Projet MO103 : Rapport de conception
Phase Génie et Intégration

Groupe 4

Janvier 2021

1 Rappel des exigences

Les sous-parties suivantes (1.1, 1.2, 1.3, 1.4, 1.5, 1.6) sont reprises telles quelles du rapport intermédiaire pour rappeler les exigences à suivre.

1.1 Exigences fonctionnelles et contraintes techniques

Fonctions principales	Description	Critères	Précision
FP 1	Se déplacer		
FP 2	Attaquer	Vitesse, force, équilibre...	v>8cm/s
FP 3	Se protéger	Résister à une attaque frontale	
FP 4	Détecter, repérer des objets	Portée du capteur IR	20cm

Table 1 – Fonctions principales et exigences

Fonctions contraintes	Description	Critères	Précision
FC 1	Rester dans le dohyo	Taille du terrain : 1m x 1m	20%
FC 2	Attendre au début du combat	Temps : délai au démarrage	>3s
FC 3	Ne pas endommager le matériel	Matériaux de construction	
FC 4	Ne pas perdre au poids	Masse du robot	<2kg
FC 5	Le robot doit rentrer sur le terrain	Taille du robot	<1m

Table 2 – Fonctions contraintes et exigences

1.2 Exigences opérationnelles

L'aspect opérationnel le plus important est l'autonomie du robot lors du combat. Ainsi il faut qu'il puisse se déplacer seul et trouver son adversaire seul. Pour se faire, plusieurs stratégies ont été testées. De plus, il doit rester alimenter pour fonctionner. On observe donc les exigences opérationnelles suivantes :

Exigence	Besoin	Critères	Précision
Pouvoir suivre la stratégie de combat	Être autonome	Code	Aucune erreur
Maintenir l'alimentation du robot pendant le combat	Fonctionner	Câble d'alimentation	>3m

Table 3 – Exigences opérationnelles

1.3 Exigences d'environnement

La zone de combat, constituée du dohyo et du robot adverse, est l'environnement principal de "vie" de notre robot. Ceci implique certaines exigences.

Exigence	Besoin	Critères	Précision
Pouvoir réutiliser les pièces des robots (allié et adverse)	Ne pas les endommager	Matériaux	
Limiter les frottements sur la moquette	Avancer	Contact avec le sol	Aucun (hors roues)

Table 4 – Exigences d'environnement

1.4 Exigences d'interface

La stratégie retenue a donc dû être codé sur l'unité de commande du robot et un travail d'équipe a dû être fourni. Ceci a imposé certaines contraintes.

Exigence	Besoin	Critères	Précision
Coder la stratégie	Être autonome	Coder en C	Uniquement en C
Constituer une équipe de 10	Travailler en équipe	Humain	+/- 1 personne
Ne pas être plus de 6 dans une salle	Respecter les normes sanitaires	Humain	Aucune

Table 5 – Exigences d'interface

1.5 Exigences d'études et solutions imposées

Ce projet utilise les ressources fournies par l'école et s'inscrit dans le cadre d'un tournoi (interne à l'école) qui possède donc son propre règlement. Ceci génère certaines exigences.

Exigence	Besoin	Critères	Précision
Utiliser la boîte "Bioloid Beginner Kit"	Construire un robot	Boîte	1 seule boîte

Table 6 – Exigences d'études et solutions imposées

1.6 Exigences d'assurance de résultat

Comment s'assurer que notre robot soit bien capable de remporter un combat de sumo ? Tout cela ne peut se faire que par des tests (détaillés dans le plan de validation) qui devront vérifier certaines exigences.

Exigence	Besoin	Critères	Précision
Détecter l'autre robot	Gagner le combat	Porté du capteur IR et vitesse de rotation	<6s/tour
Détecter les limites	Gagner le combat	Seuil du capteur IR	Détection possible
Pousser le robot adverse en dehors du dohyo	Gagner le combat	Vitesse en translation	>8cm/s

Table 7 – Exigences d'assurance de résultat

2 Recherche des solutions techniques

2.1 Architecture

L'architecture du robot a été un des premiers éléments sur lesquels il a fallu réfléchir pour avoir une ébauche de direction à suivre pour débuter. En se basant sur le calcul suivant, nous avons conclut que le robot avait nécessairement besoin d'utiliser ses quatre moteurs pour se déplacer afin de lutter contre une attaque adverse typique.

