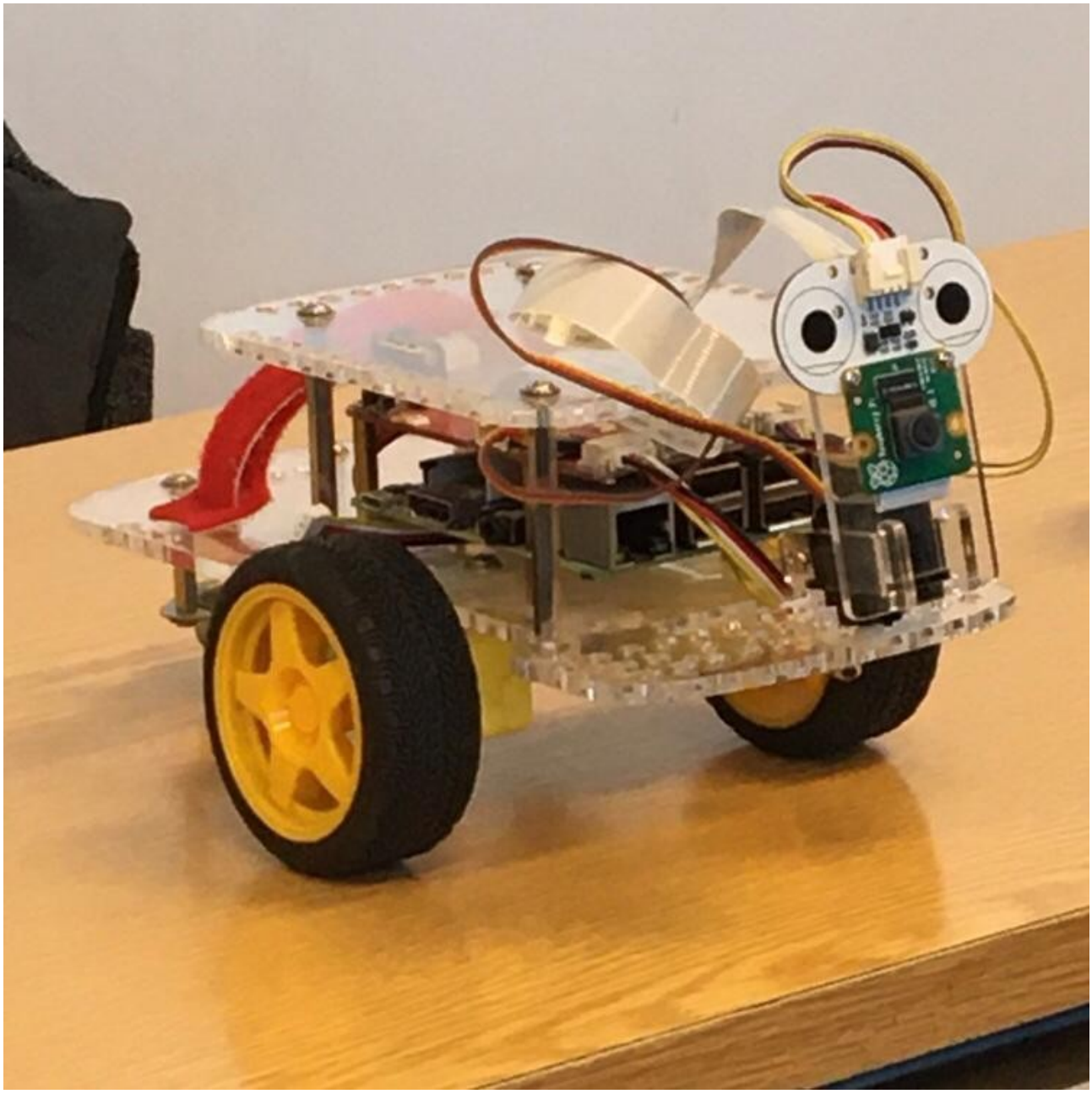
groupe DEXTER2-0 :
Bengana Massyl 21107297
Lahbib Yassin 28718491
Munsup Warintara 28707636
Tighrine Dilia 28721597
van der Meer Zélie 28723936

