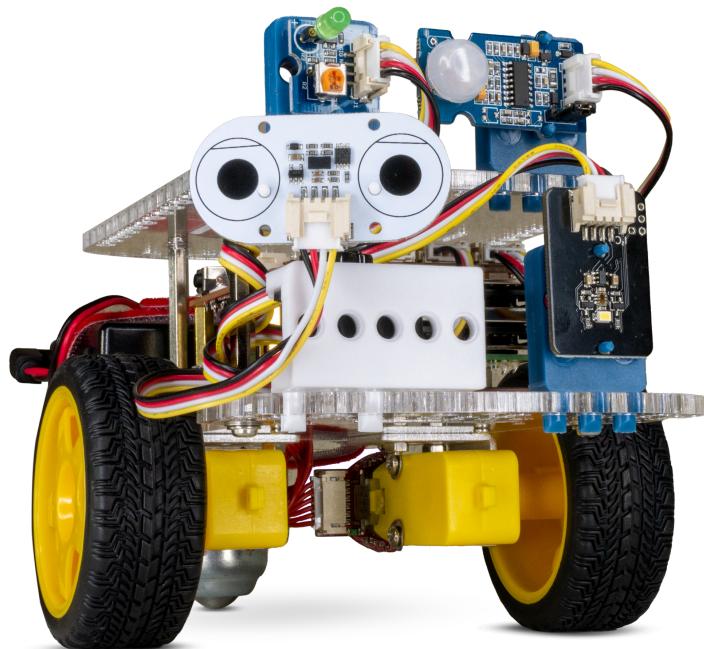


Rapport final du projet robotique



Étudiants :

LI Laurrene
BAGADAD SAIB Rasheequa
AMADO Hugo
KESKIN Derya
BOUDALIL Somaya
FUTURAMA TEAM 7

Encadrants :

BASKIOTIS Nicolas
SIGAUD Olivier

L2 - Mono info 3

TABLE DES MATIÈRES

1. INTRODUCTION.....	3
1.1 Présentation du projet.....	3
1.2 Méthodologie de développement.....	3
2. CONCEPTION ET IMPLÉMENTATION.....	4
2.1 Choix de représentation.....	4
2.2 Choix des logiciels et langage.....	5
2.3 Structure et description du code.....	5
3. FONCTIONNALITÉS DE LA SIMULATION.....	7
4. PASSAGE AU RÉEL.....	9
5. PROBLÈMES ET PROGRESSION.....	10
6. RÉSULTATS ET ANALYSES.....	11
6.1 Comparaison avec les attentes.....	11
7. CONCLUSION.....	15

1. INTRODUCTION

1.1 Présentation du projet

La robotique est un domaine interdisciplinaire qui englobe la conception, la construction, le fonctionnement et l'utilisation des robots. C'est un domaine en constante évolution qui attire beaucoup d'intérêt dans de nombreux domaines. Ces machines autonomes ou semi-autonomes sont capables d'effectuer des tâches programmées par nous, humains. C'est ce que nous avons essayé d'entreprendre dans le cadre de ce projet : le développement et l'implémentation d'une simulation d'un robot, capable d'effectuer des tâches simples, dans un environnement en 2D et 3D. Pour ensuite transposer cela sur un vrai robot physique.

De manière plus générale, parmi les objectifs de cette UE de projet, il y a apprendre à mener entièrement la réalisation d'un projet, à appréhender un nouveau langage de programmation et à travailler en groupe.

1.2 Méthodologie de développement

Afin de mener à bien notre projet, il a fallu dans un premier temps, mettre en place une méthodologie de travail. Pour ce faire, on s'est servi de la méthode dite **AGILE** apprise en classe. Cette méthode s'organise en cycle à intervalles réguliers. Dans notre cas, chaque semaine, l'équipe se concerte, discute de ce qui a été fait par chacun, des problèmes rencontrés, répartis et fixe les nouveaux objectifs à atteindre pour la semaine suivante. Pour nous aider dans cet objectif, on s'est aidé de deux sites de gestion de projet et de développement logiciel appelé "trello" et "github".

Le site trello permet une visualisation des tâches à faire pour pallier au plus urgent et d'avoir un aperçu global de l'état du projet et de l'avancement du groupe. Et github, permet de stocker du code, de travailler à plusieurs sur un projet de programmation et partager facilement les modifications.

