RAPPORT

PROJET ROBOT LU2IN013

DRIFTATOR

Membres du groupe :

- LAHKIM BENNANI Laila 21102544
- DAUTRICHE Maxime 21105872
- OPPISI Vincent 21100703
- CHEN Laurent 21101875
- VOLLAND Guillaume 21112373

Sommaire :

Introduction :

Dans le cadre de notre UE Projet Robotique (LU2IN013), nous avons développé un package python contenant un certain nombre de fonctions et outils afin de rendre notre robot fonctionnel et facilement utilisable pour le client.
Nous avons réalisé une simulation en 2D et 3D, en parallèle des fonctionnalités pour le robot réel.
La structure principale de notre projet se trouve dans le package Driftator. Ce dernier est divisé en trois parties: IA, Simulation et Affichage, lesquelles seront détaillées par la suite.
Notre script principal (main.py) se trouve à la racine du projet, il permet d'exécuter le code sur le robot, ou de lancer une simulation sur n'importe quel ordinateur. Nous avons également créé une interface graphique (fichier main_ui.py) afin d'avoir une vision globale sur les différentes configurations et IA disponibles pour la simulation, et pouvoir les gérer plus facilement.
Il est possible de paramétrer plusieurs valeurs pour la simulation et le robot à travers un fichiers de paramètres (settings.json).
Nous fournissons une documentation Pydoc complète avec l'ensemble des fonctions implémentées dans notre projet.
Nous avons réalisé les trois tâches principales : tracer un carré, s'approcher le plus vite possible et le plus près d'un mur sans le toucher, et suivre une balise.
Enfin, notre projet final était de développer un circuit sur lequel plusieurs robots pourraient circuler. Il sera présenté dans la dernière partie de ce rapport.

I. Le package “Driftator”

A. Structure générale du package

Le package Driftator est subdivisé en 3 modules : un module dédié à l’affichage, un module dédié à la simulation et un module dédié à l’IA du robot.

1) Le module IA

Le module IA permet de transmettre les instructions au robot. Il s’agit de la seule partie du code qui s'exécute sur le vrai robot, et il est constitué de 4 scripts.
Le script ia.py est chargé de gérer l’intelligence du robot: c’est ici que sont définies les classes correspondant aux instructions de bases (avancer, tourner, …) qui seront utilisées plus tard dans le projet.
Le script controleur.py est un intermédiaire entre l’IA et le robot. Il reçoit et traite les informations transmises par l’IA, avant de les transmettre au robot virtuel ou physique, si l’on est dans la simulation ou non.
Le fichier position_balise.py contient des fonctions qui permettent d’analyser les images enregistrées par la caméra du robot et d’y déterminer le type et la position de diverses balises.
Enfin, le script parser_ia.py permet de lire une suite d’instructions depuis des fichiers .ia et les convertir en instances des classes d’IA.

2) Le module “Simulation”

Le module Simulation n’est pas exécuté sur le robot physique, mais contient un ensemble de scripts permettant d’exécuter une simulation de l’environnement et du robot sur un ordinateur, ce qui est indispensable pour pouvoir effectuer un grand nombre de tests avant de les mettre en œuvre dans la réalité. Il est constitué de 2 scripts.
Le fichier objets.py contient la définition de plusieurs classes permettant de simuler le comportement de plusieurs objets de la réalité. On y retrouve des classes pour représenter le robot, des obstacles (ronds et rectangulaires), ou encore un terrain d’une certaine superficie. Les caractéristiques du vrai robot ont été reproduites le plus fidèlement possible afin d’assurer que les tests soient proches de la réalité.

Le script simulation.py utilise toutes les classes définies dans le script objets.py pour exécuter une simulation. Il met à jour les états des différents objets au fil du temps et vérifie notamment si des évènements ont eu lieu, comme des collisions entre les objets.

3) Le module “Affichage”

Pour finir, le module Affichage permet d’afficher en temps réel une simulation à l’écran. Nous retrouvons dans ce module plusieurs ressources nécessaires à l’affichage (images et modèles 3D), ainsi qu’un unique script affichage.py.
Ce script contient 2 classes correspondant aux deux types d’affichage: 2D et 3D.
L’affichage 2D est effectué à l’aide de la librairie Pygame, et nous avons choisi la librairie Panda3D pour gérer notre affichage 3D.

B. Les outils annexes facilitant l’utilisation du package

Nous avons développé plusieurs outils pour faciliter grandement l’utilisation du package, utilisables sur le robot ou pour des simulations.

