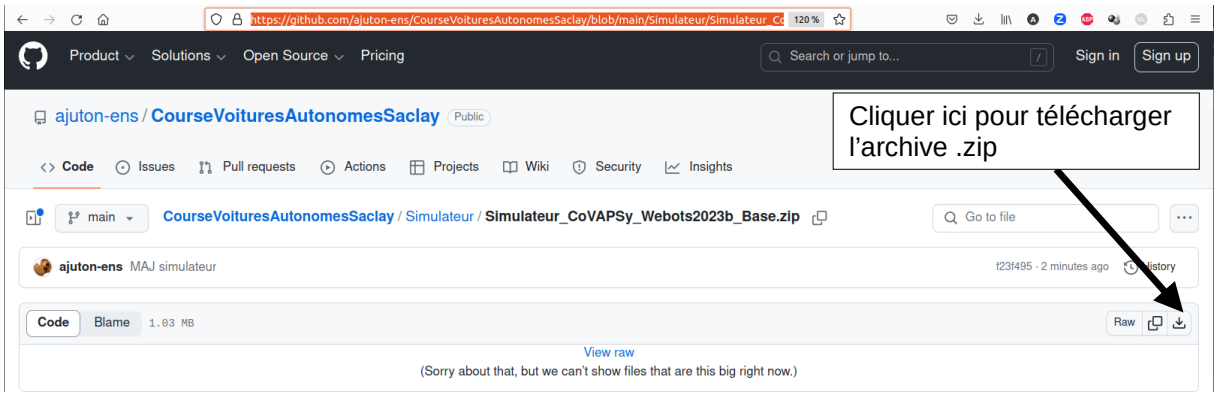


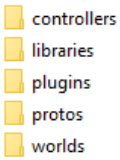
Figure 2: Téléchargement de l'archive contenant le projet webots de base

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.