La communication était essentielle pour ce projet. Elle permet de revoir rapidement notre stratégie en cas de mauvaise compréhension ou mauvaise direction prise. De ce fait nous avons communiqué via un groupe whatsapp et avons organisé des rendez-vous virtuels ou présentiel en milieu de semaine. De plus, un compte rendu était fait chaque semaine afin d'avoir une trace écrite de notre avancement.

2. CONCEPTION ET IMPLÉMENTATION

2.1 Choix de représentation

Pour l'environnement, nous avons choisi une représentation continue plutôt qu'une représentation matricielle et opté pour des coordonnées cartésiennes. Ces choix offrent plus d'avantages, notamment une plus grande précision pour le déplacement du robot et pour les collisions entre ce dernier et les obstacles. Le robot ainsi que les obstacles ont été représentés sous forme de carré. La tête du robot est colorée en rouge pour indiquer sa direction. Pour les bordures de l'environnement, ce sont des obstacles sous forme de rectangle très fins. Tous nos choix ont pour but de rester fidèle à la réalité.

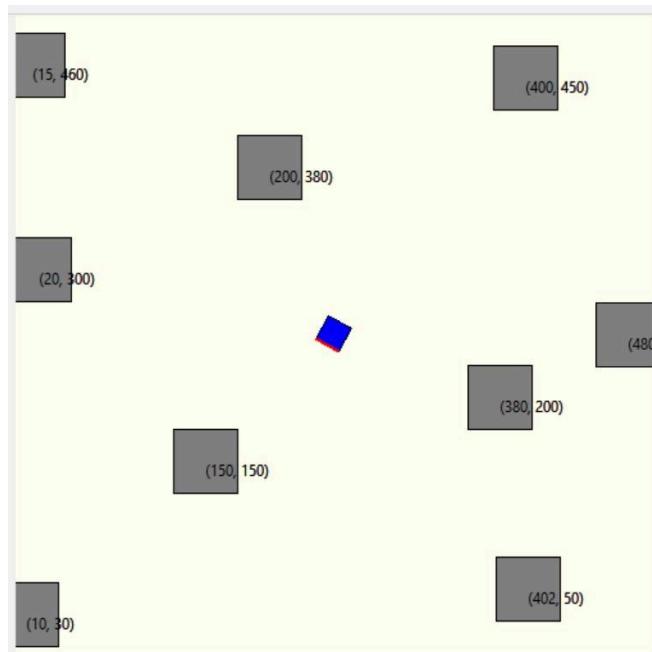


Fig 2.1 - Affichage 2D du robot dans le monde

2.2 Choix des logiciels et langage

Comme langage de programmation, nous avons utilisé Python car il possède une syntaxe claire et lisible et une vaste bibliothèque standard. Pour la simulation en 2D, nous avons opté pour Tkinter qui est livré avec la distribution standard de Python, ce qui signifie qu'il n'y a pas besoin d'installer des paquets supplémentaires pour créer des interfaces graphiques. De plus sa syntaxe est simple et intuitive, ce qui permet de créer des interfaces graphiques de manière simple et efficace.

Pour la représentation 3D, nous avons d'abord hésité entre Pygame, Ursina et Panda3D. Finalement, nous avons opté pour Panda3D, un framework de rendu 3D et de développement de jeux conçu pour être utilisé avec Python. Panda3D a grandement simplifié notre travail grâce à ses fonctionnalités avancées et sa compatibilité avec Python. Nous l'avons choisi en particulier pour la présence d'une playlist YouTube complète et concise sur son utilisation, ce qui a facilité notre apprentissage et la mise en œuvre de la simulation.

