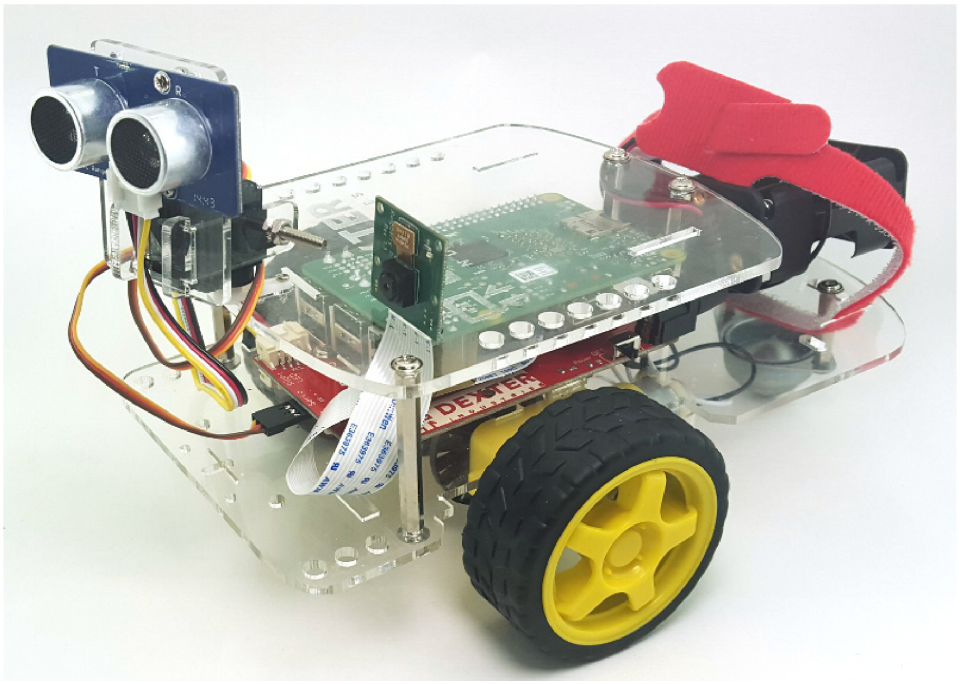
Rapport de projet

LU2IN013
Projet de Développement: Robotique

Sanchez & Co

Le robot



ADJLIA Jed Daryl [21114048]
FONTAINE Corentin [21135945]
LI Elisa [21110736]
MAMLOUK Haya [21107689]
OULFID Hamza [28711045]

Sommaire :

Introduction

En tant qu'entreprise Sanchez&Co, nous avons été missionnés sur ce projet qui a pour but de fournir un outil permettant de contrôler un robot de type GoPiGo. L'objectif est donc que l'utilisateur puisse donner des instructions, celles-ci seront traitées par notre système qui transmettra à la carte arduino du robot des ordres pour faire évoluer ce robot dans son environnement.

L'un des points majeurs de ce projet et qui a permis son développement, est l'évolution permanente. La méthodologie est souvent la même : tester une idée, voir les imperfections, corriger le tir. C'est là toute l'essence de ce projet.

Les défis de ce projet sont multiples. Tout d'abord, il y a l'organisation. En tant qu'équipe de jeunes développeurs, il nous a fallu apprendre à travailler les uns avec les autres, en cohésion, mais que chaque individu ait tout de même une certaine liberté dans sa manière de coder. En effet, cela influence notre productivité, un facteur déterminant de l'avancée de notre code. Car bien que ce projet se soit étalé sur une longue période, celui-ci demandait un investissement personnel constant afin de pouvoir délivrer de nouvelles fonctionnalités aux clients de semaine en semaine.

Lâcher dans l'univers de la robotique, de nombreuses difficultés techniques sont également survenues. Pour réaliser nos objectifs, il est souvent question de trouver la meilleure façon de faire et la technologie, la plateforme, la meilleure astuce associée, puis se former sur ce sujet pour pouvoir ensuite l'implémenter. S'adapter et se remettre en question, ainsi que ses connaissances, est donc crucial pour fournir les meilleures solutions pour le client.

