Apprentissage par renforcement et transfert simulation vers réalité pour la conduite de voitures autonomes

Culture Sciences de l'Ingénieur

La Revue 3E.I

Édité le 22/06/2023

école normale supérieure paris-saclay

Rania BENNANI[1] – Kevin HOARAU[1] – Anthony JUTON[3]

[1] *Elève de 3ème année au département Nikola Tesla de l'école Normale Supérieure Paris-Saclay*

[3] *Professeur agrégé de physique appliquée au département Nikola Tesla de l'école Normale Supérieure Paris-Saclay*

Cette ressource fait partie du N°111 de La Revue 3EI de janvier 2024 et s'intègre au dossier « Intelligence Artificielle » de Culture Sciences de l'Ingénieur.

Cet article présente l'apprentissage par renforcement de la conduite sur circuit d'une voiture autonome 1/10ème, en simulation, puis le transfert du réseau de neurones du simulateur dans la voiture réelle, en utilisant webots, gym et stable baselines3. Il est issu du travail de Kévin Hoarau pour sa participation à la course de voitures autonomes de Paris Saclay, CoVAPSy 2023 *[6]*.

Figure 1: Voiture en apprentissage sur le simulateur webots

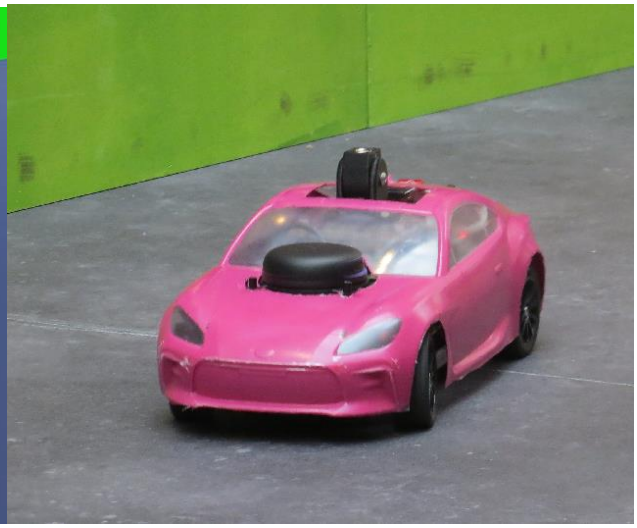


Figure 2: Voiture réelle, avec le réseau de neurones issu de la simulation

Le simulateur utilisé, webots, peu gourmand en ressource et open source, permet à chacun d'expérimenter l'apprentissage par renforcement profond sur cet exemple réaliste, même sans disposer de voiture pour le passage à la réalité.

La voiture 1/10ème instrumentée d'un coût modeste (moins de 1000 euros) permet à travers cet exercice d'appréhender le transfert simulation →réalité et les difficultés associées pour une mise en œuvre concrète et matérielle de l'intelligence artificielle. La voiture et le simulateur sont présentés en détails dans les articles CoVAPSy0_Course_de_voitures_autonomes_1_10eme.pdf et les suivants.

L'article « Apprentissage par renforcement de la conduite d'un véhicule sur AirSim » de Ludovic de Mattteis et Sasa Radosalvjevic (mettre le lien) a servi de point de départ à ce travail. Webots a été préféré à AirSim pour sa légèreté et sa facilité de mise en œuvre et l'expérience acquise précédemment à permis d'aller jusqu'au transfert de la simulation à la réalité.

1 - Introduction

L'apprentissage par renforcement ("*Reinforcement Learning*" en anglais) est une catégorie de Machine Learning. L'article « *Introduction à l'apprentissage par renforcement* » *[1] du « Dossier Intelligence Artificielle* » du numéro 109 de la revue 3EI / Culture Sciences de l'ingénieur présente cette méthode en détail. Cet article vient en complément, en proposant un exemple avancé de mise en œuvre de l'apprentissage par renforcement pour la conduite de voitures autonomes sur un circuit.

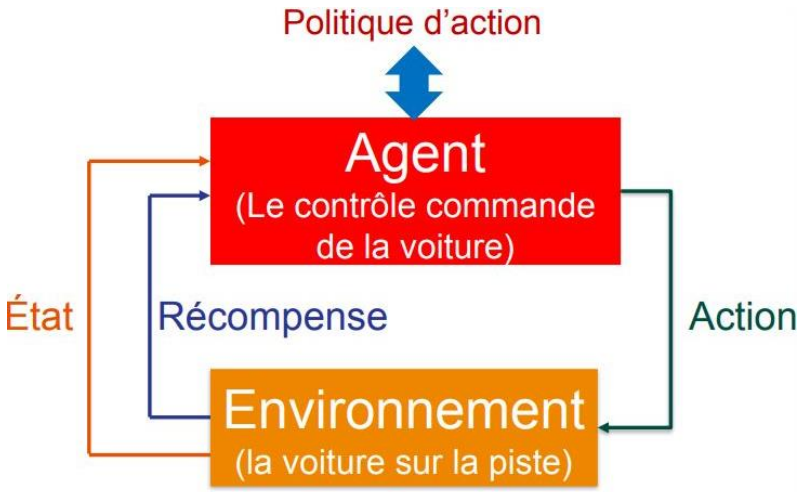


Figure 3 :Schéma explicatif de l'apprentissage par renforcement

En quelques mots, on se place dans un ensemble {**agent**, **environnement**} où une **action** choisie et réalisée par l'agent, en fonction de l'**état** de l'environnement, peut entraîner une modification de l'environnement.

Le processus d'apprentissage vise à doter l'agent d'une politique d'action lui permettant de faire les meilleurs choix. Une **récompense** est alors attribuée à l'agent dont la valeur dépend de si l'action est positive ou négative pour l'agent. Lors de chaque étape de l'apprentissage, l'agent reçoit une observation de l'environnement dans lequel il évolue. Suivant cette observation, l'agent prend une décision d'action. La décision est prise dans un ensemble d'actions appelé espace des actions. Cet espace peut dépendre de l'état.

Un exemple simple est celui d'un jeu d'échec dans lequel l'observation correspond à la position de chacune des pièces de l'échiquier et l'espace des actions est l'ensemble des déplacements possibles des pièces (un fou ne peut pas être déplacé au lancement de partie par exemple). Naturellement, on souhaite que l'agent réalise la meilleure action possible suivant l'observation reçue. L'agent, pour atteindre ce but, applique une politique d'action (notée par la suite π) qu'il utilise pour sa prise de décision. A chaque récompense obtenue, cette politique est mise à jour. Au fil des épisodes d'apprentissage, on espère ainsi atteindre une politique optimale menant à la victoire, quel que soit l'adversaire.

Dans cet article, la voiture doit, à partir de l'observation de l'environnement par son capteur lidar, agir sur la propulsion et la direction pour parcourir le plus vite possible la piste.

L'environnement est constitué de la voiture, de la piste et des voitures adverses et l'agent est le programme de pilotage de la voiture.

Des bibliothèques existent pour l'apprentissage par renforcement profond. Ici est utilisée la bibliothèque Stable Baselines 3 de PyTorch, référence dans le domaine de l'IA. Après plusieurs essais, c'est l'algorithme PPO (Proximal Policy Optimization) qui a donné les meilleurs résultats et est donc retenu pour cet article, associé à Gym, ensemble d'outils développés par OpenAI pour l'apprentissage par renforcement.

2 - Les 4 étapes

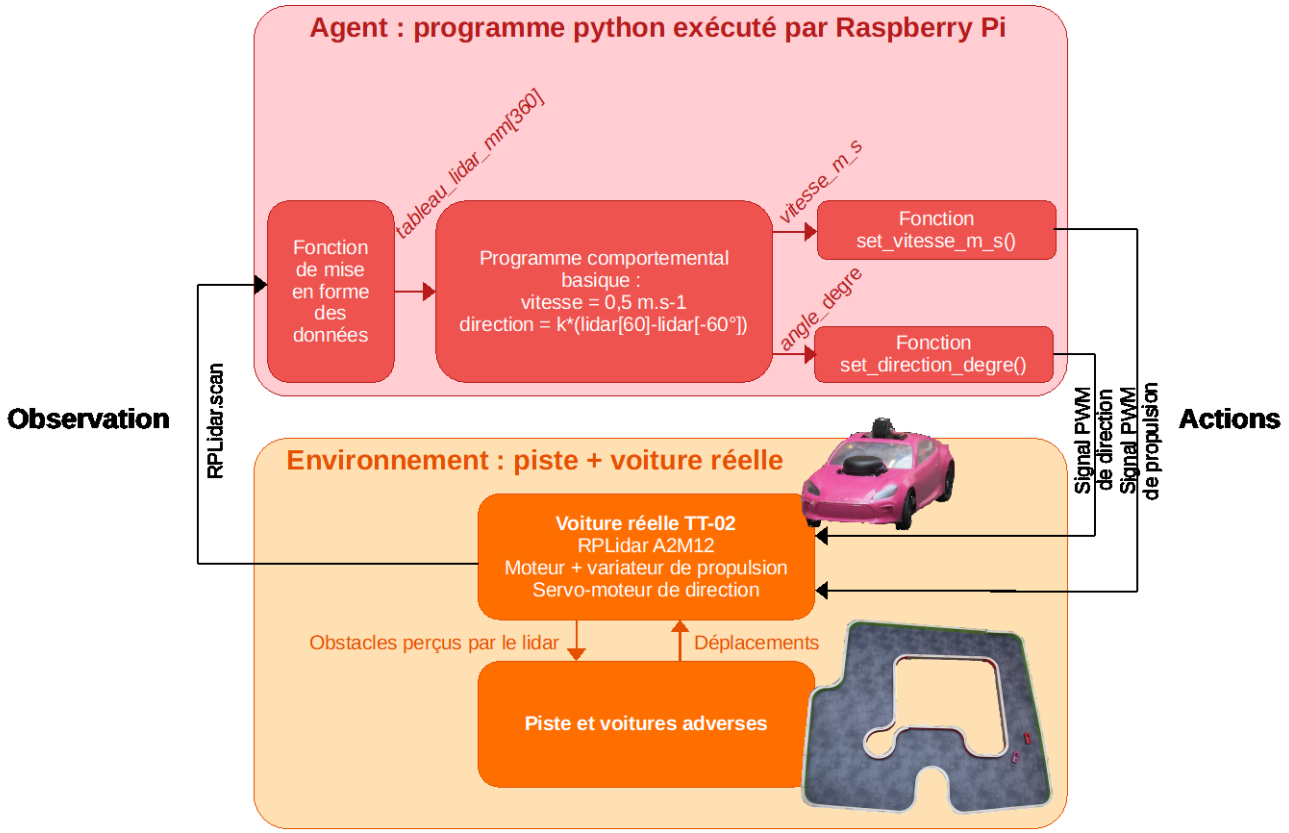
Le travail se présente en 4 étapes, les 2 premières s'intéressant à la mise en place de la voiture réelle et de son modèle simulée, la 3ème à l'apprentissage automatique sur le simulateur et la dernière au transfert du réseau du simulateur vers la voiture réelle.

2.1 - Étape 1 : fonctions de base sur la voiture réelle

La première étape consiste à équiper la voiture des capteurs et actionneurs nécessaire à la conduite autonome et à développer les fonctions lidar, direction, propulsion pour la voiture réelle, avec un algorithme de conduite basique. Cette étape est expliquée dans l'article :

CoVAPSy2_Premiers_programmes_python_sur_la_voiture_reelle

Les fonctions sont fournies sur le dépôt git *[3]*.



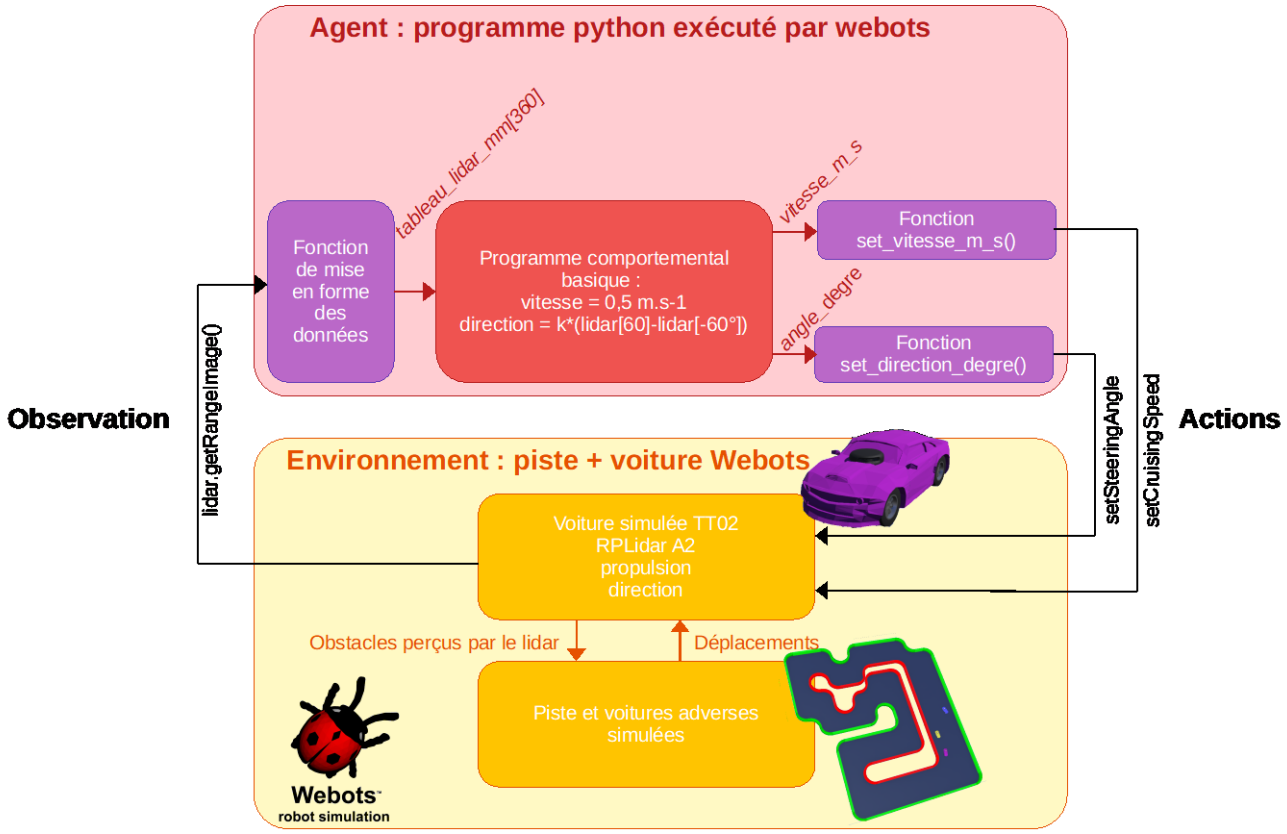
2.2 - Étape 2 : fonctions de base sur le simulateur

La 2nde étape consiste à concevoir un modèle simulé de la voiture le plus proche de la voiture réelle et à développer les fonctions pour cette voiture simulée, se comportant comme les précédentes de sorte de fonctionner avec le même algorithme de conduite.