Compte-rendu
projet robotique L2IN013



Sommaire :

Rappel des objectifs

Trello

Présentation de la structure du github

Travail en groupe

Présentation de l’implémentation du robot

Présentation du senseur de distance

Présentation de l’implémentation des obstacles

Présentation de l’affichage

Présentation de l’implémentation des ia

Parallélisation

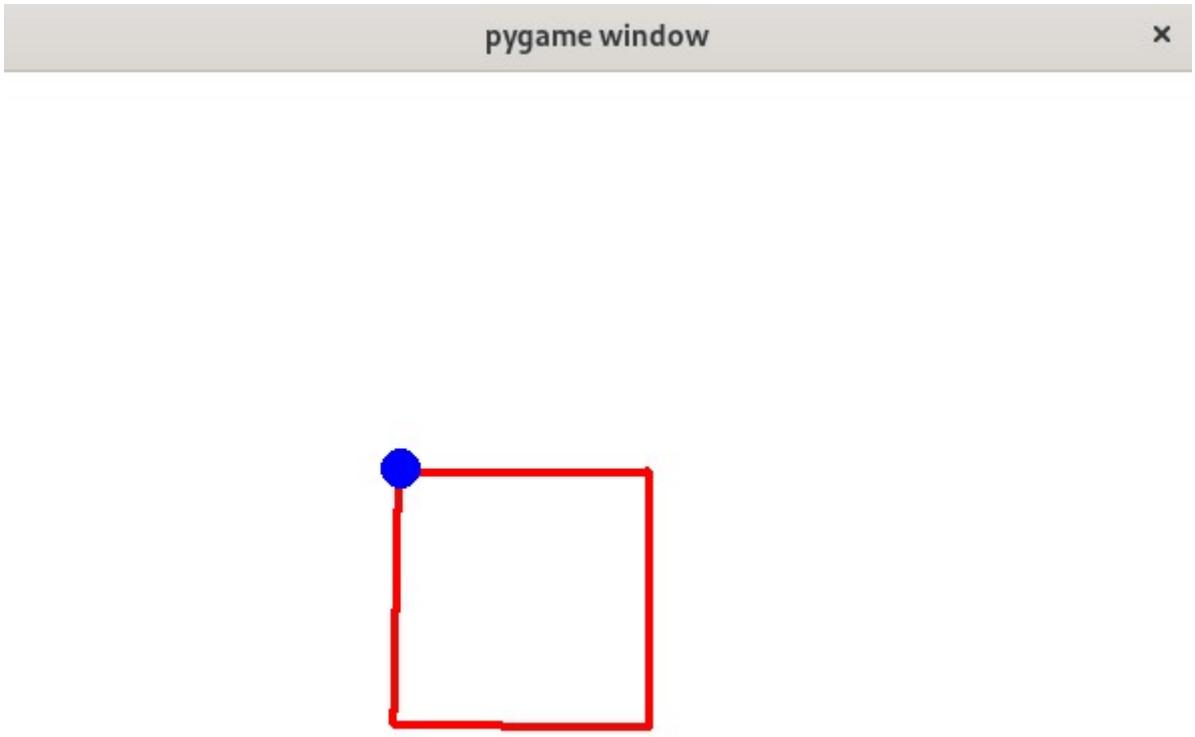
Présentation du traitement d’image

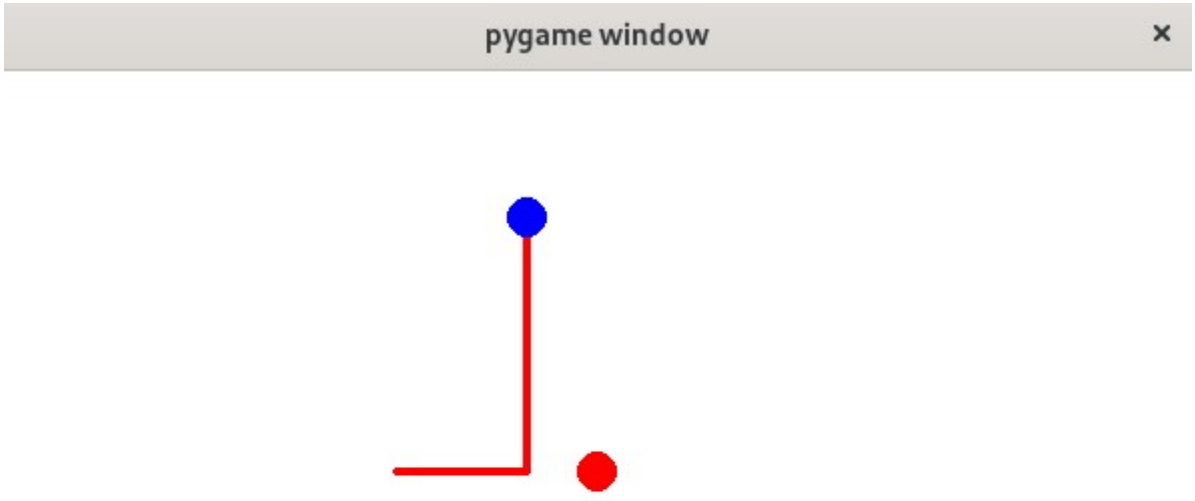