I. Prise en main des outils

Afin de répondre à notre besoin d'organisation, nous avons dû choisir une méthodologie de travail pour assurer le bon déroulement du projet. Sur de nombreux projets, la gestion en V est appliquée, or celle-ci présente de nombreux inconvénients. C'est pour cela que nous avons opté pour les principes de l'Agile, plus axé vers le besoin du client. Ceux-ci évoluent régulièrement avec le temps. Nous voulions donc rester à l'écoute du client, à l'aide de rencontres régulières, chaque semaine, pour s'assurer de travailler dans la direction souhaitée. Cette méthode est également plus dynamique, puisqu'on a pu se concentrer sur l'implémentation permanente de fonctionnalités importantes. L'idée était le dynamisme et l'écoute. Pour cela Trello et le framework Scrum agile étaient des outils nécessaires.

Au-delà de l'organisation, il fallait maintenant nous intéresser au code en lui-même. Un choix de langage était nécessaire et ce fut Python qui a été choisi, pour sa simplicité et ses fonctionnalités. De plus, il était impératif de pouvoir coder en équipe de manière efficace et ne pas avoir une tonne de versions ce qui rendrait le développement impossible. Ici, l'utilisation de Github nous a été vitale.

A. Trello et méthode Scrum

Trello est un outil de gestion de projet en ligne qui utilise un système de tableaux pour organiser les tâches. Il nous a permis d'avancer en suivant la méthode framework Scrum agile.

Le framework Scrum Agile est une méthodologie de gestion de projet qui met en œuvre la vision Agile. Il se compose du "Product Backlog", des réunions de planification, des sprints et de la livraison de nouvelles versions du produit. Nous avons utilisé Trello pour représenter ces étapes et avons créé un tableau appelé "Planning".

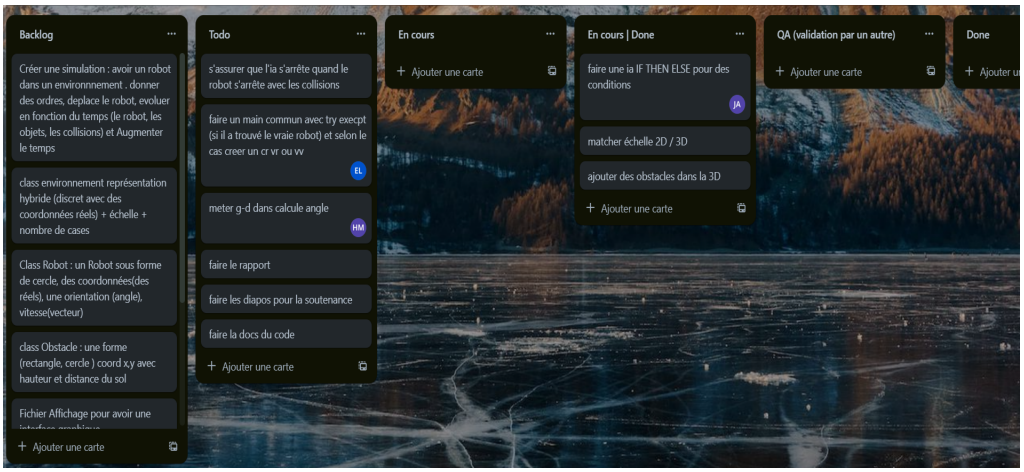


figure 1 : Notre Planning du Trello

À l'intérieur du tableau, nous avons structuré les différentes étapes du framework Scrum Agile avec des listes telles que "Backlog", "Todo", "En cours", "En cours|Done", "QA (validation par un autre)" et "Done". Chaque liste avait une utilité spécifique pour suivre notre progression hebdomadaire. Nous avons créé des cartes pour chaque tâche à réaliser, en commençant par le "Backlog" pour définir les exigences du produit et garder nos objectifs en vue. Les tâches hebdomadaires étaient ensuite placées dans la liste "Todo" pour répondre aux objectifs du "Backlog". Les tâches étaient attribuées à chaque membre de l'équipe et au fur et à mesure de notre progression dans la semaine, nous déplacions les cartes d'une liste à une autre pour refléter leur état d'avancement. Les cartes étaient déplacées vers la liste "QA" une fois terminées, puis vers "Done" après validation.