En plus de la présentation du simulateur Webots, cette étape est expliquée dans l'article :

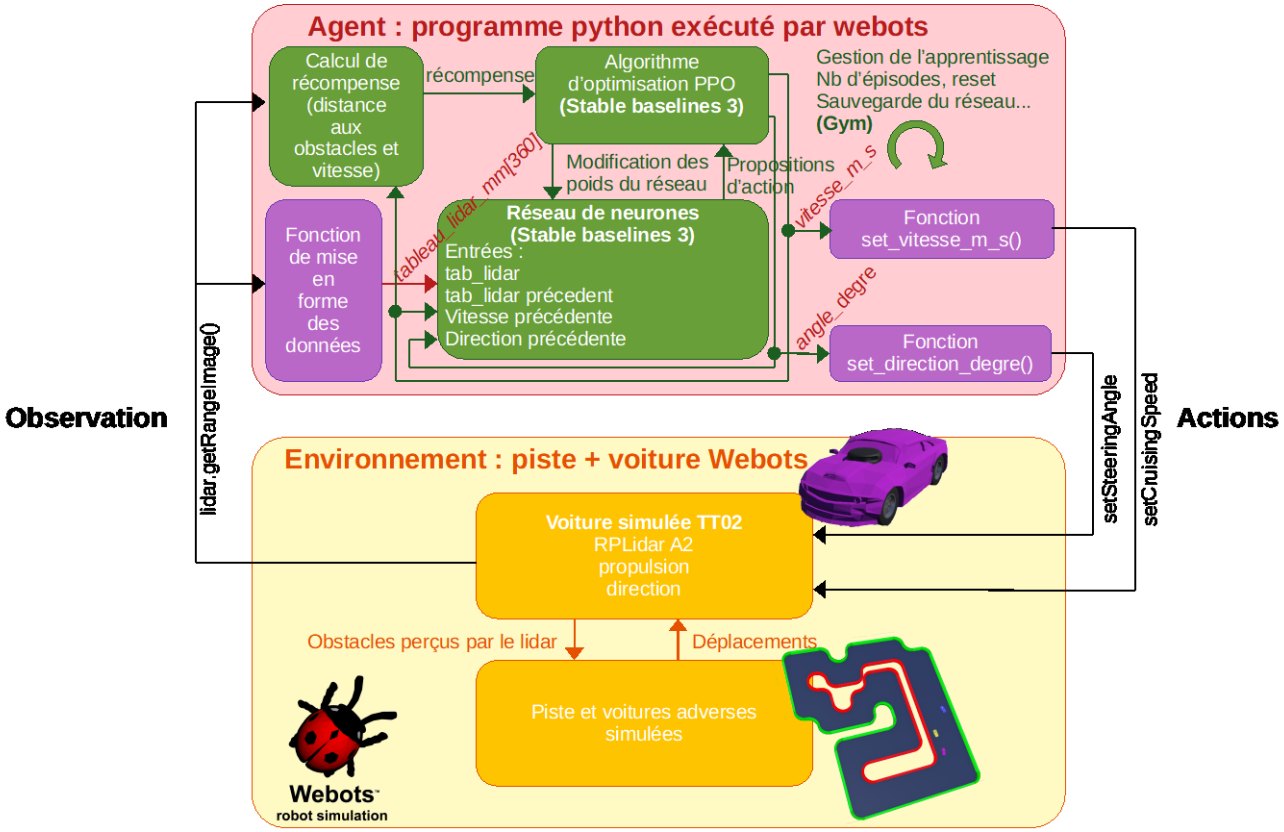
CoVAPSy1_Mise_en_oeuvre_du_simulateur_Webots

Les fonctions et le modèle de voiture TT-02 simulée sont fournies avec le projet de base du simulateur, sur le dépôt git *[3]*.



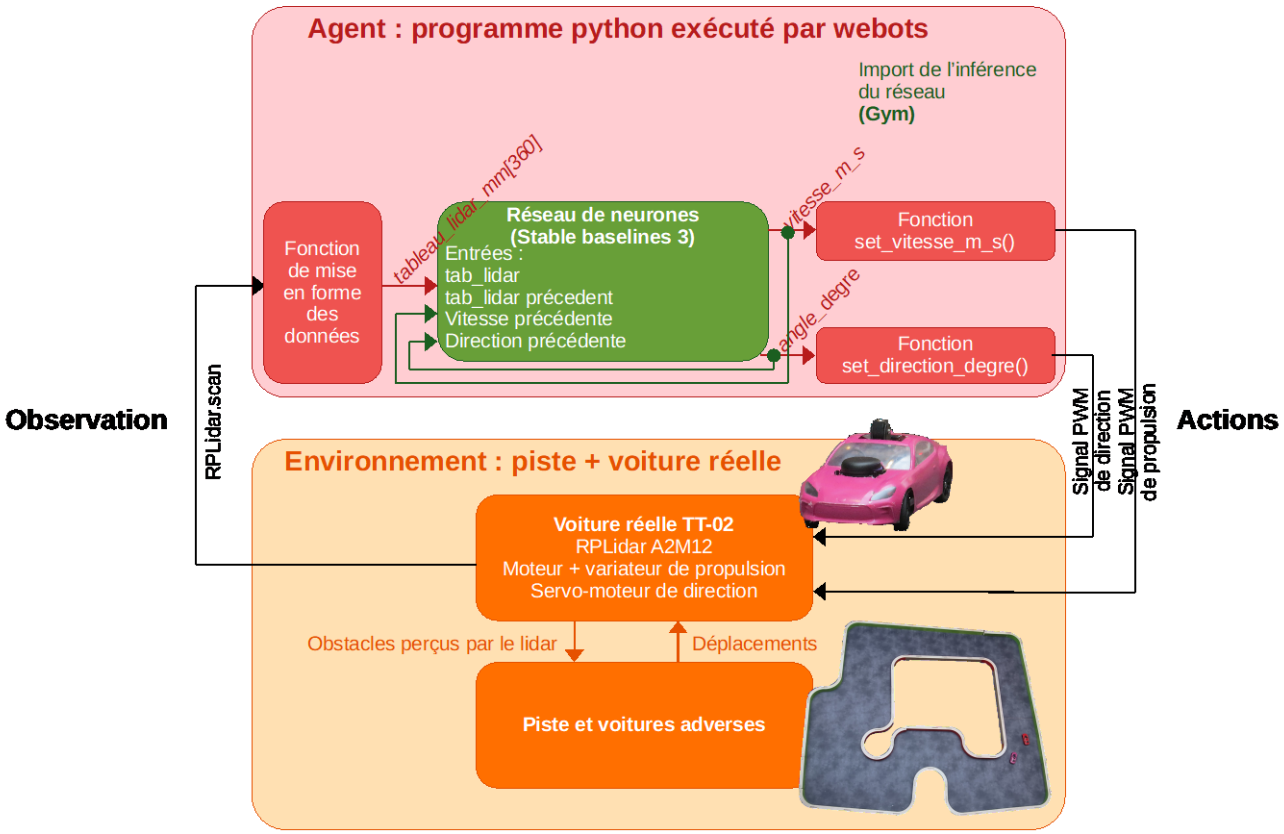
2.3 - Étape 3 : Apprentissage automatique de la conduite sur simulateur

La 3ème étape remplace l'algorithme basique de conduite par un réseau de neurones et met en place l'apprentissage par renforcement de ce réseau de neurones pour aboutir à une conduite performante. C'est l'objet de la partie 3 de cet article.



2.4 - Étape 4 : transfert du réseau de neurones dans la voiture réelle

Enfin, la dernière étape consiste à transférer le réseau de la voiture simulée dans la voiture réelle pour permettre à la voiture de concourir lors de la course de voitures autonomes. C'est l'objet de la partie 4 de cet article.



3 - Apprentissage par renforcement sur simulateur

L'utilisation d'un simulateur est indispensable pour réaliser les milliers d'essais d'un apprentissage par renforcement. Pour que le transfert du simulateur à la réalité soit possible, une première exigence est d'avoir un modèle simulé proche de la voiture réelle Tamiya TT-02. La première étape du travail a donc consisté à développer ce modèle et à ajuster ses paramètres à partir des voitures des bibliothèques Webots et de la documentation de la voiture réelle. L'ensemble est décrit dans l'article CoVAPSy1_Mise_en_oeuvre_du_simulateur_Webots.pdf et le projet de base est disponible en téléchargement sur le dépôt github. *[3]*

Cet article utilise le projet *Simulateur_CoVAPSy_Webots2023b_RL.zip* issu de celui de l'article cité ci-dessus, avec en plus le superviseur et des voitures *sparring partners [3]*. Le simulateur utilisé est Webots, de Cyberbotics, dans sa version R2023b. C'est un logiciel open source permettant de créer un environnement dans le but de simuler des machines robotiques. Le programme pilotant la voiture est écrit en Python. Il est également possible de programmer en Java, C++ ou Matlab.

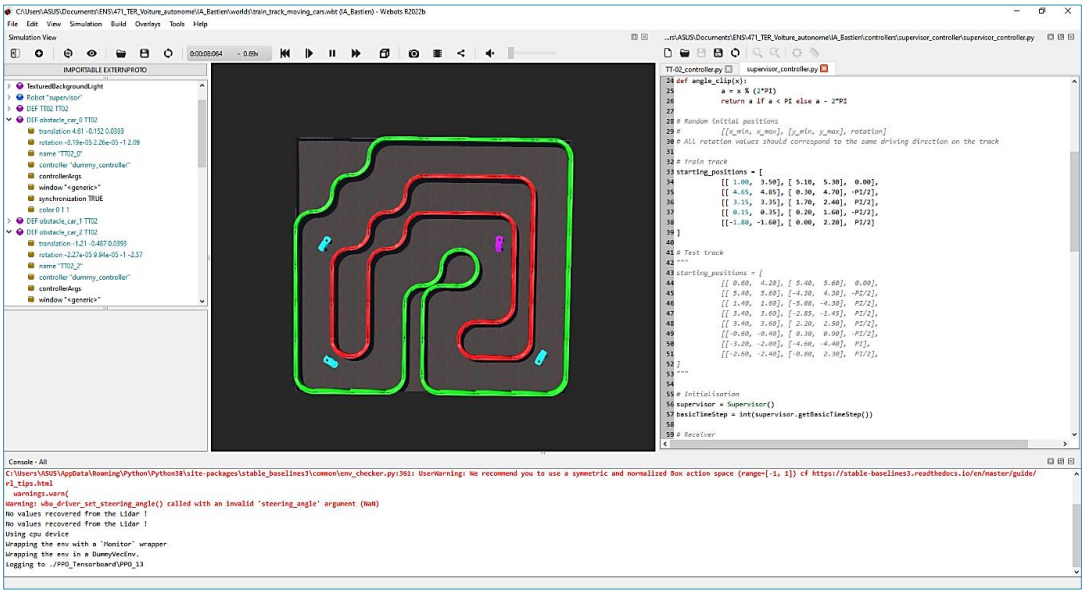


Figure 2 : Interface du simulateur Webots

Le code étant assez complexe, il est recommandé, au moins pour l'agent (le *controller* de la voiture) d'utiliser un environnement de développement python plus complet que l'éditeur de webots. La démarche est expliquée en détail ici : https://cyberbotics.com/doc/guide/running-extern-robot-controllers?

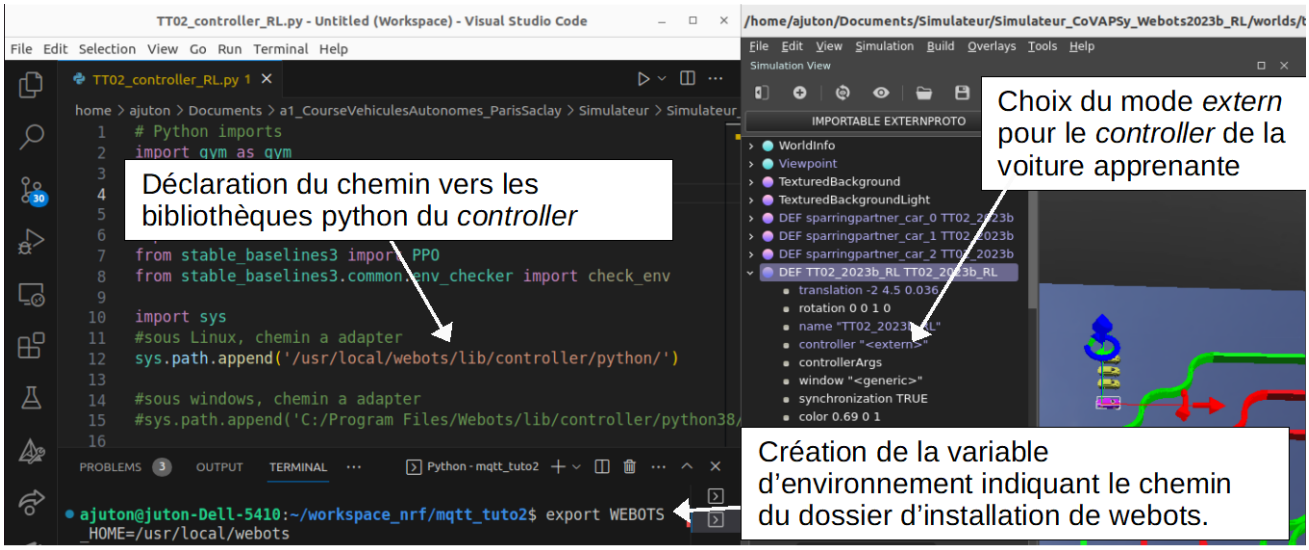


Figure 4: configuration de webots et du programme python pour utiliser un environnement de développement extérieur à webots

En plus du modèle de la voiture, pour pouvoir entraîner efficacement celle-ci sur le simulateur, il est nécessaire de disposer de plusieurs éléments qui seront détaillés par la suite :

- Les pistes, à la fois pour l'apprentissage et l'évaluation.
- Un superviseur capable de replacer les voitures pour les réinitialisations.
- La bibliothèque Stable Baselines 3 pour la génération du réseau de neurones et son apprentissage avec l'algorithme PPO.
- La bibliothèque Gym pour gérer le processus d'apprentissage, le lien avec l'environnement webots, et l'utilisation de l'inférence.