Vrai robot

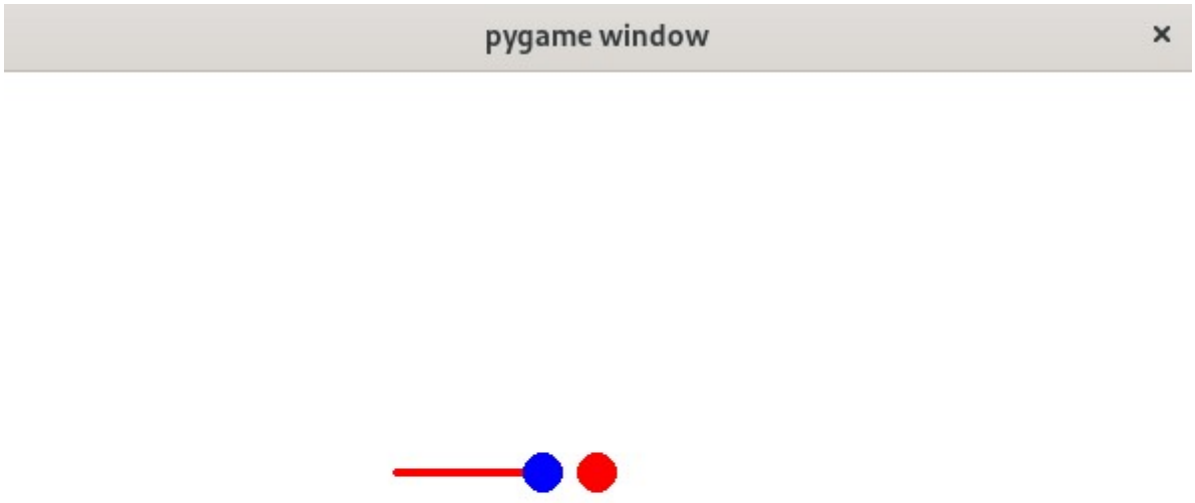
Bibliographie

Rappel des objectifs

Les trois premiers objectifs de ce projet sont que le robot réel puisse tracer un carré, s’approcher le plus vite possible et le plus proche d’un mur sans le toucher et suivre une balise. Pour ce faire, il faut coder une simulation du robot et de son environnement afin de pouvoir tester ces objectifs en virtuel et ainsi se rapprocher d’un projet industriel.