1) Fichiers .ia

La première fonctionnalité annexe de notre projet est la création de fichiers textes “.ia”. Ces fichiers peuvent être lus par le package, et permet de donner facilement un certain nombre d’instructions au robot (réel ou simulé).
Les fichiers “.ia” peuvent être lancés depuis le script main, avec l’argument “-ia” (exemple: python main.py -ia mon_ia.ia).
Les instructions au robot peuvent être données à l’aide du langage “Driftatorian”, un langage de notre création se voulant facile d’accès pour rendre l’utilisation du robot accessible au plus grand nombre.
Les différentes instructions pouvant être écrites dans les fichiers .ia sont:

- L’instruction **avancer** (exemple: avancer d=10 v=200 a=0)
  permet de faire avancer un robot sur une distance d *(en centimètres)*, avec une vitesse v *(vitesse angulaire, degrés par seconde)* avec un angle a *(entre -100 et 100, plus la valeur est*

proche de -100/100, plus le robot tourne sur sa roue gauche/droite).

- L'instruction **tourner** (exemple: tourner a=90 v=100)
  permet de faire tourner le robot sur lui-même d'un angle v *(en degrés)*, avec une vitesse v *(vitesse angulaire, degrés par seconde)*.

- L'instruction **tourner_tete** (exemple: tourner_tete a=20)
  permet de faire tourner la tête du robot (comprenant le capteur de distance et la caméra) à l'angle a *(en degrés, correspondant à la toute gauche et 180 à la toute droite)*.

- L'instruction **stop** (exemple: stop)
  permet de stopper instantanément les déplacements du robot, en mettant la vitesse de ses deux moteurs à 0.

- L'instruction **for** (exemple: for(50){ … })
  permet de répéter un certain nombre de fois le bloc présent à l'intérieur du for. Il est possible de mettre une valeur fixe (comme ici 50), ou une variable.

- L'instruction **while** (exemple: while(2 >= 1 and 1 < 2){ … })
  permet de répéter le bloc présent à l'intérieur du while tant que la condition est vérifiée. La syntaxe des conditions est la même que celle du python.

- L'instruction **if/else** (exemple: if(1 < 2){ … }else{ … })
  permet d'exécuter le contenu du premier bloc si la condition est vérifiée, et le contenu du deuxième sinon.

- L'instruction **alterner/else** (exemple: alterner(...){ … }else{ … })
  fonctionne de la même façon que le if. Cependant, le bloc à exécuter peut changer en cours d'exécution si le résultat de la condition est modifié.

- L'instruction **print** (exemple: print(ma_variable))

permet d'afficher dans la console le contenu d'une variable ou une valeur particulière.

- Les **affectations de variables** (exemple: var_a = 12)
  permettent de stocker des valeurs dans des variables pour les réutiliser plus tard (dans des conditions, par exemple).
  Il existe plusieurs variables particulières qui possèdent des fonctions spécifiques.
  true: constante True en python
  false: constante False en python
  null: constante None en python
  capteurs_background: permet d'activer ou non l'exécution en arrière plan des capteurs de distance et balise (false par défaut)
  capteur_distance: renvoie la valeur du capteur de distance (en cm)
  capteur_balise: renvoie la position horizontale de la balise (rouge-bleu-vert-jaune) si elle a été détectée (entre -1 et 1)
  type_balise: si une balise (jaune-bleu) a été détectée, renvoie son type (entre 1 et 4)
  pos_balise: si une balise (jaune-bleu) a été détectée, renvoie sa position horizontale (entre -1 et 1)
  random: renvoie un nombre aléatoire (entre 0 et 10000000)

Toutes ces instructions peuvent être librement combinées pour développer des IA plus ou moins complexes pour le robot.

```
1  //On trace le carré
2  for (4){
3      avancer d=40 v=200 a=0
4      tourner a=90 v=60
5  }
6
7  //On avance jusqu'à être à 5cm du mur.
8  while(capteur_distance > 5){
9      avancer d=5 v=300 a=0
10 }
```

Exemple de fichier .ia

Remarque: il est possible d'écrire des commentaires en précédent les lignes d'un double slash.

2) Extension VSCode “Driftatorian”

Afin de rendre plus agréable l’utilisation nos fichiers .ia décrits précédemment, nous avons réalisé une extension “Driftatorian”, disponible pour l’IDE Visual Studio Code.
Cette dernière permet d’ajouter une coloration syntaxique dans ces fichiers.
Elle est disponible gratuitement sur le marketplace de Visual Studio, à l’adresse suivante:
https://marketplace.visualstudio.com/items?itemName=Lo-ran.driftatorian

3) Interface “Driftator Editor” (en plus ça rime)

Pour simplifier l’édition et le lancement des simulations, nous avons développé une interface graphique. Cette dernière peut être ouverte en exécutant le script “main_ui.py”.