Cette approche nous permettait de suivre visuellement l'avancement du projet en consultant les tâches du "Backlog" déjà terminées et celles restant à accomplir. Trello facilite également la collaboration entre les membres du groupe, avec des commentaires partagés, des discussions sur les tâches et un suivi en temps réel des progrès. Nous avons ajouté une estimation du temps nécessaire pour chaque tâche, ainsi que le temps réellement passé, afin de mieux gérer les délais et de répartir équitablement les tâches.

En résumé, l'utilisation de Trello et de la méthode Scrum Agile a amélioré la gestion du temps, la répartition des responsabilités, la communication entre les membres du groupe et l'organisation globale du projet.

B. Github

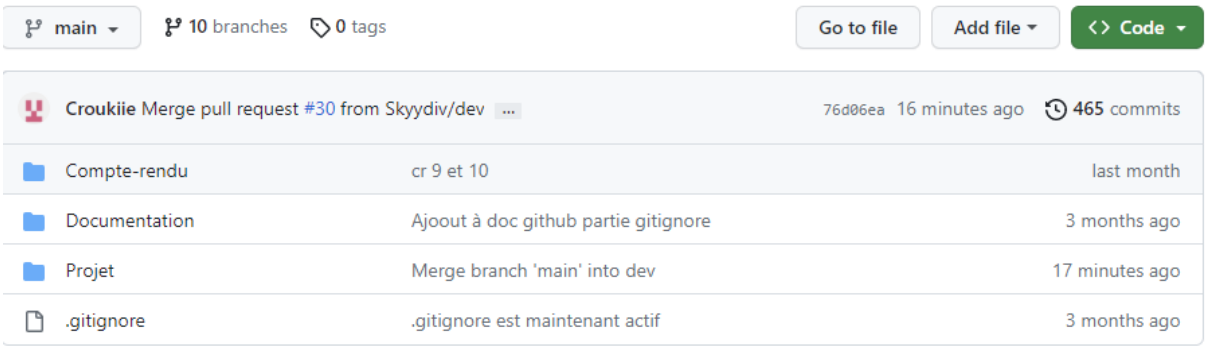


figure 2 : Notre repository Github

Github propose une interface pour le système de gestion de version Git. C'est grâce à cette dernière que l'on a pu monter le projet. Github nous a permis de toujours tous travailler sur la même version du projet, étant la plus récente. On était alors capable de travailler à plusieurs, simultanément sur des fonctionnalités différentes et le tout à distance.

Nous avons commencé par la mise en place du repository : https://github.com/Skyydiv/Sanchez-co.git. Il a ensuite fallu se familiariser avec l'outil. Assez rapidement un problème de taille apparaît : les conflits. Le mot d'ordre pour les régler, faire des tâches simples et courtes ! Ainsi, notre routine était "pull/ code / add / commit / push".

Nous avons choisi d'utiliser plusieurs branches pour avoir plusieurs états de développement du projet à tout moment. Il y en a deux principales. La première la "main", celle-ci contient toujours une version présentable pour une démonstration. La seconde "dev" est la branche qui contient le code en développement pour la semaine en cours. La branche "dev" est merge, c'est-à-dire son contenu est déplacé, à la branche "main" en fin de semaine avant réunion avec le client.

Une attention particulière a été mise afin d'avoir des commentaires clairs et cohérents lors des commits. En effet, cela permet de garder une trace sur chaque changement et de pouvoir revenir à un état précis du projet en cas d'apparition de bug. On voit l'évolution du projet également au niveau du code de cette manière.