pygame window


pygame window

Présentation de la structure du github

lien de notre github : ssh : git@github.com:DEXTER2-0/simulation.git
https : https://github.com/DEXTER2-0/simulation.git

simulation :
- rapport.odt
- rapport.pdf
- support.pdf
- compte-rendu :
  - comptes-rendu des rendez-vous pendant les séances de TME
- src :
  - arretRobotReel.py (permet d'arrêter le robot réel)
  - Code :
    - affichage :
      - affichage.py
      - __init__.py
      - __pycache__
    - ia :
      - IA.py
      - __init__.py
      - __pycache__
      - robot2IN013
      - Traducteur_proxy.py
    - image :
      - Capture_image.py
      - __init__.py
    - __init__.py
    - __pycache__
    - robot2IN013
    - simulation :
      - constantes.py
      - __init__.py
      - Obstacle.py
      - __pycache__
      - Robot.py
      - Simulation.py
      - Terrain.py
      - Vecteur.py
    - TestScript :
      - tests unitaires et autres tests de nos méthodes
  - main.py (main avec le robot réel)
  - mainSimu.py (main avec la simulation)
  - setup.py

Trello

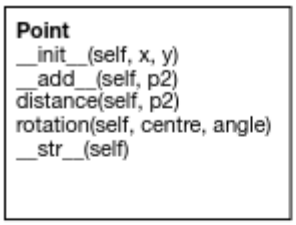
lien de notre Trello : https://trello.com/b/pXFVUTQ1/progression
Les plus grandes difficultés que nous ayons eu à utiliser cet outil de travail sont : garder une organisation claire tout au long du projet, garder trace de nos heures de travail ainsi que réussir à prévoir le bon temps de travail nécessaire selon les tâches prevues. On a utilisé Trello afin de mettre en place les méthodes agile.Il est idéal pour gérer les sprints car il permet aux membres d'etre constamment à jour.Il nous a aidé à visualiser facilement tous les détails des differentes phases du développement, ce qui nous a permis de de résoudre les problèmes complexes. Notre Trello est composé chaque semaine de:
-une liste de backlog pour le Sprint : Celle-ci est pleine en début de sprint et vide (avec beaucoup d'espoir) à la fin du sprint.
-une liste To do : qui contient toutes les taches à réaliser pour la semaine en cours.
-une liste En cours : constituées des taches en cours de réalisation.
-une liste Done : composées des taches terminées et pas encore validées.
-validation : composée des taches validées.

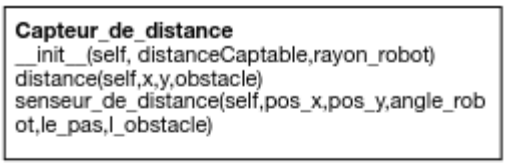
Travail en groupe

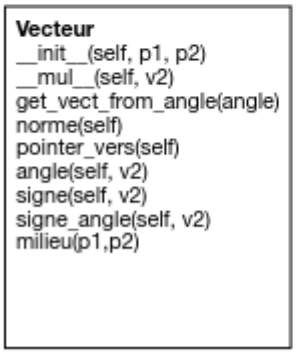
Dans le cadre de cette UE, nous avons appris à construire un projet en groupe, en utilisant de nouveaux outils (Github et Trello).
Nous avons d'abord commencé par apprendre à utiliser ces outils lors des premières semaines du semestre. Nous avons appris à tester nos codes afin de mieux cerner les erreurs et d'éviter de perdre du temps à débeugger (tests unitaires) et nous avons appris à écrire du code qui doit être compréhensible par d'autres (commentaires, noms de variables et méthodes clairs, commits clairs).
La partie la plus difficile a été de trouver le bon équilibre entre se répartir le travail afin de pouvoir avancer à un bon rythme et faire en sorte que tout soit compréhensible et utilisable par tous. Nous nous sommes retrouvé à la faculté une à plusieurs fois par semaine afin de travailler en groupe, mais nous avons aussi du apprendre à travailler en petit groupe ou seul (sinon on perdait du temps) sans que cela ne hache les codes ou qu'ils soient répétitifs.
Nous avons souvent passé du temps à revenir sur l'organisation, la propreté de nos codes ainsi que sur leur utilité car ce n'était pas des automatismes pour nous.
La simulation nous a pris les neuf premières semaines car nous sommes souvent partis dans de mauvaises directions, puis nous avons pus tester nos IA sur le vrai robot.
Pour finir, ce projet était basé sur une grande entraide au sein du groupe.