Afin de satisfaire la contrainte FC1 et ne pas sortir du dohyo en dérapant, le robot doit satisfaire l'inégalité mécanique suivante, issue de la projection d'un principe fondamental de la dynamique :

$$n_{roues} \cdot K_{pertes} \cdot \frac{U \cdot I}{v} + \mu_f \cdot m \cdot g \geq 4\, K_{pertes} \cdot \frac{U \cdot I}{v}$$

Si nous nous limitons à trois roues, l'égalité précédente devient alors

$$m \geq K_{pertes} \cdot \frac{U \cdot I}{\mu_f \cdot v \cdot g} \geq 0.2 \cdot \frac{11.1 \cdot 0.52}{0.6 \cdot 8 \cdot 10^{-2} \cdot 9.81} = 2.45\, kg$$

NB : la valeur de K_{pertes}, modélisant la relation entre la puissance électrique du moteur et sa puissance mécanique, a volontairement été sous-estimée car nous ne disposions pas de sa valeur exacte

Afin de satisfaire la contrainte FC4, nous avons imposé à notre structure de comporter quatre roues motrices.

La question de la disposition optimale des roues sur le robot s'est alors posée, et plusieurs choix s'offraient à nous :

— placer les roues de part et d'autre d'un "corps" rigide à la manière d'une voiture ;
— placer les roues à l'arrière du corps comme une brouette ;
— remplacer les roues par des hélices nucléaires qui explosent – mais malheureusement il n'y avait pas un tel dispositif dans le boîte fournie par l'école (il faudrait y penser pour les années futures) ;
— disposer les roues le long des "bras".

Nous avons d'abord réalisé que la disposition en brouette imposerait d'éviter les frottements avec le sol par une structure les diminuant, comme une bille ou un patin glissant, structure que nous n'étions pas capables de réaliser avec le contenu de la boîte. De plus, une telle structure serait assez fine pour limiter la surface de contact, ce qui fragiliserait l'équilibre et la solidité du robot.

Le placement de roues sur les bras (une roue par bras et les deux dernières vers l'arrière du robot) nous a longtemps occupé, mais nous avons fini par abandonner l'idée en raison des problèmes de solidité de la structure qui relierait les moteurs latéraux aux roues. En effet, nous ne disposions pas de données suffisantes pour quantifier la résistance en déformation du bras et nous avons donc abandonné cette piste de façon préventive en vue de la contrainte FC3.

Enfin, si l'idée d'un robot volant à l'hélium était à la fois amusante et inventive (la limite verticale du dohyo n'ayant jamais été établie et l'exigence FC4 précisant explicitement que seul le poids effectif du robot compte en cas d'égalité), elle était irréalisable avec nos moyens.

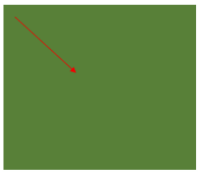
Nous avons finalement opté pour la solution de type "voiture" qui, en plus de présenter une structure compacte nécessitant peu de pièces, favorise la solidité et la compacité globale du robot.

Enfin nous avons cherché à satisfaire les exigences FP2 et FP3 le plus simplement possible afin d'éviter d'alourdir inutilement le robot en l'encombrant. Diverses techniques d'attaques (pelle, rampe ou plaque pour faire glisser l'adversaire) et de défense (leurres ou simulation de bordures de terrain) nous sont venues assez rapidement, mais nous nous sommes orientés vers une structure de bras qui apportait une solution légère mêlant défense et attaque : la majeure partie de nos efforts de conception puis d'assemblage s'est concentrée sur cette structure. En effet, moins lourds qu'une pelle, moins encombrants qu'une rampe et moins fragiles qu'une plaque, les bras nous permettaient en théorie de nous orienter vers l'adversaire par effet de levier.

2.2 Stratégie de déplacement

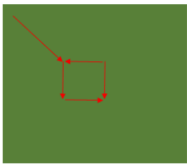
Le robot étant en totale autonomie sur le dohyo, il fallait lui élaborer une stratégie de combat qui soit à la fois efficace contre un maximum d'adversaire et en adéquation avec les paramètres mécaniques et techniques du robot. Sachant que le robot sumo allait pouvoir adapter sa stratégie uniquement lorsqu'il aurait détecté le robot adverse à l'aide de ses capteurs, il fallait trouver une stratégie qui permettait de repérer l'adversaire dans les meilleures conditions. En effet, une fois que le robot ennemi a été détecté, le robot sumo doit entrer dans une autre phase du programme pour qu'il se prépare au combat.

Stratégie naïve : la première stratégie que nous avons ébauchée était simplement de se diriger vers le centre du carré en espérant que l'adversaire adopte une stratégie similaire et que l'on pourrait alors le détecter vers le centre du dohyo.