Ces éléments composent l'environnement d'apprentissage.

TODO : enlever le terme environnement d'apprentissage

3.1 - Pistes utilisées

Pour un apprentissage par renforcement, il est nécessaire de disposer d'un maximum de situations différentes, représentatives des situations dans lesquelles se trouvera la voiture réelle. Dans le cas présent, il faut une piste avec de longues lignes droites, des virages dans chaque sens, des virages à 180°,etc. On propose donc la piste suivante comme étant la piste d'entraînement :

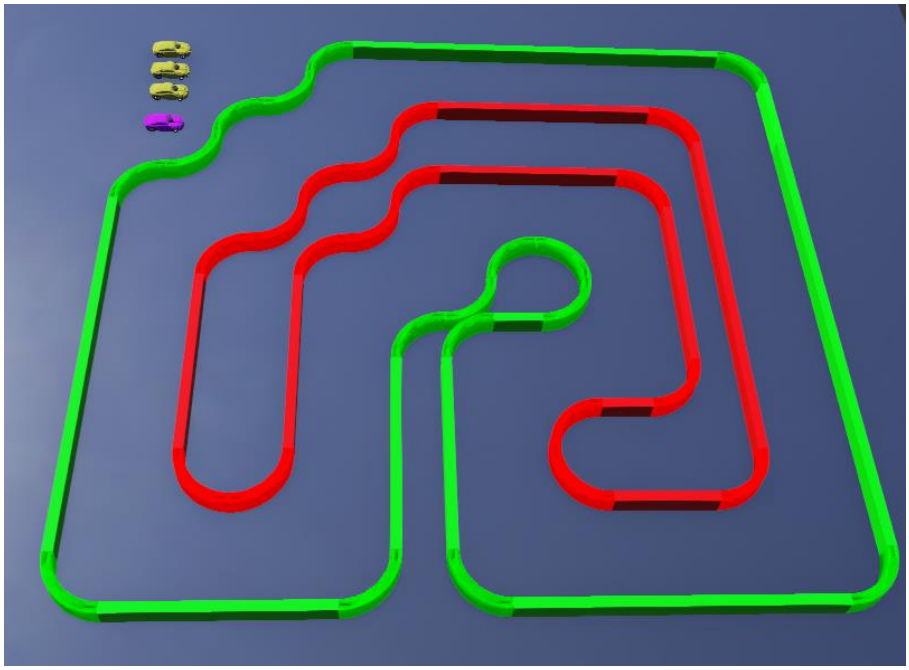


Figure 4 : Piste d'entraînement pour l'apprentissage par renforcement

Sur cette piste, on rajoute trois voitures jaunes ayant pour modèle la TT-02 et ayant un *controller* (programme de conduite) très simple (la vitesse est constante et l'angle de la direction est proportionnel à la distance mesurée à 60° moins celle mesurée à -60°). En démarrant de manière aléatoire sur la piste, on obtient ainsi une piste représentative de la course.

Dans un processus d'apprentissage, il est nécessaire de valider ce qui a été appris sur la piste d'entraînement sur une autre piste, pour vérifier notamment qu'il n'y a pas eu de sur-apprentissage (la voiture apprend à aller très vite sur la piste d'essai mais est incapable de rouler sur une autre piste). Pour cela, est créée une piste de validation avec cinq voitures en plus.

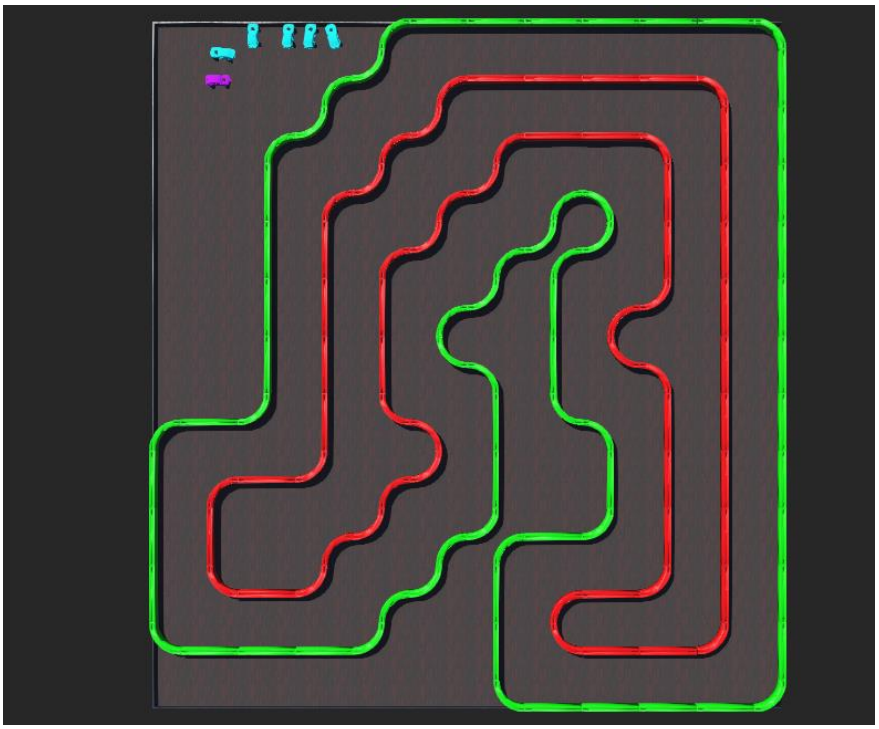


Figure 5 : Piste de validation pour l'apprentissage par renforcement

TODO : Tester la piste d'apprentissage

3.2 - Le superviseur :

Sur ces pistes, pour pouvoir lancer l'apprentissage, il est nécessaire de gérer l'ensemble des voitures lors d'une phase de réinitialisation. Pour cela est ajouté un robot "**superviseur**" qui permet de repositionner toutes les voitures.

Le superviseur reçoit, via un récepteur, un message de l'agent de la voiture en apprentissage lors d'une collision pour remettre l'ensemble des voitures en place de manière aléatoire pour un nouvel épisode. Il possède pour cela un tableau d'encadrements de positions dans lesquelles est choisie aléatoirement une valeur pour affecter « intelligemment » une position aléatoire à chaque véhicule. Cela évite que 2 voitures ne se touchent au départ, que des voitures ne suivent la piste en sens inverse ou qu'une voiture soit face au mur. Le choix de valeurs de positions et de sens de rotation aléatoires évite le sur-apprentissage (la voiture apprend un algorithme efficace uniquement pour un départ dans la position initiale de l'apprentissage).

Une fois la mise en place effectuée, le superviseur, via un émetteur, envoie un message à l'agent indiquant que les voitures sont prêtes pour redémarrer. Les voitures *sparring partners* sont repositionnées par le superviseur mais ne communiquent pas. La voiture apprenant la conduite est un nœud TT02_2023b_RL différent du TT02_2023b par la présence d'un émetteur et d'un récepteur.

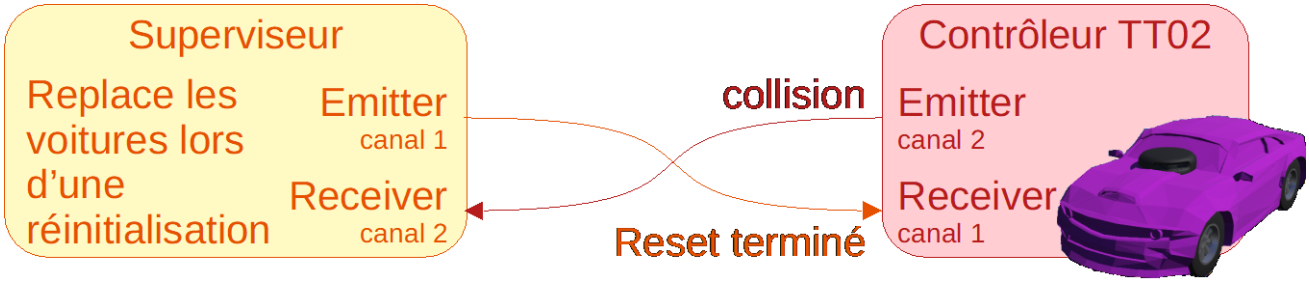


Figure 5: messages échangés par les émetteurs/récepteurs du superviseur et de la voiture en apprentissage

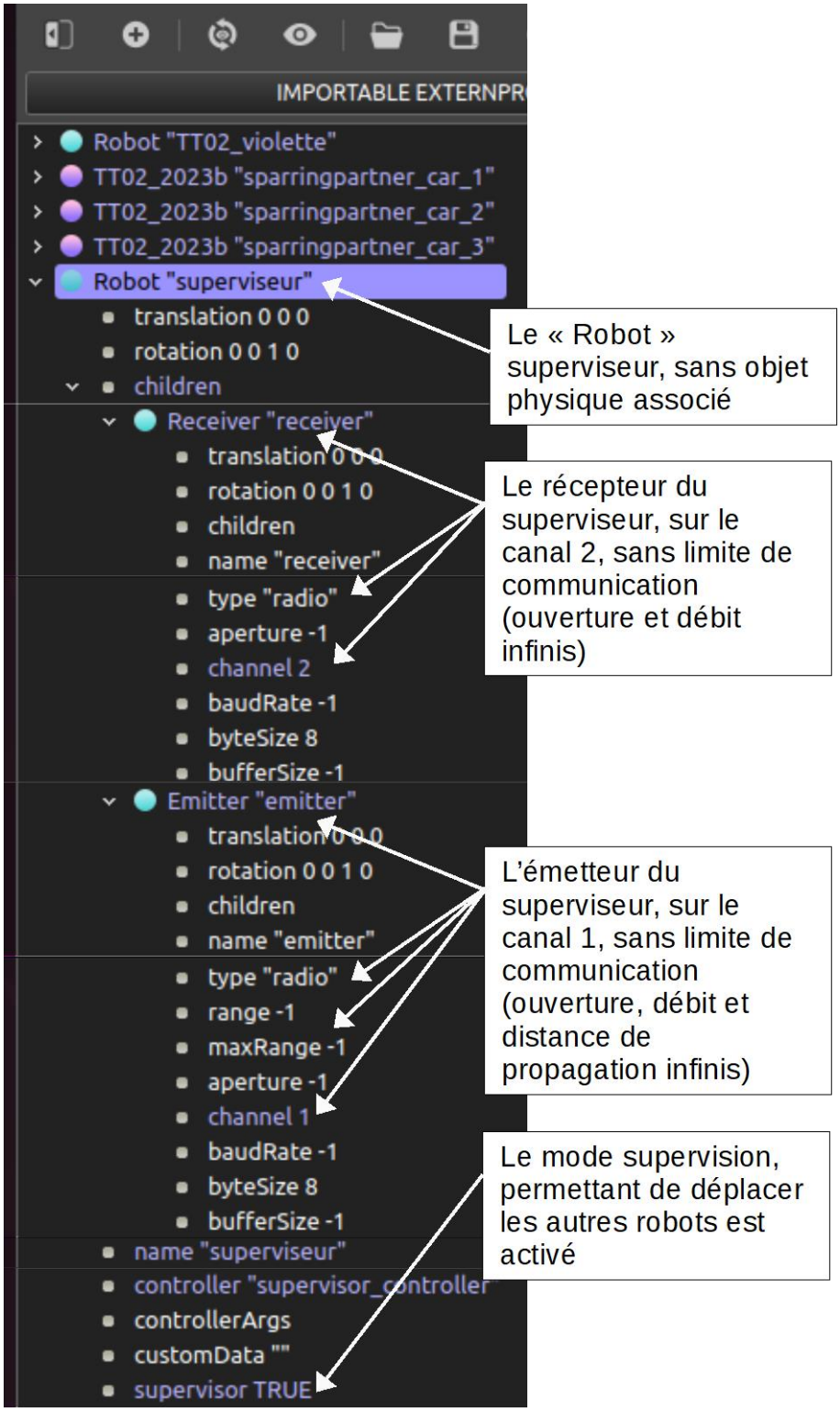


Figure 6: Le superviseur dans l'arborescence des éléments du projet

On détaille ici le code du superviseur :

```python
# Python imports
import random

# Webots imports
from controller import Supervisor

# Global constants
PI = 3.141592653589793
RECEIVER_SAMPLING_PERIOD = 64 # In milliseconds

# /!\ Set the number of sparring partner cars
NB_SPARRINGPARTNER_CARS = 3

# Clipping function
def value_clip(x, low, up):
    return low if x < low else up if x > up else x

# Angle normalization function (Webots angle values range between -pi and pi)
def angle_clip(x):
    a = x % (2*PI)
    return a if a < PI else a - 2*PI

# Positions intiales aléatoires
# (plusieurs positions initiales auxquelles s'ajoute un peu d'aléatoire) [[x_min, x_max], [y_min, y_max], rotation]
# Les angles correspondents au même sens de rotation pour les voitures

# Train track
starting_positions = [
    [[ 0.00,  4.00], [ 5.10,  5.30],  0.00],
    [[ 4.95,  5.15], [ 0.30,  4.00], -PI/2],
    [[ 2.70,  4.50], [-1.25, -0.75],  PI  ],
    [[ 3.20,  3.45], [ 2.00,  2.40],  PI/2],
    [[ 1.90,  2.70], [ 3.10,  3.30],  PI  ],
    [[ 0.00,  0.25], [-0.50,  1.50], -PI/2],
    [[-2.20, -1.90], [-0.50,  2.30],  PI/2]
]

"""
Test track
starting_positions = [
    [[ 0.60,  4.20], [ 5.40,  5.60],  0.00],
    [[ 5.40,  5.60], [-4.30,  4.30], -PI/2],
    [[ 1.40,  1.60], [-5.00, -4.30],  PI/2],
    [[ 3.40,  3.60], [-2.85, -1.45],  PI/2],
    [[ 3.40,  3.60], [ 2.20,  2.50],  PI/2],
    [[-0.60, -0.40], [ 0.30,  0.90], -PI/2],
    [[-3.20, -2.00], [-4.60, -4.40],  PI],
    [[-2.60, -2.40], [-0.80,  2.30],  PI/2],
]
"""

# Initialisation
supervisor = Supervisor()
basicTimeStep = int(supervisor.getBasicTimeStep())

# Receiver et emitter
receiver = supervisor.getDevice("receiver")
receiver.enable(RECEIVER_SAMPLING_PERIOD)
emitter = supervisor.getDevice("emitter")
packet_number = 0

# Recuperation des liens vers les noeuds voitures
tt_02 = supervisor.getFromDef("TT02_2023b_RL")
tt_02_translation = tt_02.getField("translation")
tt_02_rotation = tt_02.getField("rotation")
sparringpartner_car_nodes = [supervisor.getFromDef(f"sparringpartner_car_{i}") for i in range(NB_SPARRINGPARTNER_CARS)]
sparringpartner_car_translation_fields = [sparringpartner_car_nodes[i].getField("translation") for i in range(NB_SPARRINGPARTNER_CARS)]
sparringpartner_car_rotation_fields = [sparringpartner_car_nodes[i].getField("rotation") for i in range(NB_SPARRINGPARTNER_CARS)]

erreur_position = 0
```

Positions de départ. Le superviseur choisit pour la voiture apprenante et pour les voitures sparring partner une position dans les plages de valeur proposées avec un angle aléatoire autour de l'angle proposé.