C. Langage Python

Python, en tant que langage orienté objet, s'est révélé être un choix approprié pour notre projet. Ses fonctionnalités orientées objet ont grandement facilité notre travail. Grâce à Python, nous avons pu organiser notre code en classe, ce qui nous a permis de naviguer plus facilement entre les différentes parties de notre projet.
De plus, l'API étant en Python, son importation et sa compréhension ont été plus simples pour nous qui avons déjà initié notre projet en ce langage. En utilisant ce langage, nous avons pu développer notre projet de manière efficace et productive, en exploitant pleinement les avantages offerts par la programmation orientée objet.

I. Simulation virtuelle

Le projet, comme mentionné précédemment, consiste à développer un code permettant de donner des instructions au robot. Cependant, il peut être délicat de développer un code à partir de zéro et de l'exécuter directement sur le robot. Cela est dû à plusieurs facteurs. Tout d'abord, nous avions besoin d'être plusieurs à effectuer des tests rapidement, en simultané ainsi qu'en distanciel, ce qui serait difficile s'il fallait être présent et attendre le temps de remettre en place le robot entre chaque test. De plus, il est crucial de prendre en compte le coût élevé du robot et de prendre des précautions pour éviter de l'endommager.

Pour remédier à ces problèmes, il était essentiel de développer une simulation dans laquelle nous pouvions tester le comportement du robot dans un environnement spécifique avec différents paramètres et conditions.

A. Premiers pas

En utilisant Python, un langage orienté objet, nous avons créé deux objets principaux : le premier est un objet "robot" représentant le robot réel, et le deuxième est un objet "environnement" représentant l'environnement dans lequel le robot évolue. Nous avons commencé par modéliser un environnement discret, composé de lignes et de colonnes représentant des cases. Le robot était alors représenté par des coordonnées sous forme d'entiers, correspondant aux indices des cases, et des fonctions de déplacement très basiques lui permettant de se déplacer uniquement en ligne droite, en arrière, vers le haut ou vers le bas en incrémentant ou en diminuant sa position.

Cependant, nous avons réalisé que cette représentation était loin de la réalité et qu'il était difficile de la reproduire fidèlement avec ce modèle. C'est pourquoi nous avons décidé de revoir notre approche en attribuant au robot des caractéristiques plus réalistes, semblables à celles du robot réel, et en transformant l'environnement en un environnement continu.

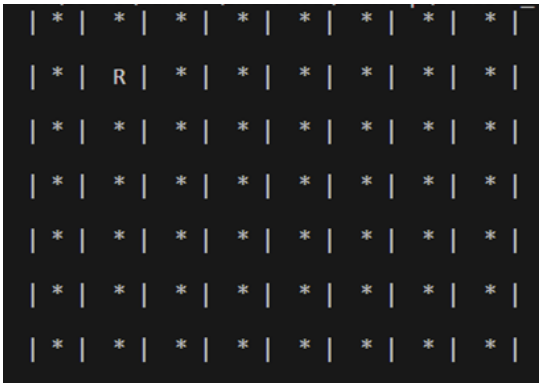


Figure 3 : Notre Environnement discret

B. Robot virtuel

Pour rendre la représentation du robot aussi fidèle que possible à la réalité, il était important de lui attribuer les mêmes caractéristiques physiques. Ainsi, dans notre objet "robot", nous avons inclus les paramètres de vitesse des roues gauches et droites, ainsi que le rayon. Bien que le robot réel ait généralement une forme carrée, nous avons choisi de le modéliser sous forme circulaire afin de simplifier les calculs de déplacement et de collisions. Le rayon de cette forme circulaire correspond à la moitié de la distance entre les deux roues du robot réel.

De plus, dans la réalité, le robot peut se déplacer dans n'importe quelle direction et orientation. Pour cela, nous avons attribué au robot des coordonnées réelles x et y, ainsi qu'une orientation exprimée en radians.

Afin d'obtenir un déplacement réaliste du robot, des calculs appropriés sont nécessaires. Ces calculs prennent en compte uniquement les données disponibles pour le robot. Les déplacements s'effectuent en calculant la distance parcourue pendant un intervalle de temps et en mettant à jour les coordonnées en conséquence. Nous avons donc implémenté une fonction qui calcule la distance parcourue par une roue durant un laps de temps donné. De plus, nous disposons de constantes représentant des mesures telles que la distance entre les roues gauches et droites, le diamètre de la roue, le périmètre du cercle de rotation et le périmètre de la roue, tous exprimés en millimètres. Ces constantes sont fournies par l'API du robot, ce qui nous permet de garantir que nos calculs se rapprochent de la réalité.