2.3 Structure et description du code

Le projet est constitué de plusieurs répertoires. Le code source se trouve dans le répertoire “src” qui contient 4 sous répertoires :

```
/contrôleur
    adaptateur.py
    robotReel.py
    strategie.py
/graphique
    graphique.py
    graphique3D.py
/model
    (modele_3D.egg)
/blender
    (modele_3D.blender)
/tex
    (texture_pour_la_3D.png)
/univers
    monde.py
    obstacle.py
    robot.py
/camera
    camera.py
```

Dans le répertoire “**univers**” nous retrouvons les modules qui composent l’environnement. Le fichier robot.py dans lequel on retrouve toutes les fonctions nécessaires pour faire avancer notre robot, le capteur de distance, des setters/getters et une fonction pour la création d’un robot. Le fichier obstacle.py qui possède également des getters. Le fichier monde.py qui importe Obstacle pour reconstituer le monde à l’aide des fonctions d’attribution de robot et d’obstacle, ainsi que les fonctions qui permettent de détecter les éventuelles collisions entre les deux éléments.

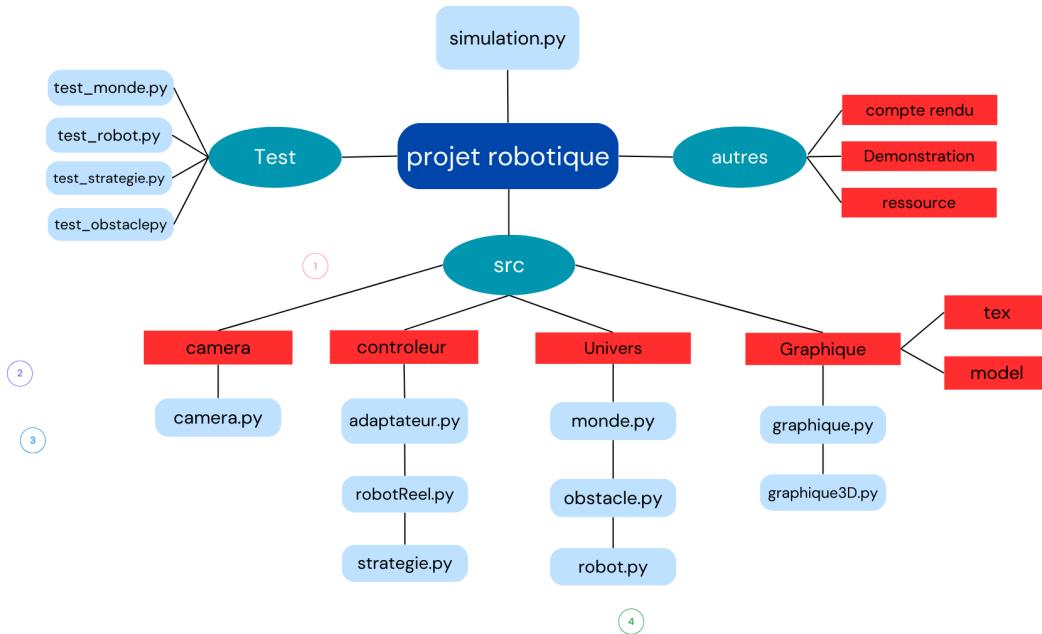
Le répertoire “**graphique**” comporte les classes graphiques. Ces classes se chargent de représenter les éléments de la simulation tels que les obstacles, le robot, les mur du monde, etc...

Le répertoire “**contrôleur**” permet de faire effectuer au robot réel et simulé des tâches simples avec les stratégies de déplacement contenu dans le fichier strategie.py et le fichier adaptateur.py a pour but de traduire ces mêmes instructions au robot réel. Le fichier robotReel.py (le mockup) fourni par les professeurs comporte l’API du robot.

Le répertoire “**camera**” contient une fichier camera.py qui permet de reconnaître la balise dans un environnement via la caméra du robot.

Le fichier **simulation.py** présent à la racine est considéré comme étant le main de notre programme, il permet l’exécution de la simulation en 2D, 3D et sans interface graphique également, ainsi que l’exécution sur le robot réel.

Enfin on retrouve le répertoire “**test**” contient tous les unités tests réalisés afin de tester certaines fonctions, et également un répertoire “**autres**” dans lequel nous avons placé nos comptes rendus hebdomadaires, les ressources utilisées pour ce projet et la démonstration du déplacement physique du robot.