Menu principal de l’éditeur

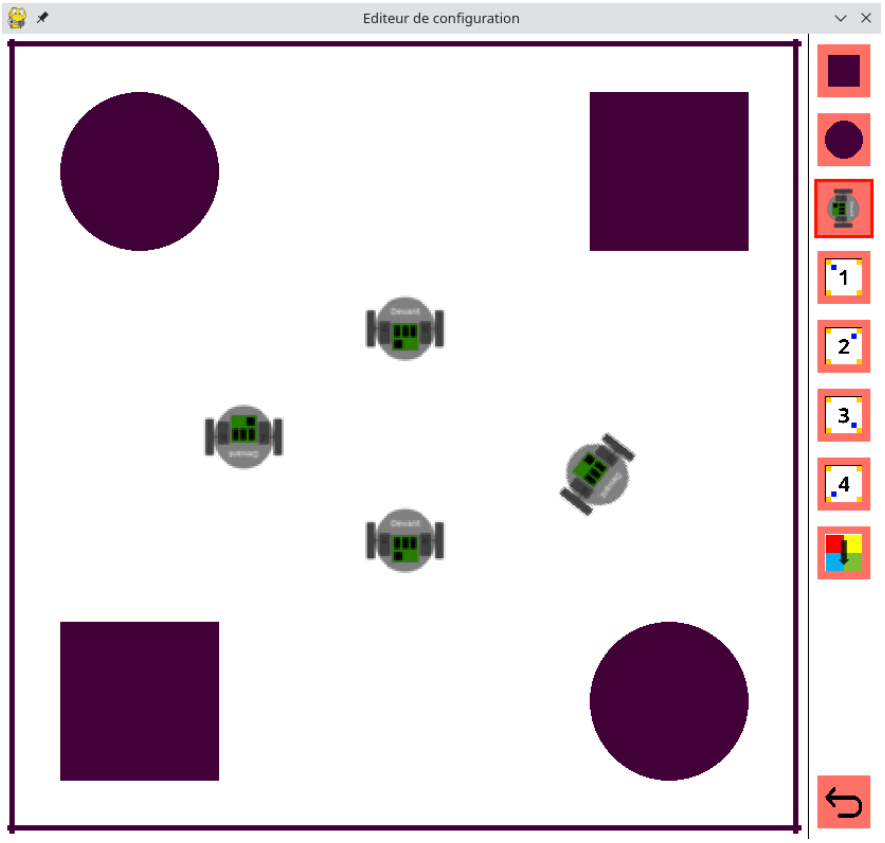
- Menu principal

Ce menu principal offre plusieurs fonctionnalités. Premièrement, il est possible de sélectionner l'un des fichiers de configurations, une IA, la vue (2D ou 3D), et de lancer directement une simulation. Il est de plus possible de créer directement un nouveau fichier de configuration ou une nouvelle IA en cliquant sur le bouton "+" à la fin d'une des deux listes (la navigation entre les pages se faisant par l'intermédiaire des flèches directionnelles). Enfin, il est possible de supprimer une configuration ou une IA en la sélectionnant, puis en pressant les touches "CTRL+SUPPR".

En effectuant un CTRL+CLIC sur un fichier IA, ce dernier sera ouvert dans VS Code. De même, effectuer un CTRL+CLIC sur un fichier de configuration permet de l'ouvrir dans l'éditeur de configuration.

- Editeur de configuration

Pour faciliter la création de plusieurs configurations, nous avons également créé une interface graphique dans ce but.



Interface de création de configuration

Sur le menu latéral, nous retrouvons plusieurs outils permettant d'éditer rapidement des configurations. Il est possible de dessiner des obstacles rectangulaires et ronds, de placer des robots, ou encore de disposer tous les types de balises sur le terrain. Un CTRL+S permet de sauvegarder la configuration, et le bouton "retour" en bas à droite permet également de l'enregistrer et de retourner au menu principal.
Enfin, lancer le script edit_config.py avec l'option -f (ou –fond) permet d'afficher une image de fond en tant que repère, pour placer des obstacles de façon fidèle à la réalité par exemple.

II. L’ environnement de simulation

A. Simulation du robot et de son environnement.

Pour effectuer des tests qui soient le plus pertinents possibles, nous avons essayé de reproduire au mieux la réalité dans notre environnement de simulation. Nous avons donc réalisé 2 types d’obstacles (ronds et rectangulaires, pouvant être combinés pour réaliser des formes plus complexes), et reproduit au mieux les fonctionnalités du robot.

Comme dans la réalité, les contrôles du robot se font en attribuant des valeurs de vitesses à chacune de ses deux roues (il s’agit de vitesses angulaires, en degrés par seconde). En attribuant des valeurs différentes, il est donc possible de faire avancer ou reculer le robot avec un degré de rotation plus ou moins élevé. Cependant, nos IA de base (avancer, tourner) permettent de donner des ordres plus précis (avancer d’une certaine distance, tourner d’un certain angle), sans avoir à se soucier des valeurs à attribuer.