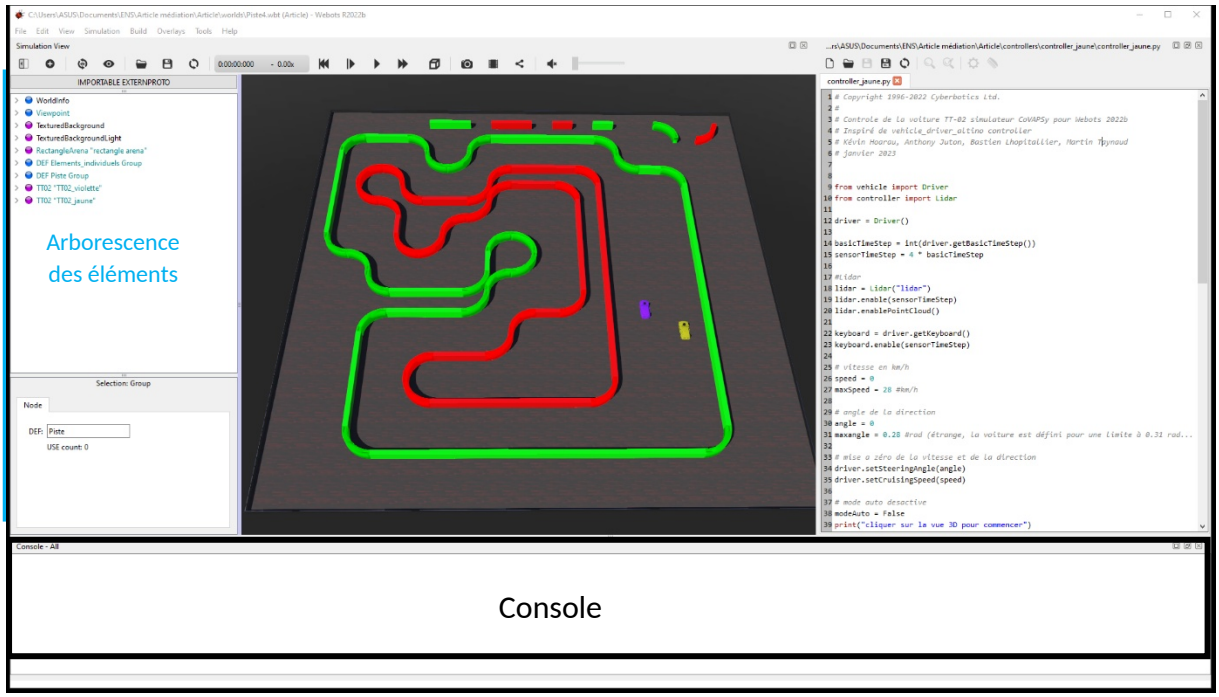


Figure 3: Les différents panneaux de Webots

- 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 trouve dans ***« DEF Elements_individuels 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.

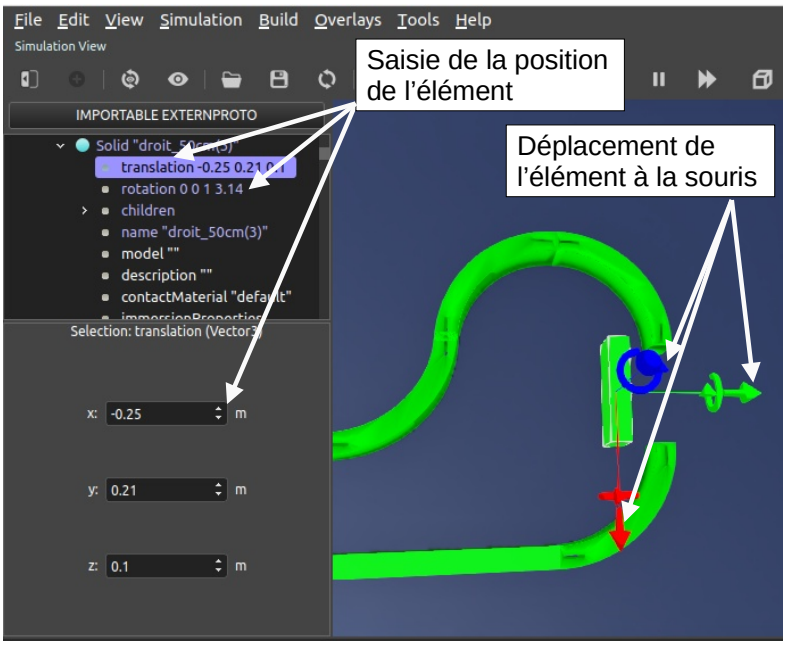


Figure 4: Modification de la piste

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.

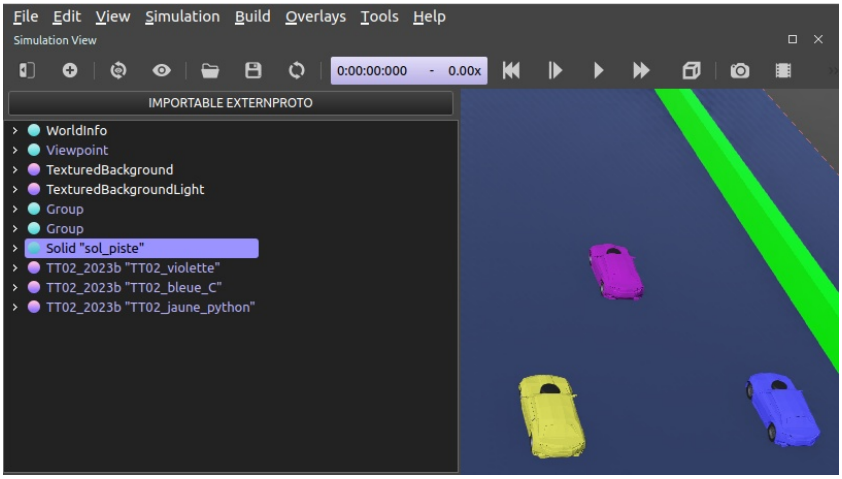


Figure 5: 3 voitures, pour la programmation en C et en python

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 pyton, 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).

