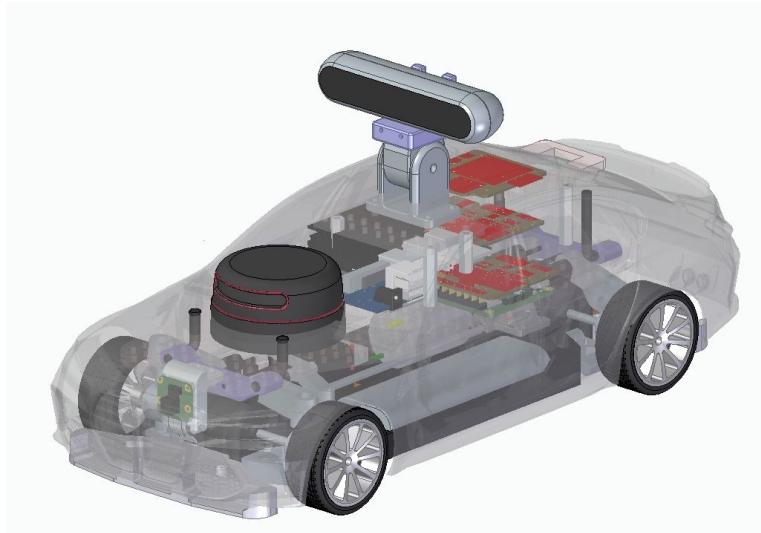


FACULTÉ DES SCIENCES D'ORSAY
UNIVERSITÉ PARIS SACLAY

M1 E3A
ÉLECTRONIQUE, ÉNERGIE ÉLECTRIQUE, AUTOMATIQUE
SAIETI
TRAVAUX D'ÉTUDES ET RÉALISATIONS
RAPPORT DE PROJET

Course de Voitures Autonomes COVAPSY



Étudiants :

Eric LOUANGPRASEUTH
Amen Allah KRISSAAN
Junle ZHONG
Kouachi Corneille EKON
Mohamed Yacine BOUSHAKI

Tuteurs :

Abdelhafid EL-OUARDI
Nicolas GAC

Table des matières

Remerciements, Résumé, Abstract, Préambule	4
I La course de voitures autonome et notre voiture	6
1 Présentation de la course	6
1.1 Éléments de règlements	6
2 Le déroulement du projet	8
2.1 Organisation temporelle	8
2.2 Organisation des tâches prévues	9
3 Le principe de fonctionnement du robot coureur	9
3.1 Matériel proposé et plan de fonctionnement	10
II Partie décisionnelle, support embarqué intelligent	11
1 Système temps réel	11
2 Choix du SBC	12
2.1 Caractéristiques de la Raspberry pi 4B	13
3 Acquisition des données de la position	14
4 Partie logicielle	15
4.1 Distributions Linux pour le Raspberry Pi	15
4.2 Mise en place et bibliothèques nécessaires pour notre application	15
5 Simulation de la course sur Webots	16
6 Apprentissage par renforcement	18
6.1 PPO	18
7 Application de l'apprentissage	20
7.1 Programme Webots	20
7.2 Entrainement du modèle	22
III Partie fonctionnelle, matérielle	23
1 Composants auxiliaires spécifiques à cette partie	23
1.1 Servomoteur	23
1.2 Support et châssis	24
1.3 La Batterie	24
1.4 Acquisiton au point mort	25
2 La carte Microcontrôleur	25
3 Liaisons microcontrôleur	26
3.1 Protocole de communication SPI	26
3.2 Protocole de communication I2C	27
3.3 Liaisons analogiques et digitales auxiliaires	28
4 Utilisation du microcontrôleur	29
4.1 Outils de programmation	29
4.2 Fonctionnalités utiles du logiciel	30
4.3 Test du microcontrôleur	32
4.4 Utilisation pour la transmission de données du microcontrôleur	33

IV Réalisation et mise en place	36
1 Construction et mise en fonctionnement du robot	36
1.1 Les débuts avec Raspberry Pi OS	36
1.2 Tests des composants et ajustement du matériel	37
1.3 Les problèmes rencontrés	37
1.4 Un premier fonctionnement par asservissement linéaire	37
2 Application de l'apprentissage par renforcement	40
3 Mise en fonctionnement réelle	42
3.1 Choix et observations du jour de la course	42
3.2 Discussion sur l'efficacité des solutions	43
4 Projections	44
Conclusion	45
Références	46
Annexe	49

Table des figures

1	Dimensions de la voiture	6
2	Photo de la piste	7
3	Timeline prévisionnel	9
4	Illustration du principe de fonctionnement du robot	10
5	Illustration du système embarqué	11
6	Composants et interfaces du Raspberry Pi 4 Vue de dessus	13
7	Acquisition des données d'un Lidar	14
8	Slamtec A2M8	14
9	Interface RPi	15
10	Piste de simulation sur Webots	16
11	Simulation sur Webots	17
12	Processus d'apprentissage par renforcement	18
13	Journal sur Webots	19
14	Servomoteur	23
15	Châssis de la voiture Tamiya TT02	24
16	Position du moteur et du variateur à droite	24
17	SRF10 Télémètre	25
18	STM32L432KC	25
19	Illustration du protocole SPI associé à ses maîtres et esclaves	26
20	Illustration du protocole I ² C associé à ses maîtres et esclaves	27
21	Trame du protocole I ² C	28
22	Schéma des liaisons auxiliaires	28
23	Schéma du fonctionnement des éléments principaux de la partie microcontrôleur	29
24	STM32CubeIDE	29
25	Interface principale MX	30
26	Interface horloge	31
27	Variables en direct (Live Expressions)	31
28	Affichage des données à visualiser rapidement	32
29	raspi-config	33
30	MX avec Pins	34
31	Voiture assemblée	36
32	Véhicule de course	43

Remerciements

Nous tenons à exprimer notre profonde gratitude à toutes les personnes dont le soutien moral et concret a été indispensable à la réalisation de notre projet de fin d'études. Il est essentiel pour nous de reconnaître leur contribution, car ce travail est autant le leur que le nôtre.

Nos sincères remerciements s'adressent tout d'abord à Monsieur **Nicolas Gac**, Responsable du Master Systèmes embarqués et traitement de l'information (SETI) et notre encadrant professionnel, pour sa disponibilité, sa patience et son engagement inébranlable envers la réussite de notre projet.

Un grand merci à Monsieur **Abdelhafid El Ouardi**, qui a accepté de diriger ce projet malgré ses nombreuses responsabilités. Sa guidance sans faille et ses efforts inlassables ont grandement contribué à notre formation et à notre réussite.

Nous exprimons également notre reconnaissance envers Monsieur **Adrien Mercier**, responsable de la formation E3A au sein de notre université. Son accueil chaleureux, sa confiance et ses précieux conseils ont été des piliers dans notre parcours académique.

Nos remerciements vont aussi à Monsieur **Anthony Juton** promoteur du projet, qui a toujours répondu présent à chaque fois que nous avions eu besoin de son aide.

Enfin, nos pensées reconnaissantes se dirigent vers nos familles et nos amis ainsi que toutes les personnes et collègues de travail ayant pu assister et/ou nous entraider durant la totalité du projet notamment les groupes auxquels nous avons été mis en concurrence au sein de la Faculté et extérieure. Leur soutien indéfectible, leurs encouragements et leur aide ont été cruciaux dans l'aboutissement de ce travail. Leur présence et leur soutien ont illuminé notre parcours, apportant réconfort et motivation dans les moments les plus difficiles.

Résumé du sujet

La Course de voitures autonomes Paris-Saclay ou CoVAPSy est une compétition de voitures autonomes réunissant de grandes institutions d'Ecoles supérieures dont la Faculté des Sciences. Ainsi l'objectif est de réaliser et programmer une voiture autonome et obtenir la plus haute place possible.

Abstract

Course de Voitures Autonomes Paris-Saclay, (CoVAPSy), is a competition of autonomous vehicles bringing together major institutions of higher education, including the Faculty of Sciences. The objective is to design and program an autonomous vehicle to achieve the highest possible ranking.

Préambule

Ce document est le rapport de projet de noms et résume le déroulé du "Travail d'études et Réalisations" (TER) qu'effectue les cinq étudiants dans le cadre du M1 "Électronique, Énergie électrique, Automatique" (E3A), parcours "Systèmes Autonomes, Intelligence Embarquée, Traitement de l'Information". Ce projet s'est déroulé en même temps que les cours magistraux de la formation contrairement à la majorité des projets proposés par la formation. Elle a débuté le 12 octobre 2023 et des séances sont organisées afin de se préparer pour la course le 30 mars 2024.

L'objectif principal est évidemment la réalisation de la Voiture Autonome afin de concourir le jour J. Afin de pouvoir réaliser ceci, des notions acquises au milieu de l'année nous ont été requises dans différents domaines, principalement l'Informatique Industrielle et l'Intelligence Artificielle par renforcement (RL).

Dans un premier temps, il a fallu se renseigner sur la compétition, les règles et le matériel autorisé puis les étudier. Ensuite, il a fallu appliquer les différents outils et techniques afin de réaliser une conduite autonome. Certains outils sont déjà maîtrisés par certains membres de l'équipe, d'autres ont du être entièrement appris en autodidacte.

En résumé, ce rapport retrace les différentes étapes du projet, les défis rencontrés et les solutions apportées, mettant en lumière la richesse de cette expérience formatrice qui combine théorie et pratique dans le cadre d'un projet collaboratif et innovant.

I La course de voitures autonome et notre voiture

1 Présentation de la course

L'événement accueille depuis 2020 des équipes d'étudiants (IUT, écoles d'ingénieurs, universités) pour une course de voitures 1/10ème autonomes, dans un esprit convivial et de collaboration. La perspective de la victoire finale n'empêche pas les équipes de s'entraider pendant l'année dans l'optique de progrès communs. L'édition 2024 a eu lieu le samedi 30 mars à l'ENS Paris Saclay.

1.1 Éléments de règlements

Le véhicule

Chaque véhicule participant doit présenter clairement un rectangle à son extrémité arrière, les dimensions étant fixées à 150 mm de large par 110 mm de haut, avec une tolérance pour un espace minimal sous la voiture. La carrosserie ne doit pas être de couleur verte, rouge, grise ou transparente. Un dispositif Lidar est utilisé pour confirmer que l'arrière du véhicule est suffisamment visible. Si le dispositif ne détecte pas l'arrière du véhicule, il faudra y apposer un ruban adhésif blanc pour augmenter la visibilité.

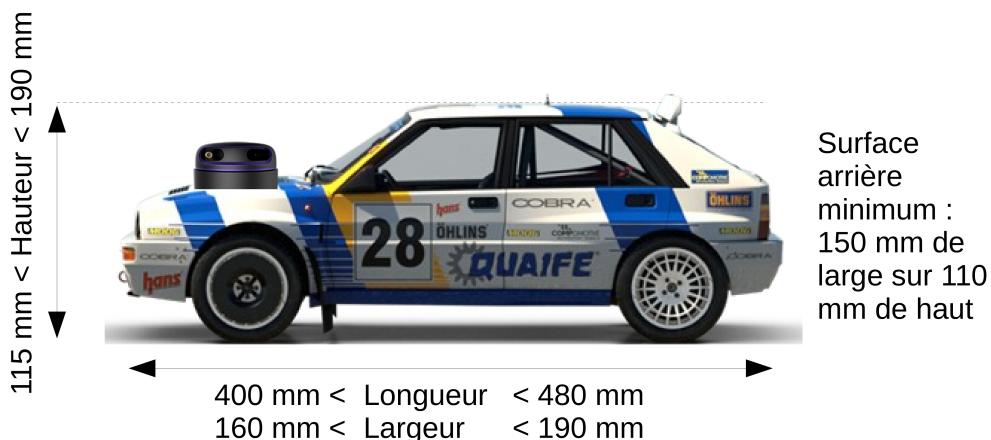


FIGURE 1 – Dimensions de la voiture

Les équipes peuvent modifier la configuration de traction des véhicules pour améliorer la maniabilité, tout en assurant que le véhicule puisse avancer et reculer. Les améliorations autorisées de la direction peuvent inclure l'échange de pièces en plastique par des pièces en aluminium, disponibles sur le marché. Des modifications majeures nécessitent une validation collective après présentation des détails des changements prévus.

La communication avec le véhicule doit se restreindre aux commandes de démarrage et d'arrêt. Toute commande externe influençant d'autres fonctions du véhicule est interdite et entraînera une disqualification. Les véhicules peuvent utiliser une batterie supplémentaire pour les systèmes électroniques, mais la tension alimentant le moteur ne doit pas excéder celle de la batterie principale.

La piste

Le tracé de la piste reste un mystère jusqu'au jour de la compétition, donc il est strictement interdit de filer des informations sur la configuration de la piste aux voitures à l'avance. Pour s'assurer que personne ne triche, des vérifications peuvent être faites sur une piste secondaire. Une fois la course lancée, les voitures ont l'occasion de se familiariser avec la piste lors de leurs premiers tours.



FIGURE 2 – Photo de la piste

La piste est bordée de barrières de 200 mm de haut, vertes à droite et rouges à gauche, suivant le sens de la course. Elle comprend des lignes droites et des courbes avec un rayon de courbure minimal de 400 mm. Le revêtement est en lino gris et la largeur de la piste dépasse toujours 800 mm, bien qu'il puisse y avoir quelques obstacles ici et là.

Pour améliorer la visibilité, une ligne blanche continue de 19 mm de haut a été ajoutée le long des bordures depuis 2023, et ils envisagent même d'ajouter une ligne pointillée au milieu pour mieux guider les voitures.

Homologations

L'homologation consiste à vérifier plusieurs éléments du véhicule : les dimensions, la batterie, le châssis, les fonctions de démarrage et d'arrêt à distance, et la visibilité de la couleur par Lidar. En plus, la voiture doit montrer qu'elle peut rouler sur une ligne droite de la piste, prendre un virage sans toucher les bordures, et reculer en cas de blocage contre un obstacle, tant qu'il n'y a pas de voiture derrière.

Des pénalités peuvent être appliquées pour des infractions mineures, selon la décision des arbitres. Par exemple, si la voiture ne démarre pas ou ne s'arrête pas correctement à distance, ou si la marche arrière ne fonctionne pas bien.

Qualifications

Les qualifications se déroulent voiture par voiture. La première qualification se fait sur une piste sans obstacles (piste A). La deuxième qualification se déroule sur une piste avec

des obstacles fixes plus grands que les voitures (piste B).

Chaque voiture doit effectuer deux tours pour chaque qualification et dispose de deux essais pour chaque phase. Le meilleur temps des deux essais est retenu, ce qui permet de compenser tout problème technique éventuel lors du premier essai. Si une voiture ne parvient pas à terminer les deux tours, un temps de 120 secondes lui est attribué.

Les résultats des qualifications, basés sur la somme des meilleurs temps des deux phases, déterminent l'ordre de départ. La voiture la plus rapide partira en tête. Les six premières voitures participeront à la première course, tandis que les six suivantes prendront part à la deuxième course, et ainsi de suite pour chaque manche.