Lors de nos tests en conditions réelles, nous nous sommes aperçus que les valeurs renvoyées par l’appel aux moteurs pour obtenir l’angle des roues étaient mis à jour moins fréquemment que nos IA. C’est pourquoi nous avons également reproduit ce comportement dans notre simulation, en bridant volontairement la vitesse d’actualisation des valeurs de rotation des roues du robot simulé. De plus, nous avons mis en place un mécanisme de prédiction, permettant d’estimer la position des roues en fonction de leur vitesse et du temps passé depuis leur dernière actualisation, ce qui nous a permis d’obtenir des résultats encore plus précis (et ce quelque soit la fréquence d’actualisation des valeurs de rotation des roues).

Nous avons également reproduit le capteur de distance du robot. Pour cela, nous effectuons un projeté de rayon virtuel en fonction de l’orientation du robot, mesurant la distance jusqu’au premier obstacle rencontré.

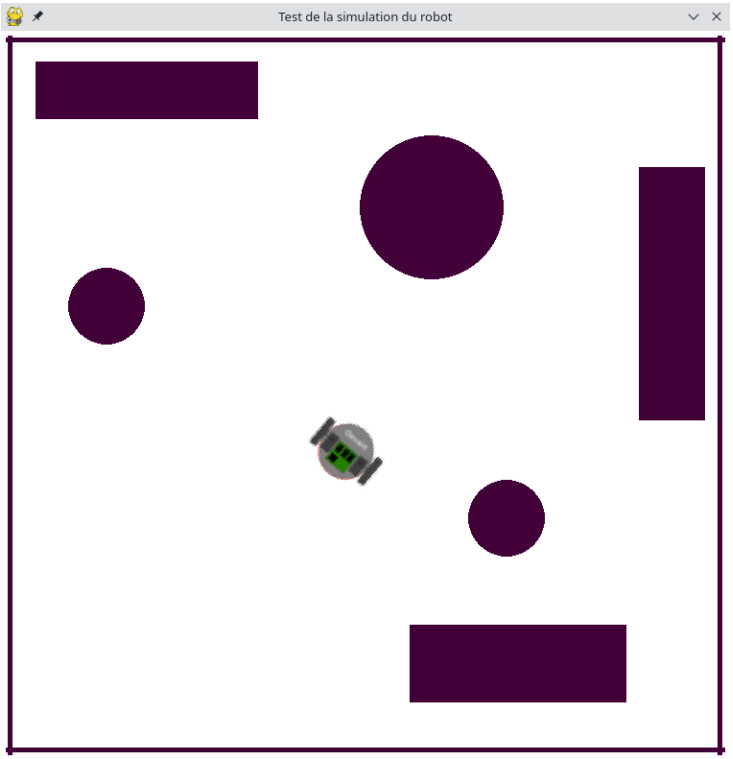
Enfin, la simulation de la caméra du robot est effectuée par l’intermédiaire de la librairie Panda3D. En effet, grâce à notre simulation 3D et la possibilité d’afficher la vue du robot à la première personne, nous avons pu reproduire la “vue” du robot réel à travers sa caméra.

Cela nous a permis d'effectuer de nombreux tests pour la détection de la balise.

B. Représentation graphique de la simulation

1) Représentation 2D

Comme nous l'avons dit précédemment, nous avons fait le choix de pygame pour notre affichage 2D. Le paramétrage de l'affichage est possible par l'intermédiaire du fichier "settings.json" situé à la racine du projet.
Ces différents paramètres permettent de modifier l'échelle d'affichage, le nombre d'images par secondes, l'affichage ou non du rayon du capteur de distance et la trace du robot.



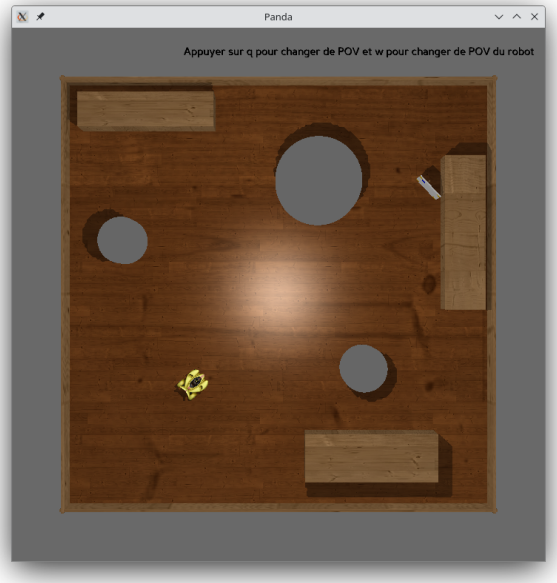
Exemple d'affichage d'une simulation en 2d