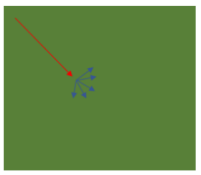


Cependant, une telle stratégie se doit d'être complétée au cas où l'adversaire ne se trouverait pas au centre du dohyo. De ce fait, nous avons imaginé deux suites à cette stratégie :

Stratégie carré : la première était de parcourir un carré autour du centre du repère en espérant découvrir le robot ennemi lors de ce parcours ;

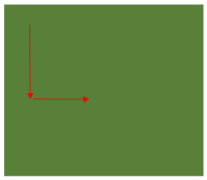


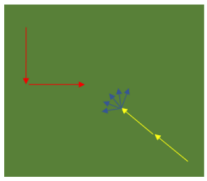
Stratégie de balayage : la seconde était de s'arrêter, de faire de petites rotations du robot de sorte à balayer un cône devant le robot en supposant que le robot adverse ne puisse pas se trouver alors derrière nous.



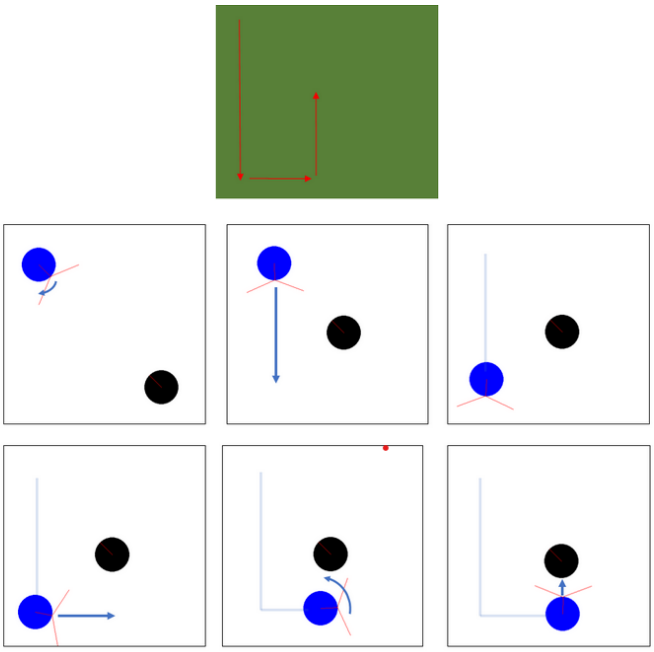
Cependant, il existe d'autres stratégies permettant d'aller vers le centre du terrain.

Stratégie dite en L : une d'entre elles serait non pas de suivre la diagonale mais de suivre un bord puis d'effectuer une rotation d'un angle de 90° puis d'avancer à nouveau tout droit afin d'atteindre le centre. Cette stratégie de déplacement pourrait contrer la première que nous avions ébauché. Ainsi, pour tenter de contrer la situation où l'adversaire se déplacerait selon "un L" et nous en diagonale, nous avons pensé à nous déplacer en diagonale mais plus lentement afin que l'adversaire se trouve déjà au centre du terrain lorsque nous arriverions pour pouvoir ensuite enchaîner sur la phase d'assaut.





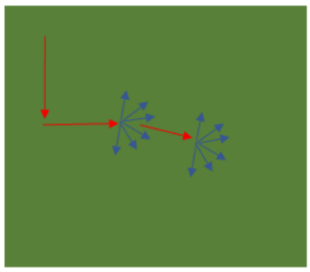
Stratégie dite en J : une autre stratégie envisageable serait de longer un des bords du carré, d'effectuer une rotation de 90°, de revenir vers l'intérieur du carré, toujours en longeant le bord puis de faire une dernière rotation à 90° et enfin de revenir vers le centre pour tenter de prendre l'adversaire par derrière. Nous appellerons plus tard cette stratégie la stratégie "en J".



En effet, la géométrie de notre robot sumo fait qu'il est très vulnérable s'il se fait prendre par surprise et il

faut donc à tout prix tenter d'éviter une telle situation.

Après avoir envisagé toutes ces options, nous avons trouvé qu'une approche "en L" était celle qui était la plus adaptée à notre robot car les bras dont dispose celui-ci lui offraient un meilleur angle d'attaque. Ainsi, il nous fallait encore déterminer notre stratégie de recherche entre le balayage sur place ou le fait de parcourir un carré. Pour faire ce choix entre autres, nous nous sommes appuyés sur une simulation de combat qui sera détaillée dans le paragraphe 4.3. Nous avons alors réalisé que parcourir un carré autour du centre du terrain rendait notre robot trop vulnérable car on perdait alors notre avantage que sont nos bras à l'avant en exposant les flancs et l'arrière de notre robot. Nous avons donc opté pour une stratégie de recherche avec un balayage puis une phase où l'on avance si jamais nous ne trouvions pas notre adversaire.