La course

Les équipes disposent de 3 minutes pour installer leur véhicule sur la piste. Les véhicules sont placés sur la grille de départ en fonction des résultats des qualifications. Une fois que toutes les équipes déclarent être prêtes, il est interdit de toucher aux véhicules. Le signal de départ est donné oralement par l'arbitre. L'ordre d'arrivée est noté après un nombre de tours prédéfini (généralement 5). Une voiture qui ne termine pas le nombre de tours requis n'est pas classée.

Les véhicules ne doivent pas adopter un comportement agressif envers les autres voitures. Il est interdit d'empêcher délibérément un autre véhicule de dépasser.

L'arbitre peut disqualifier et retirer de la piste un véhicule pour comportement agressif.

Si un véhicule reste immobilisé sur la piste pendant plus de 10 secondes, sans être bloqué par une autre voiture, l'arbitre ou un de ses assistants le retirera. Il en va de même pour un véhicule parcourant plus de 2 mètres à contresens.

Deux manches sont organisées (chaque manche pouvant inclure plusieurs courses). Les points sont attribués comme suit : 1er - 25 points ; 2ème - 18 points ; 3ème - 15 points ; 4ème - 12 points ; 5ème - 10 points ; 6ème - 8 points ; 7ème - 6 points ; 8ème - 4 points ; 9ème - 2 points ; 10ème - 1 point.

En cas d'égalité, les temps des qualifications seront utilisés pour départager les équipes.

2 Le déroulement du projet

Le déroulement du projet commence par la réception des voitures par toutes les équipes, ce qui marque le point de départ des préparations.

2.1 Organisation temporelle

Le projet est structuré selon un emploi du temps précis, avec une organisation logistique détaillée. Dès le début, un programme prévisionnel est établi pour planifier les différentes phases du projet, de la conception à la course finale. Cette planification permet de suivre l'avancement des tâches et d'assurer une gestion efficace du temps.

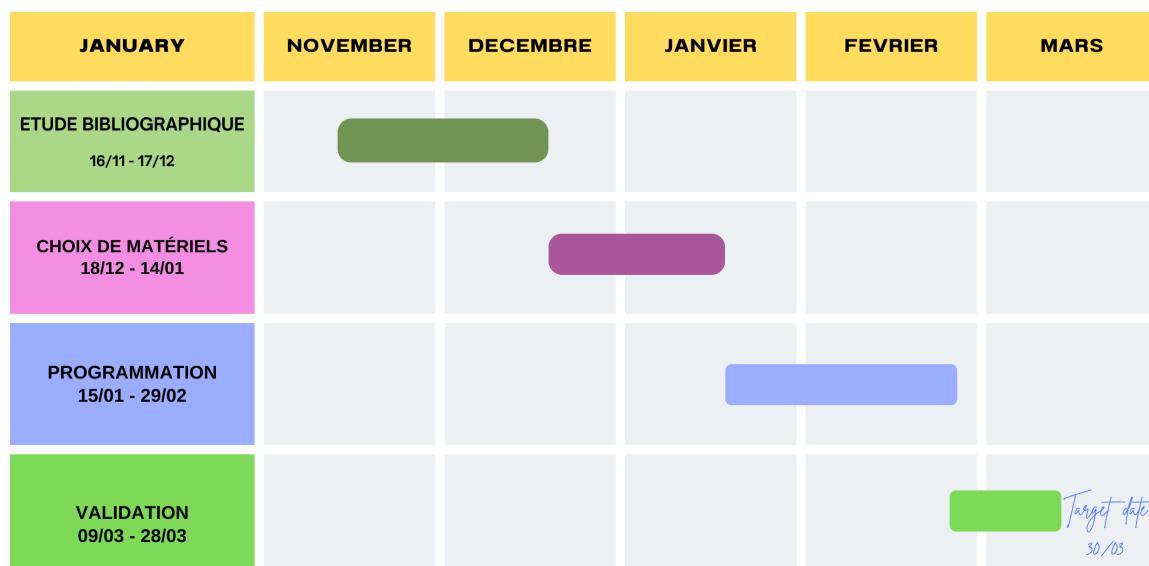


FIGURE 3 – Timeline prévisionnel

2.2 Organisation des tâches prévues

Les tâches sont initialement attribuées à chaque membre de l'équipe en fonction de leurs compétences et expertises. Chaque individu reçoit des responsabilités spécifiques, formant ainsi une lignée de pensée cohérente pour mener le projet à terme. Cette organisation garantit que chaque aspect du projet est couvert, des aspects techniques de la voiture aux stratégies de course et à la gestion des imprévus.

La répartition initiale est la suivante :

1. Deux personnes gérant la partie théorique, comment atteindre l'objectif de conduite autonome ;
2. Deux personnes gréant la partie pratique technique, le matériel qui permet de faire rouler le véhicule et ses périphériques.

3 Le principe de fonctionnement du robot coureur

Le robot coureur roule de façon autonome. Cette autonomie relève d'un plan de fonctionnement basé sur deux parties. La première partie est constituée d'une unité centrale et la seconde partie est constituée d'un ordinateur mono-carte qui effectue les calculs de prise de décisions.

3.1 Matériel proposé et plan de fonctionnement

On obtient le plan suivant :

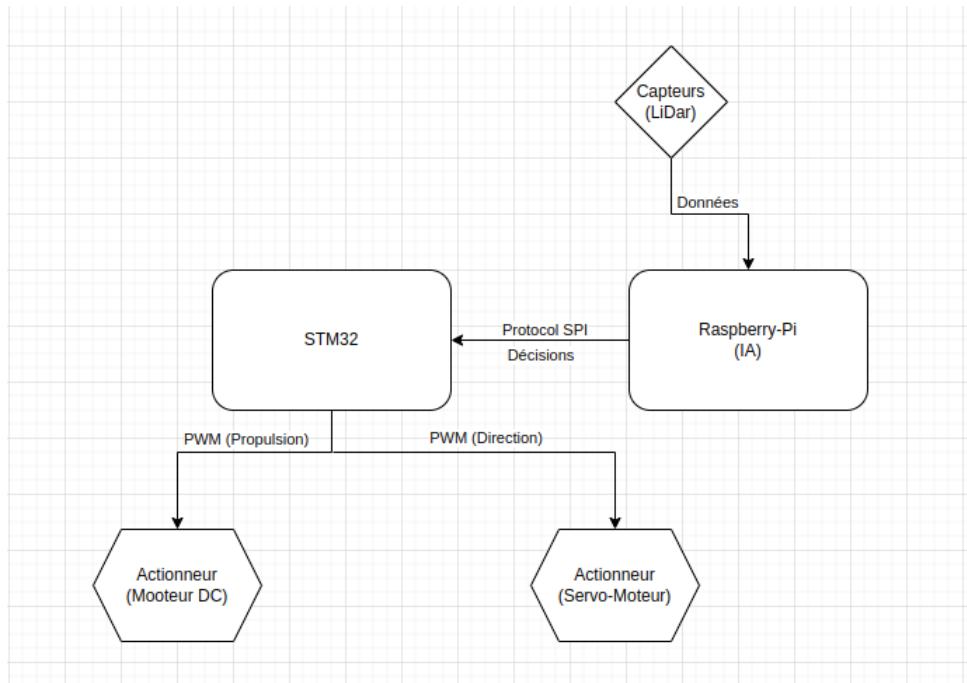


FIGURE 4 – Illustration du principe de fonctionnement du robot

L’unité centrale est un microcontrôleur qui se charge de l’application des décisions sur les différents actionneurs du véhicule. Les décisions lui parviennent à travers un protocole de communication. L’ordinateur mono-carte utilise les données de capteurs pour effectuer les calculs de prise de décision avec de l’intelligence artificielle.

On peut séparer donc en deux parties distinctes : Une partie décisionnelle qui va donner une instruction au véhicule afin de se déplacer correctement ainsi qu’une partie capture qui récupérera les données de capteurs ce qui libérera la charge au niveau de la partie décisionnelle. Ces deux parties ainsi que le matériel exact pour effectuer toutes les tâches sera détaillé dans les parties II et III.

II Partie décisionnelle, support embarqué intelligent

Dans ce chapitre, on va détailler la mise en place d'un système embarqué adéquat et performant pour les voitures autonomes en se basant sur les deux approches algorithmiques, déjà présentées et expliquées, dans le chapitre précédent. Nous allons mettre en œuvre toutes les conditions (côté matériel et logiciel) pour répondre à l'exigence de cahier des charges.

1 Système temps réel

Un système embarqué est un système complexe qui intègre du logiciel et du matériel conçus ensemble afin de fournir des fonctionnalités données. Il contient généralement un ou plusieurs microprocesseurs destinés à exécuter un ensemble de programmes définis lors de la conception et stockés dans les mémoires.

Comme montré sur la Figure 5, un système embarqué est construit autour d'un système informatique qui reçoit des informations provenant de capteurs et qui interagit avec l'environnement à partir d'afficheurs. Les capteurs mesurent des informations qui caractérisent l'environnement afin de déterminer son état courant. Ces informations sont converties numériquement pour être traitées par le système informatique qui va produire un résultat en fonction de l'état de l'environnement. Ce résultat est converti et transmis à l'afficheur.

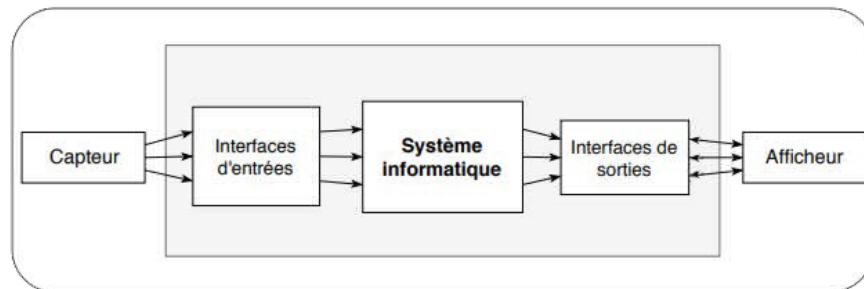


FIGURE 5 – Illustration du système embarqué

Pour notre cas, le système est formé par :

- Un dispositif d'acquisition des informations
- Une unité pour le traitement et l'exécution des algorithmes .
- Une interface graphique de paramétrage et visualisation des résultats de course .

Le dispositif à concevoir doit répondre à des exigences strictes et quantifiées par des critères de performances. La démarche de conception consiste à :

- Quantification des critères de performances à partir des existences du cahier de charge.
- Cibler la partie matérielle qui répond au mieux à nos exigences.
- Implanter les algorithmes d'apprentissage profond et de contrôle .
- Valider le dispositif par des tests réels.

2 Choix du SBC

Suivant les critères du cahier des charges, nous allons étudier les propositions sur le marché de carte de développement. Nous sommes arrivés au constat que les cartes grand public dominant dans le marché international et qui répondent particulièrement à nos besoins sont la carte Raspberry Pi et Nvidia Jetson Xavier. Les cartes grand public offrent plus de souplesse d'utilisation, sont moins confidentielles et permettent de bénéficier d'une communauté active de développeurs.

Feature	Nvidia Jetson Nano	RPi 4
SoC	NVIDIA Tegra X1	Broadcom BCM2711
CPU	x4 Cortex-A57 @ 2.26 GHz	x4 Cortex-A72 64-bit @ 1.5 GHz
GPU	128-core Maxwell	VideoCore VI 3D @ 500MHz
RAM	Variable (16/32 Go LPDDR4x)	Variable (16/32 Go LPDDR4x)
Storage	16/32 Go eMMC, microSD	16/32 Go eMMC, microSD
Ethernet	Gigabit Ethernet (10-1000 BASE-T)	Gigabit Ethernet
GPIO	40 pins, compatible Raspberry Pi	40 pins GPIO header
Affichage	HDMI 2.0, DP 1.2	2 × micro-HDMI ports
Vidéo	HDMI 2.0, DP 1.2 (support 4K 60Hz)	2 USB 3.0 ; 2 USB 2.0
Prix	\$99	\$55

TABLE 1 – Comparaison between Nvidia Jetson Xavier and Raspberry Pi 4

Les caractéristiques, le prix et la disponibilité sur le marché sont des avantages pour la carte Raspberry Pi 4B. Aussi les cartes Raspberry offrent plus de souplesse d'utilisation, et une communauté active de développeurs. C'est donc la **Raspberry Pi 4B** qui nous a paru la meilleure candidate par rapport aux critères du projet.

Présentation de la Raspberry Pi

Le Raspberry Pi est un appareil remarquable : un ordinateur entièrement fonctionnel dans un boîtier minuscule et peu coûteux. Contrairement à un ordinateur traditionnel, qui cache son fonctionnement interne dans un boîtier, un Raspberry Pi a tout ce qu'il faut pour être efficace. Ses composants, ses ports et ses fonctions.

Le Raspberry Pi est connu comme un ordinateur mono-carte (Single Board Computer) ; un nano-ordinateur construit sur un seul circuit imprimé. Comme la plupart des ordinateurs mono cartes, le Raspberry Pi est petit - à peu près de même taille d'une carte de crédit - mais cela ne veut pas dire qu'elle n'est pas puissante : un Raspberry Pi peut faire l'affaire ; tout ce qu'un ordinateur plus gros et plus gourmand en énergie peut faire, mais pas nécessairement aussi rapidement.

Le Raspberry Pi a été rendu possible en partie grâce aux progrès des puces pour ordinateurs mobiles de ces dernières années. À son cœur se trouve une puce Broadcom BCM283X (X : 5,6,7...) qui contient un processeur central ARM, une unité de traitement (CPU) et une unité de traitement graphique (GPU) se partageant la mémoire entre elles. Le GPU est suffisamment puissant pour pouvoir gérer une lecture vidéo de qualité Blu-ray. Il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux, notamment Debian.