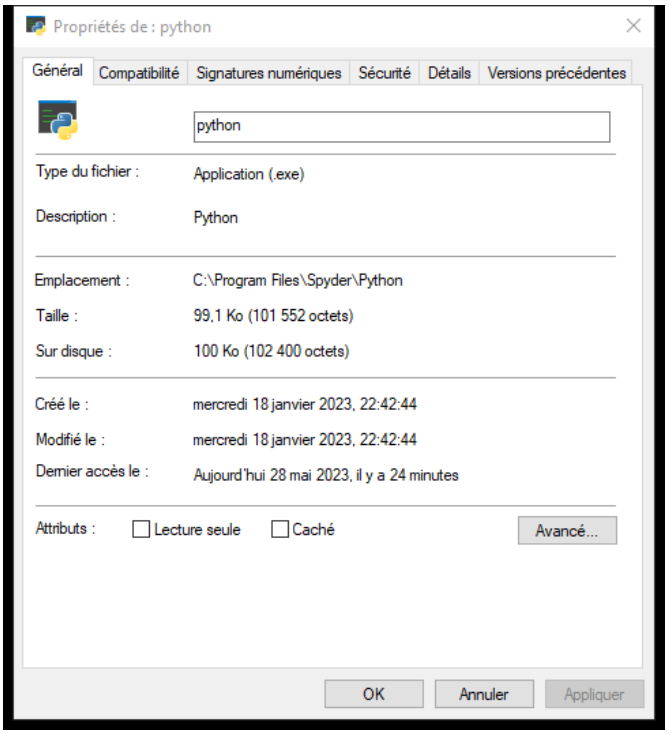
Aucune connaissance sur la bibliothèque du simulateur n'est requise. Webots supporte aussi le java. Pour aller plus loin, il est possible de se référer à la documentation de *Webots* :(https://www.cyberbotics.com/doc/guide/tutorials)

3.1 Programmation en python

1) Configuration

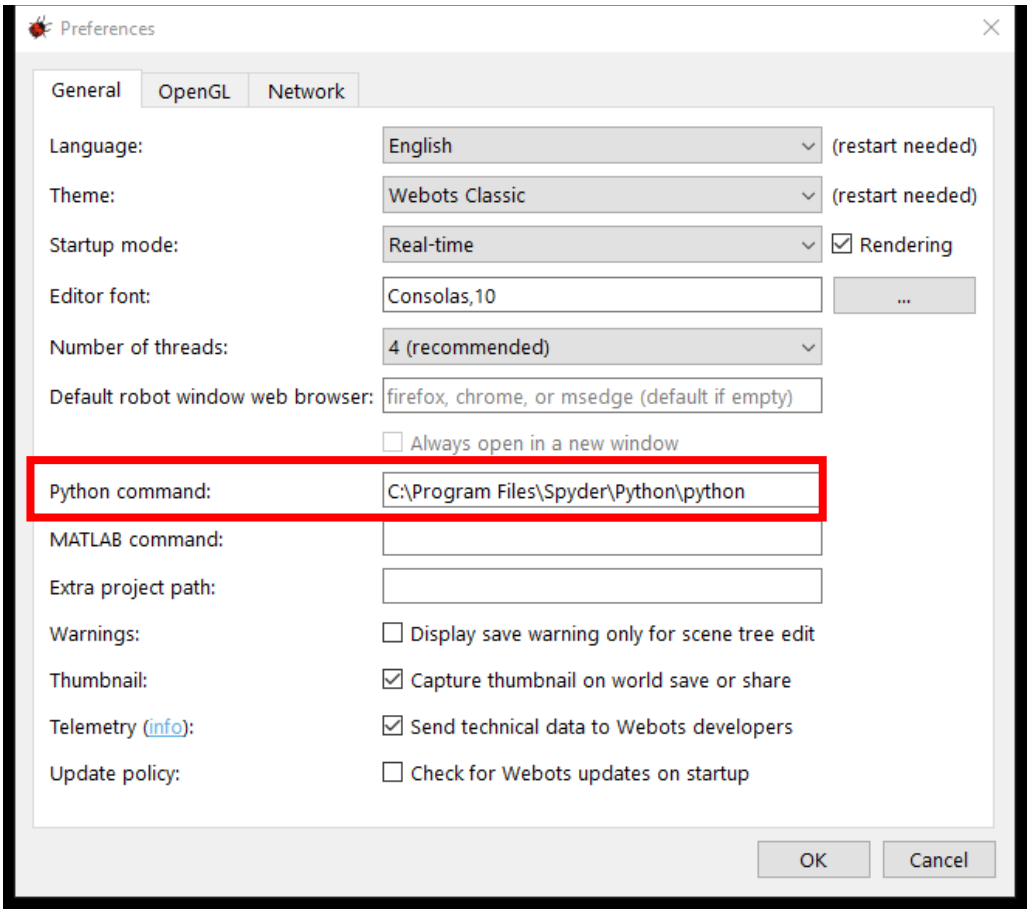
Linux ne demande pas de configuration particulière. Passer directement à « Modifier un programme dans l'éditeur de texte du simulateur »
Sous windows, il faut indiquer l'interpréteur *Python* qui sera utilisé par le logiciel pour lancer les différents programmes.