Fig 2.2 - Arborescence du projet

3. FONCTIONNALITÉS DE LA SIMULATION

Dans notre simulation, le robot possède deux roues motrices, une gauche et une droite, et un capteur de distance. On fait déplacer le robot en donnant à chacune des roues une vitesse et c'est en fonction de cette dernière qu'on détermine la trajectoire du robot. Il est donc possible de choisir leur vitesse angulaire et de connaître de combien de degrés elles ont tourné. Le capteur de distance permet de savoir à quelle distance se trouve l'obstacle le plus proche par rapport au robot et d'arrêter le robot s'il y a un risque de collision avec un obstacle.

Par la suite on a dû implémenter une **IA** dont le but est de donner des ordres au robot via ce qu'on appelle des stratégies. Ce sont des fonctions qui une fois exécutées font faire au robot un ensemble de mouvements. Chaque stratégie correspond à une classe qui elle-même est composée d'un constructeur et d'une méthode `start()`, `step()` et `stop()`. Il nous a été demandé de faire des stratégies d'actions élémentaires. Nous avons pu faire les suivantes :

- **AvancerToutDroit** est une stratégie qui permet de faire parcourir au robot une certaine distance qu'on transmet en argument. Dans sa fonction step(), elle fait appelle à la fonction qui permet de modifier les vitesses des roues. La stratégie arrête le robot une fois que la distance passée en argument a été parcourue ou avant qu'il rentre en collision avec un obstacle avec le capteur de distance.
- **Avancer** est similaire à **AvancerToutDroit** mais sans distance spécifique et s'arrête seulement si le robot est trop proche d'un obstacle.
- **Tourner** est une stratégie qui permet de faire tourner le robot d'un certain angle. Le robot arrête sa rotation une fois que l'angle parcouru dépasse l'angle à tourner.
- **ListeStrat** est une stratégie séquentielle qui à partir d'une liste de stratégies les exécute une par une. On la retrouve dans la stratégie **TracerCarre**
- **TracerCarre** est une stratégie qui permet au robot de faire un déplacement en carré avec une certaine distance pour chaque côté, cette même stratégie utilise les précédentes à savoir **AvancerToutDroit** et **Tourner**.

De plus, parmi les tâches simples avec le robot qu'on a pu réaliser, il y a s'approcher le plus vite possible et le plus près d'un mur sans le toucher.