2.1 Caractéristiques de la Raspberry pi 4B

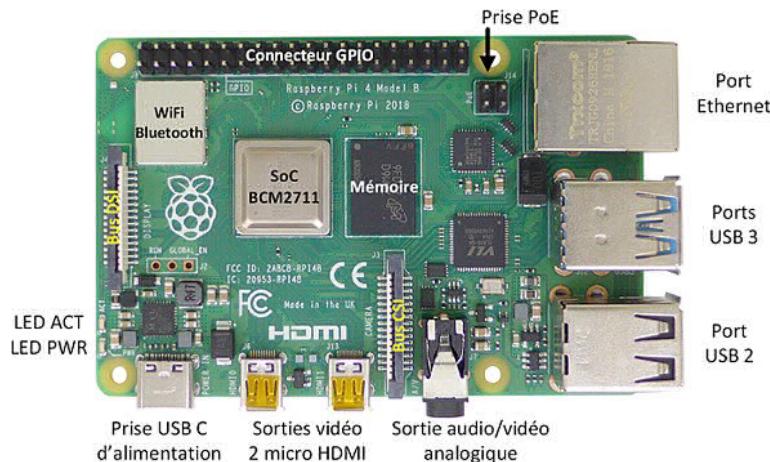


FIGURE 6 – Composants et interfaces du Raspberry Pi 4 Vue de dessus

Processeur	Broadcom BCM2711 ARM Cortex-A72 4 coeurs 1,5 GHz 64 bits, 1 Mo cache
Mémoire	4GB LPDDR4
Connectivité	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports
GPIO	Standard 40-pin GPIO header
Vidéo & Son	2 × micro-HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimédia	H.265 (4Kp60 décode) ; H.264 (1080p60 décode 1080p30 encode) OpenGL ES, 3.0 Graphics
Support SD	Micro SD card slot for loading operating system and data storage
Alimentation	5V DC via USB-C connector (min 3A) 5V DC via GPIO header (minimum 3A) Power over Ethernet (PoE)-enabled (requires separate PoE HAT)
Environnement	Operating température 0–50°C

TABLE 2 – Caractéristiques de Raspberry pi 4B

3 Acquisition des données de la position

Plusieurs solutions s'offre a nous afin de détecter les obstacles et les murs de la course, on a notamment :

- Plusieurs choix de Lidar à des prix différents,
- Des caméras

Notre choix s'est porté sur le Lidar. Le Lidar (Light detection and ranging), essentiel à l'architecture de conduite autonome, est crucial pour la détection d'objets et l'estimation de la profondeur. Sa sensibilité et précision en font un élément clé en robotique, dans les systèmes avancés d'aide à la conduite et dans d'autres domaines nécessitant l'interprétation d'images 3D.

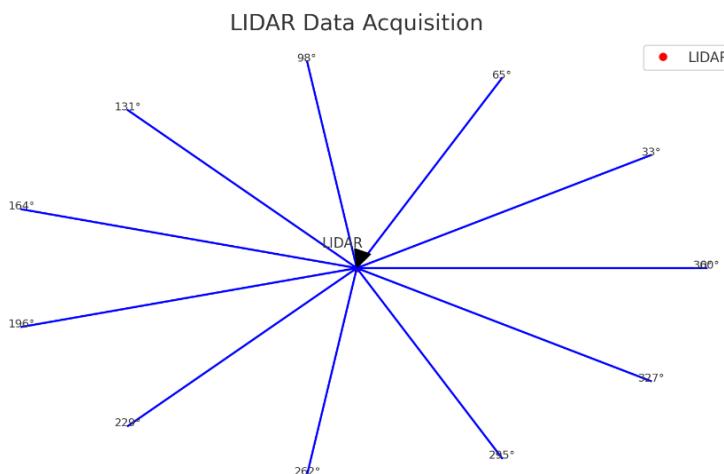


FIGURE 7 – Acquisition des données d'un Lidar

Pour le choix du Lidar, nous avons opté pour le modèle "Slamtec A2M8" pour ses caractéristiques techniques adaptées à notre application. Avec une fréquence de rafraîchissement de 15Hz et une portée effective de 12 mètres, il offre une détection précise des objets environnants. Les données du Lidar sont transmises à la Raspberry Pi 4 via une interface USB, assurant ainsi une communication rapide et fiable.



FIGURE 8 – Slamtec A2M8

4 Partie logicielle

Pour atteindre notre objectif, la démarche suivante (méthode descendante) est retenue :

- Installation des outils logiciels compatibles avec le choix matériel déjà fixé.
- Installation des bibliothèques nécessaires.
- Adéquation de l'algorithme pour la cible et l'outil de compilation.
- Programmation de l'algorithme en utilisant le langage Python .
- Simulation et présentation des résultats.
- La validation sera abordée dans la partie IV.

4.1 Distributions Linux pour le Raspberry Pi

Notre système utilise la distribution Linux officielle pour la carte cible Raspberry Pi, incluant tous les outils nécessaires au développement natif directement sur la carte RPi : la distribution Raspbian. Raspbian est une version modifiée de Debian spécifiquement conçue pour les SoC de type ARMv6, combinant les mots "Raspberry Pi" et "Debian".

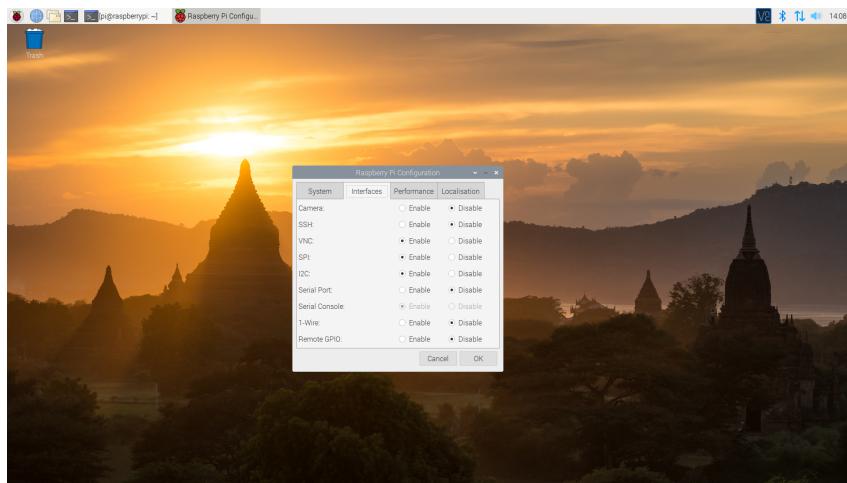


FIGURE 9 – Interface RPi

4.2 Mise en place et bibliothèques nécessaires pour notre application

Le développement se faisant en Python pour des raisons de praticité de code et d'implémentation sur cette plate-forme, nous devons procéder à l'installation des librairies suivantes :

stable_baselines : Il s'agit d'une bibliothèque d'apprentissage par renforcement basée sur TensorFlow qui fournit plusieurs algorithmes classiques d'apprentissage par renforcement tels que PPO, SAC pour faciliter la formation et l'utilisation de modèles d'apprentissage par renforcement.

RPILidar : Il permet de contrôler et d'acquérir des données Lidar et de les traiter en temps réel pour les utiliser dans des algorithmes de conduite autonome.

SPIdev : SPI (Serial Peripheral Interface) est un protocole de communication série synchrone largement utilisé pour connecter des microcontrôleurs et des périphériques pour l'échange de données.

rpi_hardware_pwm : Sur Raspberry Pi, les bibliothèques Python pour contrôler le PWM matériel (modulation de largeur d'impulsion) permettent de réaliser un contrôle plus précis et stable des dispositifs de sortie par rapport aux signaux analogiques générés par le Raspberry Pi lui-même. En utilisant le PWM matériel, il est possible de contrôler simultanément plusieurs canaux PWM, offrant une meilleure précision et stabilité que le PWM logiciel.

5 Simulation de la course sur Webots

Afin de mener à bien notre projet, nous avons commencé par faire divers tests de simulations avec le logiciel Webots. Webots est un simulateur de robot open-source puissant et polyvalent, permettant la création et l'expérimentation dans des environnements virtuels sophistiqués. Cet outil offre une plate-forme flexible pour le développement et le test d'algorithmes de contrôle et de navigation, ainsi que des fonctionnalités avancées pour simuler des capteurs et des actionneurs permettant ainsi une évaluation complète et précise de notre système.

L'environnement de simulation de Webots qui a été fourni par Monsieur Anthony Juton nous a permis d'avoir une piste de course très réaliste comme illustré dans la figure ci-dessous, afin d'effectuer divers tests et de perfectionner notre système avant de passer à la phase de mise en œuvre réelle.



FIGURE 10 – Piste de simulation sur Webots

Une des premières étapes de notre projet a été la mise en place d'un système de contrôle de base utilisant les données du LIDAR avec un contrôleur PID (Proportionnel-Integral-Dérivé). Le contrôleur PID est un mécanisme de contrôle en boucle fermée largement utilisé dans les systèmes de contrôle automatique, il ajuste les commandes du système en fonction de trois composantes : la proportionnelle (P), qui dépend de l'erreur actuelle, l'intégrale (I), qui tient compte de la somme des erreurs passées, et la dérivée (D), qui anticipe les erreurs futures. Cette combinaison permet d'améliorer la précision et la stabilité du système en réduisant les oscillations et en minimisant l'erreur par rapport à la consigne.

En utilisant les données fournies par le Lidar, nous avons implémenté un contrôleur PID pour ajuster en temps réel la direction du véhicule afin de suivre la piste et d'éviter les obstacles de manière optimale, cette approche a permis de stabiliser le comportement du véhicule et de garantir qu'il pouvait faire un tour de piste complet sans aucun soucis et en évitant tous les obstacles.

Les codes source de nos simulations ainsi que de notre implantation du contrôleur PID sont fournis en annexe pour référence. De plus, les résultats détaillés de nos tests et simulations sont illustrés dans les figures suivantes, montrant la performance de notre système dans divers scénarios de conduite et démontrant l'efficacité de notre approche en environnement simulé sur Webots.



FIGURE 11 – Simulation sur Webots

Une fois l'étape de simulation avec le PID terminée et des résultats satisfaisants obtenus, nous sommes passés à la simulation avec des algorithmes d'apprentissage par renforcement. Ces algorithmes permettent au véhicule d'apprendre à naviguer sur la piste de manière autonome en optimisant ses actions par essais et erreurs, et en recevant des récompenses en fonction de ses performances.

6 Apprentissage par renforcement

L'apprentissage par renforcement consiste à apprendre en interagissant avec l'environnement et en observant les conséquences de certaines actions. Cela permet aux machines de déterminer automatiquement le comportement idéal dans une situation donnée pour maximiser les performances. Cela nécessite simplement de renvoyer les résultats pour apprendre à la machine comment fonctionner. C'est ce qu'on appelle « la cause et l'effet » et c'est sans doute la clé de l'acquisition de connaissances tout au long de la vie.

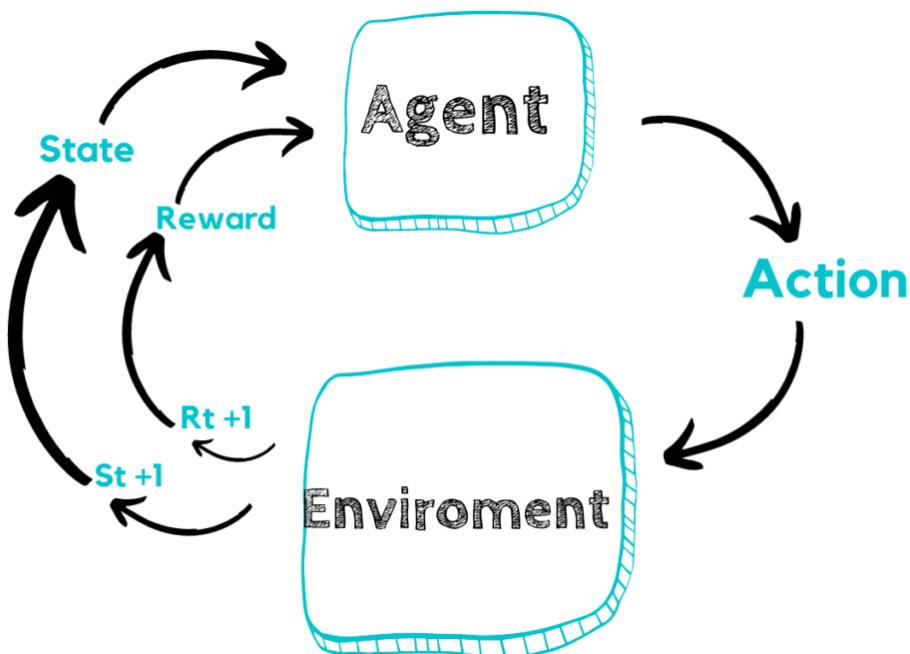


FIGURE 12 – Processus d'apprentissage par renforcement

1. L'agent surveille l'état de l'enregistrement.
2. Les actions sont déterminées par les fonctions décisionnelles (politiques).
3. Le travail est terminé.
4. L'agent obtient des résultats en fonction de son environnement.
5. Des informations sur cet état ou le résultat de l'action sont enregistrées.

6.1 PPO

PPO(Proximal Policy Optimization), est une méthode d'apprentissage par renforcement utilisée pour résoudre des problèmes de décision dans des espaces d'action continus.

Comparé aux algorithmes traditionnels d'apprentissage par renforcement, cet algorithme restreint les différences entre les nouvelles politiques et les anciennes politiques à chaque mise à jour, ce qui est généralement réalisé en introduisant un ratio $r(\theta)$, qui est le rapport des probabilités qu'une nouvelle politique et une ancienne politique prennent une certaine action donnée dans un état donné. PPO utilise deux méthodes pour limiter ce ratio.

1. *Clipping* : En limitant le ratio $r(\theta)$ dans un petit intervalle (comme $[1-\epsilon, 1+\epsilon]$) pour empêcher une mise à jour de la politique trop importante. Sinon, cela peut entraîner une instabilité dans le processus d'entraînement, rendant ainsi le processus d'apprentissage difficile à converger ou ne convergeant pas du tout, tombant dans un optimum local ou ne pouvant pas converger vers un optimum global, et avoir une faible capacité de généralisation.
2. *Surrogate Loss* : Utilisation d'une fonction de perte de substitution pour optimiser la politique, cette fonction encourage à maximiser le rendement attendu tout en maintenant la stabilité de la politique.

rollout/	
ep_len_mean	167
ep_rew_mean	358
time/	
fps	22
iterations	2
time_elapsed	367
total_timesteps	8192
train/	
approx_kl	0.004172016
clip_fraction	0.0697
clip_range	0.15
entropy_loss	-2.85
explained_variance	0.000861
learning_rate	0.0003
loss	1.1e+03
n_updates	40
policy_gradient_loss	-0.00489
std	1.02
value_loss	3.18e+03

FIGURE 13 – Journal sur Webots

L’explication des paramètres :

1. *policy* : Ici nous avons choisi MultiInputPolicy, car lors de la simulation, il est nécessaire de prendre les données du radar et la vitesse de la voiture comme entrées pour prendre des décisions.
2. *verbose* : Le niveau de détail des journaux. 0 indique aucune sortie d’informations, 1 indique la sortie d’informations de base, 2 indique la sortie d’informations détaillées.
3. *learning_rate* : Pour contrôler la taille des pas de mise à jour du réseau de politique.
4. *tensorboard_log* : Pour visualiser diverses informations statistiques lors du processus d’entraînement.
5. *batch_size* : Le nombre d’échantillons(steps) par lot.
6. *n_steps* : Le nombre de pas exécutés dans l’environnement avant chaque mise à jour de la politique, utilisé pour collecter des données.
7. *n_epochs* : Le nombre d’itérations d’entraînement utilisées lors de chaque mise à jour de la politique avec les données collectées.
8. *gae_lambda* : Un paramètre de l’estimation de l’avantage généralisé (GAE), pour équilibrer le compromis entre le biais et la variance.
9. *clip_range* : La plage de coupe. C’est pour contrôler la plage de variation du ratio entre la nouvelle et l’ancienne politique.