Cette stratégie ayant été établie, il nous fallait encore jouer sur certains paramètres afin d'avoir une chance de victoire plus élevée dans tous les scénarii possibles. Les paramètres sur lesquels nous pouvions jouer étaient : la vitesse des différentes rotations du robot, la longueur parcourue sur chaque "branche du L", l'angle de rotation lors de la phase de recherche et la distance à parcourir après une phase de recherche infructueuse.

Nous nous sommes appuyés sur la simulation en tentant d'optimiser les paramètres afin de se trouver dans les meilleures conditions pour combattre. Tout d'abord il s'est avéré qu'un angle de balayage total de 180° en utilisant notre vitesse de rotation maximale était la meilleure pour ne pas se faire surprendre et détecter l'ennemi dans un maximum de situations. De plus, ces simulations nous ont permis d'optimiser la durée lors de laquelle il fallait faire tourner les moteurs lors des phases de translation. Nous avons donc choisi une durée de 2640 ms pour la première translation puis une durée de 2350 ms pour la deuxième. Enfin, elles nous ont également fourni le tableau suivant *(les meilleurs choix sont en gras dans le tableau)* :

Stratégie alliée \ Stratégie adverse	Tout droit	Tout droit lent	J	J inversé	L	L inversé
L (balayage gauche puis droite)	Victoire	Victoire	**Victoire**	Défaite	Victoire	Victoire
L (balayage droite puis gauche)	Victoire	Victoire	Victoire	Nul : dépend de l'adversaire	**Victoire**	Victoire

TABLE 8 – Confrontation des stratégies

Ce tableau était censé nous informer sur quelle stratégie était optimale entre faire une recherche dont le balayage commençait par la gauche puis vers la droite ou dont le balayage commençait vers la droite puis vers la gauche. Cependant, on voit que ces deux stratégies semblent relativement équivalentes à l'exception de la stratégie adverse qui serait celle du J inversé.

Cependant, nous avons également remarqué grâce aux simulations que si notre balayage commençait vers la droite, alors notre robot n'allait pas observer le coin opposé du dohyo dans le temps imparti. Ainsi, si le robot adverse ne bouge pas du tout, nous n'aurions pas été mesure de le trouver et donc nous nous serions retrouvé dans une situation d'égalité. Or, nous souhaitons éviter au maximum ce cas de figure car nous n'avons pas construit un robot très léger pour faire face à cette éventualité. Donc, pour répondre au mieux à notre objectif, il fallait absolument être en mesure de vérifier ce qu'il se trouve dans le coin opposé à celui de notre départ si l'on ne trouve pas le robot adverse.

C'est pourquoi nous avons donc opté pour la stratégie d'un balayage vers la gauche puis vers la droite pour notre phase de recherche.

3 Nos choix

3.1 Architecture mécanique

Nous avons finalement choisi une architecture en forme de "crabe" : un élément central, tel une carapace, où se situe l'unité de commande, surmontant les quatre moteurs et les quatre roues. A l'avant se trouve le capteur, ainsi que deux "bras", un de chaque côté, aboutissant chacun à une "pince". Sur chaque "bras" on trouve, presque à sa base une sorte de protubérance. A l'arrière, est placé une "queue" plate surmontée d'un porte-drapeau.

Voici une photo de notre robot pour mieux visualiser cette architecture.