- 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) <u>Modifier un programme dans l'éditeur de texte du simulateur</u>

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***

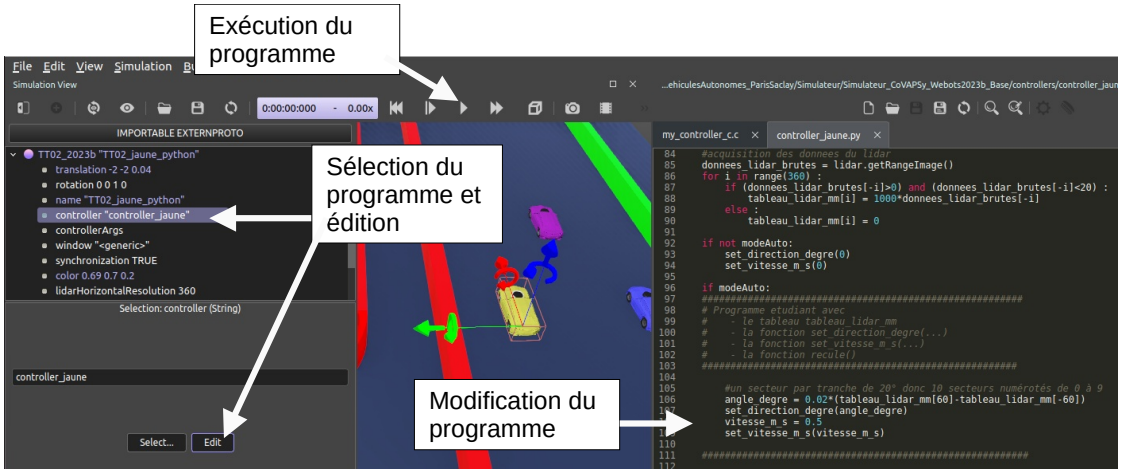


Figure 6: Modification du programme controller de la voiture jaune.

- 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'éxécution.

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.