10. *vf_coef* : Le coefficient de perte de la fonction de valeur. C'est pour équilibrer le poids entre la perte de la politique et la perte de la fonction de valeur. Si elle est grande, le poids de l'optimisation de la fonction de valeur est élevé, l'algorithme accorde davantage d'importance à l'amélioration de l'exactitude de la fonction de valeur, ce qui entraîne une mise à jour plus lente de la politique. Ainsi, le choix du coefficient approprié détermine l'équilibre entre les ressources de calcul et les objectifs d'optimisation.
11. *ent_coef* : Le coefficient de perte d'entropie. C'est pour encourager l'exploration de la politique, qui nous permet d'éviter de tomber dans un optimum local.

7 Application de l'apprentissage

7.1 Programme Webots

Une fois la simulation de la partie asservissement terminée sur Webots, nous avons entamé la phase d'apprentissage. Cette transition a nécessité une approche différente, axée sur l'implémentation d'algorithmes d'apprentissage pour améliorer les performances de notre système. Pour cette étape cruciale, nous avons bénéficié de l'expertise de M. Juton, qui nous a fourni un code de base pour l'environnement d'apprentissage, préconfiguré avec des paramètres par défaut.

Le code de base utilisé pour l'apprentissage par renforcement est écrit en Python et utilise la bibliothèque Gymnasium pour la création de l'environnement de simulation, ainsi que la bibliothèque Stable Baselines 3 pour l'implémentation des algorithmes d'apprentissage.

Configuration de l'environnement

Le code commence par l'importation des bibliothèques nécessaires installés et la configuration de l'environnement Webots pour l'apprentissage. La classe WebotsGymEnvironment hérite de la classe Driver de Webots et de la classe Env de Gym, ce qui permet de créer un environnement personnalisé voici quelques extraits de code :

```

1 class WebotsGymEnvironment(Driver, gym.Env):
2     def __init__(self):
3         super().__init__()
4         self.consigne_angle = 0.0 # en deg
5         self.consigne_vitesse = 0.1 # en m/s

```

Listing 1 – Configuration de l'environnement

Cependant, comme tout projet d'apprentissage par renforcement, l'ajustement des paramètres était nécessaire pour adapter le modèle à notre cas d'utilisation spécifique. Nous avons donc commencé à modifier ces paramètres et à introduire des conditions supplémentaires, notamment en ce qui concerne les récompenses du modèle.

Configuration des récompenses

La fonction de récompense est cruciale pour l'apprentissage par renforcement, car elle guide l'agent vers un comportement optimal. Dans notre cas la fonction `get_reward` les attribue. Nous, nous avons conçu des récompenses pour que la voiture obtienne des points supplémentaires lorsqu'elle évite correctement les obstacles et suit les virages de manière optimale.

```

1 def get_reward(self, obs):
2     is_straight = all(obs["current_lidar"][:i] > 350 for i in range(-10,
3         11))
4     reward = 0
5     done = False
6     mini = 1
7     for i in range(-30,30):
8         if (obs["current_lidar"][:i] < mini and obs["current_lidar"][:i]
9             != 0):
10            mini = obs["current_lidar"][:i]
11    if obs["current_lidar"][:0] == 0 and obs["current_lidar"][:1] == 0 and
12        obs["current_lidar"][:1] == 0:
13        self.numero_crash += 1
14        reward = -300
15        done = True

```

Listing 2 – Configuration des récompenses

Gestion des collisions

Nous avons également mis en place un système de pénalités pour les collisions, où chaque crash est enregistré et les voitures sont replacées de façon aléatoire sur la piste pour éviter le surapprentissage de certaines sections de la piste

```

1 def reset(self, seed = 0):
2     self.consigne_vitesse = 0.0
3     self.consigne_angle = 0.0
4     self.set_vitesse_m_s(self.consigne_vitesse)
5     self.set_direction_degre(self.consigne_angle)
6     for i in range(20):
7         super().step()
8     self.reset_counter = 0
9     if self.numero_crash != 0:
10        while abs(super().getCurrentSpeed()) >= 0.001:
11            super().step()
12        self.packet_number += 1
13        self.emitter.send("voiture crash envoi numero " + str(self.
packet_number))

```

Listing 3 – Gestion des collisions

Gestion des paramètres

Nous avons commencé à ajuster ces différents paramètres pour observer leurs impacts sur l'apprentissage du modèle. Par exemple, nous avons modifié le taux d'apprentissage pour trouver un équilibre entre la vitesse de convergence et la qualité de la solution finale.

De même, nous avons ajusté le nombre de pas de temps total `total_timesteps` pour garantir que le modèle ait suffisamment de temps pour apprendre efficacement tout en restant dans des limites de temps raisonnables.

```

1 def main():
2     t0 = time.time()
3     env = WebotsGymEnvironment()
4     print("environnement cree")
5     check_env(env)
6     print("verification de l'environnement")
7     model = PPO(policy="MultiInputPolicy", # Model definition
8                  env=env,
9                  learning_rate=1e-5,
10                 verbose=1,
11                 device='cpu',
12                 batch_size=64, # Factor of n_steps
13                 tensorboard_log='./PPO_Tensorboard',
14                               # Additional parameters
15                 n_steps=2048,
16                 n_epochs=10, # Number of policy updates per iteration
17                               of n_steps
18                 gamma=0.99,
19                 gae_lambda=0.95,
20                 clip_range=0.2,
21                 vf_coef=1,
22                 ent_coef=0.05)
23     print("demonstration") #Load learning data
24     model = PPO.load("PPO_results_9")
25     model.set_env(env)
26     print("debut de l'apprentissage")# Training
        model.learn(total_timesteps=800000)

```

Listing 4 – Gestion des paramètres

7.2 Entrainement du modèle

Il convient de souligner que cette étape d'apprentissage a demandé des ressources computationnelles significatives, en effet, l'entraînement d'un modèle d'apprentissage automatique, surtout dans un environnement complexe comme celui de la simulation de véhicules, nécessite un ordinateur assez puissant. Les calculs intensifs et les nombreux ajustements de paramètres exigent une puissance de traitement adéquate pour garantir des résultats efficaces et rapides. Ainsi, l'accès à un ordinateur performant était crucial pour mener à bien cette phase d'apprentissage.

Les codes sources de nos simulations ainsi que de notre implémentation du contrôleur PID et des algorithmes d'apprentissage par renforcement sont fournis en annexe pour référence.

III Partie fonctionnelle, matérielle

On s'intéresse à la partie "bas niveau" du dispositif. On va ici observer le fonctionnement des objets communicants et de la façon de les gérer. On va donc voir comment un module type microcontrôleur permettra d'interfacer des éléments permettant l'obtention de données critiques tel que les données de vitesse, possiblement les données de la batterie et les mesures du point mort du robot : La face arrière.

Il va falloir donc mélanger une exploitation "hardware" et "software", tout d'abord avec l'étude des composants et ce dont il peut nous fournir puis ensuite établir la lecture logicielle pour finalement gérer l'envoi vers l'unité centrale : la Raspberry Pi.

1 Composants auxiliaires spécifiques à cette partie

En plus du Lidar et de la Raspberry Pi énumérés plus tôt qui vont servir à aider à prendre la décision, on aura besoin de choisir notre matériel afin de faire déplacer la voiture :

1.1 Servomoteur

Un servomoteur est un moteur associé à une carte d'asservissement et à un potentiomètre prévu pour un contrôle en position angulaire. Les servomoteurs permettent d'actionner des pièces mobiles sur des projets robotiques. Ils se commandent via un signal ou grâce à un microcontrôleur. Il s'agit donc d'un élément essentiel puisque sans celui-ci, la voiture ne tourne pas...

Le servomoteur que l'on utilise est un Konekt Servo KN0913LVMG il permet d'être contrôlé numériquement ce qui permet le fonctionnement en PWM. Plus d'informations sur les valeurs à mettre dans la documentation du composant.

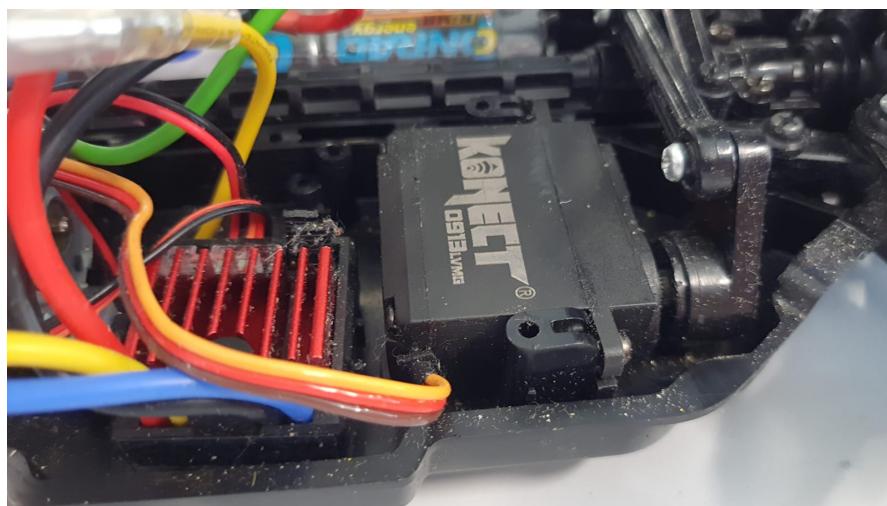


FIGURE 14 – Servomoteur

1.2 Support et châssis

Pour le choix du châssis de la voiture, nous avons opté pour le modèle "Tamiya TT02", ce choix a été dicté par les exigences spécifiques du cahier des charges de la course, en effet le châssis Tamiya TT02 est reconnu pour sa robustesse, sa fiabilité et sa compatibilité avec divers composants de contrôle et de propulsion, ce qui le rend particulièrement adapté à notre projet, la représentation de ce châssis est illustrée dans la figure suivante.



FIGURE 15 – Châssis de la voiture Tamiya TT02

Ce châssis (qui est plus en réalité le composant d'un kit voiture) est fourni avec son moteur contrôlé par PWM à l'aide du variateur lui aussi inclus, cette description restera brève car ce n'est pas dans nos enjeux de maîtriser la partie puissance du véhicule qui sera faite à notre place :



FIGURE 16 – Position du moteur et du variateur à droite

1.3 La Batterie

Le Powerhouse 7000 est une batterie puissante conçue pour fournir une source d'énergie portable dans une variété de situations. Avec une capacité élevée, il peut alimenter une gamme de dispositifs, des petits appareils électroniques aux équipements plus gros et plus énergivores. Afin de garantir un fonctionnement continu, nous avons opté pour en avoir deux au minimum.

1.4 Acquisiton au point mort

Le SRF10 est un capteur ultrasonique de distance compact et précis. Il utilise des ondes sonores pour mesurer les distances jusqu'à 6 mètres avec une précision de ± 1 cm. Avec une interface simple en I²C, il est facile à intégrer dans des projets électroniques. Sa petite taille le rend idéal pour les applications embarquées, tel que notre véhicule.



FIGURE 17 – SRF10 Télémètre

2 La carte Microcontrôleur

La famille STM32 est une série de microcontrôleurs 32- bits en circuits intégrés réalisés par la société Franco-Italienne STMicroelectronics. Les puces STM32 sont regroupées dans différentes séries proches, basées sur les processeurs d'architecture ARM 32-bits, tels que le Cortex-M7F, le Cortex-M4F, le Cortex-M3, Cortex-M0+, ou le Cortex-M0.

Nous utiliserons pour des raisons de simplicité une carte de développement NUCLEO. Cette carte contient en plus d'un microcontrôleur des composants utiles au fonctionnement du robot. Ce choix est motivé en plus par un coût plus faible par rapport à une carte Arduino ainsi que l'aisance de l'équipe au développement sur la plate-forme dédiée pour cette famille de microcontrôleur.



FIGURE 18 – STM32L432KC

La STM32L432KC est un microcontrôleur 32 bits de la famille STM32L4 de STMicroelectronics, offrant des performances élevées et une faible consommation d'énergie. Voici quelques-unes de ses spécifications principales :

1. Architecture ARM Cortex-M4 à 80 MHz.
2. Mémoire flash jusqu'à 256 Ko et Mémoire RAM jusqu'à 64 Ko.
3. Faible consommation d'énergie avec plusieurs modes de sommeil.
4. Nombreux périphériques intégrés, tels que des interfaces séries UART, SPI, I2C etc.
5. Convertisseur analogique-numérique (ADC) jusqu'à 12 bits avec jusqu'à 5 canaux.
6. Convertisseur numérique-analogique (DAC) jusqu'à 12 bits avec 1 ou 2 canaux.
7. Nombreux timers et compteurs.
8. Interfaces de communication série, y compris USART, SPI, I2C.
9. Multiples broches d'interruption externe.

Ces spécifications peuvent varier légèrement en fonction de la configuration spécifique de la puce et des fonctionnalités activées. Néanmoins, certaines de ces spécifications seront très intéressantes et utiles non seulement par leur présence mais aussi par le support logiciel énuméré plus bas.

3 Liaisons microcontrôleur

3.1 Protocole de communication SPI

Il s'agit d'un protocole de communication série synchrone largement utilisé pour l'échange de données entre différents périphériques électroniques. Il permet la transmission bidirectionnelle de données à haut débit entre un maître et plusieurs esclaves sur un bus de données. Le SPI est populaire en raison de sa simplicité, de sa flexibilité et de sa rapidité, ce qui en fait un choix courant dans de nombreux systèmes embarqués, notamment dans les domaines de l'informatique embarquée, de l'automatisation industrielle, et des communications sans fil.

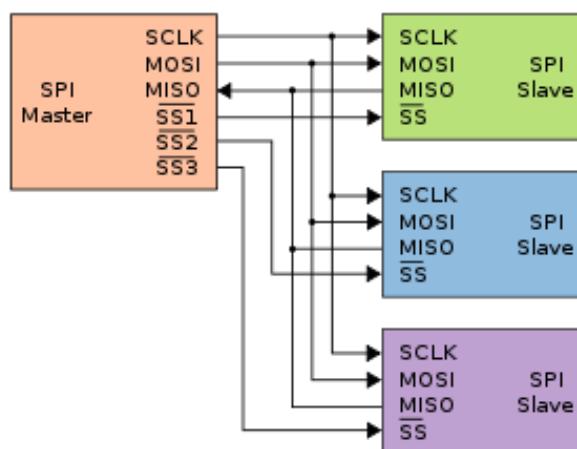


FIGURE 19 – Illustration du protocole SPI associé à ses maîtres et esclaves

Les lignes de communication dans le bus SPI (Serial Peripheral Interface) sont les suivantes :

1. **SCLK (Serial Clock)** : Cette ligne est utilisée pour synchroniser la transmission des données entre le maître et les esclaves. Le signal d'horloge est généré par le maître et contrôle la vitesse de transmission des données.
2. **MOSI (Master Out Slave In)** : Sur cette ligne, le maître envoie les données aux esclaves. Les données sont transmises du maître vers les esclaves et sont lues par ces derniers.
3. **MISO (Master In Slave Out)** : Les esclaves envoient les données au maître sur cette ligne. Les données sont transmises des esclaves vers le maître et sont lues par ce dernier.
4. **SS (Slave Select)** : Cette ligne est utilisée par le maître pour sélectionner le périphérique esclave avec lequel il souhaite communiquer. Lorsqu'elle est activée (mise à un niveau logique bas), un esclave spécifique est choisi pour la communication, tandis que les autres esclaves restent inactifs. Elle n'est pas obligatoire pour un seul esclave.