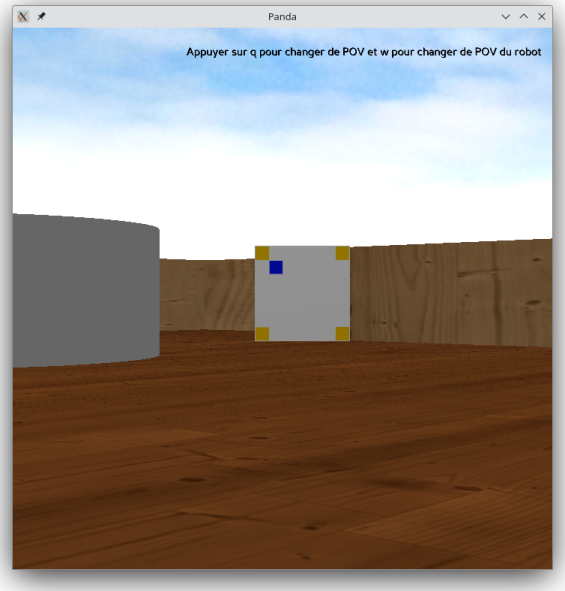
Les simulations 2D sont utiles pour analyser les déplacements du robots et s'assurer que sa trajectoire est correcte, ou encore pour effectuer des tests avec plusieurs robots tout en gardant une vue d'ensemble sur la scène.

Cependant, certaines fonctionnalités ne sont pas disponibles dans la simulation 2D. En effet, le capteur de balise nécessitant la prise d'images pour la détection n'est pas fonctionnelle dans ce modèle. Pour ce genre de tests, il faudra privilégier la simulation 3D.

2) Représentation 3D

Notre simulation 3D fonctionne avec le moteur Panda3D. Contrairement à la vue 2D, elle ne permet pas d'afficher certains éléments comme la trace du robot ou le capteur de distance, mais elle a l'avantage de proposer des fonctionnalités plus poussées.





La touche Q permet d'alterner entre une vue "première personne" du robot et une vue du dessus, et la touche W permet de changer de robot.
L'intérêt majeur de la simulation 3d est qu'elle permet, lorsque la vue "première personne" est activée, de simuler complètement la caméra du robot. Elle rend ainsi fonctionnelle la détection de la balise, ce qui n'était pas possible avec la simulation 2D.

III. Le projet

A. Les objectifs demandés

1) Tracer un carré

Le premier objectif était de tracer un carré. En combinant les IA for, avancer et tourner, nous avons pu aboutir à ce résultat:

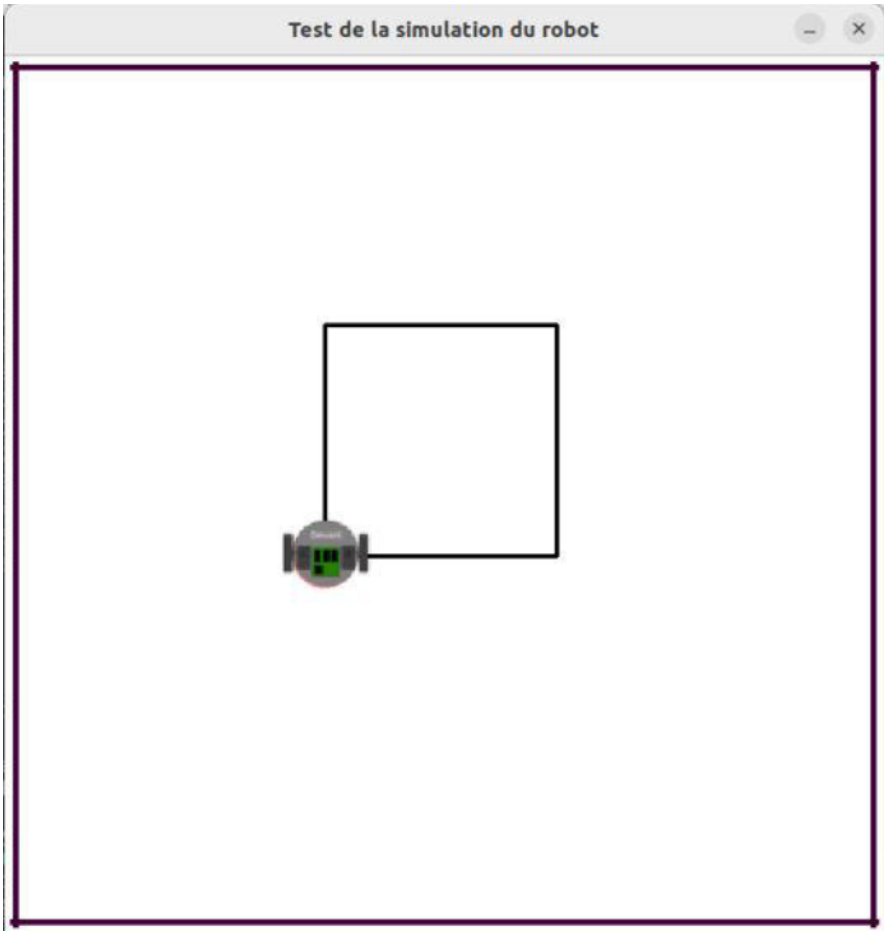


Image de la simulation 3d après le tracé du carré

```
1  //On trace le carré
2  for (4){
3      avancer d=40 v=200 a=0
4      tourner a=90 v=60
5  }
```

Fichier .ia permettant de tracer ce carré

2) Approcher le mur

Le deuxième objectif était de s'approcher le plus près possible du mur, sans le toucher.. Pour cela, nous avons utilisé nos IA while et approcher_mur.