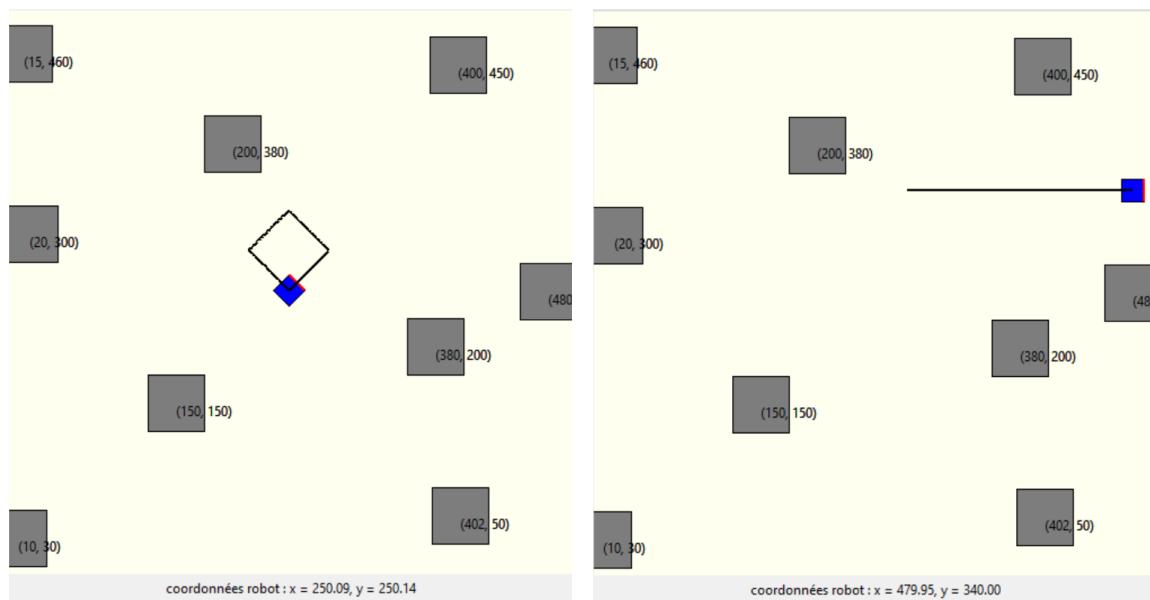


Fig 3.1 - affichage 2D de la simulation avec les différentes stratégies

En ce qui concerne l'affichage 3D, comme dit précédemment nous nous sommes aidé de la bibliothèque graphique Panda3D. Nous avons d'abord créé des modèles 3D à l'aide du logiciel Blender puis exporté sous format .egg pour pouvoir les importer dans le code. Le fichier graphique3D.py contient les fonctions d'affichages graphiques et de déplacement du robot. De plus, nous avons implémenté les angles de vues dans la classe **myRobot** pour pouvoir avoir une vue d'ensemble et pouvoir observer les déplacements du robot.

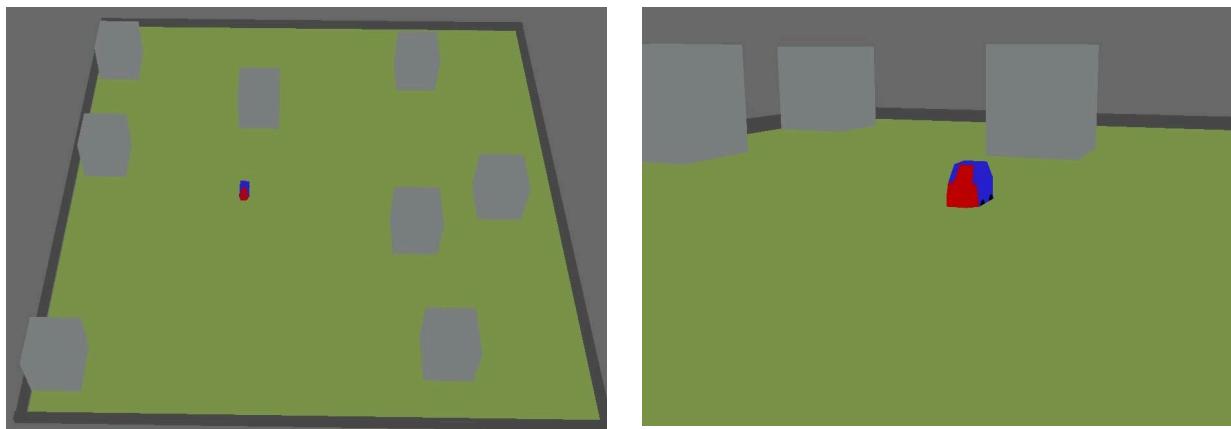


Fig 3.2 - affichage 3D de la simulation avec Panda3D

4. PASSAGE AU RÉEL

Le robot réel est équipé d'un Raspberry Pi, d'une carte contrôleur, de deux moteurs encodeurs pour contrôler les roues, et de trois capteurs : une caméra, un capteur de distance et un accéléromètre. Pour adapter notre code du robot simulé au robot réel, nous avons mis en place diverses fonctions et classes. Notamment, une classe **adaptateur** qui traduit le code du robot simulé pour qu'il soit compatible avec le robot réel. La connexion au robot réel s'effectue via **SSH**.

Les essais des différentes stratégies sur le robot réel ont demandé des ajustements supplémentaires pour tenir compte des écarts de valeurs entre la simulation virtuelle et la réalité. Les valeurs mis dans la simulation virtuelle ne représentaient aucune unité réelle et l'on se retrouvait avec de grands écarts de valeurs entre les deux situations. L'objectif était alors de concorder entre le virtuel et la réalité.

5. PROBLÈMES ET PROGRESSION

Au cours de ce projet, nous avons fait face à plusieurs difficultés, certaines plus grandes que d'autres. Au cours des premières semaines, les principales problématiques étaient la communication, la cohésion de groupe et s'habituer aux nouveaux outils organisationnels et de programmation tels que savoir rédiger la description d'une tâche sur trello, écrire systématiquement les tâches supplémentaires qu'on effectue au cours de la semaine, mieux distribuer le travail entre les membres du groupes, être joignable, savoir se faire comprendre, se familiariser avec les différentes nouvelles ressources ou logiciels de développement et bien d'autres encore.