Ces lignes de communication permettent d'établir une communication série synchrone entre le maître et les esclaves, facilitant l'échange de données et le contrôle entre les différents périphériques connectés au bus SPI.

Le principe ici est d'utiliser le microcontrôleur en mode esclave et la Raspberry Pi en mode maître. L'interface servira à envoyer des données de capteurs du microcontrôleur à l'unité intelligente et si nécessaire envoyer des données de diagnostic de cet unité au microcontrôleur.

3.2 Protocole de communication I2C

Le bus I²C (Inter-Integrated Circuit) est un protocole de communication série à deux fils, un pour les données (SDA) et un pour l'horloge (SCL). Un maître contrôle la communication, en envoyant des adresses et des commandes aux périphériques esclaves. Chaque esclave a une adresse unique. Les données sont transmises en bits, avec des cycles d'horloge synchronisés, permettant une communication efficace entre les composants d'un système.

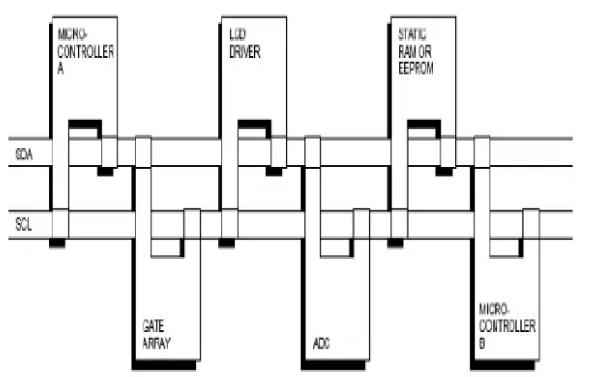


FIGURE 20 – Illustration du protocole I²C associé à ses maîtres et esclaves

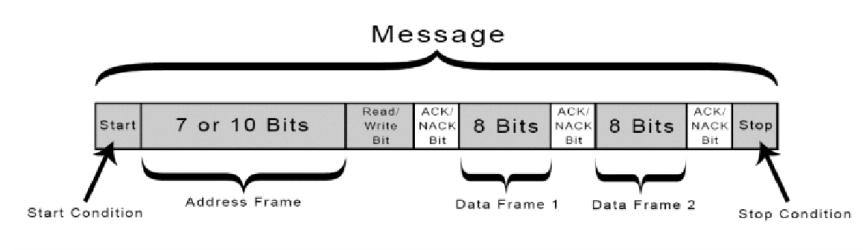


FIGURE 21 – Trame du protocole I²C

Le protocole I²C échange des messages/données sous formes de trames au début de l'échange et de l'envoi des données. Il faut que la condition de départ "START" soit réaliser, cette condition est suivit directement de l'adresse du composant qui a été appelé par le maître (Adress Frame) qui est codé sur 7 ou 10 bits, chaque composant possède une adresse unique.

Le 8ème ou 11ème bit de la trame est le bit R/W (Read/Write) qui permet de déterminer le sens du transfert si : R/W = 0 alors on est en mode écriture donc le maître parle et l'esclave écoute, une donnée est donc envoyé à l'esclave et si R/W = 1 alors on est en mode lecture et donc le maître écoute et l'esclave parle, une donnée est reçue par le maître. Seul les esclaves dont les adresses auront été reconnus participeront à l'échange le reste ne participera pas, on utilise le bit ACK pour la confirmation de la réception du message.

Ce type de bus est notamment utile pour les transmissions d'un capteur à un autre et sera la liaison pour récupérer les données à partir d'un composant métrologique : Le télémètre laser SRF10 arrière.

3.3 Liaisons analogiques et digitales auxiliaires

Une liaison analogique sous forme d'une ADC est utilisée par le microcontrôleur pour lire la tension de la batterie. Une liaison est utilisée par l'écran u8g_II pour afficher des informations. Des liaisons digitaux sont utilisés au niveau d'un buzzer et de la fourche optique.

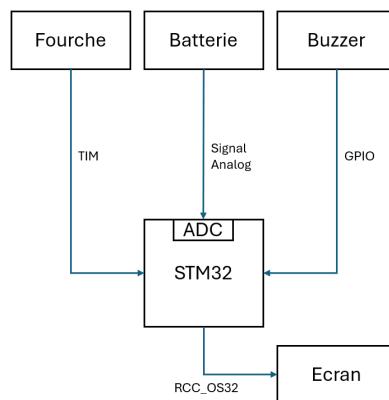


FIGURE 22 – Schéma des liaisons auxiliaires

4 Utilisation du microcontrôleur

A ce stade, nous avons toutes les informations nécessaires afin de déterminer le plan à suivre pour réaliser la tâche que doit effectuer cette partie : La gestion des données de capteurs et l'affichage diagnostic.

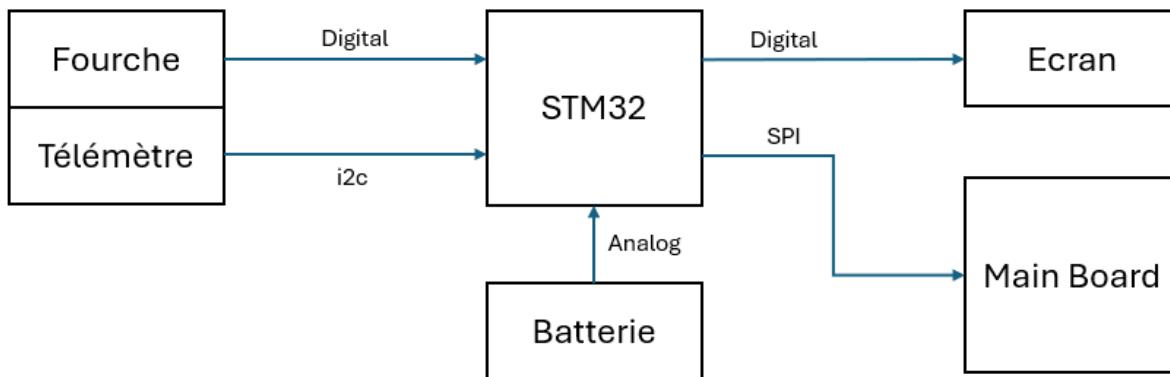


FIGURE 23 – Schéma du fonctionnement des éléments principaux de la partie microcontrôleur

Cette partie concerne également les outils utiles et utilisés pour le développement sur microcontrôleur. De nombreux outils externes à la formation ont été appris et maîtrisés par l'équipe afin de mener cette partie malgré le rapprochement des notions vu au sein de la formation (ils ont cependant été complémentaires, l'utilité de ces enseignements ont évidemment été appliqués dès que nécessaire).

4.1 Outils de programmation

Sur les cartes microcontrôleur du type que nous utilisons, il existe une multitude de programmes permettant d'écrire et compiler du code dans différents langages de programmation. Ici, on va s'intéresser au logiciel constructeur, fiable, complet et robuste pour notre démarche : STM32CubeIDE



FIGURE 24 – STM32CubeIDE

4.2 Fonctionnalités utiles du logiciel

Génération de Code par initialisation des périphériques

Ce logiciel a la particularité d'avoir un module nommé STM32CubeMX. Ce module est particulièrement intéressant car à l'aide des données du modèle du microcontrôleur, il permet la génération de code.

On peut tout d'abord sélectionner l'interface à implémenter. Les paramètres sont pré-remplis par le programme et il faut sauvegarder le fichier .ioc afin de générer automatiquement le code. Les paramètres ne sont pas idéaux cependant et il est nécessaire de retoucher certains paramètres avant de générer :

- Les fréquences d'horloges doivent être calculés avant de les générer, il faudra aussi prendre en compte des horloges modifiés si nécessaire.
 - Certaines normes par défaut ne peuvent pas correspondre. C'est le cas notamment d'une liaison SPI où certaines versions du logiciel remplissent automatiquement la taille de la trame sur 4 bits (8 bits dans notre application). Il faut souvent vérifier la bonne norme au risque de détruire le matériel auxiliaire.
 - Les périphériques sont initialisés en mode bloquant par défaut. En général les applications bloquantes sont largement suffisantes pour un bon fonctionnement. Cependant il est déconseillé dans les bus de transmission car il risque de bloquer le programme en attendant une trame. Il faut alors effectuer les réglages afin de procéder à un fonctionnement par interruption CPU ou à accès direct de la mémoire (DMA).

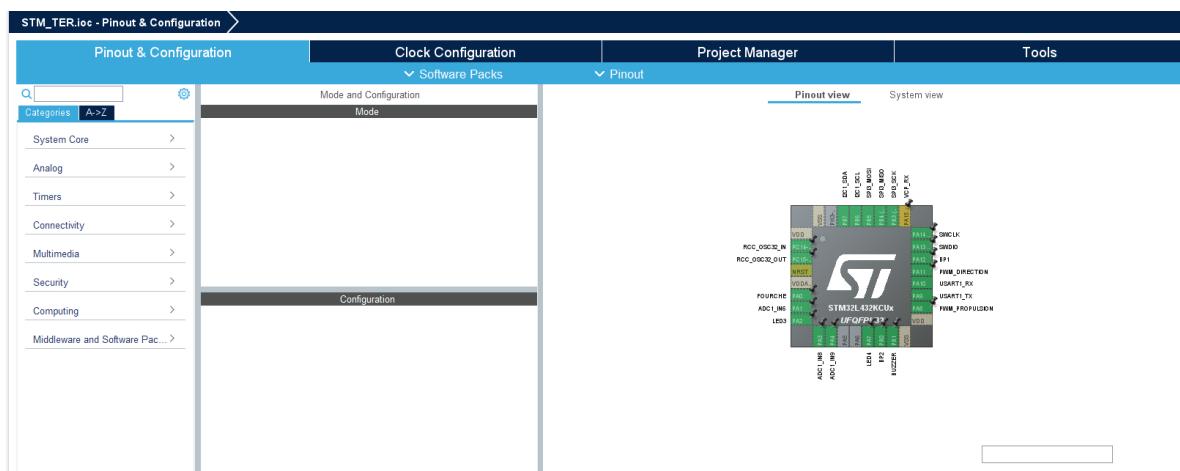


FIGURE 25 – Interface principale MX

Manipulation précise du matériel

Les paramètres du fichier .ioc permet de modifier des aspects du microcontrôleur avec précision tel que l'horloge. Ces modifications permettent plusieurs choses : Limiter la consommation électrique du programme, essentiel dans le contexte d'un véhicule fonctionnant sous batterie et dans certains cas découpler les horloges afin d'éviter les conflits. Il est également possible de manipuler des aspects de la mémoire même si ce n'est pas forcément ce que nous avons le plus besoin.

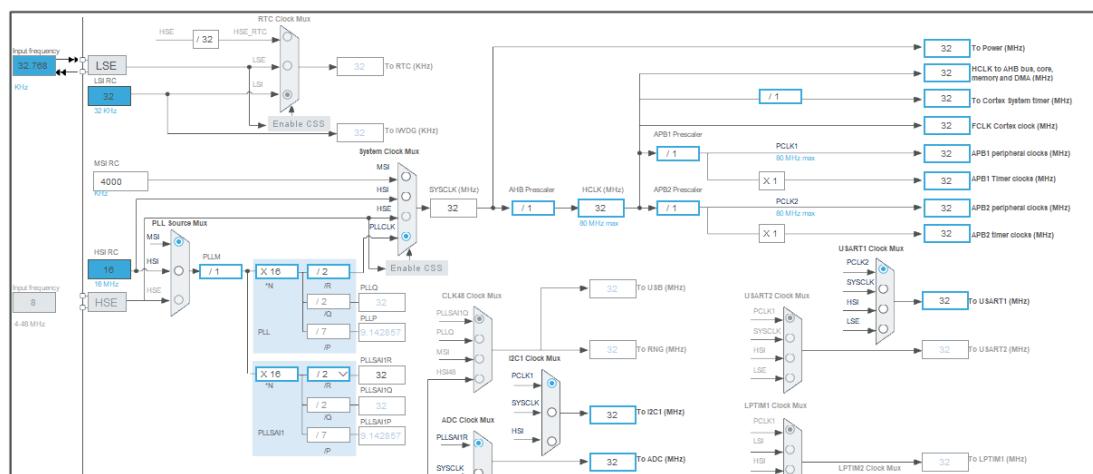


FIGURE 26 – Interface horloge

Fonction robuste de débogage

L'outil de débogage du logiciel est très intéressant. Il est équipé d'une analyse fine des variables en direct et de la mémoire. Contrairement à d'autres débogueurs, il est possible de vérifier le fonctionnement correct du programme et de ce qui se passe en mémoire ainsi que la valeur des variables en direct sans point d'arrêt. Ceci peut se révéler très pratique pour diagnostiquer les problèmes en exécution réelle notamment pour le fonctionnement d'une communication entre le microcontrôleur et un autre périphérique (SPI avec Raspberry Pi).

Live Expressions X		
(x)= Variables	Breakpoints	Expressions
Registers	SFRs	
Expression	Type	Value
✓ SPI_TxBuffer	uint8_t [16]	[16]
(x)= SPI_TxBuffer[0]	uint8_t	53 '5'
(x)= SPI_TxBuffer[1]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[2]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[3]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[4]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[5]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[6]	uint8_t	0 '\000'
(x)= SPI_TxBuffer[7]	uint8_t	0 '\000'

FIGURE 27 – Variables en direct (Live Expressions)

4.3 Test du microcontrôleur

Un tout premier programme de test existe. Il permet de vérifier l'installation de tous les périphériques nécessaires puis les affiche sur l'écran LCD relié au microcontrôleur. On va ainsi vérifier le bon fonctionnement des bus I2C, ADC et digitaux.