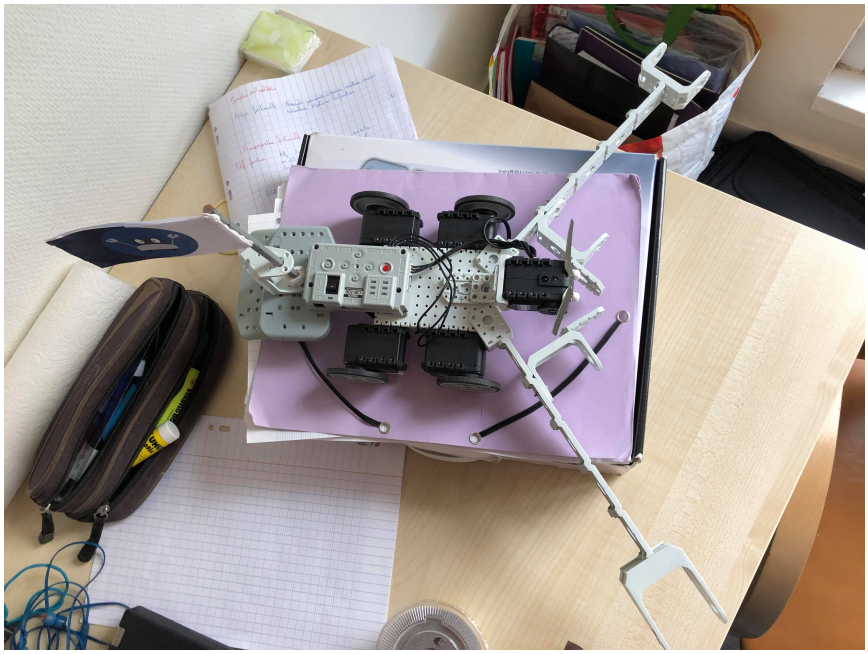


FIGURE 1 – Architecture finale du robot

Il est légitime de se demander pourquoi un tel choix d'architecture, mais rien n'a été laissé au hasard. Les roues ont été placées dans cette position pour avoir une structure la plus proche du sol possible. Il est à noter qu'il n'y a aucun élément en hauteur, à l'exception du drapeau mais qui est de masse négligeable par rapport aux autres éléments. Ainsi le centre de gravité de la structure s'en est trouvé être le plus bas possible, ce qui procure une grande stabilité.

Nous avons choisi de mettre des bras au robot pour réaliser un "suivi" de l'adversaire. En effet, en cas de rencontre entre l'adversaire et un de nos bras, notre robot pivoterait jusqu'à se trouver un face du robot ennemi pour pouvoir mieux le pousser vers l'extérieur. De plus, pousser le robot adverse lorsque celui-ci est encore de côté nous avantage. Nous avons constaté expérimentalement que le robot se réorientait effectivement vers sa cible s'il l'attaquait en diagonale grâce à la seule force de poussée de nos moteurs. Cette technique de dérapage est également valable en cas de défense, car le dérapage nous place en face de l'adversaire ce qui nous permet de réduire à moins de 20° (valeur expérimentale) l'angle formé par les deux robots, nous assurant ainsi une efficacité de poussée d'au moins $cos(\ 20°\ ) = 94\ \%$ de notre force maximale.

Le bloc sensor (Robotis AX-S1) possède des capteurs sur trois de ses faces. Il fallait au moins un capteur pointant vers le sol pour détecter les limites du dohyo et un autre pointant devant nous pour détecter un potentiel robot en face. Ceci limite déjà fortement l'orientation possible des capteurs. De plus, il ne fallait pas qu'il sature, ce qui ne laissait plus qu'une position possible : pointant vers l'avant.

Les protubérances sur les bras ont pour objectif de déjouer une stratégie adverse, surnommée "la jupe blanche". C'est-à-dire que si un robot possédait un contour blanc autour au ras du sol autour de lui, par

exemple, celui pourrait faire croire à notre robot qu'il vient de rencontrer les limites du dohyo. Ainsi ces protubérances sont là pour empêcher que cette fameuse "jupe blanche" puisse accéder jusqu'à notre capteur. Bien sûr, ceci fonctionne si le robot adverse est suffisamment haut et si la jupe n'est pas trop longue.

La queue à l'arrière sert de contre-poids. En effet, comme tous les éléments sont situés à l'avant du robot, celui-ci penchait légèrement, ce qui diminuait le contact avec le sol des roues arrières et donc la puissance disponible. Ainsi l'ensemble queue et porte drapeau sont là pour rajouter du poids et rééquilibrer le tout.

3.2 Stratégie de déplacement

Finalement, nous avons pu élaborer grâce à nos simulations et à nos tests, l'algorithme final. Son principe est détaillé sur la figure 2.

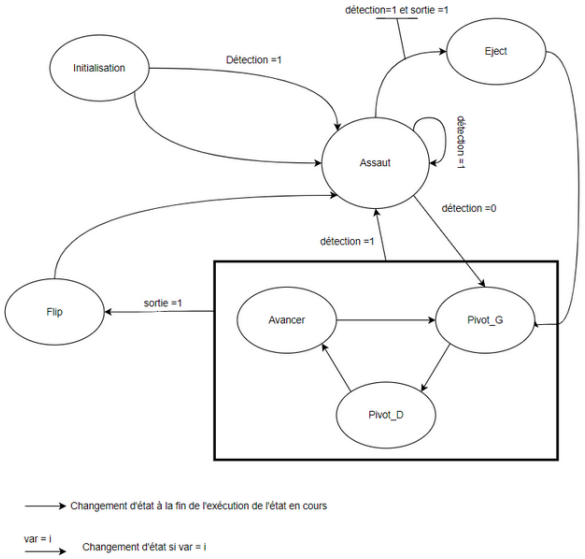


FIGURE 2 – Diagramme d'état du robot simplifié

Puis en détaillant la phase d'initialisation, qui comprend le temps d'attente de trois secondes exigé par le cahier des charges et la stratégie initiale "en L" adoptée on obtient la figure 3 :