Idem pour le circuit de validation

Configuration des émetteur et récepteur pour la communication avec la voiture apprenante

Récupération des liens vers la voiture apprenante et vers les voitures sparring partner pour contrôler ensuite leur position pour les débuts d'épisodes

Compteur des erreurs de réinitialisation

Figure 7: Première partie du code python du superviseur : initialisations

```
# Main loop
while supervisor.step(basicTimeStep) != -1:
    # detection de positions incoherentes. Ne devrait pas servir...
    if abs(tt_02_translation.getSFVec3f()[0]) > 20 or abs(tt_02_translation.getSFVec3f()[1]) > 20 or abs(tt_02_translation.getSFVec3f()[2]) > 0.1 :
        start_rot = [0.0, 0.0, 1.0, -PI/2]
        tt_02_rotation.setSFRotation(start_rot)
        tt_02_translation.setSFVec3f([2.98, 0.34, 0.039])
        erreur_position += 1
        supervisor.step(basicTimeStep)
        print("erreur position numero : ",erreur_position)

    # If reset signal : replace the cars
    if receiver.getQueueLength() > 0:
        # Get the data off the queue
        try :
            data = receiver.getString()
            receiver.nextPacket()
            print(data)
        except :
            print("souci de réception")

        # Choose driving direction
        direction = random.choice([0, 1])
        # Select random starting points
        coordinates_idx = random.sample(range(len(starting_positions)), 1+NB_SPARRINGPARTNER_CARS)

        # Replace TT-02 avec une position pseudo aleatoire
        # print("replacement de la voiture violette")
        coords = starting_positions[coordinates_idx[0]]
        start_x = random.uniform(coords[0][0], coords[0][1])
        start_y = random.uniform(coords[1][0], coords[1][1])
        start_z = 0.039

        angle = coords[2]
        start_angle = random.uniform(angle - PI/12, angle + PI/12)

        if direction:
            start_angle = start_angle + PI
        start_rot = [0.0, 0.0, 1.0, angle_clip(start_angle)]
        tt_02_rotation.setSFRotation(start_rot)
        tt_02_translation.setSFVec3f([start_x, start_y, start_z])

        packet_number += 1

        # Replace sparring partner cars
        for i in range(NB_SPARRINGPARTNER_CARS):
            coords = starting_positions[coordinates_idx[i + 1]]
            start_x = random.uniform(coords[0][0], coords[0][1])
            start_y = random.uniform(coords[1][0], coords[1][1])
            start_z = 0.039
            sparringpartner_car_translation_fields[i].setSFVec3f([start_x, start_y, start_z])
            # Rotate sparringpartner cars
            angle = coords[2]
            start_angle = random.uniform(angle - PI/12, angle + PI/12)
            if direction:
                start_angle = start_angle + PI
            start_rot = [0, 0, 1, angle_clip(start_angle)]
            sparringpartner_car_rotation_fields[i].setSFRotation(start_rot)

        #attente pour que les voitures se stabilisent dans la position de depart
        for i in range(20) :
            supervisor.step(basicTimeStep)
        emitter.send("voiture replacee num : " + str(packet_number))
```

Début de la boucle infinie

Si la voiture apprenante a une position aberrante (trop en hauteur ou en dehors de la piste, le superviseur la replace devant un mur pour provoquer un reset. Cela arrive parfois quand la voiture a une collision à grande vitesse

Si le superviseur reçoit un message (le message est envoyé par la voiture apprenante lors d'une collision)

Le superviseur récupère et affiche le message (le message est le numéro de la collision)

Le superviseur choisit un sens de circulation et 4 points de départ (pour la voiture apprenante et pour les 3 autres)

Le superviseur replace la voiture apprenante aléatoirement autour de la position de base du tableau tirée au sort ci-desssus.

Le superviseur replace les voitures saprring partners aléatoirement autour des positions de base du tableau tirées au sort ci-desssus.

Le superviseur envoie un message à la voiture apprenante pour indiquer que les replacements sont terminés

Figure 8: Seconde partie du code python du superviseur : la boucle principale

3.3 - Interface Gym :

Gym est une bibliothèque Python développée par OpenAI (dont l'évolution est depuis peu nommée Gymnasium, suivi par Farama Foundation) qui permet de gérer l'apprentissage par renforcement de manière normalisée, faisant le lien entre l'environnement (webots pour nous mais Gym en propose plusieurs) et l'algorithme d'apprentissage.

Pour plus de détails sur l'environnement Gym et son utilisation, se référer à l'article « *Introduction aux bibliothèques Gym et Stable-Baselines pour l'apprentissage par renforcement* » *[4]* du « Dossier Intelligence artificielle »

Gym (et numpy, demandé par Gym) s'installe à l'aide de la commande suivante :

```
pip3 install numpy
```

```
pip3 install gym
```

Dans le cas de la course de voitures autonomes, un premier travail est de faire le lien entre Gym et l'environnement webots décrit ci-dessus. Gym exige qu'un environnement contienne les fonctions suivantes :

- *get_observation()* : fonction renvoyant les observations de l'environnement.
- *get_reward()* : fonction donnant la récompense selon l'action effectuée par l'agent.
- *reset()* : fonction exécutant la démarche pour repartir au début d'un épisode.
- *step()* : fonction faisant évoluer l'environnement d'un pas.

Toutes ces fonctions sont rassemblées dans une classe nommée ici "WebotsGymEnvironment". Dans cette classe ont été rajoutées les 3 fonctions propres à la voiture, décrite dans l'article sur le simulateur, et commentée en annexe :

- *get_lidar_mm()* : fonction qui renvoie un tableau des valeurs acquises par le lidar dans le même format que la voiture réelle avec des valeurs cohérentes en mm.
- *set_vitesse_m_s()* : fonction qui prend en argument une vitesse en $m.s^{-1}$ pour contrôler la propulsion de la voiture. Cette fonction existe aussi sur la voiture réelle.
- *set_direction_degre()* : Fonction qui prend en argument un angle en degré pour contrôler la direction de la voiture. Cette fonction existe aussi sur la voiture réelle.

La fonction *get_observation()*

```
# Get Lidar observation
def get_observation(self,init=False):
        tableau_lidar_mm = self.get_lidar_mm()
        i = 0
        while(tableau_lidar_mm[0] == 0) and (i<50) :
                #on essaie d'avoir un tableau de valeur correct !
                self.nb_pb_acqui_lidar+=1
                print("souci d'aquisition lidar"+str(self.nb_pb_acqui_lidar))
                i = i+1
                tableau_lidar_mm = self.get_lidar_mm() #lidar en mm

        if init:
                current_lidar=tableau_lidar_mm.astype("float64")/12000
                previous_lidar=tableau_lidar_mm.astype("float64")/12000
                previous_speed=[0]
                previous_angle=[0]
        else:
                #grandeurs normalisées pour observation
                previous_lidar=self.observation["current_lidar"]
                current_lidar=tableau_lidar_mm.astype("float64")/12000
                previous_speed=[self.consigne_vitesse/VITESSE_MAX_M_S]
                # si on a un capteur de vitesse sur la voiture relle :
                # previous_speed=[(super().getTargetCruisingSpeed()/3.6)/VITESS
                previous_angle=[self.consigne_angle/MAXANGLE_DEGRE]
                # si on a un capteur de vitesse sur la voiture relle :
                # previous_speed=[(super().getSteeringAngle()*180/PI)/MAXANGLE

        observation = {
                        "current_lidar": current_lidar,
                        "previous_lidar":previous_lidar,
                        "previous_speed":previous_speed,
                        "previous_angle":previous_angle,
                      }
        # print(observation["current_lidar"])
        self.observation=observation
        return observation
```

Récupération du tableau des données lidar, en mm

Tant que le tableau est incorrect (on ne peut avoir 0 cm devant le lidar, on recommence l'acquisition

Au début d'un nouvel épisode, on met la valeur actuelle de tableau lidar dans la valeur du précédent.

On remplit les 4 champs de l'observation ; valeur actuelle du lidar, valeur précédente et valeurs des commandes de vitesse et de direction. Les valeurs sont normalisées de sorte d'être entre -1 et +1

On conserve observation dans self.observation (car la valeur actuelle du lidar sera la valeur précédente au prochain pas)

Figure 9: Code python de la fonction get_observation()

La fonction renvoie les valeurs actuelles normalisées du Lidar à "l'instant présent", les valeurs à "l'instant précédent", (récupérées de l'observation précédente), ainsi que les valeurs normalisées des commandes de direction et de vitesse. Si la fonction est appelée depuis la fonction reset() (init = True), on donne le même tableau pour les deux parties de l'espace d'observation concernant le Lidar puisque la voiture a été repositionnée.

Ce sont ces données qui seront les entrées du réseau de neurones.

La fonction get_reward()

```
# Reward function
def get_reward(self, obs):
        reward = 0
        done = False
        mini = 1
        #recherche de la distance la plus faible mesuree par le lidar entre -40et +40°
        for i in range(-40,40) :
                if (obs["current_lidar"][i] < mini and obs["current_lidar"][i]!=0) :
                        mini = obs["current_lidar"][i]
        # print(mini)
        #si le lidar touche un mur ou si la voiture va trop vite (chute en dehors du sol de la piste)
        if mini < 0.014 or super().getCurrentSpeed() > 30.0 : #0.014 <-> 160 m
                # Crash
                self.numero_crash += 1
                reward = -300
                done = True

        else:
                #Récompense pour une grande distance aux obstacles et une grande vitesse
                reward = 12 * (mini-0.014) + 3 * super().getTargetCruisingSpeed()
                #print("reward : "+str(reward))

        #Reset si la voiture a fait beaucoup de pas
        self.reset_counter += 1
        if self.reset_counter % RESET_STEP == 0:
                print("Reset")
                done = True

        return reward, done
```

On vérifie si il y a collision en regardant la distance la plus faible à l'avant de la voiture (entre -40° et +40°)

Si il y a collision (160 mm correspondant à la longueur du capot devant le lidar) ou si la vitesse de la voiture est aberrante, done = True et la récompense est très mauvaise (-300)

Si il n'y a pas collision, la récompense valorise une grande distance aux obstacles (une valeur de mini importante) et une grande vitesse.

Si la voiture a fait plus de 16384 pas, on demande à démarrer un nouvel épisode (done = True)

Figure 10: Code python de la fonction get_reward()

La fonction *get_reward()* attribue la récompense associée à l'état dans lequel se trouve la voiture et indique si l'épisode est terminé (variable done). On distingue deux états possibles pour la voiture. Le premier état est celui d'une collision. Dans le cas de la collision, on donne un malus de -300 "points". Le deuxième état regroupe toutes les situations autres que celle de collision. On donne comme récompense ici une valeur dépendant de la vitesse actuelle de la voiture ainsi que la distance minimale donnée par le tableau de Lidar à l'avant de la voiture. Les fonctions de récompenses seront détaillées plus tard. On indique aussi ici si l'on a terminé l'épisode via la variable done.

Le calcul de la récompense est très important dans l'apprentissage. Il faut donner des récompenses à la fois pour valider de petites avancées (être loin des obstacles) et pour atteindre l'objectif (aller vite). Trop d'importance donnée à la vitesse amène la voiture à aller très vite en ligne droite pour s'écraser au premier virage. Trop peu d'importance donnée à la vitesse amène la voiture à rouler moins vite, au milieu de la piste.