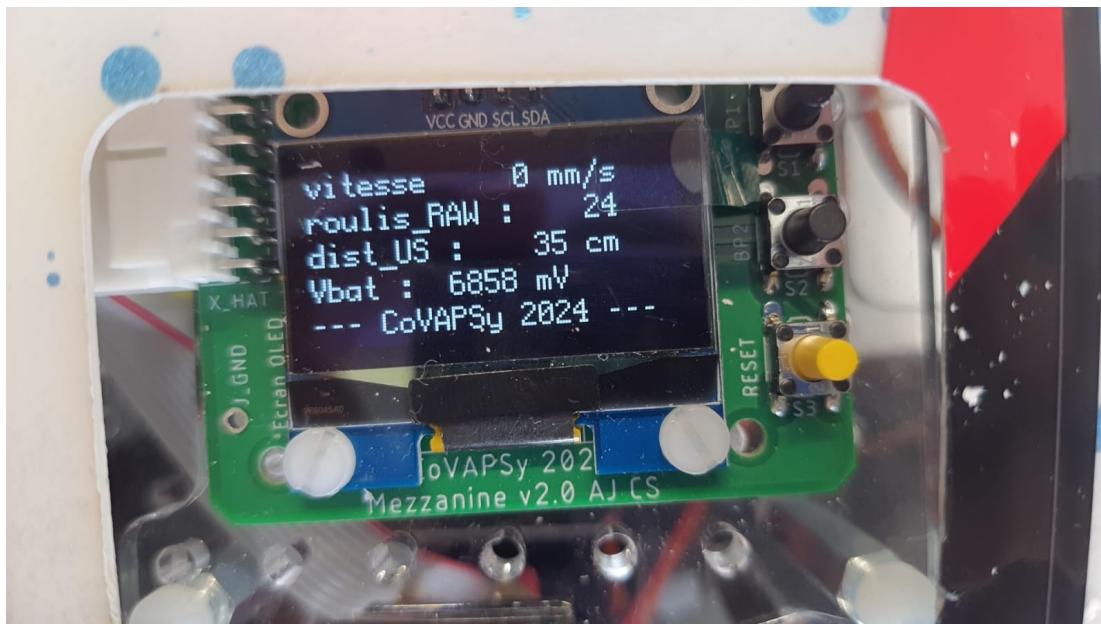


FIGURE 28 – Affichage des données à visualiser rapidement

On cherche ici à diagnostiquer également la validité des données et comment les exploiter, les données affichées sont les suivantes :

- Vitesse (mm/s)
- Roulis
- Distance arrière (cm)
- Niveau de batterie (mV), utile à détecter les chutes de tension pour le remplacement de la batterie

On peut diagnostiquer l'état des capteurs simplement en déplaçant la voiture d'avant à arrière, le faire tourner et mettre un objet au niveau du capteur SRF10.

Ce programme ayant toutes les configurations nécessaires pour récupérer les données utiles et en afficher, va nous permettre d'être la base de notre code pour la suite : Transmettre ces données.

4.4 Utilisation pour la transmission de données du microcontrôleur

La partie à programmer plus complexe par son implantation et par son diagnostic est la communication SPI entre le microcontrôleur et la Raspberry Pi. Il faut faire tout les pré-réglages nécessaires :

Mise en place SPI sur Raspberry Pi

Il faut mettre en place les liaisons SPI au niveau de la Raspberry Pi. Les réglages de ce paramètre suivent ce protocole :

1. **Ouvrir le menu de configuration** : Ouvrir un terminal et taper la commande suivante :

```
sudo raspi-config
```

2. **Accéder aux options d'interface** : Dans le menu de configuration, utiliser les touches fléchées pour sélectionner *Interfacing Options* et appuyer sur *Entrée*.
3. **Activer le SPI** : Sélectionner *SPI* et appuyer sur *Entrée*. Lorsque demandé d'activer le SPI, sélectionner *Yes* et appuyer sur *Entrée*.

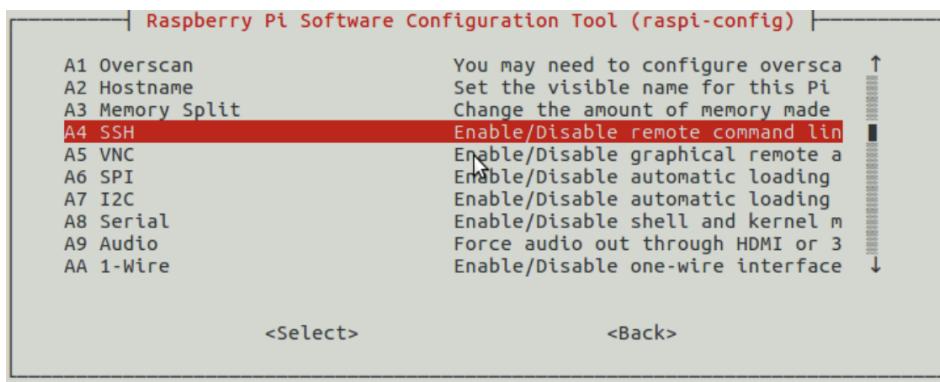


FIGURE 29 – raspi-config

4. **Quitter le menu de configuration** : Utiliser la touche *Tab* pour sélectionner *Finish* et appuyer sur *Entrée*. Un redémarrage du Raspberry Pi peut être nécessaire pour que les changements prennent effet.
5. **Vérifier l'activation du SPI** : Après le redémarrage, ouvrir un terminal et taper la commande suivante pour vérifier que le SPI est activé : Cela devrait afficher quelque chose comme `/dev/spidev0.0` et `/dev/spidev0.1`.

```
ls /dev/spidev*
```

Une fois ces étapes complétées, le SPI sera activé sur le Raspberry Pi, permettant de communiquer avec des périphériques SPI. Un code sous python pourra exploiter le SPI à l'aide de la bibliothèque *spidev*.

Mise en place SPI sur microcontrôleur

On va appliquer les générateur de code MX pour générer un préprogramme SPI : Pour générer le code à l'aide de STM32CubeMX pour configurer le SPI sur 8 bits en mode esclave sans interruptions, on va prodéder de la manière suivante

1. Ouvrir le .ioc : Ceci va démarrer MX.

2. Configurer les broches SPI :

- (a) Dans l'onglet *Pinout*, sélectionner les broches correspondantes au SPI (par exemple, SPI1_MISO, SPI1_MOSI, SPI1_SCK, et SPI1_NSS).
- (b) Les broches sélectionnées apparaîtront sur la carte, configurées pour leur fonction SPI respective. Ici, nos broches sont PB3, PB4 et PB5, nous n'avons pas besoin du signal SS car nous utilisons qu'un seul périphérique.

3. Configurer le SPI en mode esclave :

- (a) Aller dans l'onglet *Configuration*.
- (b) Cliquer sur *SPI1 Configuration* pour ouvrir les paramètres du SPI.
- (c) Régler les paramètres suivants :
 - **Mode** : Sélectionner *Slave*, très important
 - **Data Size** : Sélectionner *8 Bits*.
 - **First Bit** : Sélectionner *MSB First*, on travaille en Big Endian
 - **NSS Signal Type** : Sélectionner *Hardware NSS Signal*.

4. Générer le code : Sauvegarder simplement suffit

5. Configurer le code généré :

- (a) Ouvrir le projet généré sur l'IDE
- (b) Vérifier que la configuration du SPI est correcte dans les fichiers générés (notamment `main.c` et `stm32fxx_hal_msp.c`).

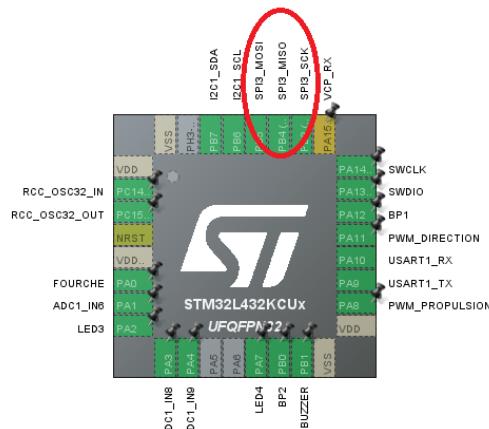


FIGURE 30 – MX avec Pins

Pour une implantation en interruption, plus robuste mais plus complexe, il faut effectuer les réglages supplémentaires :

- En mode NVIC (interruptions CPU) : Dans l'onglet *NVIC Settings*, cocher la case "*SPI global interrupt*"
- En DMA (Accès direct de la mémoire) : Dans l'onglet *DMA Settings*, activer les interruptions en mode circulaire.

Après avoir complété ces étapes, le SPI sera configuré sur 8 bits en mode esclave sans interruptions. Il ne reste que l'implantation dans le programme principal que nous allons planter à partir du programme test initial : Il faudra ajouter la commande suivante :

- Pour une transmission simple :

```
HAL_SPI_Transmit(&hspi, Tx_Data, sizeof(Tx_Data), HAL_MAX_DELAY);
```

- Pour une réception :

```
HAL_SPI_Receive(&hspi, Rx_Data, sizeof(Rx_Data), HAL_MAX_DELAY);
```

- Pour une transmission-réception avec "n" la taille commune des tableaux d'émission et réception :

```
HAL_SPI_TransmitReceive(&hspi1, Tx_Data, Rx_Data, n, HAL_MAX_DELAY);
```

Dans le cas des interruptions, le cas est plus complexe : Il faut écrire la fonction pour la fonction que l'on veut tel que pour une transmission :

```
HAL_SPI_Transmit_IT(&hspi, Tx_Data, sizeof(Tx_Data));
```

ou dans le cas DMA

```
HAL_SPI_Transmit_DMA(&hspi, Tx_Data, sizeof(Tx_Data));
```

Puis écrire dans la fonction retour "Callback" un comportement désiré, l'idéal serait la déclaration d'un "Flag" qui déclencherait un bout de code ou l'analyse de la trame reçue :

```

1 // A remplacer par Rx ou TxRx en fonction des besoins
2 void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef * hspi)
3 {
4     Flag = true; // Exemple
5 }
```

Note importante au niveau de la transmission SPI : il est nécessaire de savoir que la transmission SPI est directement influencée par l'affichage sur écran. En effet : Les trames envoyées par SPI deviennent faussées si les envois deviennent trop complexe en parallèle de l'affichage (ex : TransmitReceive). Dans ce cas il est requis de sacrifier l'affichage écran afin de transmettre des bonnes trames plus régulièrement. Il faudra donc vérifier que toutes les interfaces fonctionnent (cf Test du microcontrôleur, 4.3)

IV Réalisation et mise en place

Cette partie est consacrée à la mise en fonctionnement de la voiture autonome. À l'aide de tous les éléments énumérés dans les parties précédentes, il nous reste à implémenter toutes les liaisons au sein du robot. Il est nécessaire de préparer une mise en place préliminaire avant de pouvoir concrètement réaliser notre tâche à l'aide de nombreux tests et calibrations. On veut donc vérifier tout d'abord le bon comportement des commandes avant de passer au bon fonctionnement des communications et récupérations de données pour finir par implémenter cet apprentissage.

1 Construction et mise en fonctionnement du robot

La première partie de l'assemblage a été effectuée par l'équipe, cette partie concerne le châssis :

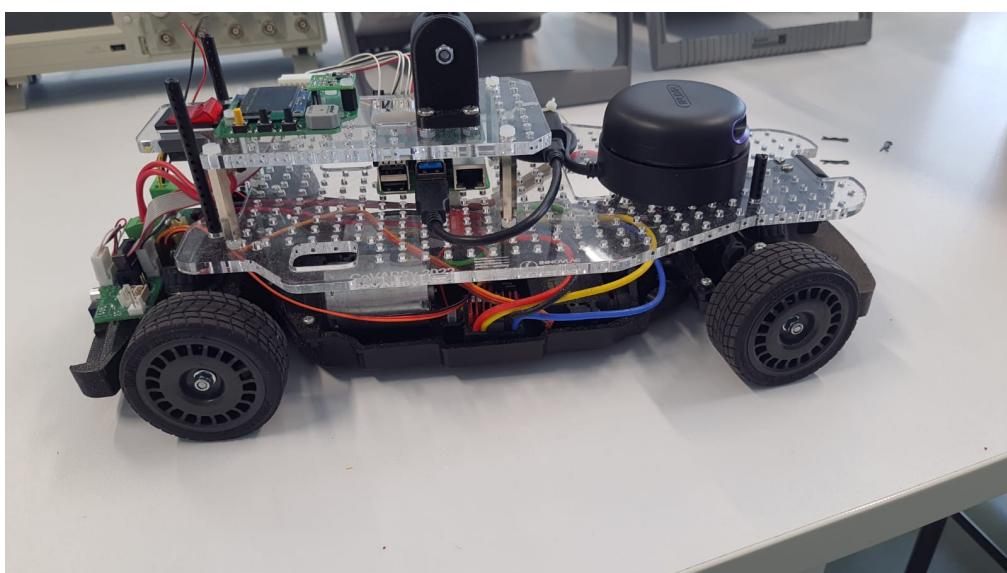


FIGURE 31 – Voiture assemblée

La construction finale du robot a été effectué par l'Innovlab de l'IUT de Cochon, ce qui veut dire que la partie électrique a été montée par autre que nous.

1.1 Les débuts avec Raspberry Pi OS

La première manipulation directe logicielle sur la voiture est la partie intelligente à savoir la Raspberry Pi que l'on manipulera de deux manières :

- **Installation initiale avec un écran** : Pour la configuration initiale, nous avons utilisé un écran connecté à la Raspberry Pi. Cela nous a permis d'installer les paquets et dépendances nécessaires pour notre projet. Cette étape était cruciale pour s'assurer que tous les composants logiciels requis étaient en place et correctement configurés.

- **Utilisation de la connexion SSH** : Pour pouvoir manipuler la raspberry le SSH est extrêmement pratique car il nous permet de modifier le code et de faire des ajustements à distance, sans avoir besoin d'un accès physique direct à la Raspberry Pi. Cette flexibilité est particulièrement utile pour les développements et les tests rapides

1.2 Tests des composants et ajustement du matériel

Puisque la fabrication du robot n'a pas été supervisé par nous, il faudra ré-effectuer le test individuel des composants du véhicule afin de détecter les défauts mécaniques et/ou électriques.

Pour s'assurer du bon fonctionnement de notre système, nous avons utilisé des programmes de test fournis par Anthony Juton, ces programmes ont été essentiels pour vérifier les capacités de propulsion et de direction de notre voiture les tests comprenaient

- **Tests de propulsion** : Ces tests consistaient à faire avancer et reculer le robot pour vérifier que les moteurs fonctionnaient correctement et que les vitesses étaient bien contrôlées.
- **Calibrage de la direction** : Pour s'assurer que le robot pouvait tourner correctement, nous avons effectué des calibrages pour les mouvements à gauche et à droite.

1.3 Les problèmes rencontrés

Lors de la mise en place initiale de la voiture, nous avons évidemment rencontré des obstacles imprévus puis réglés promptement :

- **La carte SD** : Nous avons rencontré un problème majeur avec la carte SD de la Raspberry Pi, en effet elle a surchauffé et a été endommagée, ce qui a entraîné la perte de toutes les données et configurations stockées sur celle-ci. Nous avons donc dû réinstaller le système d'exploitation, les bibliothèques nécessaires et reconfigurer tous les paramètres sur une nouvelle carte SD. Cette situation a retardé notre progression mais nous avons réussi à remettre en place l'environnement de développement et à continuer notre travail.
- **Le servomoteur** : Le servomoteur, essentiel pour la direction et le contrôle précis de notre véhicule, a surchauffé et a grillé. Cela a interrompu nos tests et nos calibrages, ce contretemps a pas mal retardé notre progression, mais une fois le nouveau servomoteur installé, nous avons pu reprendre nos essais.

1.4 Un premier fonctionnement par asservissement linéaire

Nous avons effectué des tests de simulation à l'aide de Webots en utilisant un modèle d'asservissement linéaire, maintenant, nous allons implémenter ce modèle sur la Raspberry Pi, ce qui nécessitera quelques modifications pour s'assurer que le code est compatible avec l'environnement de la Raspberry Pi.