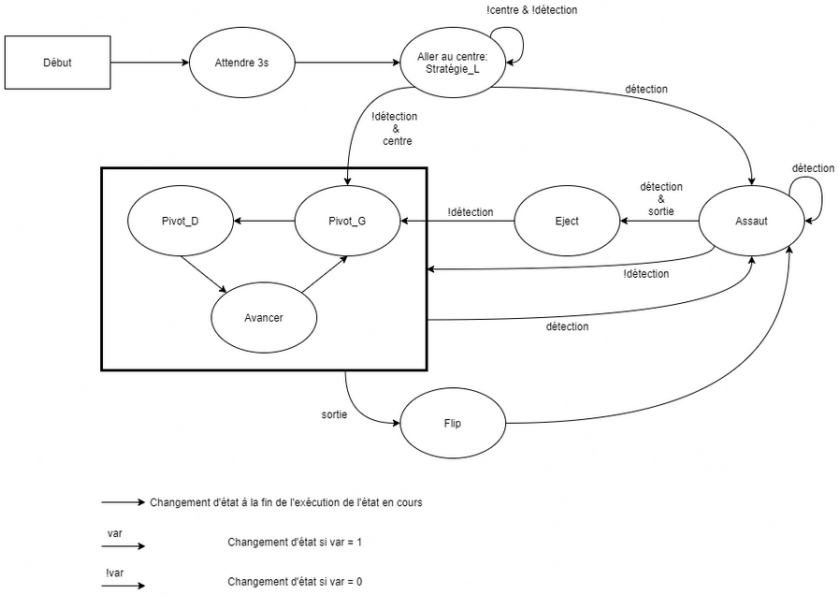


FIGURE 3 – Diagramme d'état avec la phase d'initialisation détaillée

Attendre 3s : le robot ne démarrera pas avant 3s. La musique démarre.

Aller au centre, stratégie en L : le robot positionné face à la diagonale centrale, se tourne colinéairement au bord du dohyo choisi. Puis se déplace jusqu'au milieu du bord, puis effectue à nouveau une rotation vers le centre puis avance jusqu'au centre (figure 4).

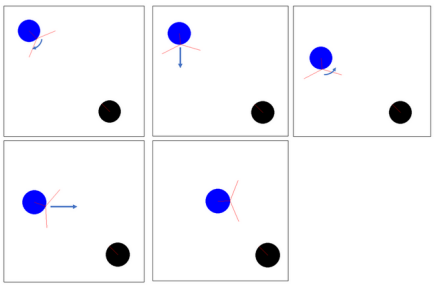


FIGURE 4 – Captures de la simulation de la phase de la stratégie finale (en L) sans détection

Ensuite vient la phase de recherche :

Phase de recherche (pivot_D ; pivot_G ; avancer) :

— pivot_G : le robot effectue une rotation de 90° dans le sens trigonométrique ;
— pivot_D : le robot effectue une rotation de 160° dans le sens horaire ;
— avancer : le robot avance en ligne droite sur 30cm.

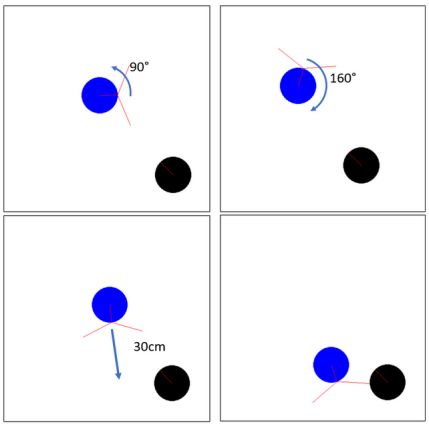


FIGURE 5 – Phase de recherche

Assaut : On se dirige vers le robot adverse tant que celui-ci est détecté. La vitesse à laquelle le robot se déplace est également augmentée.

Eject : On avance/pousse le robot adverse d'une certaine distance afin de s'assurer que notre robot reste dans le dohyo. Puis on recule, on fait demi-tour dans le sens horaire et on effectue la phase de recherche.

Flip : On recule, on fait demi-tour dans le sens direct et on entre en phase recherche.

Phase / Vitesse des roues lors de :	Initialisation	Recherche	Eject	Assaut	Flip
La translation (cm/s)	9	16	9	16	9
La rotation (rad/s)	1.14	1.94	1.14	-	1.14

TABLE 9 – Tableau des vitesses lors de la translation et de la rotation

4 Détails de la réalisation

4.1 Code

Pour simplifier les échanges de code, nous avons mis en place un dépôt git (passé public depuis la fin de la compétition) disponible à l'adresse suivante : `https://github.com/gpotter2/sumo-meca`. L'historique du code est disponible dessus.

Nous sommes partis de l'exemple du code fourni avec le SDK (Software Developement Kit) que nous avons découpé.