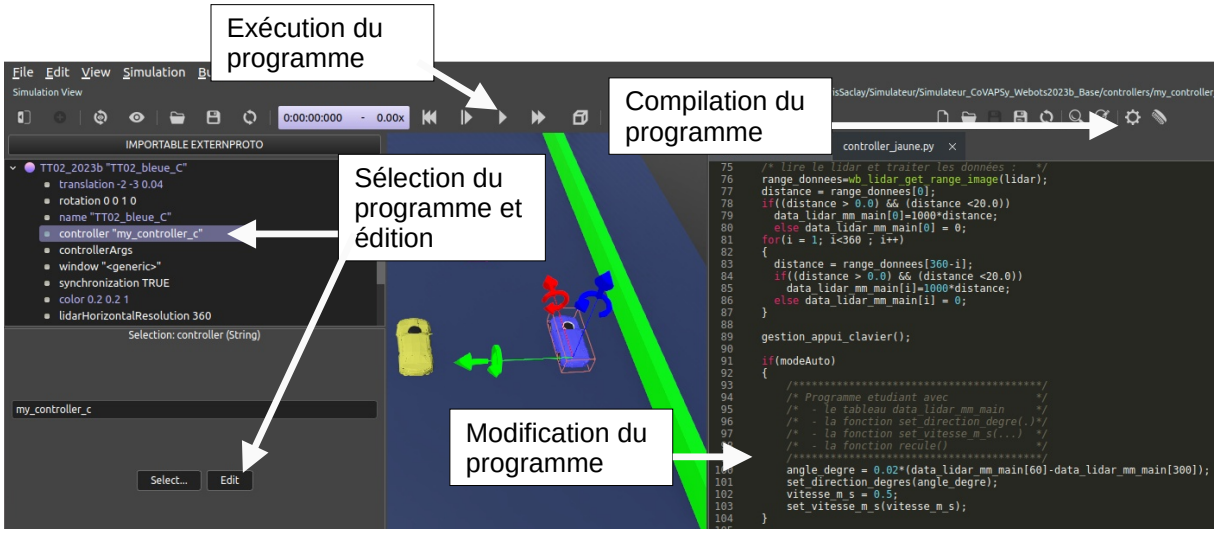


Figure 7: Modification du programme my_controller_c.c de la voiture bleue.

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 :

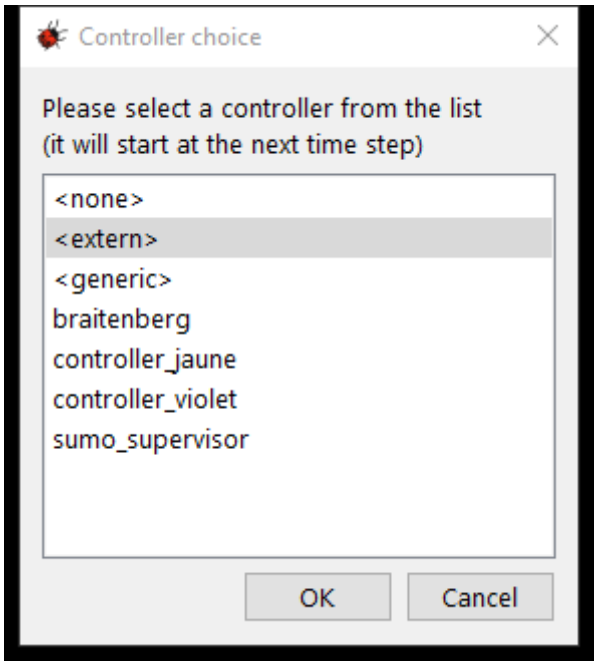
  ```
  export WEBOTS_HOME=/usr/local/webots/
  ```

- **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*.