Une boucle principale gère le démarrage et l'arrêt du LIDAR ainsi que la conduite autonome en fonction de l'entrée de l'utilisateur grâce à des threads et cela afin d'exécuter les différentes tâches de manière asynchrone, permettant ainsi une acquisition continue des données et une conduite réactive.

La première étape consiste à initialiser les moteurs de la voiture en configurant les signaux PWM nécessaires pour contrôler la propulsion et la direction, ils sont configurés pour fonctionner à une fréquence de 50 Hz, et les moteurs sont initialisés avec des valeurs spécifiques pour les états de repos et de mouvement qui ont été obtenu grâce aux tests de calibrage.

```

1 stop_prop = 7.5
2 point_mort_prop = 0.5
3 vitesse_max_m_s = 7
4 vitesse_m_s = 5
5 lspeed = 1.0
6 angle_pwm_min = 5.9 # gauche
7 angle_pwm_max = 9.2 # droite
8 angle_degre_max = 18 # vers la gauche
9 angle_pwm_centre = 7.6
10 angle_degre = 0
11 pwm_prop = HardwarePWM(pwm_channel=0, hz=50)
12 pwm_dir = HardwarePWM(pwm_channel=1, hz=50)

```

Listing 5 – Initialisation des moteurs

Le LIDAR est connecté via un port USB avec un baudrate de 256000. Une fois connecté, le LIDAR est démarré et les mesures de distance sont préparées pour être enregistrées dans un tableau global.

```

1 # Connexion et démarrage du lidar
2 lidar = RPLidar("/dev/ttyUSB0", baudrate=256000)
3 lidar.connect()
4 print(lidar.get_info())
5 lidar.start_motor()
6 time.sleep(2)
7 acqui_tableau_lidar_mm = [0] * 360 # création d'un tableau de 360 zeros
8 tableau_lidar_mm = [0] * 360
9 drapeau_nouveau_scan = False
10 Run_Lidar = False

```

Listing 6 – Connexion et démarrage du LIDAR

La fonction **lidar_scan** est responsable de l'acquisition continue des données LIDAR, les angles et les distances sont enregistrés dans un tableau global pour être utilisés par l'algorithme de conduite autonome.

```

1 def lidar_scan():
2     global drapeau_nouveau_scan
3     global acqui_tableau_lidar_mm
4     global Run_Lidar
5     global lidar
6     print("tache lidar_scan demarree")
7     scan_avant_en_cours = False
8     while Run_Lidar:
9         try:
10             for _, _, angle_lidar, distance in lidar.iter_measures(
11                 scan_type='express'):
12                 angle = min(359, max(0, 359 - int(angle_lidar)))
13                 if (angle >= 260) or (angle <= 100):
14                     acqui_tableau_lidar_mm[angle] = distance

```

Listing 7 – Acquisition des données du LIDAR

La fonction `set_direction_degre` ajuste le signal PWM pour contrôler la direction de la voiture en fonction de l'angle désiré, cette fonction calcule le signal PWM correspondant à l'angle de direction souhaité.

```

1 def set_direction_degre(angle_degre):
2     angle_pwm = angle_pwm_centre - (angle_pwm_max - angle_pwm_min) *
3         angle_degre / (2 * angle_degre_max)
4     if angle_pwm > angle_pwm_max:
5         angle_pwm = angle_pwm_max
6     if angle_pwm < angle_pwm_min:
7         angle_pwm = angle_pwm_min
8     return angle_pwm

```

Listing 8 – Contrôle de la Direction

La fonction `set_vitesse_m_s` ajuste le signal PWM pour contrôler la vitesse de la voiture, elle prend en compte les valeurs de vitesse maximale et minimale et ajuste la propulsion en conséquence.

```

1 def set_vitesse_m_s(vitesse_m_s):
2     if vitesse_m_s > vitesse_max_m_s:
3         vitesse_m_s = vitesse_max_m_s
4     elif vitesse_m_s < -vitesse_max_m_s:
5         vitesse_m_s = -vitesse_max_m_s
6     if vitesse_m_s == 0:
7         pwm_prop.change_duty_cycle(stop_prop)
8     elif vitesse_m_s > 0:
9         vitesse = vitesse_m_s * 1.5 / 8
10        pwm_prop.change_duty_cycle(stop_prop + point_mort_prop + vitesse)
11    elif vitesse_m_s < 0:
12        vitesse = vitesse_m_s * 1.5 / 8
13        pwm_prop.change_duty_cycle(stop_prop - point_mort_prop + vitesse)

```

Listing 9 – Contrôle de la Vitesse

La fonction `recule` permet à la voiture de reculer en ajustant la vitesse à une valeur négative pendant une durée déterminée, elle utilise également les données du télémètre pour s'assurer que la marche arrière peut être enclenchée sans risquer de heurter un obstacle à l'arrière.

```

1 def recule():
2     set_vitesse_m_s(-vitesse_max_m_s)
3     time.sleep(1.5)
4     set_vitesse_m_s(0)
5     time.sleep(0.1)
6     set_vitesse_m_s(-5)
7     time.sleep(1)

```

Listing 10 – Fonction de Recule

Toutes ces fonctions sont incorporées dans la fonction principale, `conduite_autonome`, qui est le cœur de notre asservissement, cette fonction analyse les données du LIDAR pour détecter les obstacles, décide de la direction et de la vitesse appropriées, et ajuste les PWM en conséquence. On a pris des secteurs de 20 degrés afin de trouver la distance minimale et prendre des décisions basées sur ces valeurs.

Ainsi nous avons réussi à implémenter un modèle d'asservissement linéaire testé en simulation, permettant à la voiture de détecter et d'éviter les obstacles en temps réel, grâce

aux données du LIDAR pour assurer une réaction rapide et autonome aux changements dans l'environnement.

La structure du code, avec des fonctions dédiées pour le contrôle de la vitesse, de la direction, la marche arrière et la gestion des données du LIDAR, assure une grande flexibilité et facilité de maintenance.

2 Application de l'apprentissage par renforcement

Création du programme sur Raspberry Pi à partir du programme Webots

Il s'agit de faire fonctionner l'asservissement sur le véhicule physique. Nous avons un moyen de faire rouler le robot à l'aide d'un asservissement linéaire et un moyen d'utiliser un modèle d'apprentissage. Il est temps de combiner les deux. Une partie préliminaire consiste en l'identification de la partie qui permet le déplacement du véhicule dans le code :

Comme nous l'avons fait précédemment, dans la simulation Webots, nous utilisons l'API Webots pour simuler les capteurs afin de générer les données et d'accéder directement aux données des capteurs simulés via l'interface fournie par Webots, et pour simuler le contrôle du véhicule, les fonctions Webots (par exemple, setCruisingSpeed() et setSteeringAngle()) sont utilisées pour le contrôler. Mais lorsque nous contrôlons réellement le chariot, les signaux que nous capturons proviennent de l'environnement physique réel, de sorte que tous les signaux de contrôle analogiques et les capteurs doivent être implémentés par le biais de dispositifs matériels réels (tels que rplidar et rpi.hardware_pwm). Pendant le processus de contrôle réel, l'interface de contrôle doit être interconnectée avec la bibliothèque matérielle (STM32), nous avons décrit la communication entre eux plus tôt, nous devons l'initialiser, pour l'importation du modèle, la bibliothèque stable_baselines3 est utilisée pour charger le modèle entraîné.

```

1 from rplidar import RPLidar
2 import numpy as np
3 import time
4 from rpi.hardware_pwm import HardwarePWM
5 import threading
6 from rpi.hardware_pwm import HardwarePWM
7 from stable_baselines3 import PPO, SAC
8 import spidev

```

Listing 11 – Importation des modules Python

```

1 bus = 0
2 spi_len = 6
3 spi = spidev.SpiDev()
4 TxBuffer = [0]*spi_len
5 RxBuffer = [0]*spi_len
6 valeur_fourche = 0
7 valeur_arriere = 0

```

Listing 12 – Initialisation des paramètres pour la communication SPI

```

1 def set_direction_degre(self, angle_degre) :
2
3     angle_pwm = self.angle_pwm_centre + self.direction * (self.
4         angle_pwm_max - self.angle_pwm_min) * angle_degre / (2 * self.
5             angle_degre_max)
6     if angle_pwm > self.angle_pwm_max :
7         angle_pwm = self.angle_pwm_max
8     if angle_pwm < self.angle_pwm_min :
9         angle_pwm = self.angle_pwm_min

```

Sur le Raspberry Pi, contrairement à Webots, nous utilisons la modulation de largeur d'impulsion (PWM) pour ajuster la vitesse et la direction du véhicule. Pour ajuster ces deux paramètres, nous avons besoin de deux signaux PWM distincts.

```

1 self.lidar = RPLidar("/dev/ttyUSB0", baudrate=256000)

```

Cela a pour effet de créer un nouvel objet RPLidar et d'établir une communication avec le dispositif physique via le chemin de fichier et le débit en bauds du premier dispositif série USB.

Le code Python ci-dessous, destiné à être exécuté sur la Raspberry Pi, illustre comment nous avons implémenté ces fonctionnalités. Il est structuré pour gérer à la fois les actions de déplacement et la détection des obstacles, tout en intégrant un modèle d'apprentissage par renforcement pour une conduite autonome.

Initialisation et Configuration Le programme commence par l'importation des bibliothèques nécessaires et l'initialisation des composants :

```

1 from rplidar import RPLidar
2 import numpy as np
3 import time
4 from rpi_hardware_pwm import HardwarePWM
5 import threading
6 from stable_baselines3 import PPO
7 import spidev

```

Listing 13 – Importation des bibliothèques

La classe **Driver** gère les commandes de propulsion et de direction du véhicule. Elle configure les PWM (Pulse Width Modulation) pour le contrôle de la vitesse et de la direction.

La classe **car_lidar** gère la détection des obstacles en utilisant un LIDAR. Elle effectue des scans et fournit les données de distance pour la navigation autonome.

```

1 model = PPO.load("PP04s")

```

Listing 14 – Importation de modèle PPO

Grâce à l'importation du modèle, la fonction `conduite_autonome` utilise le modèle chargé pour effectuer des prédictions et des contrôles. En utilisant les mêmes espaces d'observation et d'action dans les environnements Webots et Raspberry Pi, et en assurant la cohérence du traitement des données et des signaux de contrôle, il est possible d'obtenir une précision dans les prédictions et les contrôles.

3 Mise en fonctionnement réelle

A ce stade, tout est prêt afin de pouvoir réaliser la course. Si on exclut les optimisations nécessaires et minimise l'importance de la performance, la voiture permet de rouler automatiquement et de détecter des obstacles. Certains choix restent à faire et des optimisations en direct ont été mis en place.

3.1 Choix et observations du jour de la course

Contre la montre

Après des tests intensifs et des comparaisons, il est apparu que l'algorithme d'apprentissage qui offrait les meilleurs résultats en termes de prise de virages était celui que nous devions choisir pour la course contre la montre. Cet algorithme avait démontré une meilleure capacité à anticiper et à naviguer efficacement à travers les virages serrés de la piste, ce qui était crucial pour maximiser notre vitesse et réduire notre temps au tour.

On note la difficulté de l'algorithme d'apprentissage à aborder les lignes droites et certains séries de virages ce que l'algorithme d'asservissement aurait pu corriger d'une meilleure façon. Malgré cela, l'algorithme d'apprentissage nous a permis de garantir le tour puisque sa robustesse notamment aux demi tours a été cruciale.

Tour d'obstacle

Pour l'épreuve contre des obstacles statiques, nous avons opté pour l'asservissement classique. Cet asservissement s'était montré plus efficace pour éviter les obstacles fixes et plus rapide sur les lignes droite que l'algorithme d'apprentissage, et cela est du à sa réponse rapide et précise aux changements détectés par le LIDAR. La fiabilité de l'asservissement dans des environnements où les obstacles ne bougent pas a donc orienté notre choix pour cette épreuve spécifique.

Cependant l'asservissement a des difficultés à observer des évènements successifs rapides et peut donc ignorer un ordre qu'il aurait fait si ceci étaient plus espacés la ou l'apprentissage aurait pu observer d'une meilleure manière. De plus, la programmation du recul arrière étant basé sur les évènements Lidar à certains points, il est fortement possible que même si le véhicule se retrouve dans une situation où il devrait reculer, les données captées par le Lidar n'enregistrent pas la bonne instruction.

Course

Lors de la course finale, où nous étions opposés à d'autres voitures, nous avons pris la décision stratégique d'utiliser le modèle d'asservissement linéaire pour maximiser notre vitesse, même si cela signifiait une précision moindre dans les virages et le fait que nous avions qu'une seule chance au niveau des collisions (si on rentre dans un mur, le recul risque de ne pas fonctionner correctement et coincer la voiture dans une boucle).

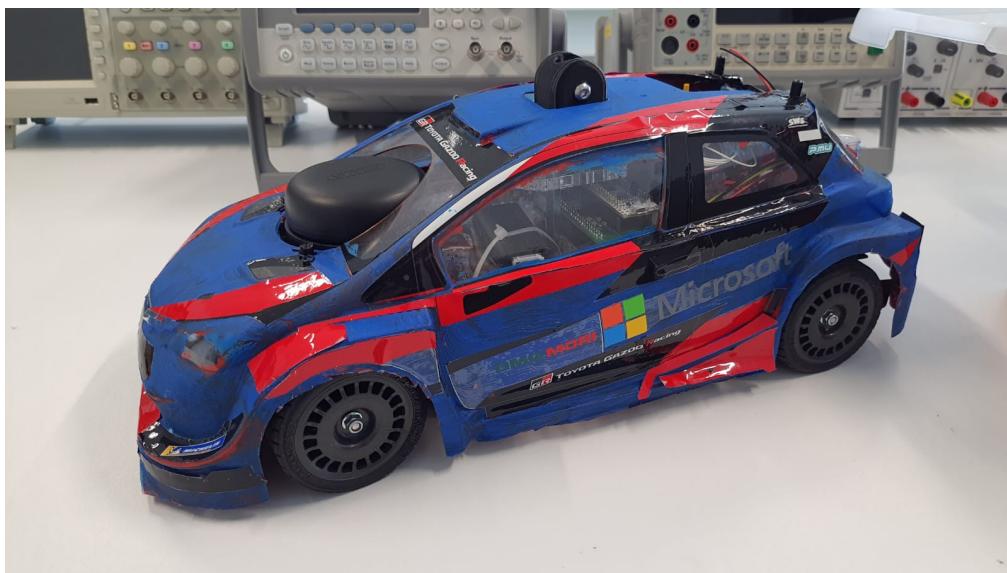


FIGURE 32 – Véhicule de course

3.2 Discussion sur l'efficacité des solutions

En ayant utilisé les deux solutions, nous pouvons observer les différences fondamentales de fonctionnement apparaissent entre le modèle d'apprentissage et un asservissement linéaire. Ces différences surviennent sous la forme d'une différence en comportement du véhicule :

- L'asservissement permet de courir sur la piste en étant seule très efficacement, plus que le modèle d'apprentissage.
- Le modèle d'apprentissage est théoriquement plus efficace dans la détection d'obstacles, mais cela reste à vérifier car ce modèle est plus lent en vitesse au vu de nos calibrations. Il est cependant robuste lors de virages et de détections plus complexes à effectuer (recul et virages successifs).