L'état du robot est stocké dans une variable. À chaque état possible est attribué une fonction qui prend en paramètre un pointeur vers cet état.

La fonction main initialise les horloges, les moteurs et place cet état sur INIT. Commence alors une boucle infinie qui démarre la fonction correspondant à l'état actuel. Pour changer d'état, une fonction doit simplement changer l'état en utilisant le pointeur puis retourner. Cette méthode permet de pouvoir accéder à n'importe quel état dans n'importe quel ordre tout en conservant un code propre (pas tout dans un seul main...).

Le contenu de notre main est alors extrêmement simple à lire :

```
int state = INIT;
while(state != STOP)
{
  if(state == INIT){
    initSequence(&state);
  }
  if(state == SEEKING){
    seekSequence(&state);
  }
  if(state == CHASING){
    chaseSequence(&state);
  }
  if(state == FLIP){
    flipSequence(&state);
  }
}
```

Nous avons implémenté de nombreux utilitaires sous forme de fonctions afin de jamais avoir à répéter du code. On trouve par exemple :

- `spin(int speedright, int speedleft)` : permet de tourner
- `forward(int speed)` : permet d'avancer
- `detectWhiteBorder()` : retourne 1 s'il y a détection de la bande blanche
- etc...

Notre robot doit être capable d'effectuer des stratégies (= successions d'actions préprogrammées) tout en restant attentif à son environnement. Ne disposant pas de capteurs sur les roues, les stratégies sont implémentées avec des timers : on attend un certain nombre de secondes pendant que les roues tournent dans une direction. Le problème est alors que ces timers (`mDelay()`) sont bloquants : il nous a donc fallu modifier cette fonction pour y ajouter la détection des bords, afin de pouvoir interrompre le timer si besoin.

Finalement, les fonctions d'état pouvaient alors être implémentées assez simplement. Voici par exemple la fonction (commentée) de l'état `FLIP`, qui se déclenche lorsqu'une bande blanche est détectée :

```
void flipSequence(int* state){
  while (*state == FLIP){
    forward(-speed_ini); // Reculer
    mDelayMusic(1000); // Pendant une seconde
    *state = SEEKING; // Et se remettre à chercher le robot
    spin(-speed_ini, -speed_ini);  // Tout en effectuant un demi-tour
    mDelayMusicLogic(4000, state);  // pendant 4 secondes
    if (*state != FLIP) // mDelayMusicLogic a peut-être détecté la ligne
      break;
    // la ligne a été détectée: on avance une seconde (pour pousser le robot)
    // puis on recommence à reculer
    forward(speed_ini);
    mDelayMusicLogic(1000, state);
  }
}
```

Implémentation de la musique. Une particularité de notre robot était qu'il soit capable de jouer de la musique en parallèle de l'exécution (et non seulement pendant les trois premières secondes). Il a fallu pour cela implémenter une **horloge** ainsi qu'une fonction de gestion des notes, qui était appelée par toutes les boucles bloquantes et qui jouait les notes lorsque l'horloge atteignait certains instants.

Pour les détails techniques, nous avons modifié `__ISR_DELAY()` qui est appelée à chaque milliseconde pour incrémenter une variable d'horloge, et également modifié `mDelay` pour utiliser cette horloge (plutôt que d'utiliser un nombre décroissant de millisecondes restantes). Nous avons également du découper les fonctions de buzzer pour pouvoir démarrer et éteindre des notes indépendamment.

4.2 Les tests

Grâce à notre implémentation fonctionelle, il est très simple de tester les différentes parties du code. Il suffisait d'insérer un morceau de code bloquant dans la phase d'initialisation. Cela nous a permi de :

— tester les parties du code ;
— mesurer la vitesse de déplacement du robot et ainsi régler les timers.

Voici des exemples de tests que nous avons effectués, ainsi que leurs résultats :

Vitesse linéaire maximale. Mesure de la vitesse linéaire maximale du robot

```
  // La documentation donnée indiquait qu'il était possible d'augmenter encore la vitesse
  // par rapport aux exemples donnés.
  #define speed_very_max 0x3ff
  forward(speed_very_max);
  while(1){}
```

Résultat : 9.47cm/s

Vitesse de rotation maximale. Mesure de la vitesse maximale de rotation du robot

```
  spin(speed_very_max, speed_very_max);
  while(1){}
```

Résultat : $65°/\text{s}$

Seuil de détection de la bande blanche. Mesure le seuil de détection de la bande blanche

```
  // test du capteur principal, orienté vers le bas
  unsigned char field;
  while(1){
    centerInfraRed(SENSOR, &field);
    TxDString("InfraRed center: ");
    TxDByte16(field);
    TxDString("\n");
  }
```