La fonction reset()

```python
# Reset the simulation
def reset(self):
        # Reset speed and steering angle et attente de l'arrêt de la voiture
        self.consigne_vitesse = 0.0
        self.consigne_angle = 0.0
        self.set_vitesse_m_s(self.consigne_vitesse )
        self.set_direction_degre(self.consigne_angle)
        for i in range(20) :
                        super().step()
        self.reset_counter = 0
        print("crash num : " +str(self.numero_crash))

        if(self.numero_crash != 0):
                #attente de l'arrêt de la voiture
                while abs(super().getTargetCruisingSpeed()) >= 0.001 :
                        print("voiture pas encore arrêtée")
                        super().step()

                # Return an observation
                self.packet_number += 1
                # Envoi du signal de reset au Superviseur pour replacer les voitures
                self.emitter.send("voiture crash numero " + str(self.packet_number))
                super().step()
                #attente de la remise en place des voitures
                while(self.receiver.getQueueLength() == 0) :
                        self.set_vitesse_m_s(self.consigne_vitesse )
                        super().step()

                data = self.receiver.getString()
                self.receiver.nextPacket()
                print(data)

                self.consigne_vitesse = 0
                self.consigne_angle = 0
                self.set_vitesse_m_s(self.consigne_vitesse )
                self.set_direction_degre(self.consigne_angle)
                #on fait quelques pas à l'arrêt pour stabiliser la voiture si besoin
                while abs(super().getTargetCruisingSpeed()) >= 0.001 :
                        print("voiture pas arrêtée")
                        self.set_vitesse_m_s(self.consigne_vitesse )
                        super().step()

        return self.get_observation(True)
```

Arrêt de la voiture (avec quelques pas de simulation pour qu'elle s'arrête réellement), remise à zéro des compteurs

Attente de l'arrêt de la voiture (a priori inutile désormais)

Envoi du message au superviseur indiquant qu'il faut lancer un nouvel épisode (et donc repositionner les voitures)

Attente du message du superviseur indiquant que les voitures sont repositionnées (on en profite pour arrêter de nouveau la voiture, ce qui devrait être inutile normalement…)

Affichage du message reçu

Arrêt de la voiture, parfois mise en mouvement par le déplacement par le superviseur

Figure 11: Code python de la fonction reset()

La fonction *reset()* est lancée dans le cas d'une collision de la voiture ou si le nombre d'actions autorisées par épisode est atteint. La fonction reset() démarre un nouvel épisode. Pour cela, elle envoie un message au superviseur et attend qu'il ait fini le repositionnement des voitures. En plus de cela, plusieurs instructions visent à arrêter les voitures. En effet, le déplacement par le superviseur d'une voiture non arrêtée engendre des mouvements incohérents, d'où les multiples instructions d'arrêt de la voiture et d'attente qu'elle soit stabilisée.

la fonction *step()*

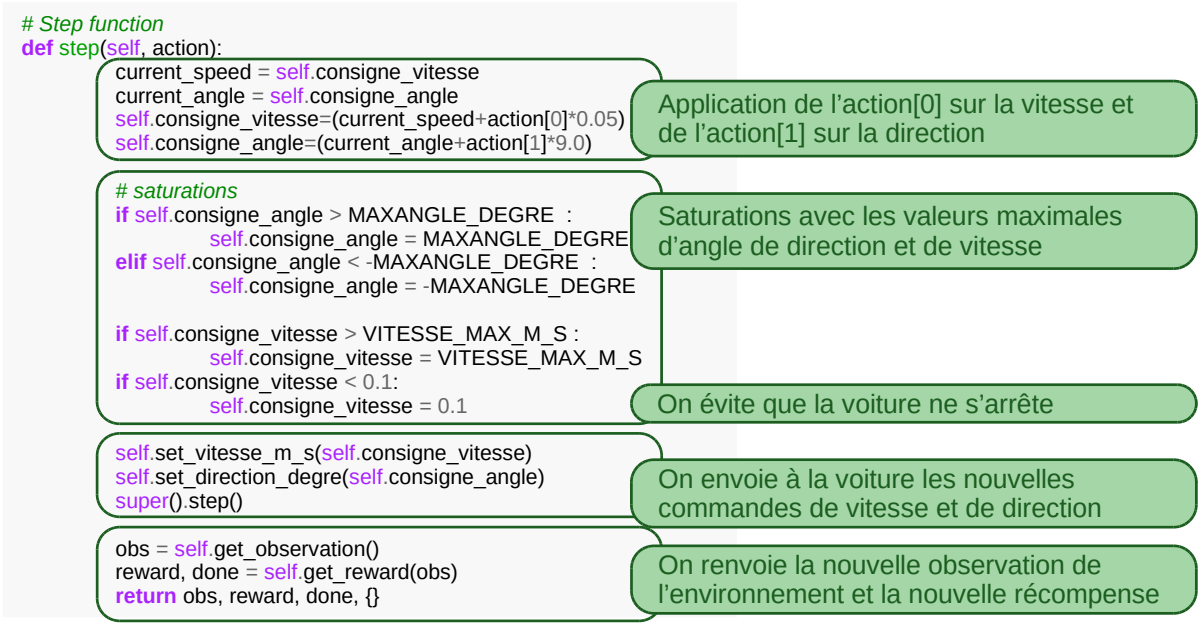


Figure 12: Code python de la fonction step()

La fonction *step()* correspond à un pas dans le processus d'apprentissage. Dans cette fonction on fait avancer la voiture avec les actions issues de la politique d'action de l'agent (compromis entre hasard/exploration et exploitation du réseau de neurones pendant l'apprentissage). Ensuite, on récupère une observation depuis le Lidar et on calcule la récompense avant de retourner toutes les informations obtenues.

Une fois l'interface avec l'environnement au format Gym, il est possible d'utiliser une bibliothèque d'apprentissage par renforcement.

3.4 - Bibliothèque Stable BaseLines 3 :

La bibliothèque Stable-Baselines 3 propose plusieurs algorithmes d'apprentissage par renforcement, plus ou moins adaptés pour des problèmes différents. Après plusieurs essais, c'est l'algorithme PPO (Proximal Policy Optimization) qui a donné les meilleurs résultats. Il se caractérise par une évolution des paramètres évitant les discontinuités dans les résultats.

Pour installer stable baselines 3, on utilise la commande :

```
pip3 install stable-baselines3
```

La bibliothèque Stable-Baselines 3 permet de plus de créer et gérer le réseau de neurones avant d'utiliser l'algorithme d'apprentissage pour en optimiser les poids. Elle propose un large choix de paramètres pour l'apprentissage.

Pour plus de détails sur l'environnement Gym et son utilisation, se référer à l'article « Introduction aux bibliothèques Gym et Stable-Baselines pour l'apprentissage par renforcement » [4] du « Dossier Intelligence artificielle »

Voici un descriptif de ces paramètres ainsi que la valeur utilisée dans le programme de référence :

- ***policy*** : Type de politique utilisée. Ici, pour de multiples entrées ('MultiInputPolicy')
- ***env*** : Environnement d'apprentissage Gym

- ***learning_rate*** : Facteur d'apprentissage, il correspond à l'importance donné à un nouveau pas par rapport à ce qui a été acquis avant. Plus il est faible, plus l'apprentissage est lent et stable (5.10−4)
- ***n_steps*** : nombre d'actions autorisées par épisode (2048 par défaut)
- ***batchsize*** : Taille du batch (64)
- ***n_epochs*** : Nombre d'epoch (10)

— *gamma* : Facteur d'actualisation (0.99)

— *gae_lambda* : Facteur permettant de choisir entre le biais et la variance de l'estimateur de récompense (0.95)

— *clip_range* : équivalent à la valeur ϵ décrite dans le paragraphe PPO. Une fonction peut être renseignée (0.02)

— *clip_rang_vf* : La même que précédemment mais spécifique à OpenAI. (None)

— *normalize_advantage* : Si l'on normalise l'avantage ou pas (True)

— *ent_coef* : Coefficient d'entropie pour la fonction de perte (0)

— *vf_coef* : Coefficient de la fonction de perte (0.5)

— *max_grad_norm* : Valeur maximale du gradient de la fonction clipping (0.5)

— *use_sde* : Utilisation de la fonction gSDE au lieu de l'exploration aléatoire d'action (False)

— *sde_sample_freq* : Fréquence d'échantillonnage de matrice de bruit lorsque la fonction gSDE est utilisée (-1)

— *target_kl* : Valeur limite de la divergence KL entre deux mise à jour. Il n'y a pas de limite par défaut. (None)

— *state_window_size* : Taille de la fenêtre affichant la longueur moyenne d'épisode et la récompense moyenne

— *tensorboard_log* : Emplacement pour enregistrer les données de Tensorboard (None)

— *policy_kwargs* : Arguments additionnels lors de la création de la politique (None) — verbose : Affichage d'informations (1)

— *device* : Composant sur lequel faire tourner l'algorithme ("cpu")

— *_init_setup_model* : Construction du réseau à la création de l'instance. (True)

Il est nécessaire de décrire les espaces d'action et d'observation dans la classe de l'environnement Gym mentionné plus haut. Dans notre cas, nous optons pour l'espace d'action :

— un scalaire compris entre -1 et 1 pour l'incrément en vitesse

— un scalaire compris entre -1 et 1 pour l'incrément en angle

Les valeurs sont normalisées car Stable-Baselines3 préfère des valeurs normalisées. Ce n'est que par la suite que l'on multiplie par 0.5 m/s pour la vitesse et 9° pour l'angle.

Concernant l'espace d'observation, nous avons choisi le suivant :

— Un tableau de scalaire compris entre 0 et 1 pour les données actuelles du Lidar

— Un tableau de scalaire compris entre 0 et 1 pour les données précédentes du Lidar

— Un scalaire compris entre 0 et 1 pour la vitesse actuelle de la voiture

— Un scalaire compris entre -1 et 1 pour la direction actuelle de la voiture

Là encore on normalise les valeurs en divisant par leur valeurs maximales possibles respectifs. Concernant la vitesse et l'angle, on utilise les fonctions de Webots qui permettent d'accéder à ces grandeurs.

La libraire permet de lancer l'apprentissage grâce à la fonction learn(). Cette fonction prend en paramètre le nombre d'épisodes que va comporter la phase d'apprentissage. Une fois l'apprentissage terminé, il est possible de sauvegarder le réseau de neurones ce qui est particulièrement intéressant pour notre projet dans le but de charger ce réseau dans la voiture réelle. La fonction load() permet notamment de charger un réseau de neurones.

3.5 - Tensorboard

```
pip3 install tensorboard
```

3.6 - Fonctions de récompense :

La fonction de récompense est un des hyperparamètres les plus importants dans un processus d'apprentissage par renforcement. Elle est aussi un des plus problématiques étant donné son côté aléatoire. En effet, c'est la récompense qui influe sur comment l'apprentissage s'effectue. Par exemple, si l'on ne sanctionne pas suffisamment le crash de la voiture alors celle-ci pourrait avoir tendance à foncer dans le mur. Autre cas : si l'on ne favorise pas la vitesse alors la voiture aura tendance à rouler lentement voire être complètement à l'arrêt. Il est donc nécessaire de trouver une bonne fonction de récompense pour avoir le meilleur comportement possible. Dans notre cas, on a choisi de donner un malus de -300 lors d'un crash et dans toutes les autres situations la fonction suivante :

$$reward = 12 \times mini + 3 \times vitesse$$

avec *mini* est la distance la plus courte du tableau de Lidar.

Ici cherche à fortement pénaliser en cas de collision avec le décor ou une autre voiture. De plus, on veut que la voiture aille le plus vite possible et que celle-ci s'éloigne le plus possible des bords. Nous verrons par la suite dans les résultats si cette fonction de récompense a été concluante.

4 - Exemples d'apprentissage :

Après avoir expliqué le principe du simulateur et des bibliothèques nécessaires pour faire l'apprentissage par renforcement, nous détaillerons dans cette partie les étapes que nous avons suivi pour entrainer un réseau de neurones.

Dans le cas de la voiture autonome, il est tout d'abord nécessaire de définir un espace d'observation cohérent avec la réalité car toutes les grandeurs ne sont pas mesurables sur la voiture. Ainsi, pour l'espace d'observation, nous pouvons nous baser sur le Lidar qui permet de donner la distance des différents obstacles. De plus, si l'on avait mis en place l'asservissement de vitesse, alors la vitesse réelle pouvait être un élément de cet espace. Dans le cas de l'espace d'action, seuls deux éléments sont possibles. En effet, les actions commandent les actionneurs de la voiture, à savoir le servomoteur pour la direction ainsi que le moteur à courant continu pour la propulsion. Il a donc été décidé que le réseau de neurones donnera un incrément d'angle pour la direction et un incrément de vitesse pour la propulsion.

Quelques algorithmes d'entraînement de réseau de neurones sont présents dans la Library Python Stable-Baselines3 :

— "Proximal Policy Optimization" (PPO) : Algorithme de type "policy-based"
— "Deep Q-Network" (DQN) : Algorithme de type "valued-based"
— "Soft Actor-Critic" (SAC) : Algorithme de type "actor-critic"

Au regards, de nombreux tests, l'algorithme qui a été retenu pour la suite est l'algorithme PPO, qui converge, entre autres, le plus rapidement.

4.1 - Fonctionnement de l'algorithme PPO :

L'algorithme PPO *[5]* a pour but d'être facile à implémenter, d'être facile à paramétrer et d'avoir une bonne efficacité au niveau des échantillons. Cet algorithme est un algorithme de type "policy-based" avec lequel on peut utiliser un ensemble d'action discret ou continu. On cherche à calculer la récompense cumulative Gt avec pour expression :

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

Ici r_k est la récompense obtenue à l'instant k et le facteur d'actualisation $\gamma \in [0, 1]$. La convergence est assurée car $\gamma < 1$. On indique ici que les récompenses obtenues dans longtemps soient moins impactantes.

PPO possède une partie qui a pour but de choisir l'action suivant l'état obtenu en entrée. Une autre partie est chargée de déterminer Gt à l'instant t. On définit alors l'estimation d'avantage At qui est la différence entre la valeur de Gt réelle calculée après la prise de décision et la valeur estimée par le réseau de neurones. Ainsi on cherche à savoir si l'action a été meilleure ou pire par rapport à ce qui était attendu.

On définit la perte comme étant $E_t\left[log\left(P_\pi(a_t|s_t)\right)A_t\right]$ où $P_\pi(a_t|s_t)$ est la probabilité que l'action a_t est choisie dans l'état st par le réseau de neurones suivant la politique π.

L'algorithme PPO utilise le principe de *"Trust Region Policy Optimization"*. Ce principe s'assure que la politique π_{old} ne soit pas trop de la politique actualisée π.