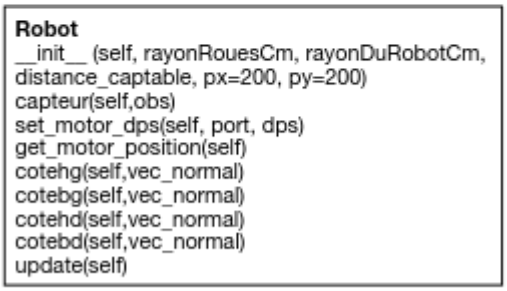
Présentation de l’implémentation du robot et de ses mouvements

Le robot est représenté par classe contentant les informations sur ses dimensions, sa position, les vitesses de ses roues, sa direction, son sens et son capteur de distance.
Nous avons choisi de représenter le robot par un cercle de rayon de 10 cm avec des roues de rayon de 2,5 cm. Chaque roue peut avoir sa propre vitesse. Cependant, nous utilisons que deux cas de figure : soit le robot avance ou recule et dans ce cas, les roues ont la même vitesse, soit le robot tourne et une des roues est immobile alors que l’autre tourne (roue droite pour aller à gauche et inversement).
Mathématiquement, le robot est représenté par un point situé au centre du cercle. Son mouvement est représenté par un vecteur pour chacun des côté du carré inscrit dans le cercle. Toutes ses informations sont recalculées au cours des déplacements (avancer et tourner) du robot. Les calculs des vecteurs dépend de la position actuelle, du rayon des roues et du vecteur normal (dépendant de l’angle du robot).
Nous avons simulé un robot parfait, soit sans frottement des roues sur le sol et aucun bruit dans les mesures du capteur.









Présentation du senseur de distance

Le senseur de distance est l'un des composants essentiels du robot utilisé dans notre projet. Son rôle est de mesurer la distance entre le robot et les obstacles présents dans son environnement. Cela permet au robot de détecter les obstacles et d'ajuster ses mouvements en conséquence pour éviter les collisions.
Nous avons choisi d'implémenter le senseur de distance en utilisant une approche basée sur la géométrie et les calculs mathématiques. Nous avons utilisé la méthode des pas pour estimer la distance entre le robot et les obstacles se trouvant en face de lui.
Grâce au senseur de distance, le robot est capable de détecter les obstacles et de modifier ses mouvements en conséquence pour éviter les collisions. L'implémentation basée sur les pas permet d'obtenir des mesures précises et fiables de la distance entre le robot et les obstacles.
Au départ, nous définissons les paramètres nécessaires, tels que la position du robot (pos_x, pos_y), l'angle de direction du robot (angle_robot), le pas de mesure (le_pas) et la liste des obstacles (l_obstacle).
Dans la boucle principale, à partir de la position du robot, nous avançons progressivement dans la direction spécifiée par l'angle_robot en utilisant des pas de longueur le_pas. À chaque pas, nous vérifions si les coordonnées actuelles se trouvent à l'intérieur d'un obstacle. Pour chaque obstacle de la liste l_obstacle, nous calculons la distance entre les coordonnées actuelles du capteur du robot et les coordonnées de l'obstacle. Si cette distance est inférieure ou égale au rayon de l'obstacle, cela signifie que le robot a rencontré un obstacle.
Nous retournons la distance parcourue jusqu'à l'obstacle le plus proche si un obstacle est détecté. Sinon, si aucun obstacle n'est rencontré avant d'atteindre la distance maximale captable (distanceCaptable), nous retournons la distance maximale moins le rayon du robot.

Présentation de l’implémentation des obstacles

On a choisi d'implémenter les obstacles ainsi :
-Si le type = 0 c'est un mur.
-Si le type = 1 c'est un cercle.
-Si le type= 2 c'est un rectangle.
On gère les collisions grâce à la méthode collision implémentée dans le fichier simulation.py ainsi :
-s'il ne y a pas le robot dans le terrain, la fonction retourne False.
-s'il atteint les balises, la fonction retourne True.
-On parcourt la liste des obstacles présents sur le terrain, si la distance entre le centre du robot et la position de l'obstacle est inférieure à la somme du rayon du robot et celle de l'obstacle en question alors on détecte une collision et on retourne True.