En attribuant à l'objet “robot” ces caractéristiques physiques et en mettant en place les calculs nécessaires, nous visons à obtenir une représentation fidèle des mouvements et des comportements du robot dans la simulation.

C. Environnement

Nous avons choisi de représenter l'environnement à l'aide d'un vecteur de coordonnées maximales. Cette approche présente plusieurs avantages. Tout d'abord, elle simplifie la représentation spatiale en définissant la longueur et la largeur maximales de l'environnement, créant ainsi un espace borné pour le robot et les obstacles. Cette simplification facilite la visualisation de la disposition de l'environnement.

De plus, l'utilisation de coordonnées maximales permet une détection claire des collisions. Si le robot dépasse ces coordonnées, cela est considéré comme une collision avec un mur, ce qui simplifie la gestion des collisions.

Pour rendre l'environnement plus réaliste et créer un défi pour le robot, il était évident d'inclure des obstacles. L'environnement est donc composé d'un ensemble d'obstacles. Chaque obstacle est représenté par des coordonnées réelles x et y, un rayon pour sa forme circulaire, une hauteur et une distance par rapport au sol. Ces caractéristiques permettent de facilement détecter des collisions et de concevoir une simulation réaliste avec différents types d'obstacles et de créer divers environnements .

En matière d'efficacité computationnelle, cette représentation simplifie les calculs liés au mouvement du robot et aux interactions avec les obstacles, car il n'est pas nécessaire d'effectuer des vérifications complexes des limites.

D. Simulation

La simulation intervient lorsque le robot et l'environnement interagissent. Pendant une simulation, le robot est placé dans un environnement contenant des obstacles et évolue sur une période de temps. Nous mettons à jour les différents éléments de la simulation à chaque pas de temps. Tant que tout se déroule sans problème, c'est-à-dire tant que le robot n'entre pas

en collision, la simulation continue de s'exécuter. À chaque pas de temps, le robot se déplace dans son environnement en mettant à jour ses coordonnées à intervalles réguliers, cet intervalle s'agit du temps écoulé depuis le dernier déplacement du robot.

La flexibilité de pouvoir choisir les dimensions de l'environnement ainsi que le nombre d'obstacles nous permet d'effectuer divers tests dans différentes situations et d'observer l'évolution et le comportement du robot. Cela constitue l'intérêt principal d'avoir une simulation : pouvoir explorer différentes configurations et scénarios pour mieux comprendre les capacités et les performances du robot.

C'est pourquoi, en développant une simulation avec un objet robot plus réaliste et un environnement continu, nous avons cherché à créer une représentation plus précise du comportement du robot et de son environnement.

II. Affichage de simulations

L'affichage joue un rôle crucial dans le processus de test de nos simulations pour plusieurs raisons. Tout d'abord, il permet une visualisation claire et intuitive de l'environnement dans lequel le robot évolue. Grâce à un affichage, nous pouvons représenter graphiquement les obstacles, les déplacements du robot et d'autres éléments pertinents, ce qui facilite l'analyse et la compréhension des résultats.

De plus, l'affichage permet également de repérer visuellement d'éventuelles erreurs ou anomalies dans la simulation. En observant les déplacements du robot et les interactions avec les obstacles à l'écran, nous pouvons rapidement repérer des comportements indésirables ou des problèmes de collision, ce qui facilite le processus de débogage et d'amélioration du code.

A. Affichage 2D

L'affichage 2D offre une représentation simplifiée de la réalité, ce qui permet de se concentrer sur les aspects essentiels de la simulation. Nous pouvons nous concentrer sur la logique du comportement du robot et des interactions avec l'environnement.