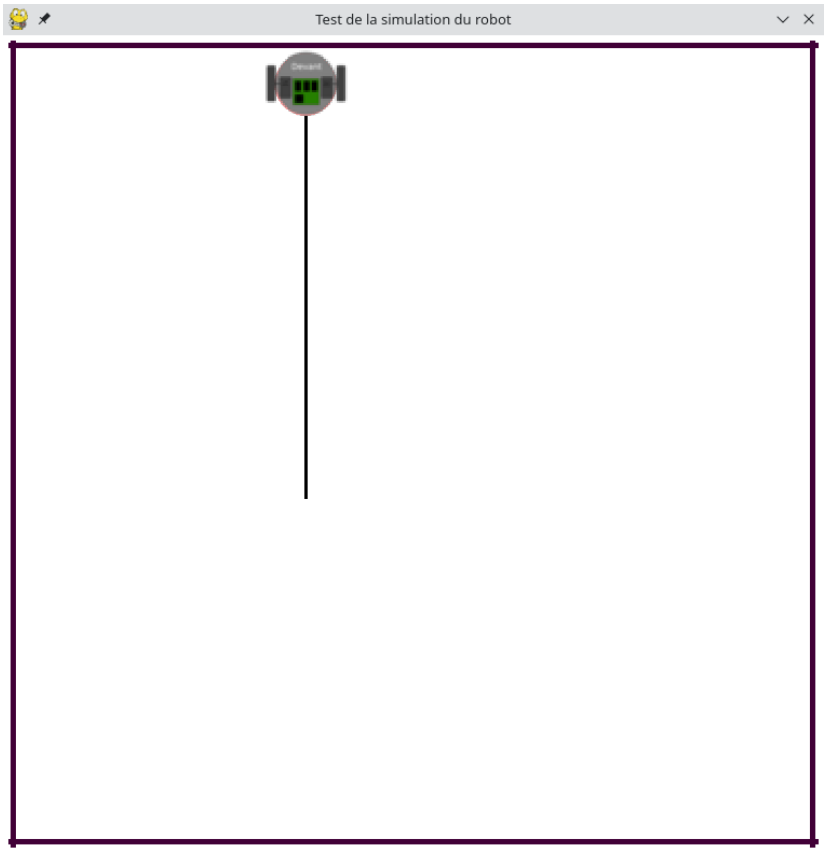


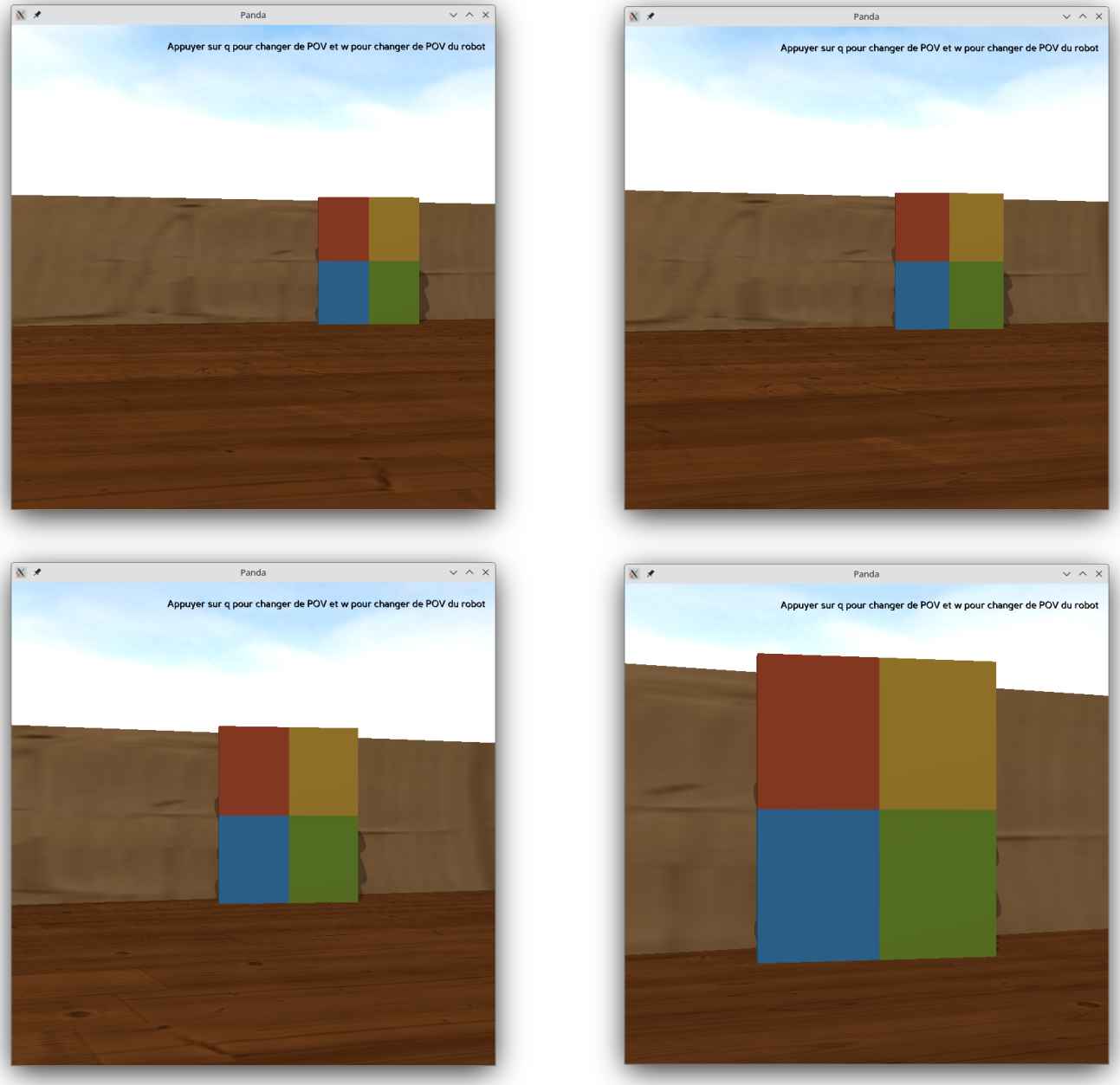
Image de la simulation après l'approche du mur

```
1  //On approche jusqu'à être à 1cm du mur
2  while(capteur_distance > 1){
3      avancer d=10 v=200 a=0
4  }
```

Fichier .ia permettant d'arriver à ce résultat

3) Suivre la balise

Le dernier objectif était de détecter et suivre une balise. Nous avons pour cela développé une fonction qui analyse la couleur des pixels en utilisant la librairie numpy. En ne retenant que les pixels de couleur rouge, jaune, vert et bleu, nous pouvons ainsi déterminer la position horizontale de la balise ce qui permet de la faire suivre par le robot.



Images de l'avancée du robot vers la balise dans la simulation 3D

```
1   //On cherche et on suit la balise
2   while(capteur_distance > 3){
3       b = capteur_balise
4       if(b == null){
5           tourner a=3 v=100
6       }else{
7           angle = 50 * b
8           avancer d=2 v=200 a=angle
9       }
10  }
```

Fichier .ia permettant de suivre la balise

B. Projet final

Le but de notre projet final est d'utiliser l'ensemble des fonctions et outils développés précédemment afin de réaliser un circuit sur lequel vont pouvoir interagir plusieurs robots.