Grâce à la méthode **AGILE**, nous avons pu identifier rapidement ces différents problèmes et trouver des solutions pour les surmonter et nous améliorer continuellement au fur et à mesure du projet. Par exemple, il y avait souvent des écarts importants entre les temps estimés et les temps réalisés de certaines tâches, particulièrement celles qu'on avait du mal à visualiser. Cela posait problème car mal estimer une tâche c'est passer trop de temps sur cette dernière et par conséquent faire qu'une partie du travail attendu ou à l'inverse ne pas en faire assez. Mettre une tâche de brainstorming en amont était une bonne initiative qui nous a permis de mieux faire nos estimations par la suite.

Un des autres problèmes rencontrés est la réorganisation du code en modules et packages. Malgré le temps passé à comprendre le fonctionnement des imports, nous n'avions que partiellement réussi. Nous n'avons pu régler le problème qu'après l'intervention des professeurs. Le problème était une mauvaise compréhension de ce qui devait être exécuté ou pas. Le code source n'est pas fait pour être exécuté mais seulement importé, seul le main devait être à la racine et être exécuté et c'est dans celui-ci qu'on importe les modules.

Enfin, la dernière grande difficulté, à laquelle nous avons dû être confrontés, était de garder une constance dans le travail à fournir chaque semaine. En période de partiels ou d'examens, il était compliqué de concilier révisions et tâches à accomplir pour l'avancement du projet. On a trouvé que la charge de travail était parfois compliquée à gérer mais on a essayé de faire du mieux qu'on pouvait.

6. RÉSULTATS ET ANALYSES

6.1 Comparaison avec les attentes

Dans le cadre de notre projet robotique, nous avions pour objectif de créer une représentation 2D et une représentation 3D. Notre but était de fournir une interface pour visualiser des modèles robotiques dans un environnement virtuel. Pour la partie 2D, nous avons utilisé la bibliothèque **Tkinter**, et pour la partie 3D, nous avons choisi **Panda3D**. Nos objectifs ont été atteints.

Pour la partie caméra, le robot doit reconnaître une balise dans l'environnement et la suivre. Pour ce faire, nous avons développé un programme en utilisant des notions de base en traitement d'images ainsi que plusieurs bibliothèques Python (opencv, matplotlib, numpy, etc..).

Le programme convertit d'abord une image **RVB** (rouge, vert, bleu) en image **TSL** (teinte, saturation, luminance) (*Cf. figure 6.1*). Ensuite, l'image est redimensionnée et floutée pour éliminer les couleurs perturbatrices, facilitant ainsi l'extraction des couleurs de la balise. La segmentation des couleurs de la balise se fait à l'aide des valeurs basées sur le TSL (*Cf. Figure 6.2*).

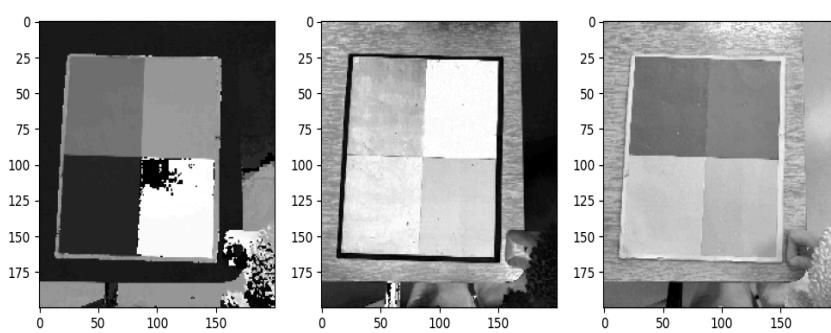


Fig 6.1 - TSL sur la balise

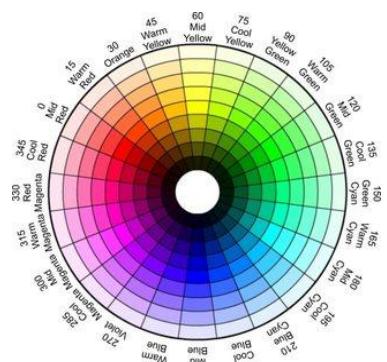


Fig 6.2 - Valeurs TSL

Nous calculons ensuite les barycentres de chaque couleur afin d'obtenir leurs coordonnées (*Cf. figure 6.3*). Grâce à ces coordonnées, nous pouvons déterminer si une couleur est adjacente à une autre. Si les quatres couleurs de la balise (jaune, rouge, vert, bleu) remplissent cette condition, alors la balise est présente dans l'image perçue par la caméra, sinon, elle ne l'est pas.