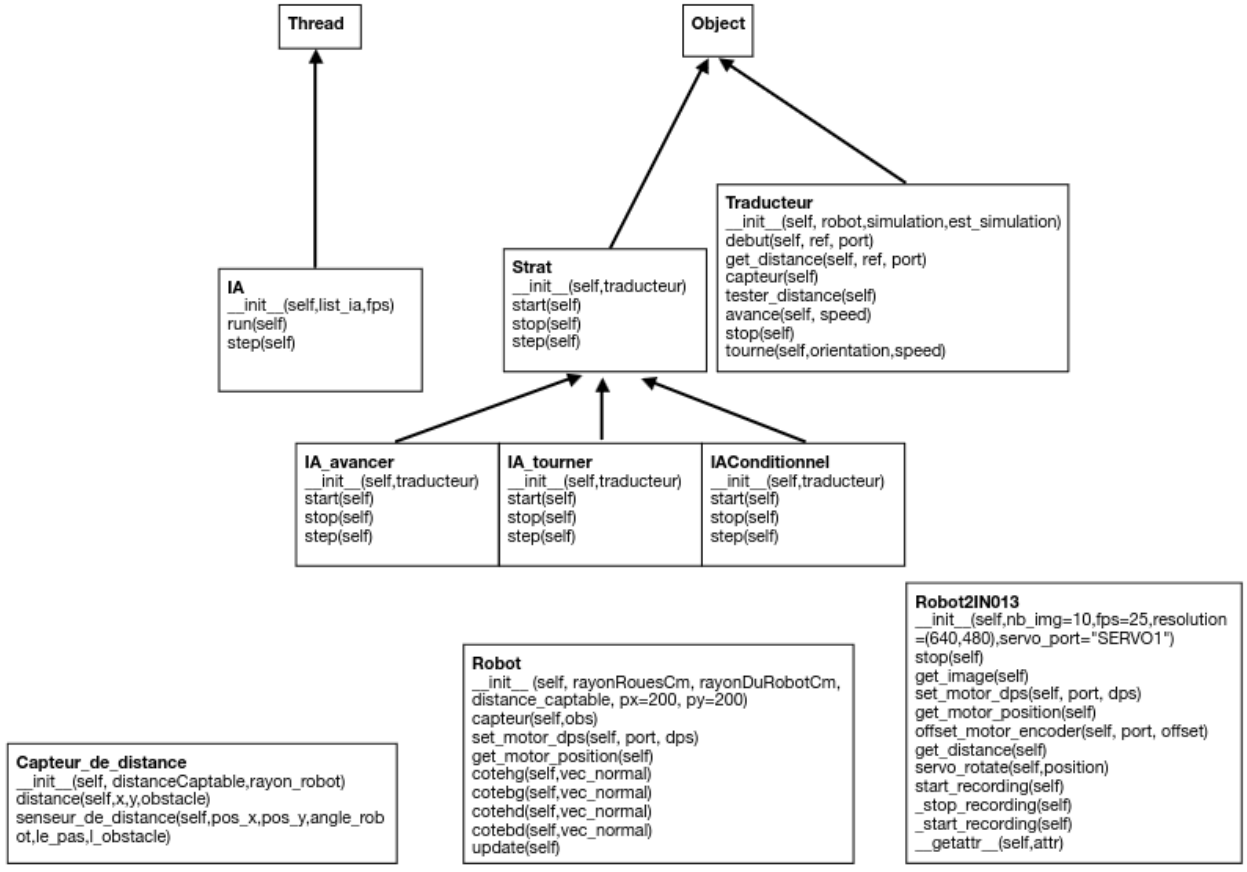
Présentation de l’affichage

On a choisi d’implémenter notre classe d’affichage graphique en utilisant le module Pygame pour l’affichage du robot, des obstacles et des mouvements, tout en se basant sur le principe de parallelisation mentionné ci-dessous afin d’éxecuter la simulation et l’affichage en parallèle.
Indépendamment du projet, l’utilisation de pygame passe nécessairement par quatre étapes :

- Initialisation de pygame
- Appel des modules nécessaires (ex: pygame.display)
- Boucle infinie : dans la fonction update .
- Fermeture du programme : En gérant les événements grâce à la méthode events.