Nous avons utilisé Tkinter, une bibliothèque graphique intégrée à Python, pour implémenter l'affichage de notre simulation. Ce choix s'est basé sur la facilité d'accès et la simplicité de Tkinter pour créer des interfaces graphiques.

En ce qui concerne la visualisation, nous avons opté pour une représentation similaire au jeu Pac-Man. Le robot est représenté par une forme semblable à celle de Pac-Man, avec un

fond noir et un cadre fin et blanc. Les obstacles, quant à eux, sont représentés par des cercles blancs.

Lorsque nous lançons l'affichage, nous créons une zone dédiée à l'affichage qui représente l'environnement de la simulation, en définissant sa taille et sa forme.

Ensuite, à chaque pas de temps, nous mettons à jour les différents éléments de l'affichage. Nous ajoutons la position actuelle du robot ainsi que sa trajectoire, car il se déplace au cours de la simulation. De même, nous tenons compte des obstacles pour détecter d'éventuelles collisions.

Il est important de souligner que l'affichage est une représentation visuelle de la simulation. La simulation et l'affichage sont exécutés séparément, mais ils partagent des dimensions représentatives de l'environnement, une trajectoire de robot identique et un nombre et une position d'obstacles identiques. Il n'y a pas de communication directe entre les deux parties, il s'agit simplement de récupérer les données de la simulation pour les afficher à l'écran.

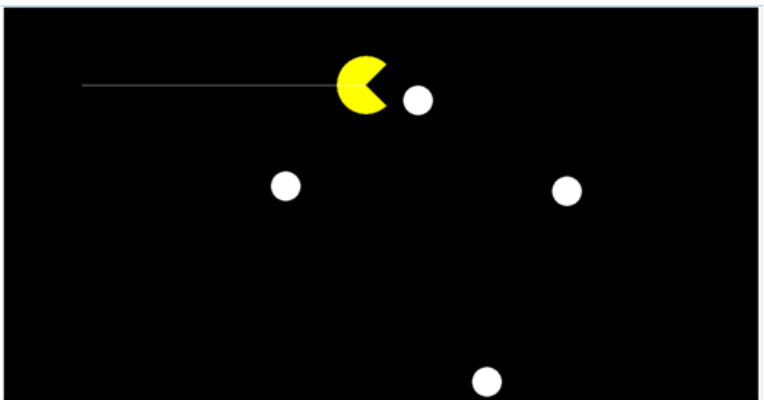


Figure 4 : Affichage 2D d’une simulation

B. Affichage 3D

La 2D nous a permis de tester des fonctionnalités du robot. Cependant, cette représentation reste loin de la réalité. Pour aller plus loin et se rapprocher d’un rendu réaliste, il nous faut un affichage 3D. Pour cela, nous avons utilisé Panda3D, une bibliothèque python qu’il fallait installer. Nous avons fait ce choix pour la simplicité de prise en main ainsi que les fonctionnalités de base qui correspondaient à nos besoins pour ce projet.

Pour la visualisation, nous avons voulu passer à un rendu plus réaliste tout en restant un peu dans la fantaisie. Notre robot est ainsi représenté par un panda et les obstacles sont des rochers. L’utilisateur a le choix entre 2 visions pour observer le panda évoluer dans son environnement. La première, une vue “fps”, c'est-à-dire à la première personne, on voit au

travers les yeux du panda. La seconde est la vue de “haut”, où l’on se place au-dessus de la scène pour avoir une vue d’ensemble. Cette vue est choisie au lancement de la simulation.

À la création de l’affichage 3D, une scène est créée à laquelle est attaché un acteur, le panda. C’est également à ce moment que les obstacles créés par la simulation sont représentés à leurs coordonnées respectives. Le panda et les obstacles sont des modèles 3D qui sont chargés, redimensionnés et placés à leur création. Le panda a par ailleurs une animation de marche qui lui est attaché.

À tous les intervalles de temps de l’affichage, on récupère la position du robot dans la simulation et on change les coordonnées du panda en conséquence. C’est ensuite au rafraîchissement de la caméra, c'est-à-dire à chaque frame, que l’on va pouvoir voir les différences appliquées. Avec un rafraîchissement et un intervalle d’affichage suffisamment petit, on a une impression réelle de mouvement.