Quoi qu'il en soit, les deux algorithmes ont tous deux des avantages et défauts. On peut cependant retenir une chose : L'algorithme d'asservissement est un algorithme simplifié et l'algorithme d'apprentissage est un algorithme plus complexe. Dans un contexte de course de voitures autonomes à une piste simple et limitée en obstacles, l'asservissement est plus adapté. Le modèle d'apprentissage, bien calibré, serait clairement bien plus efficace dans une piste plus complexe : Embranchements, obstacles serrés, faux virages (si des éléments de ce type surviennent dans une course future).

4 Projections

Le robot actuellement remplit ses fonctions, il permet de rouler sur une piste à une certaine vitesse quoique modérée à l'aide d'un apprentissage et un fonctionnement plus basique mais plus robuste par un asservissement linéaire. Il y a des éléments développés et des éléments qui n'ont pas eu l'occasion d'être terminés et/ou implémentés correctement pour le jour de la course.

La possibilité que le modèle final pourrait être amélioré est à approfondir. En effet : il est fort probable qu'un entraînement plus rigoureux avec possiblement un type d'apprentissage autre que le PPO aurait un impact inconnu mais possiblement mélioratif, par exemple SAC. On pourrait aussi ajouter plus de paramètres au modèle tel que le roulis pour les virages ou même fusionner avec un algorithme de classification avec une caméra frontale ce qui malheureusement augmenterait grandement la complexité du programme.

Il est aussi envisageable de "fusionner" le fonctionnement par apprentissage et l'asservissement linéaire. On a constaté que le robot fonctionne très bien au niveau de l'asservissement en ligne droite et que si le robot aborde un virage au centre en mode apprentissage, il l'aborde efficacement.

Finalement, il serait possible d'améliorer tous les aspects du robot. On pourrait optimiser les consommations de la RPi et de la carte microcontrôleur par exemple en diminuant les vitesses d'horloge. On pourrait augmenter la vitesse en allégeant le robot de composants inutilisés (RPi caméra, quelques éléments imprimés 3D qui nous n'ont peu ou pas servi).

Quoi qu'il en soit, ces optimisations et propositions pourront peut-être améliorer le comportement de la voiture et la rendre plus fiable pour une prochaine itération de la course à condition que les règles ne changent pas.

Conclusion

Ce projet se résumant à la préparation d'une course de voitures autonomes nous a permis d'acquérir de l'expérience en termes de gestion de projet, gestion, coordination et organisation de plutôt larges équipes de travail et gestion d'organisation temporelle tel qu'une "deadline" qu'a été le jour de la course. Il a fallu s'organiser au niveau du temps de préparation qui nous sont proposés par l'organisation et nos emplois du temps mais aussi en externe pour mener à bien le projet notamment pour les tests en plus d'une gestion autonome.

Nous avons pu exploiter les acquis de la formation lors de la création du véhicule mais nous avons surtout du acquérir des compétences en plus : L'apprentissage de logiciels de programmation, de bibliothèques, de logiciels de gestion de travail (SSH), d'un système d'exploitation et des méthodes précises d'apprentissage. Nous avons du aussi développer un esprit de recul et prendre des décisions par rapport à des choix décisifs autant pour le matériel que pour les choix et ajustement de fin ce qui est crucial de bien choisir dans un projet de cet envergure.

Enfin, ce projet nous introduit bien à plusieurs domaines : Le domaine de l'embarqué intelligent sous forme de la Raspberry Pi et de ce qu'elle peut faire, l'embarqué plus orienté industriel avec les capteurs et le microcontrôleur et surtout le domaine de l'automobile et des véhicules autonomes. Ce projet a donc aussi vocation de nous inspirer pour un choix de parcours professionnel que ce soit dans un poste en automobile ou même pour un choix de poursuites d'études.

Finalement, ce projet fut une bonne application de notions relatifs au domaine de l'embarqué et du "Machine Learning" et est une mise en bouche de projets professionnels à l'avenir. Et malgré les difficultés surmontés lors de la réalisation de véhicule et les nombreuses heures supplémentaires effectués, nous avons acquis une bonne expérience pour un futur projet de ce type et pour certains passionnés de l'équipe, une future course de voitures.

Bibliographie

Références

- [1] J. Doe, J. Smith, and A. Brown, “Deep learning for vision and decision making in self-driving cars : Challenges with ethical decision making,” *International Journal of Advanced Transportation Systems*, vol. 34, no. 2, pp. 145–159, 2020. [Online]. Available : <https://ieeexplore.ieee.org/document/9498342>
- [2] S. O. A. Chishti, S. Riaz, M. B. Zaib, and M. Nauman, “Self-driving cars using CNN and Q-learning,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2019, pp. 789–794. [Online]. Available : <https://ieeexplore.ieee.org/document/8595684>
- [3] D. Wilson, L. Martinez, and R. Thompson, “Design and implementation of autonomous car using Raspberry Pi,” *Journal of Embedded Systems and Applications*, vol. 16, no. 3, pp. 223–238, 2021. [Online]. Available : https://www.researchgate.net/publication/276027654_Design_and_Implementation_of_Autonomous_Car_using_Raspberry_Pi
- [4] M. Garcia, T. Anderson, and O. White, “Self-driving car : A deep-learning approach,” *Journal of Artificial Intelligence Research*, vol. 42, no. 1, pp. 101–117, 2018. [Online]. Available : https://www.researchgate.net/publication/357569914_Self-Driving_Car_A_Deep-Learning_Approach
- [5] J. Doe, “Autonomous driving object detection on the Raspberry Pi,” *Instructables*, 2024. [Online]. Available : <https://www.instructables.com/Autonomous-Driving-Object-Detection-on-the-Raspber/>
- [6] D. Benedetti, J. Agnelli, A. Gagliardi, and P. Dini, “Design of a digital dashboard on low-cost embedded platform in a fully electric vehicle,” in *Proceedings of the 2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / ICPS Europe)*, June 2020. [Online]. Available : https://www.researchgate.net/publication/342816735_Design_of_a_Digital_Dashboard_on_Low-Cost_EMBEDDED_Platform_in_a_Fully_Electric_Vehicle
- [7] M. Gupta and R. Kumar, “Real-time road lane detection for self-driving cars using computer vision,” in *Machine Intelligence for Research and Innovations*, January 2024, pp. 157–168. [Online]. Available : https://www.researchgate.net/publication/377370481_Real-Time_Road_Lane_Detection_for_Self-driving_Cars_Using_Computer_Vision
- [8] M. Calcroft and A. Khan, “LiDAR-based obstacle detection and avoidance for autonomous vehicles using Raspberry Pi 3B,” *Journal of Robotics and Autonomous Systems*, vol. 134, pp. 103–112, 2020. [Online]. Available : <https://ieeexplore.ieee.org/document/9781465>
- [9] T. Nguyen, P. Tran, and H. Pham, “Robotic path planning based on episodic memory fusion,” in *International Conference on System Science and Engineering (ICSSE)*, 2021, pp. 546–551. [Online]. Available : <https://ieeexplore.ieee.org/document/9486489>

- [10] N. Ragheb and M. Mahmoud, "Implementing deep reinforcement learning in autonomous control systems," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, vol. 41, no. 1, pp. 168–178, Mar. 2024. [Online]. Available : <https://doi.org/10.37934/araset.41.1.168178>
- [11] "Raspberry Pi Documentation," *Raspberry Pi Foundation*. [Online]. Available : <https://www.raspberrypi.org/documentation/>
- [12] "Self-Driving Car Engineer Nanodegree," *Udacity*. [Online]. Available : <https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013>
- [13] " Make your own self-driving RC car using Python and Raspberry Pi," *Towards Data Science*. [Online]. Available : <https://medium.com/@zaid.killam/make-your-own-self-driving-rc-car-using-python-and-raspberry-pi-part-1-introduct>
- [14] "Autonomous Vehicles Research," *IEEE Xplore*. [Online]. Available : <https://ieeexplore.ieee.org/Xplore/home.jsp>
- [15] "OpenCV for Autonomous Vehicles," *OpenCV*. [Online]. Available : <https://360digitmg.com/blog/opencv-for-autonomous-vehicle>
- [16] "Proximal Policy Optimization algorithm," *Stable-Baselines3*. [Online]. Available : <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>
- [17] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *ArXiv*, abs/1604.07316, 2016.
- [18] "Python Pickle module document," *Python*. [Online]. Available : <https://docs.python.org/3/library/pickle.html>
- [19] "Waymo open dataset," *Waymo*. [Online]. Available : <http://waymo.com/open>
- [20] F. Yu, et al., "BDD100K : A Diverse Driving Video Database with Scalable Annotation Tooling," *ArXiv*, abs/1805.04687. [Online]. Available : <https://arxiv.org/abs/1805.04687>
- [21] "Sunfounder company," *Sunfounder*. [Online]. Available : <https://www.sunfounder.com/>
- [22] "OpenCV color conversions document," *OpenCV*. [Online]. Available : https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
- [23] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.
- [24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, and H. Adam, "Mobilenets : Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [27] "Keras document," *Keras*. [Online]. Available : <https://keras.io/configuring-your-keras-backend>

- [28] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.
- [29] N. L. Hjort, “Pattern recognition and neural networks,” *Cambridge University Press*, 1996.
- [30] “About Train, Validation and Test Sets in Machine Learning,” *Towards Data Science*. [Online]. Available : <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>

Annexe

Livrables

Tous nos codes RaspberryPi, Webots et STM32CubeIDE sont disponibles ici :

Cliquez moi !

Documentations de composants importants

Pour réaliser notre projet de voiture autonome, nous avons utilisé plusieurs composants matériels critiques. Vous trouverez ci-dessous les liens vers la documentation de chaque composant, qui fournissent des informations détaillées sur leurs spécifications et leur utilisation :

- Raspberry Pi 4 model B - 8GB Datasheet
- STM32L432KC Datasheet
- RPLIDAR A2M12 Datasheet
- SRF10 Datasheet

Débuter avec Raspberry Pi

Pour démarrer avec Raspberry Pi, **après avoir formatté la carte SD** :

1. Branchez la carte Raspberry avec :
 - Une alimentation minimale de 5V, 700mA pour assurer une tension minimale sûre aux sorties USB de la carte permettant d'alimenter les périphériques.
 - Périphériques : Écran, Clavier, Souris ...
2. Branchez un écran, un clavier et un réseau sur le RPI. Ensuite, alimentez-le en branchant sa prise de type C.
3. Au cours de ce démarrage (et des suivants), vous verrez de nombreuses lignes de commandes défiler toutes seules. Il s'agit des commandes exécutées par le système afin de bien démarrer.
4. Une fois le système démarré, le login par défaut est « pi », et le mot de passe est « Raspberry ». Vous avez maintenant accès au système, Raspbian est installé !
5. Activez le SSH. SSH est un protocole qui permet d'autoriser l'accès à distance à Raspberry. Créez un nouveau document texte. Renommez-le en "ssh" et enregistrez-le. Maintenant, un fichier nommé "ssh" est créé sans aucune extension de fichier.
6. Pour qu'une liaison réseau fonctionne entre deux ordinateurs, ils doivent utiliser la même plage d'adresses. La plage d'adresses autorisée est déterminée par le masque de sous-réseau. 255.255.0.0 signifie que tous les octets des adresses doivent être identiques à l'exception des deux derniers de l'adresse IP, sinon ils seront filtrés. La plupart des PC connectés directement à un autre ordinateur attribueront l'adresse IP dans la plage 192.168.x.x (avec un masque de sous-réseau de 255.255.0.0). Pour que RPI puisse communiquer par la liaison directe, il doit avoir une adresse IP fixe dans la même plage d'adresses 192.168.x.x.
7. Saisissez cette adresse « ip=192.168.137.2 » dans un fichier appelé « cmdline.txt ». Utilisez un client SSH gratuit pour vous connecter à Pi.

Ouvrir un projet avec STM32CubeIDE

Pour créer un projet sous STM32CubeIDE :

1. Assurez-vous que STM32CubeIDE est installé sur votre machine et que vous possédez la dernière version.
2. Lancez STM32CubeIDE lors de la première utilisation, il peut vous demander de sélectionner un espace de travail (workspace), choisissez un dossier de votre choix pour stocker vos projets.
3. Maintenant on va créer un nouveau projet pour se faire on doit :
 - Cliquez sur File -> New -> STM32 Project.
 - Une fenêtre Target Selection s'ouvre, vous permettant de choisir le microcontrôleur ou la carte de développement que vous utilisez, vous pouvez chercher votre microcontrôleur par la référence ou sélectionner directement votre carte de développement si elle est listée.
 - Sélectionnez votre microcontrôleur ou votre carte, puis cliquez sur next.
4. Configurer le projet
 - Donnez un nom à votre projet dans le champ Project Name.
 - Sélectionnez Target Language (généralement C).
 - Choisissez les Firmware Packages que vous souhaitez utiliser. Par défaut, STM32Cube HAL est sélectionné, mais vous pouvez également inclure d'autres packages si nécessaire.
 - Cliquez sur Finish pour créer le projet. STM32CubeIDE génère automatiquement les fichiers nécessaires et ouvre l'interface de configuration STM32CubeMX.
5. Configurer les périphériques et les broches
 - L'interface STM32CubeMX vous permet de configurer les périphériques et les broches de votre microcontrôleur. Sélectionnez les périphériques que vous souhaitez utiliser en cliquant sur les broches correspondantes ou en utilisant les menus sur la gauche
 - Configurez les paramètres des périphériques (comme les horloges, les modes de communication, etc.) en cliquant sur les périphériques dans l'onglet Configuration.
6. Générer le code d'initialisation
 - Une fois que vous avez configuré les périphériques et les broches, cliquez sur l'icône Generate Code en haut de la fenêtre STM32CubeMX.
 - STM32CubeIDE génère le code d'initialisation basé sur vos configurations et ouvre le projet avec le code source prêt à être modifié.
7. Écrire, modifier, compiler, Déboguer et téléverser le code
 - Naviguez dans l'arborescence du projet dans l'onglet Project Explorer. Vous y trouverez des dossiers comme Core, Drivers, et d'autres qui contiennent les fichiers de votre projet, notamment le fichier **main.c**
 - Cliquez sur l'icône Build (marteau) pour compiler votre projet.
 - Pour déboguer votre projet, cliquez sur l'icône Debug (scarabée) pour passer en mode débogage, vous permettant d'utiliser des points d'arrêt, d'inspecter les variables et de suivre l'exécution du code pas à pas.

-
- Assurez-vous que votre carte de développement STM32 est connectée à votre ordinateur via USB et cliquez sur l'icône Run (flèche verte) pour téléverser le code sur votre microcontrôleur et démarrer l'exécution.