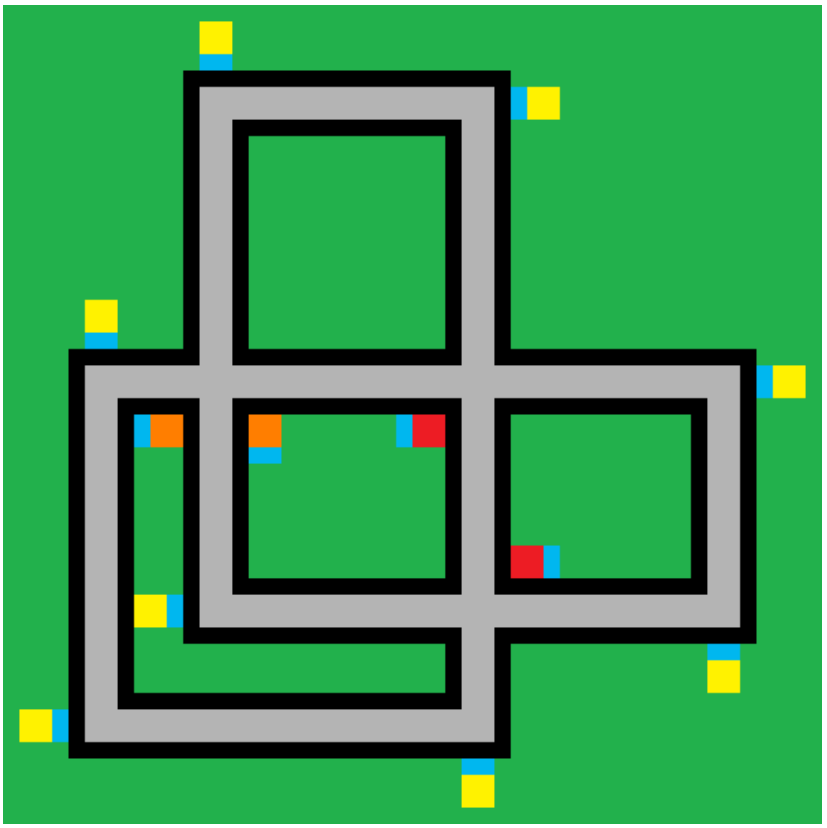


Image d'un circuit concept

Nous avons pour cela conçu un nouveau type de balises: les balises jaune-bleu.

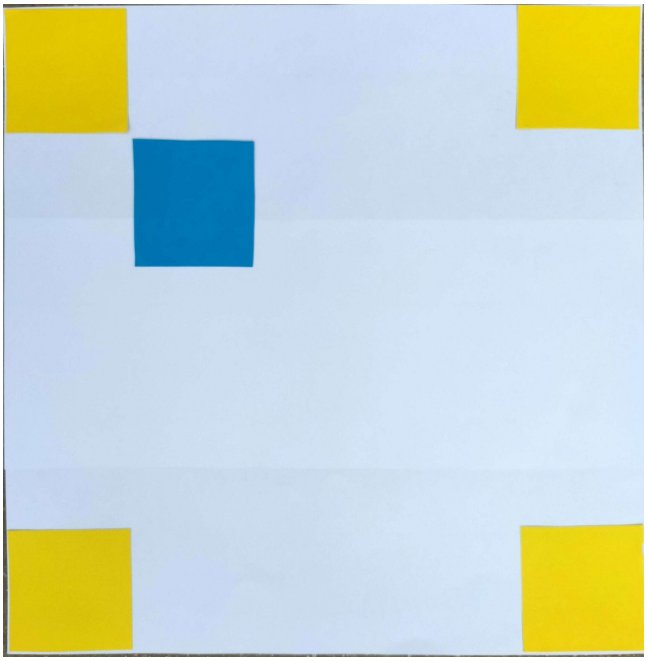


Photo d'une balise JAUNE-BLEU

Ces balises peuvent être détectées par le robot, et ont la particularité de pouvoir représenter 4 types de balises différentes selon leur orientation (et donc la position du carré bleu sur la balise).
Nous avions donc l'intention de donner plusieurs ordres au robot. Les balises représentées en jaune et bleu sur l'image correspondent à des panneaux "tourner". Le robot est censé avancer jusqu'à elles, puis tourner à droite, ce qui lui permet de suivre le tracé du circuit.
Les balises représentées en jaune et orange correspondent à des feux rouges. Lorsqu'elles sont aperçues, si le feu est rouge, alors le robot doit attendre qu'il passe au vert avant d'avancer. L'alternance entre les feux rouges et verts est gérée par le robot, à l'aide d'une horloge interne.
Enfin, les balises représentées en jaune et rouge correspondent à des panneaux "stop". Lorsqu'elles sont aperçues, le robot doit s'arrêter, et s'assurer qu'aucun autre robot n'approche avant de poursuivre son chemin (à l'aide de son capteur de distance).

Malheureusement, bien que la détection des balises fonctionne dans la simulation 3D, elle s'est révélée trop approximative dans la réalité pour qu'elle puisse être vraiment utilisable.

Nous n’avons donc pas pu achever ce projet avec le robot physique.
Une configuration de démo du circuit ainsi qu’un fichier .ia permettent cependant de lancer une simulation dans laquelle le robot suit le circuit. Les balises correspondant aux panneaux “stop” et aux feux rouges n’ont pas été implémentées.

Conclusion :

Bien que nous ne soyons pas arrivés au terme de notre projet final, nous avons tout de même pu développer un environnement de simulation complet et fonctionnel, ainsi que de nombreux outils annexes facilitant son utilisation.
L'ensemble du code du projet, les fichiers de configurations et .ia de démonstration, ainsi que la documentation peuvent être retrouvés sur GitHub. Ils sont accessibles grâce au lien suivant:
https://github.com/MaxDtrc/Projet-Robot-LU2IN013

Pour la réalisation de ce projet, nous avons utilisé les librairies suivantes:

- Panda3D (affichage 3D de la simulation)
  https://www.panda3d.org/

- PyGame (affichage 2D de la simulation)
  https://www.pygame.org/

- pynput (contrôle de la simulation avec un clavier)
  https://pypi.org/project/pynput/

- numpy (détection de la balise)
  https://numpy.org/

- pillow (manipulation d'images pour la détection de la balise)
  https://pypi.org/project/Pillow/