On considère deux distributions de probabilités P et Q sur le même espace x. On définit alors la divergence Kullback-Leibler (divergence KL) de Q à P comme :

$$KL[P, Q] = E_{xP}\left[log\left(\frac{P(X)}{Q(X)}\right)\right]$$

Cela permet de mesurer la distance de la distribution Q par rapport à la véritable distribution P. L'objectif de TRPO est alors de maximiser la quantité $E_t\left[\frac{P_\pi(a_t|s_t)}{P_{\pi_{old}}(a_t|s_t)}A_t\right]$ étant donné un paramètre δ tel que :

$$E_t\left[KL\left[P_{\pi_{old}}(.|s_t), P_\pi(.|s_t)\right]\right] \leq \delta$$

On a donc pour objectif de ne pas trop s'éloigner des anciennes politiques et de trouver la meilleure politique π

Pour ce qui de l'algorithme PPO, l'objectif devient le suivant : Maximiser la fonction L CLIP (π) avec

$$L^{CLIP}(\pi) = E_t[min(r_t(\pi)A_t, clip(r_t(\pi), 1-\varepsilon, 1+\varepsilon)A_t)]$$

Avec $r_t(\pi) = \frac{P_\pi(a_t|s_t)}{P_{\pi_{old}}(a_t|s_t)}$ et clip la fonction qui borne la fonction rt(π) entre 1 − ϵ et 1 + ϵ

4.2 - Modèle de réseau de neurones :

Le réseau de neurones que nous avons utilisé est composé de deux sous-couches("Hidden layers" en anglais). Les neurones d'entrée ("Input layer") sont au nombre de 360, qui correspondent aux 360 valeurs données par le Lidar. Les neurones de sortie sont au nombre de deux correspondant à la commande de vitesse et la commande de direction. Le modèle de réseau de neurones par défaut de l'algorithme PPO implémenté dans la librairie Stable-baselines3 possède déjà deux sous-couches. Il s'agit d'un élément qui n'a pas été modifié au cours du projet.

Dans le but d'améliorer le comportement de la voiture, d'autres données ont été rajoutées dans l'espace d'observation : d'une part les données du Lidar à "l'instant précédent" ont été ajoutées afin de donner une continuité dans l'observation des obstacles. De l'autre, la vitesse et la direction de la voiture avant une nouvelle commande du réseau de neurones ont été ajoutées.

Le schéma ci-dessous illustre le réseau de neurones utilisé. Ce réseau de neurones est celui qui a fourni les meilleurs résultats par ceux qui sont détaillés dans cette partie, notamment au niveau de la voiture réelle.

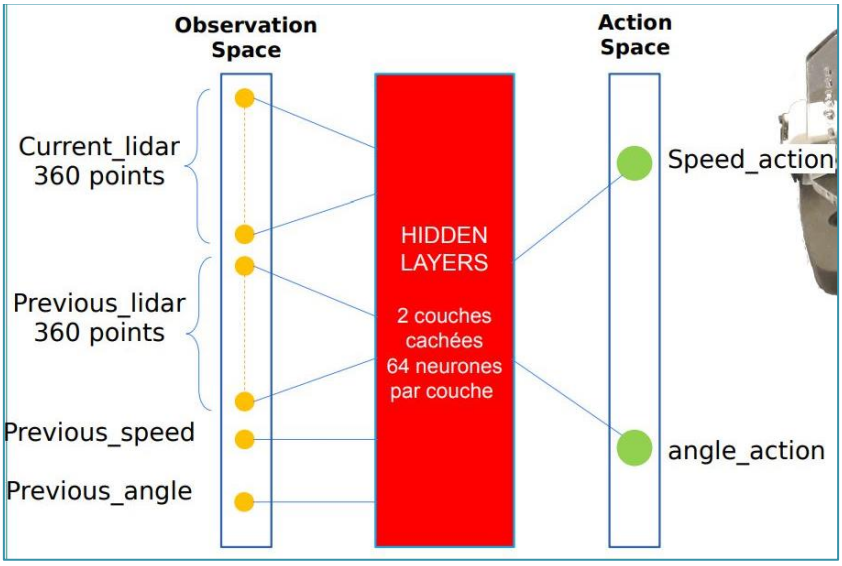


Figure 6 : Modèle du réseau de neurones utilisé

4.3 - Mise en place de l'entraînement du réseau de neurones:

Dans un premier temps, on déclare un environnement Gym, qui est celui détaillé dans la partie précédente. On utilise la fonction check() de Gym pour vérifier si la classe d'environnement est conforme à ce qu'attend la bibliothèque d'un environnement créé. Une fois cette étape passée, on déclare le modèle PPO avec de multiples grandeurs d'entrée ainsi que spécifier l'environnement instancié précédemment.

On lance un apprentissage avec 1.000.000 d'épisodes. Ce nombre peut sembler assez élevé, mais il est nécessaire de donner un grand nombre d'épisodes pour que la voiture puisse avoir le temps d'apprendre.

4.4 - Résultats obtenus :

Le tableau ci-dessous présente d'une part les résultats obtenus avec le modèle de réseau de neurones sur la piste d'entrainement, de l'autre les résultats obtenu en contre-la-montre sur la piste de validation :

Réseau de neurones sur la piste d'entrainement		
Réseau N°	**Particularité**	**Observation sur la piste d'entrainement**
1	Configuration de base	- Evite les obstacles - Percute parfois les autres voitures - Dépasse les voitures même dans les lignes droites
2	Lors d'un crash : $reward = -300 - n°_{crash}$	- Capable de faire plusieurs tours sans se crasher - Peut éviter les autres voitures et les dépasser - Ralenti pas mal dans les virages
3	500 000 épisodes pour l'apprentissage	- Pas de différence notable avec les autres réseau de neurones
4	*gae_lambda=0.90*	- Mouvement un peu hasardeux - Ne tourne pas certaines fois
5	*ent_coef=0.02*	- Fait plusieurs tour sans crash - Roule un peu moins vite que les autres réseau des neurones

6	*ent_coef=0.02* *angle entre -2° et 2°*	- A réussi à faire un tour complet - Lente dans les virages - Assez réactive dans les lignes droites

Réseaux de neurones sur la piste de validation

Réseau N°	Particularité	Temps de passage (en minutes)	Observation sur la piste de validation (sans obstacles)
1	Configuration de base	1 :03 :74	- Ne s'est pas crashée - Ralenti dans les virages
2	Lors d'un crash : $reward = -300 - n°_{crash}$	1 :31 :26	- S'est crashée une fois - A réussi mieux dans un sens - Ralenti dans les virages
3	500 000 épisodes pour l'apprentissage	2 :05 :93	- S'est crashée une fois - A réussi mieux dans un sens - Ralenti dans les virages
4	*gae_lambda=0.90*	Aucun	- S'est crashée plusieurs fois - N'arrive pas à gérer les virages après les lignes droites - Voiture la plus rapide mais incapable de faire un tour de piste
5	*ent_coef=0.02*	2 :25 :27	- Ralenti beaucoup dans les lignes droites - Réagi bien dans les virages
6	*ent_coef=0.02* *angle entre -2° et 2°*	2 :17 :49	- N'avance pas très vite - Oscille moins dans les lignes droites

Au vu de ces résultats, l'on peut en conclure que le réseau de neurone n°1 est celui qui donne les résultats les plus satisfaisant. En revanche, lors du passage à la réalité, c'est le réseau n°5 qui a tout de même été choisi puisqu'il a un meilleur comportement sur le simulateur. Dans la réalité, il est préférable de choisir un réseau de neurones le plus stable possible (où la voiture ne crashe pas) au détriment d'une vitesse moins importante.

5 - Passage à la réalité:

Il s'agit ici de la dernière étape du projet, celui où l'on veut faire tourner le réseau de neurones sur la voiture réelle. Cette étape intervient après que le réseau de neurones a été entraîné sur le simulateur.

5.1 - Implémentation dans la voiture réelle :

Pour s'assurer que les commandes données par le réseau de neurones soit compréhensible par le programme de la voiture. On a donc développé deux fonctions similaires à celles présentes dans le simulateur : *set_vitesse_m_s()* et *set_direction_degre()*. Cette fois, il s'agit de passer des commandes en m/s et en degrés en commandes de PWM. De plus, pour ce qui est du Lidar, on s'assure de ne pas laisser de valeur à 0 dans la partie "utile" du Lidar (les valeurs des indices 0 à 99 et celles de 260 à 359). Pour cela, si une valeur 0 apparaît entre deux valeurs non nulles, ont la considère comme étant la moyenne de ces deux dernières. Dans le cas où il y a une série de 0, on copie la valeur non nulle précédente pour compenser. On se retrouve ainsi dans les mêmes dispositions que la simulation au niveau du Lidar.

Concernant le réseau de neurones, il a pu être charger grâce la fonction load() de Stable-Baselines3. Il est possible d'utiliser le réseau grâce à la fonction predict() qui prend en argument

les observations et qui renvoie les actions. Les observations sont stockées sous forme de dictionnaire. Cependant, contrairement à la simulation, l'accès direct à la vitesse et à la direction n'est pas possible car il n'y a pas d'asservissement. Nous ne pouvons donner que les consignes en vitesse et en angle à l'instant précédent en tant qu'observation. Cela marque la principale différence entre la simulation et les conditions réelles.

Pour utiliser le Lidar, une classe Lidar() a été créee permettant de l'initialiser, le démarrer ainsi qu'acquérir les valeurs. Au démarrage du programme, on lance un thread *thread_scan_lidar* qui permet de faire l'acquisition des données du Lidar en continue. On active et désactive un drapeau dans le code pour indiquer lorsque l'on veut récupérer ces données acquises. Le traitement de ces données comme décrit plus haut se fait en dehors de ce thread.

En parallèle de ce thread, on lance un second thread *thread_conduite_autonome* qui est le programme de pilotage de la voiture. Dans la fonction *conduite_autonome(),* on commence par récupérer les données du Lidar puis on procède au traitement de ces données. On récupère les valeurs traitées tous les 20 degrés pour faciliter l'analyse des obstacles. Dans le cas où l'on considère qu'il y a collision (valeur minimale inférieur à un certain seuil), on recule dans la direction opposée à l'obstacle détectée (indice de cette valeur minimale). Si ce n'est pas la cas, on laisse la main au réseau de neurones pour le pilotage.

Il se passe un phénomène qui n'est pas voulu après le recule où, au démarrage, la voiture n'avance plus. En effet, le réseau de neurones envoie des incréments négatifs de vitesse comme consigne. Cependant, nous n'autorisons pas le réseau de neurones à effectuer la marche arrière. On se retrouve donc bloquer car, les observations restant les mêmes, rien ne permet au réseau de neurones de faire augmenter la vitesse. Il a donc été décidé de mettre une valeur minimale de 0.1 m/s pour éviter cette situation de blocage.

5.2 - Résultats obtenus:

Les résultats présentés ici sont ceux de la voiture réelle avec le réseau de neurones avec *ent_coef = 0.02*. En effet, il s'agit de celui qui a montré le meilleur comportement en réel.

Les résultats sont ceux réalisés le jour de la course de voitures autonomes. Une première phase concerne une piste sans obstacle où le temps comptabilisé est le temps que prend la voiture pour réaliser deux tours.

Figure 7 : 1ère piste de qualification

Les résultats obtenus sont les suivants :

	1er passage	**2ème passage**
Temps de passage (2 tours de piste)	31 sec	25 sec

Nous avons pu observer que sans aucun obstacle, la voiture réussit très bien à se diriger et ne prend aucun mur pendant son tour. L'apprentissage est donc concluant sur ce point puisque la conduite est très satisfaisante et conforme à ce qui est attendu. En revanche pour atteindre ce résultat, une réduction du coefficient la consigne de vitesse s'imposait. En effet, sans cette correction, la voiture allait trop vite et se prenait quelques fois les murs dans les tests que j'effectuais. Le problème de cette correction est que sans asservissement de vitesse, celle-ci dépend de l'état de charge de la batterie. In fine, la voiture réagit bien mais est assez lente en comparaison avec les autres voitures. Aussi, on peut observer que dans les lignes droites, la voiture a tendance à vaciller contrairement à la simulation.

La deuxième phase se fait avec une nouvelle piste et des obstacles fixes, comme présenté ci-dessous.

Figure 8 :2ème piste de qualification

Les résultats sont les suivants :

	1er passage	**2ème passage**
Temps de passage (2 tours de piste)	32 sec	34 sec

En présence d'obstacles fixes, la voiture se comporte un peu moins bien. En effet, celle-ci aura tendance à ne pas suffisamment tourner pour éviter l'obstacle. En revanche, cela est une bonne situation pour tester la marche arrière couplée au réseau de neurones. La voiture a pu finir ses deux tours malgré des collisions avec les obstacles. On peut conclure que la phase d'entraînement sur simulateur n'a pas suffisamment d'obstacle fixe. Enfin, la dernière partie de l'évènement était la course en elle-même. Là encore, il y a eu deux courses. Ici pas de chrono à présenter mais quelques observations sont possibles. Lors de la première course, la voiture n'a pas pu finir la course à la suite d'un carambolage et une conduite à contre-sens. Cela permet de conclure qu'il pourrait être utile d'inclure ce genre de situation dans le simulateur. Durant la deuxième course cependant, la voiture a pu terminer la course en 2e position.

Conclusion :

Pistes d'améliorations : mesure de la vitesse réelle et mesure de la position réelle de la direction (super().getTargetCruisingSpeed() et semblable pour direction dans le simulateur)

Ajouter une détection de l'angle aberrant de la voiture apprenante dans le superviseur.