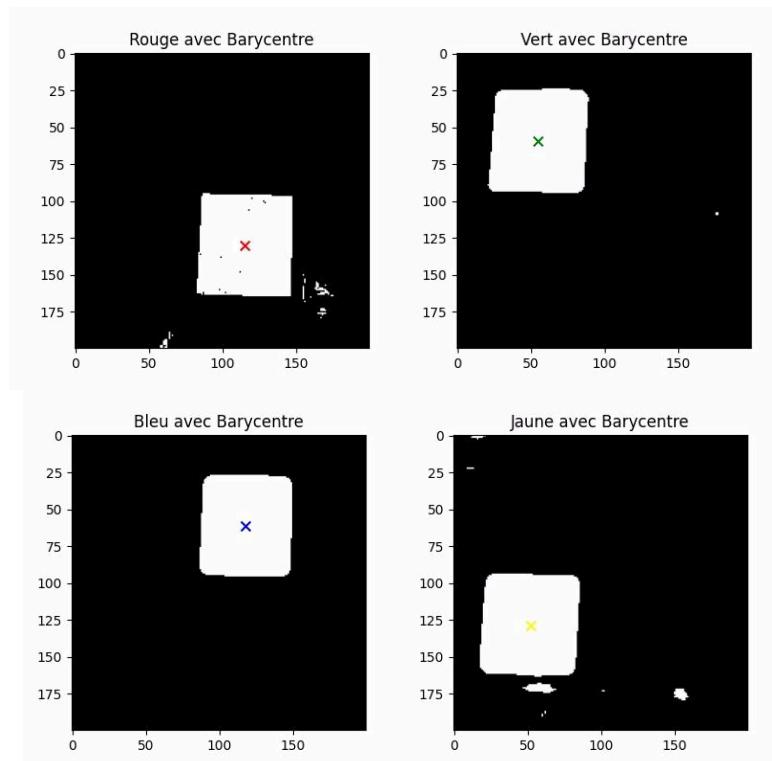
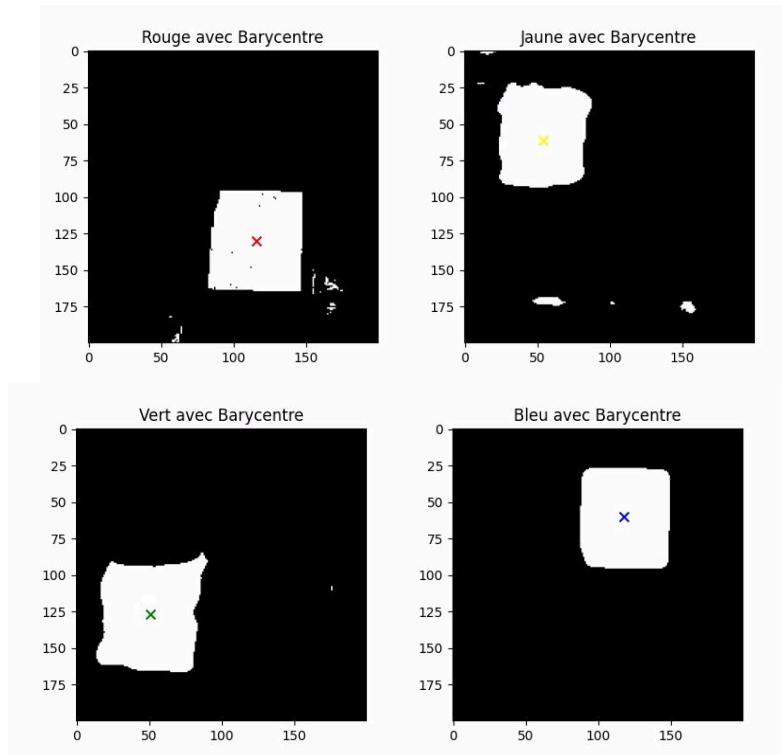