Présentation de l’implémentation des IA

Les IA, ou plutôt les stratégies, permettent de créer de donner les ordres nécessaires à n’importe quel robot afin de créer des mouvements plus complexes que d’avancer sans s’arrêter et de tourner sans s’arrêter. La classe IA est associée à la classe Traducteur qui est la classe qui appelle les bonnes méthodes des classes Robot (réel et simulé) et dans laquelle sont effectués les calculs (la distance parcourue par exemple).
Nous avons implenté plusieurs IA différentes : IA_avancer qui fait avancer le robot sur une distance donnée, IA_tourner qui fait tourner le robot d’un angle donné, IAConditionnelle qui permet au robot d’éviter les obstacles. Chaque IA est décomposée en trois parties : start, qui initialise les calculs et donne l’ordre du mouvement, step, qui demande au Traducteur de faire les calculs et selon le résultat, donne les ordres de continuation du mouvement ou l’ordre d’arrêt en appelant la dernière partie, stop, qui donne l’ordre d’arrêt.
Notre classe Traducteur est également un proxy, ce qui nous permet de donner les ordres aux deux robots (réel et simulé). Pour permettre cela, il fallait que les noms de méthodes des deux robots soient identiques.

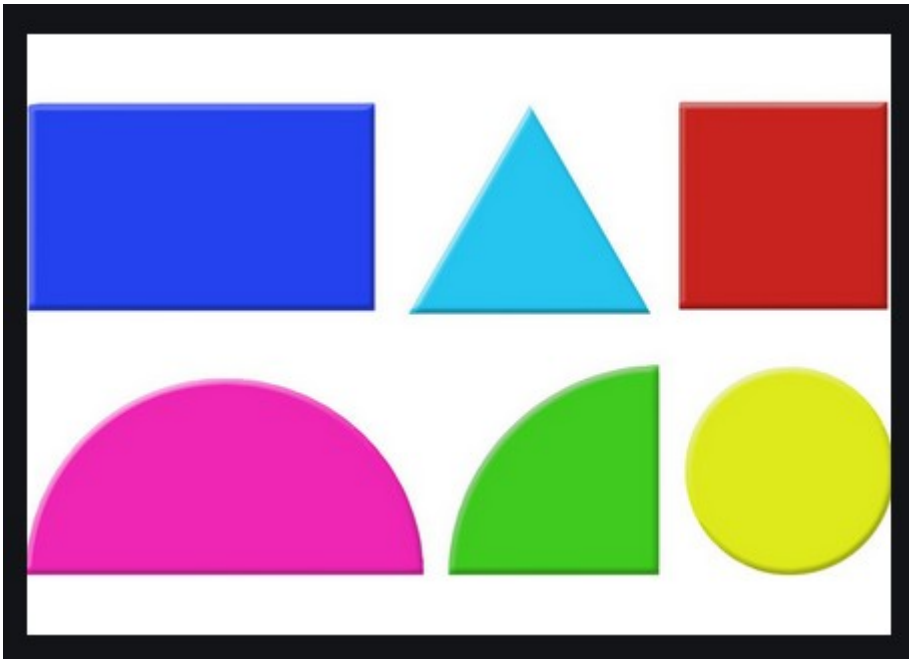


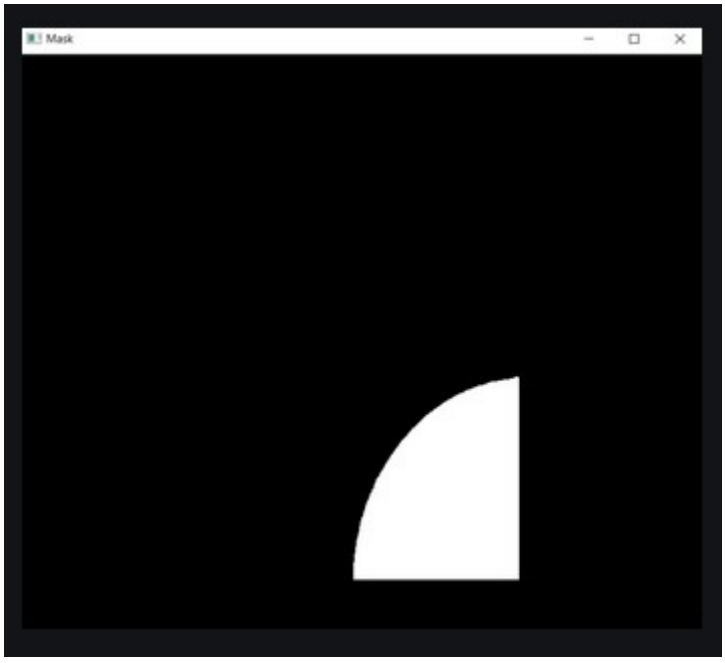
Parallélisation

La parallélisation des différentes méthodes des différentes classes étaient nécessaires car la simulation et l’affichage graphique de celle-ci sont séparées, elles doivent être indépendantes l’une de l’autre, mais doivent effectuer les calculs et produire les mêmes résultats en même temps
Afin de paralléliser nos différentes classes (simulation, affichage,) on a utilisé la classe Thread qu'on a importé du module Threading. On a choisi de spécifier l'activité en utilisant l'héritage et en réimplémentant la méthode run dans la sous classe (classe héritante).

Présentation du traitement d’image

Afin de pouvoir réussir la demande que le robot suive une balise, il faut que le robot puisse la percevoir et donc que les images qu’il prend soient traitées afin de pouvoir la reconnaître.
Nous avons utilisé deux bibliothèques pour cette partie qui sont cv2 et PIL. Nous n’avons utilisé que Image de PIL afin de pouvoir jouer sur le format de l’image (image prise par le robot comme un array et converti en png afin de pouvoir être traitée).