Références :

[1]: Introduction à l'apprentissage par renforcement
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14756/14756-introduction-lapprentissage-par-renforcement-ensps_0.pdf

[2]: Wei Zhu, Xian Guo, Dai Owaki, Kyo Kutsuzawa, Mitsuhiro Hayashibe. "A Survey of Sim-to-Real Transfer Techniques Applied to Reinforcement Learning for Bioinspired Robots". In : IEEE Transactions on Neural Networks and Learning Systems (sept. 2021).
https://ieeexplore.ieee.org/document/9552429

[3]: GitHub public sur les voitures autonomes
https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay

[4]: Introduction aux bibliothèques Gym et Stable-Baselines pour l'apprentissage par renforcement
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14764/14764-introduction-aux-bibliotheques-pour-lapprentissage-par-renforcement-ensps.pdf

[5]: John Schlman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. "Proximal Policy Optimization Algorithms". In : (2017).
https://arxiv.org/pdf/1707.06347.pdf

[6]: Site web de la course de voitures autonomes de Paris Saclay : https://ajuton-ens.github.io/CourseVoituresAutonomesSaclay/

Ressource publiée sur Culture Sciences de l'Ingénieur : https://eduscol.education.fr/sti/si-ens-paris-saclay

6 - Annexes

La fonction get_lidar_mm()

```
def get_lidar_mm(self):
        tableau_lidar_mm = np.zeros(360)
        try :
                donnees_lidar_brutes = np.array(self.lidar.getRangeImage()) #lidar en mm
                for i in range(0,101) :
                        if (donnees_lidar_brutes[-i]>0) and (donnees_lidar_brutes[-i]<12) :
                                tableau_lidar_mm[i] = 1000*donnees_lidar_brutes[-i]
                        else :
                                tableau_lidar_mm[i] = 0
                for i in range(101,260) : #zone correspondant à l'habitacle de la voiture
                        tableau_lidar_mm[i] = 0
                for i in range(260,360) :
                        if (donnees_lidar_brutes[-i]>0) and (donnees_lidar_brutes[-i]<12) :
                                tableau_lidar_mm[i] = 1000*donnees_lidar_brutes[-i]
                        else :
                                tableau_lidar_mm[i] = 0
        except :
                self.nb_pb_lidar+=1
                print("souci lidar"+str(self.nb_pb_lidar))
        return tableau_lidar_mm
```

Récupération des données du lidar webots (distance en m et angles dans le sens horaires

Insertion des données acquises entre -100° et +100° dans un tableau en mm. Les indices du tableaux sont les angles de mesure, dans le sens trigonométrique

Gestion des éventuelles erreurs

Figure 13: Code python de la fonction get_lidar_mm

Les fonctions set_vitesse_m_s() et set_direction_degre()

```
def set_vitesse_m_s(self,vitesse_m_s):
        speed = vitesse_m_s*3.6
        if speed > MAXSPEED :
                speed = MAXSPEED
        if speed < 0 :
                speed = 0
        super().setCruisingSpeed(speed)
```

Conversion de la vitesse en km.h-1 et saturations

Exécution de la fonction webits de contrôle de la direction de la voiture simulée

```
def set_direction_degre(self,angle_degre):
        if angle_degre > MAXANGLE_DEGRE:
                angle_degre = MAXANGLE_DEGRE
        elif angle_degre < -MAXANGLE_DEGRE:
                angle_degre = -MAXANGLE_DEGRE
        angle = -angle_degre * PI/180
        super().setSteeringAngle(angle)
```

Saturation de l'angle aux valeurs maximales acceptables par la direction et conversion en radians

Exécution de la fonction webits de contrôle de la direction de la voiture simulée

Figure 14: Code python des fonctions set_vitesse_m_s() et set_direction_degre()