III. Passage au robot avec contrôleur et proxy

Le proxy et le contrôleur sont deux éléments essentiels de notre système de contrôle des robots. Ils fonctionnent de manière complémentaire pour nous permettre de contrôler efficacement nos robots. Le proxy facilite la communication entre le contrôleur et les robots, tandis que le contrôleur gère les actions et les mouvements spécifiques des robots. Ensemble, ces deux éléments offrent un contrôle complet et précis des robots.

A. Proxy

Le proxy agit comme un intermédiaire entre notre système de contrôle et les robots, en facilitant la communication et la transmission des instructions. Il permet d'établir une connexion entre le contrôleur et les robots, ce qui nous donne la possibilité de contrôler à distance les actions et les mouvements des robots. Il joue donc un rôle essentiel dans le contrôle de notre robot virtuel et de notre robot réel.

Pour le robot virtuel, nous avons développé des fonctions dans le fichier virtuel.py, en utilisant les fonctionnalités spécifiques de ce robot. Grâce à ces fonctions, nous sommes en mesure de faire avancer et tourner le robot virtuel.

Quant au robot réel, nous avons dû étudier attentivement l'API associée afin de mieux le comprendre. Nous utilisons les principales fonctions fournies par l'API pour contrôler les mouvements du robot réel, lui permettant d'avancer et de tourner.

Dans notre fichier proxy.py, nous avons créé la classe mère appelée Controleur. Cette classe regroupe des méthodes permettant de configurer les vitesses, de faire avancer le robot en ligne droite, ainsi que de le faire tourner à gauche ou à droite. De plus, nous avons ajouté

une fonction principale "update" qui met à jour la distance parcourue et l'angle de rotation du robot, ce qui nous permet de suivre sa progression et de déterminer quand il doit s'arrêter.

Ensuite, nous avons deux classes dérivées de la classe mère, qui contiennent des fonctions spécifiques de configuration des vitesses adaptées à chaque type de robot. Chacune de ces classes implémente également des fonctions de calcul de la distance et de l'angle en fonction du robot utilisé. Pour le robot virtuel, nous utilisons les données fournies par le fichier virtuel.py, que nous avons programmé tout au long du projet. En revanche, pour le robot réel, nous exploitons les données fournies par l'API spécifique au robot. Finalement, nous utilisons le proxy et le contrôleur, d'une façon complémentaire, pour établir une communication avec les robots.

B. Contrôleur

Le contrôleur, comme son nom l'indique, permet le contrôle du robot. C'est un processus qui fonctionne en parallèle des autres processus, car il doit être capable de donner des ordres au robot à tout moment et ce processus s'arrête uniquement quand l'ordre donné est terminé.

Le contrôleur a différentes phases : une phase de vérification des conditions d'arrêt (mise à jour) et une phase de pause pendant un court instant pour laisser au robot le temps d'avancer dans la tâche qui lui a été donnée et permet au système de faire les calculs nécessaires. De plus, le fait que le processus tourne en arrière-plan n'est pas très utile, donc le mettre en veille pendant quelques secondes permet d'améliorer la fluidité.

Les ordres donnés ne sont pas donnés au hasard, ils dépendent de ce que l'on souhaite faire faire au robot. C'est pourquoi il existe différentes fonctions dans le contrôleur qui permettent d'indiquer au robot quelle action effectuer. Par exemple, si l'on souhaite que le robot avance tout droit sur une certaine distance à une certaine vitesse, on utilise la fonction "avancer_tout_droit" à laquelle on passe la distance et la vitesse souhaitées en paramètre. Ensuite, à l'aide des valeurs que le contrôleur demande au proxy, telles que la distance parcourue ou l'angle parcouru, le contrôleur peut donner l'ordre d'arrêter ou de poursuivre la tâche qui lui a été donnée.