C’est grâce à la bibliothèque cv2 que nous avons pu traiter les images. Tout d’abord, nous avons augmenter la saturation des couleurs dans l’image (image 1) afin qu’elles soient plus reconnaissables, puis pour chaque couleur présente sur la balise, nous vérifions qu’elle est bien présente sur l’image, puis nous analysons sa forme (image 2) et si nous avons bien un carré entièrement dans l’image alors nous considérons que cette partie de la balise est présente. C’est seulement dans le cas où les quatre couleurs ont bien été trouvées (et de la bonne forme) que nous décidons que la balise est présente dans l’image capturée par le robot. De plus, comme la perception des couleurs présentes sur la balise dépendent de la luminosité, nous avons crée une gamme pour chaque couleur afin qu’elles puissent être reconnaissable au maximum.





Vrai robot

Après avoir connecté notre projet au vrai robot. Les fonctionnalités telles que avancer, tourner, dessiner le carrée, détecter l’obstacle, éviter la collision et le capteur de distance fonctionnent sur celui- ci. Le robot peut avancer à la vitesse et à la distance souhaitées. Il peut également s'approcher du mur le plus proche possible avec une vitesse de 490 degrés/sec sans toucher le mur ni dévié à cause d’une vitesse trop forte .
Dans la vraie vie, le robot semble tourner moins bien que dans la simulation. La distance effectuée (l'angle total effectué par le robot) est plus petite que l’angle que l’on a demandé à faire. Comme notre modèle est parfait, lorsque nous avons essayé sur le robot, les frottements ont eu un impact sur l'angle.
Nous avons également un problème d'enregistrement des images du robot (get_image) et en raison de ce problème, nous ne pouvons pas tester notre stratégie de suivi d'une balise.

Bibliographie

cours et tuto Github présentés en TD : https://github.com/baskiotisn/2IN013robot2022

cours de physique et de mathématiques de PTSI (modélisation mathématiques des mouvements du robot et exercices de compréhension)

tuto traitement d'image : https://www.geeksforgeeks.org/color-identification-in-images-using-python-opencv/

documentation PIL : https://pillow.readthedocs.io/en/stable/

documentation cv2 : https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html