Sans step après observation 146 erreur de position t avec angle max = 16

```
crash num : 4003
souci de réception
crash num : 4004
souci de réception
crash num : 4005
souci de réception
----------------------------------------
| rollout/              |              |
|    ep_len_mean        | 1.01e+03     |
|    ep_rew_mean        | 5.87e+03     |
| time/                 |              |
|    fps                | 83           |
|    iterations         | 458          |
|    time_elapsed       | 11210        |
|    total_timesteps    | 937984       |
| train/                |              |
|    approx_kl          | 0.018797053  |
|    clip_fraction      | 0.168        |
|    clip_range         | 0.2          |
|    entropy_loss       | 0.204        |
|    explained_variance | 0.884        |
|    learning_rate      | 0.0005       |
|    loss               | 276          |
|    n_updates          | 4570         |
|    policy_gradient_loss | 0.00151    |
|    std                | 0.219        |
|    value_loss         | 2.59e+03     |
----------------------------------------
crash num : 4006
souci de réception
crash num : 4007
souci de réception
crash num : 4008
souci de réception
crash num : 4009
souci de réception
----------------------------------------
| rollout/              |              |
|    ep_len_mean        | 979          |
|    ep_rew_mean        | 5.59e+03     |
| time/                 |              |
|    fps                | 83           |
|    iterations         | 459          |
|    time_elapsed       | 11229        |
|    total_timesteps    | 940032       |
| train/                |              |
|    approx_kl          | 0.014305837  |
|    clip_fraction      | 0.151        |
|    clip_range         | 0.2          |
|    entropy_loss       | 0.224        |
|    explained_variance | 0.925        |
|    learning_rate      | 0.0005       |
|    loss               | 760          |
|    n_updates          | 4580         |
|    policy_gradient_loss | -0.00319   |
|    std                | 0.216        |
|    value_loss         | 2.39e+03     |
----------------------------------------
crash num : 4010
souci de réception
crash num : 4011
souci de réception
crash num : 4012
souci de réception
crash num : 4013
souci de réception
crash num : 4014
souci de réception
crash num : 4015
souci de réception
----------------------------------------
| rollout/              |              |
|    ep_len_mean        | 962          |
|    ep_rew_mean        | 5.5e+03      |
| time/                 |              |
|    fps                | 83           |
|    iterations         | 460          |
|    time_elapsed       | 11250        |
|    total_timesteps    | 942080       |
| train/                |              |
|    approx_kl          | 0.039682604  |
|    clip_fraction      | 0.229        |
|    clip_range         | 0.2          |
|    entropy_loss       | 0.235        |
|    explained_variance | 0.935        |
|    learning_rate      | 0.0005       |
|    loss               | 182          |
|    n_updates          | 4590         |
|    policy_gradient_loss | 0.00216    |
|    std                | 0.217        |
|    value_loss         | 1.32e+03     |
----------------------------------------
crash num : 4016
souci de réception
crash num : 4017
souci de réception
crash num : 4018
souci de réception
crash num : 4019
souci de réception
crash num : 4020
souci de réception
crash num : 4021
souci de réception
```

```
crash num : 4022
souci de réception
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 853       |
|    ep_rew_mean        | 5.03e+03  |
| time/                 |           |
|    fps                | 83        |
|    iterations         | 461       |
|    time_elapsed       | 11270     |
|    total_timesteps    | 944128    |
| train/                |           |
|    approx_kl          | 0.048693582 |
|    clip_fraction      | 0.202     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.234     |
|    explained_variance | 0.97      |
|    learning_rate      | 0.0005    |
|    loss               | 99.2      |
|    n_updates          | 4600      |
|    policy_gradient_loss | 0.00254 |
|    std                | 0.216     |
|    value_loss         | 1.06e+03  |
-----------------------------------------
crash num : 4023
souci de réception
crash num : 4024
souci de réception
crash num : 4025
souci de réception
crash num : 4026
souci de réception
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 784       |
|    ep_rew_mean        | 4.73e+03  |
| time/                 |           |
|    fps                | 83        |
|    iterations         | 462       |
|    time_elapsed       | 11289     |
|    total_timesteps    | 946176    |
| train/                |           |
|    approx_kl          | 0.026715096 |
|    clip_fraction      | 0.193     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.237     |
|    explained_variance | 0.957     |
|    learning_rate      | 0.0005    |
|    loss               | 506       |
|    n_updates          | 4610      |
|    policy_gradient_loss | 0.00143 |
|    std                | 0.216     |
|    value_loss         | 2.07e+03  |
-----------------------------------------
crash num : 4027
souci de réception
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 794       |
|    ep_rew_mean        | 4.84e+03  |
| time/                 |           |
|    fps                | 83        |
|    iterations         | 463       |
|    time_elapsed       | 11306     |
|    total_timesteps    | 948224    |
| train/                |           |
|    approx_kl          | 0.012957906 |
|    clip_fraction      | 0.124     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.234     |
|    explained_variance | 0.907     |
|    learning_rate      | 0.0005    |
|    loss               | 1.01e+03  |
|    n_updates          | 4620      |
|    policy_gradient_loss | -0.00285 |
|    std                | 0.217     |
|    value_loss         | 4.31e+03  |
-----------------------------------------
crash num : 4028
souci de réception
crash num : 4029
souci de réception
crash num : 4030
souci de réception
crash num : 4031
souci de réception
crash num : 4032
souci de réception
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 712       |
|    ep_rew_mean        | 4.36e+03  |
| time/                 |           |
|    fps                | 83        |
|    iterations         | 464       |
|    time_elapsed       | 11326     |
|    total_timesteps    | 950272    |
| train/                |           |
|    approx_kl          | 0.012017993 |
|    clip_fraction      | 0.151     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.236     |
|    explained_variance | 0.875     |
|    learning_rate      | 0.0005    |
|    loss               | 445       |
|    n_updates          | 4630      |
```

```
|    policy_gradient_loss | -0.00331    |
|    std                  | 0.216       |
|    value_loss           | 2.63e+03    |
-----------------------------------------
crash num : 4033
souci de réception
crash num : 4034
souci de réception
crash num : 4035
souci de réception
crash num : 4036
souci de réception
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 712         |
|    ep_rew_mean          | 4.4e+03     |
| time/                   |             |
|    fps                  | 83          |
|    iterations           | 465         |
|    time_elapsed         | 11345       |
|    total_timesteps      | 952320      |
| train/                  |             |
|    approx_kl            | 0.012295746 |
|    clip_fraction        | 0.126       |
|    clip_range           | 0.2         |
|    entropy_loss         | 0.247       |
|    explained_variance   | 0.865       |
|    learning_rate        | 0.0005      |
|    loss                 | 830         |
|    n_updates            | 4640        |
|    policy_gradient_loss | -0.00311    |
|    std                  | 0.215       |
|    value_loss           | 5.4e+03     |
-----------------------------------------
crash num : 4037
souci de réception
crash num : 4038
souci de réception
crash num : 4039
souci de réception
crash num : 4040
souci de réception
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 698         |
|    ep_rew_mean          | 4.4e+03     |
| time/                   |             |
|    fps                  | 83          |
|    iterations           | 466         |
|    time_elapsed         | 11364       |
|    total_timesteps      | 954368      |
| train/                  |             |
|    approx_kl            | 0.014074434 |
|    clip_fraction        | 0.146       |
|    clip_range           | 0.2         |
|    entropy_loss         | 0.254       |
|    explained_variance   | 0.941       |
|    learning_rate        | 0.0005      |
|    loss                 | 1.32e+03    |
|    n_updates            | 4650        |
|    policy_gradient_loss | 0.000352    |
|    std                  | 0.214       |
|    value_loss           | 3.03e+03    |
-----------------------------------------
crash num : 4041
souci de réception
crash num : 4042
souci de réception
crash num : 4043
souci de réception
crash num : 4044
souci de réception
crash num : 4045
souci de réception
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 642         |
|    ep_rew_mean          | 4.07e+03    |
| time/                   |             |
|    fps                  | 84          |
|    iterations           | 467         |
|    time_elapsed         | 11384       |
|    total_timesteps      | 956416      |
| train/                  |             |
|    approx_kl            | 0.012574561 |
|    clip_fraction        | 0.136       |
|    clip_range           | 0.2         |
|    entropy_loss         | 0.267       |
|    explained_variance   | 0.881       |
|    learning_rate        | 0.0005      |
|    loss                 | 1.33e+03    |
|    n_updates            | 4660        |
|    policy_gradient_loss | -0.00371    |
|    std                  | 0.213       |
|    value_loss           | 3.48e+03    |
-----------------------------------------
crash num : 4046
souci de réception
crash num : 4047
souci de réception
crash num : 4048
souci de réception
crash num : 4049
souci de réception
crash num : 4050
souci de réception
```

```
crash num : 4051
souci de réception
crash num : 4052
souci de réception
-----------------------------------------
| rollout/              |             |
|    ep_len_mean        | 578         |
|    ep_rew_mean        | 3.78e+03    |
| time/                 |             |
|    fps                | 84          |
|    iterations         | 468         |
|    time_elapsed       | 11404       |
|    total_timesteps    | 958464      |
| train/                |             |
|    approx_kl          | 0.032191094 |
|    clip_fraction      | 0.188       |
|    clip_range         | 0.2         |
|    entropy_loss       | 0.266       |
|    explained_variance | 0.955       |
|    learning_rate      | 0.0005      |
|    loss               | 195         |
|    n_updates          | 4670        |
|    policy_gradient_loss | -0.000113 |
|    std                | 0.213       |
|    value_loss         | 1.76e+03    |
-----------------------------------------
crash num : 4053
souci de réception
crash num : 4054
souci de réception
crash num : 4055
souci de réception
crash num : 4056
souci de réception
crash num : 4057
souci de réception
-----------------------------------------
| rollout/              |             |
|    ep_len_mean        | 579         |
|    ep_rew_mean        | 3.8e+03     |
| time/                 |             |
|    fps                | 84          |
|    iterations         | 469         |
|    time_elapsed       | 11423       |
|    total_timesteps    | 960512      |
| train/                |             |
|    approx_kl          | 0.0109035475 |
|    clip_fraction      | 0.123       |
|    clip_range         | 0.2         |
|    entropy_loss       | 0.262       |
|    explained_variance | 0.949       |
|    learning_rate      | 0.0005      |
|    loss               | 805         |
|    n_updates          | 4680        |
|    policy_gradient_loss | -0.00726  |
|    std                | 0.214       |
|    value_loss         | 3.48e+03    |
-----------------------------------------
crash num : 4058
souci de réception
crash num : 4059
souci de réception
crash num : 4060
souci de réception
-----------------------------------------
| rollout/              |             |
|    ep_len_mean        | 585         |
|    ep_rew_mean        | 3.82e+03    |
| time/                 |             |
|    fps                | 84          |
|    iterations         | 470         |
|    time_elapsed       | 11442       |
|    total_timesteps    | 962560      |
| train/                |             |
|    approx_kl          | 0.012086901 |
|    clip_fraction      | 0.113       |
|    clip_range         | 0.2         |
|    entropy_loss       | 0.27        |
|    explained_variance | 0.939       |
|    learning_rate      | 0.0005      |
|    loss               | 898         |
|    n_updates          | 4690        |
|    policy_gradient_loss | -0.00397  |
|    std                | 0.212       |
|    value_loss         | 2.66e+03    |
-----------------------------------------
crash num : 4061
souci de réception
crash num : 4062
souci de réception
crash num : 4063
souci de réception
crash num : 4064
souci de réception
crash num : 4065
souci de réception
crash num : 4066
souci de réception
crash num : 4067
souci de réception
-----------------------------------------
| rollout/              |             |
|    ep_len_mean        | 523         |
|    ep_rew_mean        | 3.45e+03    |
| time/                 |             |
|    fps                | 84          |
```

```
|    iterations          | 471        |
|    time_elapsed        | 11463      |
|    total_timesteps     | 964608     |
| train/                 |            |
|    approx_kl           | 0.015234785 |
|    clip_fraction       | 0.146      |
|    clip_range          | 0.2        |
|    entropy_loss        | 0.277      |
|    explained_variance   | 0.95       |
|    learning_rate       | 0.0005     |
|    loss                | 497        |
|    n_updates           | 4700       |
|    policy_gradient_loss | -0.00144   |
|    std                 | 0.213      |
|    value_loss          | 2.07e+03   |
-----------------------------------------
crash num : 4068
souci de réception
crash num : 4069
souci de réception
crash num : 4070
souci de réception
crash num : 4071
souci de réception
crash num : 4072
souci de réception
-----------------------------------------
| rollout/               |            |
|    ep_len_mean         | 502        |
|    ep_rew_mean         | 3.36e+03   |
| time/                  |            |
|    fps                 | 84         |
|    iterations          | 472        |
|    time_elapsed        | 11482      |
|    total_timesteps     | 966656     |
| train/                 |            |
|    approx_kl           | 0.018570533 |
|    clip_fraction       | 0.128      |
|    clip_range          | 0.2        |
|    entropy_loss        | 0.268      |
|    explained_variance   | 0.957      |
|    learning_rate       | 0.0005     |
|    loss                | 750        |
|    n_updates           | 4710       |
|    policy_gradient_loss | -0.00467   |
|    std                 | 0.214      |
|    value_loss          | 2.35e+03   |
-----------------------------------------
crash num : 4073
souci de réception
crash num : 4074
souci de réception
crash num : 4075
souci de réception
crash num : 4076
souci de réception
crash num : 4077
souci de réception
crash num : 4078
souci de réception
crash num : 4079
souci de réception
crash num : 4080
souci de réception
-----------------------------------------
| rollout/               |            |
|    ep_len_mean         | 495        |
|    ep_rew_mean         | 3.43e+03   |
| time/                  |            |
|    fps                 | 84         |
|    iterations          | 473        |
|    time_elapsed        | 11504      |
|    total_timesteps     | 968704     |
| train/                 |            |
|    approx_kl           | 0.02405244 |
|    clip_fraction       | 0.206      |
|    clip_range          | 0.2        |
|    entropy_loss        | 0.268      |
|    explained_variance   | 0.937      |
|    learning_rate       | 0.0005     |
|    loss                | 564        |
|    n_updates           | 4720       |
|    policy_gradient_loss | 0.00384    |
|    std                 | 0.213      |
|    value_loss          | 2.62e+03   |
-----------------------------------------
crash num : 4081
souci de réception
crash num : 4082
souci de réception
crash num : 4083
souci de réception
crash num : 4084
souci de réception
crash num : 4085
souci de réception
crash num : 4086
souci de réception
-----------------------------------------
| rollout/               |            |
|    ep_len_mean         | 459        |
|    ep_rew_mean         | 3.19e+03   |
| time/                  |            |
|    fps                 | 84         |
|    iterations          | 474        |
|    time_elapsed        | 11524      |
```

```
|    total_timesteps     | 970752      |
| train/                 |             |
|    approx_kl           | 0.009317725 |
|    clip_fraction       | 0.12        |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.269       |
|    explained_variance  | 0.923       |
|    learning_rate       | 0.0005      |
|    loss                | 950         |
|    n_updates           | 4730        |
|    policy_gradient_loss | 0.000818    |
|    std                 | 0.213       |
|    value_loss          | 4.2e+03     |
-----------------------------------------
crash num : 4087
souci de réception
crash num : 4088
souci de réception
crash num : 4089
souci de réception
crash num : 4090
souci de réception
crash num : 4091
souci de réception
crash num : 4092
souci de réception
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 436         |
|    ep_rew_mean         | 3.01e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 475         |
|    time_elapsed        | 11544       |
|    total_timesteps     | 972800      |
| train/                 |             |
|    approx_kl           | 0.014671426 |
|    clip_fraction       | 0.0997      |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.271       |
|    explained_variance  | 0.879       |
|    learning_rate       | 0.0005      |
|    loss                | 776         |
|    n_updates           | 4740        |
|    policy_gradient_loss | -0.00356    |
|    std                 | 0.212       |
|    value_loss          | 3.92e+03    |
-----------------------------------------
crash num : 4093
souci de réception
crash num : 4094
souci de réception
crash num : 4095
souci de réception
crash num : 4096
crash num : 4097
crash num : 4098
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 416         |
|    ep_rew_mean         | 2.91e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 476         |
|    time_elapsed        | 11564       |
|    total_timesteps     | 974848      |
| train/                 |             |
|    approx_kl           | 0.00990401  |
|    clip_fraction       | 0.182       |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.281       |
|    explained_variance  | 0.948       |
|    learning_rate       | 0.0005      |
|    loss                | 392         |
|    n_updates           | 4750        |
|    policy_gradient_loss | -0.000926   |
|    std                 | 0.211       |
|    value_loss          | 2.7e+03     |
-----------------------------------------
crash num : 4099
crash num : 4100
crash num : 4101
crash num : 4102
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 409         |
|    ep_rew_mean         | 2.85e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 477         |
|    time_elapsed        | 11583       |
|    total_timesteps     | 976896      |
| train/                 |             |
|    approx_kl           | 0.018657187 |
|    clip_fraction       | 0.159       |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.289       |
|    explained_variance  | 0.958       |
|    learning_rate       | 0.0005      |
|    loss                | 1.08e+03    |
|    n_updates           | 4760        |
|    policy_gradient_loss | -0.00351    |
|    std                 | 0.211       |
|    value_loss          | 3.33e+03    |
-----------------------------------------
crash num : 4103
```

```
crash num : 4104
crash num : 4105
crash num : 4106
crash num : 4107
crash num : 4108
----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 400       |
|    ep_rew_mean        | 2.83e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 478       |
|    time_elapsed       | 11603     |
|    total_timesteps    | 978944    |
| train/                |           |
|    approx_kl          | 0.018630184 |
|    clip_fraction      | 0.143     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.285     |
|    explained_variance | 0.946     |
|    learning_rate      | 0.0005    |
|    loss               | 312       |
|    n_updates          | 4770      |
|    policy_gradient_loss | -0.0018 |
|    std                | 0.211     |
|    value_loss         | 1.97e+03  |
----------------------------------------
crash num : 4109
crash num : 4110
crash num : 4111
crash num : 4112
crash num : 4113
crash num : 4114
----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 392       |
|    ep_rew_mean        | 2.82e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 479       |
|    time_elapsed       | 11623     |
|    total_timesteps    | 980992    |
| train/                |           |
|    approx_kl          | 0.012395136 |
|    clip_fraction      | 0.209     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.295     |
|    explained_variance | 0.947     |
|    learning_rate      | 0.0005    |
|    loss               | 554       |
|    n_updates          | 4780      |
|    policy_gradient_loss | 0.00282 |
|    std                | 0.21      |
|    value_loss         | 1.73e+03  |
----------------------------------------
crash num : 4115
crash num : 4116
crash num : 4117
crash num : 4118
crash num : 4119
crash num : 4120
----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 392       |
|    ep_rew_mean        | 2.84e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 480       |
|    time_elapsed       | 11643     |
|    total_timesteps    | 983040    |
| train/                |           |
|    approx_kl          | 0.013135914 |
|    clip_fraction      | 0.173     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.307     |
|    explained_variance | 0.975     |
|    learning_rate      | 0.0005    |
|    loss               | 386       |
|    n_updates          | 4790      |
|    policy_gradient_loss | -0.000179 |
|    std                | 0.209     |
|    value_loss         | 1.41e+03  |
----------------------------------------
crash num : 4121
crash num : 4122
crash num : 4123
----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 403       |
|    ep_rew_mean        | 2.89e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 481       |
|    time_elapsed       | 11661     |
|    total_timesteps    | 985088    |
| train/                |           |
|    approx_kl          | 0.008528873 |
|    clip_fraction      | 0.134     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.314     |
|    explained_variance | 0.95      |
|    learning_rate      | 0.0005    |
|    loss               | 1.43e+03  |
|    n_updates          | 4800      |
|    policy_gradient_loss | -0.00154 |
|    std                | 0.209     |
```

```
|    value_loss          | 2.35e+03    |
-----------------------------------------
crash num : 4124
crash num : 4125
crash num : 4126
crash num : 4127
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 393         |
|    ep_rew_mean         | 2.78e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 482         |
|    time_elapsed        | 11680       |
|    total_timesteps     | 987136      |
| train/                 |             |
|    approx_kl           | 0.031610988 |
|    clip_fraction       | 0.135       |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.324       |
|    explained_variance  | 0.954       |
|    learning_rate       | 0.0005      |
|    loss                | 764         |
|    n_updates           | 4810        |
|    policy_gradient_loss | 0.00136     |
|    std                 | 0.207       |
|    value_loss          | 2.07e+03    |
-----------------------------------------
crash num : 4128
crash num : 4129
crash num : 4130
crash num : 4131
crash num : 4132
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 394         |
|    ep_rew_mean         | 2.79e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 483         |
|    time_elapsed        | 11700       |
|    total_timesteps     | 989184      |
| train/                 |             |
|    approx_kl           | 0.03220329  |
|    clip_fraction       | 0.21        |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.339       |
|    explained_variance  | 0.948       |
|    learning_rate       | 0.0005      |
|    loss                | 376         |
|    n_updates           | 4820        |
|    policy_gradient_loss | -0.00154    |
|    std                 | 0.206       |
|    value_loss          | 1.83e+03    |
-----------------------------------------
crash num : 4133
crash num : 4134
crash num : 4135
crash num : 4136
crash num : 4137
crash num : 4138
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 375         |
|    ep_rew_mean         | 2.63e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 484         |
|    time_elapsed        | 11719       |
|    total_timesteps     | 991232      |
| train/                 |             |
|    approx_kl           | 0.0115434965 |
|    clip_fraction       | 0.112       |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.338       |
|    explained_variance  | 0.899       |
|    learning_rate       | 0.0005      |
|    loss                | 1.19e+03    |
|    n_updates           | 4830        |
|    policy_gradient_loss | -0.0019     |
|    std                 | 0.206       |
|    value_loss          | 4.6e+03     |
-----------------------------------------
crash num : 4139
crash num : 4140
crash num : 4141
crash num : 4142
crash num : 4143
crash num : 4144
-----------------------------------------
| rollout/               |             |
|    ep_len_mean         | 371         |
|    ep_rew_mean         | 2.61e+03    |
| time/                  |             |
|    fps                 | 84          |
|    iterations          | 485         |
|    time_elapsed        | 11739       |
|    total_timesteps     | 993280      |
| train/                 |             |
|    approx_kl           | 0.01736202  |
|    clip_fraction       | 0.159       |
|    clip_range          | 0.2         |
|    entropy_loss        | 0.345       |
|    explained_variance  | 0.958       |
|    learning_rate       | 0.0005      |
|    loss                | 546         |
```

```
|    n_updates          | 4840      |
|    policy_gradient_loss | -0.00269  |
|    std                | 0.204     |
|    value_loss         | 2e+03     |
-----------------------------------------
crash num : 4145
crash num : 4146
crash num : 4147
crash num : 4148
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 381       |
|    ep_rew_mean        | 2.65e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 486       |
|    time_elapsed       | 11758     |
|    total_timesteps    | 995328    |
| train/                |           |
|    approx_kl          | 0.009702235 |
|    clip_fraction      | 0.125     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.351     |
|    explained_variance | 0.945     |
|    learning_rate      | 0.0005    |
|    loss               | 481       |
|    n_updates          | 4850      |
|    policy_gradient_loss | -0.00518  |
|    std                | 0.205     |
|    value_loss         | 2.9e+03   |
-----------------------------------------
crash num : 4149
crash num : 4150
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 384       |
|    ep_rew_mean        | 2.69e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 487       |
|    time_elapsed       | 11776     |
|    total_timesteps    | 997376    |
| train/                |           |
|    approx_kl          | 0.012518723 |
|    clip_fraction      | 0.127     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.339     |
|    explained_variance | 0.911     |
|    learning_rate      | 0.0005    |
|    loss               | 562       |
|    n_updates          | 4860      |
|    policy_gradient_loss | -0.0024   |
|    std                | 0.208     |
|    value_loss         | 3.15e+03  |
-----------------------------------------
crash num : 4151
crash num : 4152
crash num : 4153
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 401       |
|    ep_rew_mean        | 2.78e+03  |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 488       |
|    time_elapsed       | 11795     |
|    total_timesteps    | 999424    |
| train/                |           |
|    approx_kl          | 0.022419808 |
|    clip_fraction      | 0.185     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.328     |
|    explained_variance | 0.956     |
|    learning_rate      | 0.0005    |
|    loss               | 354       |
|    n_updates          | 4870      |
|    policy_gradient_loss | -0.0011   |
|    std                | 0.207     |
|    value_loss         | 1.87e+03  |
-----------------------------------------
crash num : 4154
crash num : 4155
crash num : 4156
crash num : 4157
-----------------------------------------
| rollout/              |           |
|    ep_len_mean        | 404       |
|    ep_rew_mean        | 2.8e+03   |
| time/                 |           |
|    fps                | 84        |
|    iterations         | 489       |
|    time_elapsed       | 11814     |
|    total_timesteps    | 1001472   |
| train/                |           |
|    approx_kl          | 0.027889661 |
|    clip_fraction      | 0.174     |
|    clip_range         | 0.2       |
|    entropy_loss       | 0.326     |
|    explained_variance | 0.937     |
|    learning_rate      | 0.0005    |
|    loss               | 560       |
|    n_updates          | 4880      |
|    policy_gradient_loss | -0.00116  |
|    std                | 0.209     |
|    value_loss         | 2.02e+03  |
-----------------------------------------
```

crash num : 4157
Demo of the results.
crash num : 4158
crash num : 4159
crash num : 4160
crash num : 4161
crash num : 4162
crash num : 4163
crash num : 4164
crash num : 4165
crash num : 4166
crash num : 4167
crash num : 4168
crash num : 4169
crash num : 4170
crash num : 4171
crash num : 4172
crash num : 4173
crash num : 4174
crash num : 4175
crash num : 4176
crash num : 4177
crash num : 4178
crash num : 4179
crash num : 4180
crash num : 4181
Exiting.
66827.7093261042

TODO : expliquer super().__init()__ et pourquoi c'est super et non driver()
TODO : voir si on peut enlever la boucle sur get_obesrvation
TODO : voir si on peut enlever le test sur init
TODO : a priori, pas besoin de self.observation. Juste observation suffit
TODO : voir pour remplacer 12 par lidar_max_range
TODO : vérifier que le fait d'avoir enlever le super.step() après l'observation dans step() ne gêne pas.