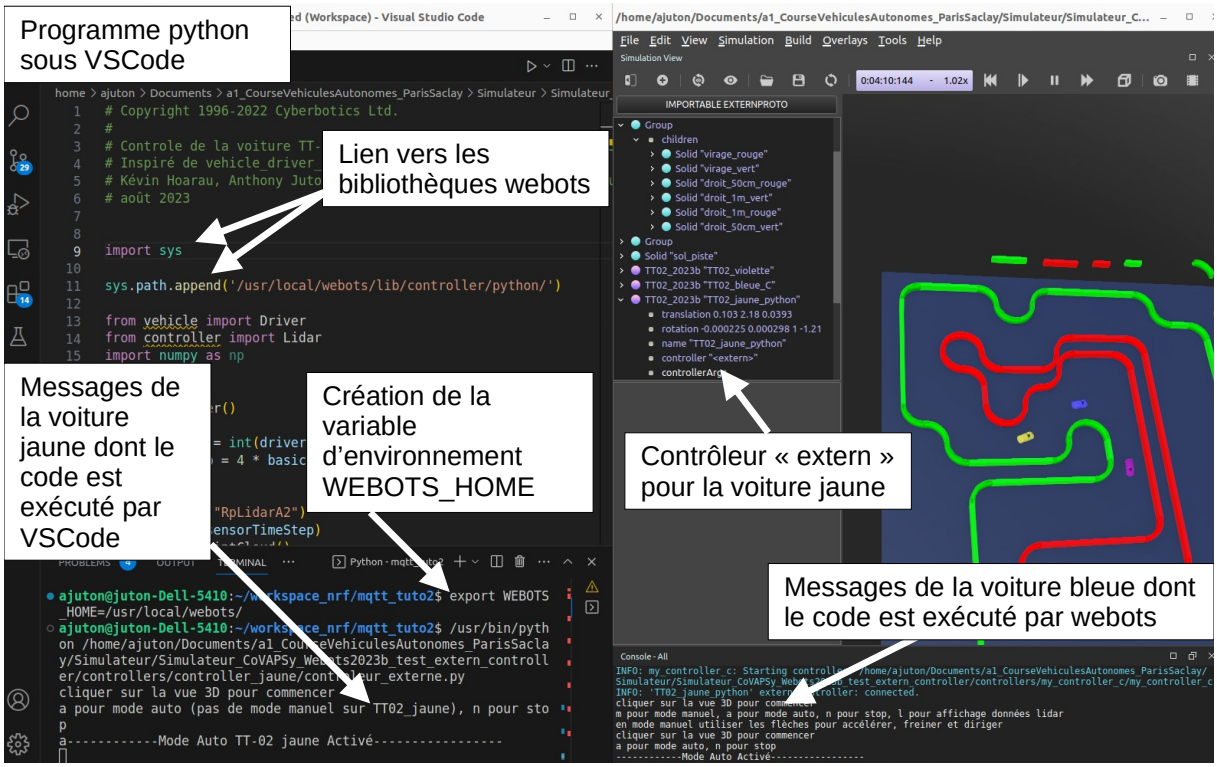


Figure 8: voiture jaune simulée sous webots avec un contrôleur exécuté sous VSCode

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 contrôler envoyé par un étudiant, soit en copiant/collant un dossier contrôleur existant. Le nom du fichier doit correspondre au nom du dossier.

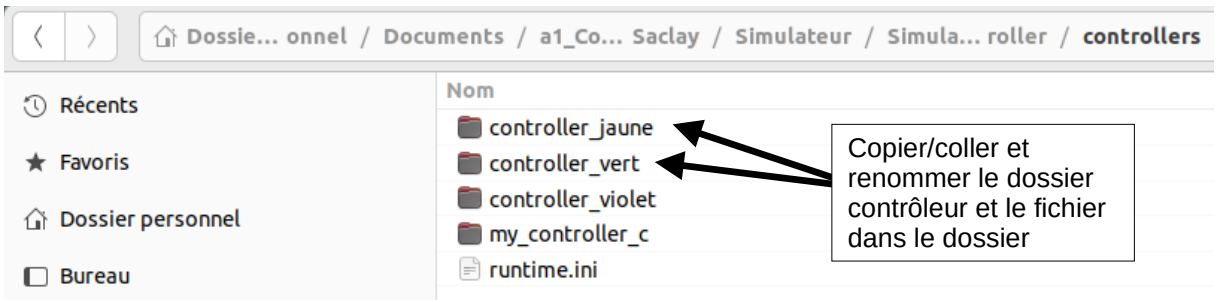


Figure 9: Ajout d'un contrôleur au projet pour une course à 4 voitures

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 le 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)

```python
# 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
```

```python
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)
```

```python
    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)

```c
/*
 * 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);
```

```c
//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 */
  //intialisation 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);
}
```