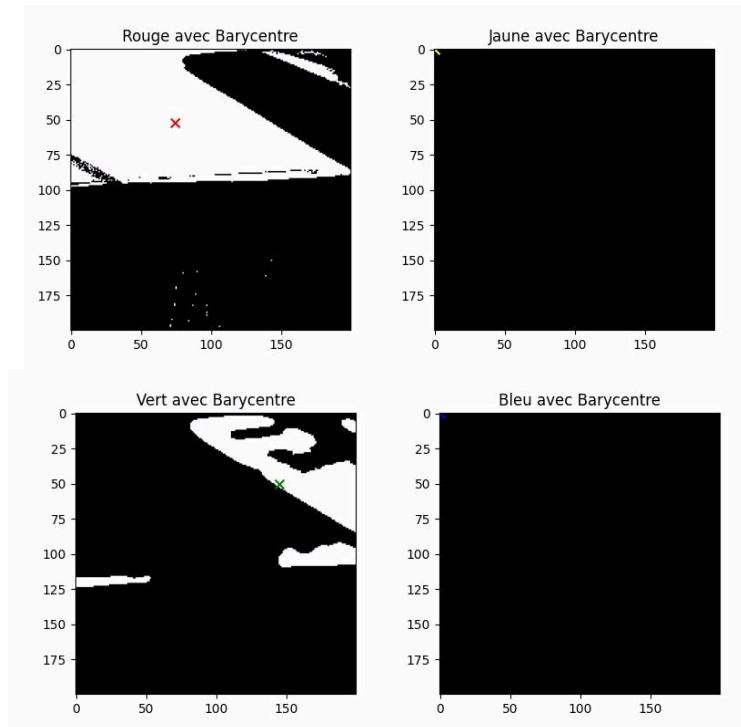
Fig 6.3 - Extraction des couleurs de la balise avec leurs barycentres

Enfin, nous avons intégré des stratégies de mouvement telles qu'**AvancerToutDroit** et **Tourner** pour permettre au robot de suivre la balise. Malheureusement, nous n'avons pas encore pu tester ces fonctionnalités sur le robot réel. Cependant, nous avons testé le programme sur la balise donnée et dans un environnement aléatoire pour démontrer que la reconnaissance de la balise fonctionne correctement (*Cf. figure 6.4 et 6.5*).



Jaune est à côté de Vert et Rouge : False
Vert est à côté de Bleu et Jaune : False
Bleu est à côté de Vert et Rouge : False
Rouge est à côté de Jaune et Bleu : False

Fig 6.4 - Extraction des couleurs de la balise avec les couleurs intervertis et non reconnaissance de la balise



Jaune est à côté de Vert et Rouge : False
Vert est à côté de Bleu et Jaune : False
Bleu est à côté de Vert et Rouge : False
Rouge est à côté de Jaune et Bleu : False

Fig 6.5 - Extraction des couleurs d'une image d'un environnement aléatoire et non reconnaissance de la balise

7. CONCLUSION

Pour conclure, ce projet était une expérience enrichissante et précieuse pour notre équipe, tant dans le développement de code en POO que dans l'utilisation de diverses ressources telles que GitHub, des bibliothèques graphiques et Blender ou encore des différents outils organisationnels.

Bien que nous ne soyons pas arrivés au terme de notre projet final, il nous a permis d'apprendre à travailler en groupe, à mieux communiquer, mieux structurer notre travail et maîtriser de nouveaux outils de programmation. Devoir repasser plusieurs fois sur notre code au fur et à mesure de notre progression nous a permis de repousser nos limites et d'acquérir de nouvelles connaissances et compétences au sujet de la robotique. Nos réunions hebdomadaires du mercredi nous ont aidées à nous organiser tout au long de la semaine et à définir de nouveaux objectifs. Grâce à cette approche, nous avons pu nous adapter rapidement en fonction des résultats obtenus.

De plus, ce projet constitue un réel atout pour nous car, en plus de nous être utile dans le domaine de la robotique et du traitement d'image, il nous a ouvert, pour la team 7 FUTURAMA, de nouvelles perspectives et opportunités à l'avenir.