Le but est d'effectuer les tâches de manière successive, donc le contrôleur dispose d'une fonction séquentielle à laquelle on donne les fonctions dans l'ordre dans lequel on souhaite qu'elles s'exécutent. Cette fonction séquentielle est également considérée comme une action et donc la fin de cette dernière signifie que toutes les autres ordres ont bien été données et effectués.

IV. Robot virtuel vs. Robot réel

Après avoir implémenté le proxy et le contrôleur, nous devons maintenant les utiliser pour communiquer avec nos robots. Pour cela, nous utilisons deux approches distinctes. La première consiste à contrôler notre robot virtuel à l'aide de stratégies qui facilitent les démonstrations avec un affichage visuel. En effet, nous avons mis en place une fonctionnalité permettant aux clients de générer facilement une simulation en définissant leurs propres paramètres. De plus, nous avons intégré une fonctionnalité de création d'affichage 2D ou 3D qui permet de suivre visuellement la simulation. Dans le but de rendre l'expérience cliente plus fluide, nous avons également développé des stratégies simplifiées utilisant des commandes du contrôleur pour contrôler le robot, telles que les stratégies "faire un carré" ou "avancer tout droit". La deuxième approche concerne notre robot réel, où nous utilisons les fonctions du contrôleur pour tester différents schémas sans nous limiter à des formes préconstruites.

Une grande partie de notre travail a été consacrée à l'implémentation de notre simulation pendant plusieurs semaines. Nous avons perfectionné les méthodes pour assurer le bon fonctionnement de notre robot virtuel et nous assurer qu'il exécute les tâches selon nos attentes. Ainsi, grâce à nos efforts jusqu'à présent, notre robot virtuel est capable d'effectuer des tâches simples telles que se déplacer en ligne droite, tourner et réaliser des formes spécifiques en utilisant des stratégies, ou encore faire plusieurs stratégies de suite sans s’arrêter.

Cependant, lors du passage au robot réel, nous nous attendions à obtenir des résultats similaires à ceux du robot virtuel. Nous avons constaté que les fonctions de l'API du robot diffèrent de celles que nous avions implémentées pour le robot virtuel. Nous avons donc dû revenir sur le proxy et le contrôleur afin d'adapter les calculs et de revoir les méthodes utilisées pour le robot réel. Lors de nos premiers essais, le robot réel exécutait des tâches simples telles que se déplacer et tourner, mais nous avons souvent rencontré des problèmes où le robot n'avançait pas à la distance ou à l'angle souhaité. Nous avons alors ajusté le pas de temps de notre contrôleur afin d'améliorer la précision des calculs entre chaque appel de fonctions et de réduire les écarts entre la simulation et le robot réel. Grâce à ces ajustements, nous avons réussi à reproduire un carré similaire à celui de la simulation avec notre robot réel.

Conclusion

Pour conclure, ce projet a été un processus de progression et d’évolution, tant sur le plan du développement de code que de l'atteinte de nos objectifs. En tant que groupe, nous avons connu une évolution remarquable au cours de cette expérience. Nous avons réalisé qu'il

était essentiel de se réunir plus fréquemment que prévu, ce qui nous a permis d'améliorer notre communication et de prendre des décisions plus rapidement. Ce projet nous a offert une précieuse opportunité d'acquérir une vision et un recul sur un travail qui reflète l'environnement d'une entreprise. Nous avons ainsi pu identifier les difficultés auxquelles les équipes sont confrontées lorsqu'elles entreprennent des projets d'envergure, ainsi que les compétences indispensables pour mener à bien ses projets. Ce fut une première étape importante vers notre préparation à la vie professionnelle.

En travaillant sur ce projet, nous avons également pu développer notre capacité à collaborer efficacement en équipe. Nous avons appris à valoriser les compétences et les perspectives de chacun, et à les intégrer de manière cohérente dans notre travail collectif. Cette évolution en tant que groupe a grandement contribué à la réussite globale de notre projet. En cours, ce projet nous a permis de réaliser un travail concret, de relever des défis techniques et de développer notre esprit d'équipe. Nous sommes fiers des résultats obtenus et confiants dans notre capacité à appliquer les compétences acquises dans nos futurs projets.