Résultat : détecter la bande blanche fait saturer la capteur infrarouge. On place donc le seuil à 255

Test musical. Vérifier que les fonctions de musique / l'horloge fonctionnent.

```
while(1){
  musicHandler();
}
```

4.3 Simulation

Motivation de la simulation. Du fait de la taille importante du groupe et de sa division en sous-sections (mécanique et programmation), la nécessité d'une simulation visuelle s'est faite ressentir pour permettre à tous d'avoir un support intuitif pour élaborer des stratégies de recherche. De plus, il devenait possible de confronter différentes stratégies de recherche (foncer tout droit, décrire un L, décrire un J,...) et d'évaluer leurs forces et faiblesses les unes face aux autres, sans avoir à manipuler le robot directement.

Spécifications de la simulation. L'objectif de la simulation était d'avoir un outil où les concepteurs de stratégie ont une partie limitée du code à modifier. Voici quelques caractéristiques de la simulation :

— les robots sont représentés par un cercle dont un des rayons est tracé en rouge pour indiquer le placement du capteur et de la face avant du robot ;

FIGURE 6 – Représentation d'un robot dans la simulation

— les robots peuvent se déplacer en translation dans la direction du capteur ou en rotation ;
— le capteur détecte l'adversaire à 20 cm devant lui et s'il est en dehors du terrain et renvoie une valeur binaire ;
— les robots ont accès à deux vitesses de translation et deux vitesses de rotation ;
— le terrain est représenté à l'échelle par l'écran (480 pixels pour 100cm) ;
— la simulation s'arrête quand les robots entrent en contact ou quand un robot sort du terrain ;
— diviser le code en une partie simulation et affichage et une partie stratégie ;
— à chaque pas de temps, les robots ont connaissance de leur vitesse de translation, de leur vitesse de rotation, des informations de son capteur, et de variables propres à la stratégie (mais pas celles propres à la simulation comme la position absolue du robot sur l'écran) ;
— à chaque pas de temps, les robots ne peuvent modifier que leur vitesse de translation, leur vitesse de rotation et les informations propres à leur stratégie.
— plusieurs paramètres sont modifiables :
  — rayon des robots ;
  — les 2 vitesses de translation ;
  — les 2 vitesses de rotation ;
  — le pas de temps de la simulation.

Choix des spécifications de la simulation. Voici certains choix qui ont été faits :

— La représentation du robot par un cercle a été choisie pour rendre plus aisé la gestion des collisions dans la simulation et car elle ne semblait pas diverger de la forme réelle du robot du point de vue du capteur infrarouge.
— La portée de 20 cm du capteur a été prise d'après les tests effectués sur la fiabilité du capteur infrarouge de l'AX-S1.

— La réponse de l'AX-S1 est binaire dans la simulation car nous l'utilisions de la même manière sur le robot, à l'aide d'une valeur de seuil (le capteur n'était pas considéré comme assez fiable pour être utilisé plus finement).

— Les deux vitesses de déplacements ont été choisies vendredi matin lorsque nous avons trouvé un moyen de faire tourner les moteurs à une vitesse supérieure à la vitesse maximale indiquée sur la fiche technique.

— L'arrêt de la simulation lors du contact des robots (c'est-à-dire sa limitation à la simulation de la recherche) est dû au manque de tests physiques sur le frottement des roues sur le dohyo et la méconnaissance de la géométrie du robot adverse. Ces facteurs inconnus nous ont empêché de prédire la position du robot après le premier contact, et donc auraient nécessité une seconde simulation plus complexe.

— Un code séparé en une partie simulation et en une partie stratégie évitait aux membres du groupe de programmation l'apprentissage de la simulation dans son ensemble et leur permettait de se concentrer sur l'élaboration de stratégies.

— La position de départ du robot a été estimée à 19 cm du coin du dohyo à l'aide de l'image de dohyo vert du fichier *cahierDesCharges.pdf* et n'était donc pas considérée comme un paramètre.

— Python a été choisi comme langage car connu de toute l'équipe et car le concepteur de la simulation maîtrisait la bibliothèque pygame pour réaliser l'interface graphique. De plus sa souplesse par rapport au C était appréciée.

Réalisation pratique *Les noms de variables sont indicés par 1 ou 2 dans le code selon le robot associé : 1 pour le robot allié et 2 pour le robot adverse.*

— Les deux robots sont repérés par leur coordonnées cartésiennes et par un angle qui indiquait la position du capteur du robot (stockés dans *posX*, *posY* et *orientation*).

— Les deux robots ont une vitesse de translation et une vitesse de rotation (stockées dans des variables *vit_trans* et *vit_rot*).

— Les robots se touchent quand la distance entre leurs centres $d \leq R_1 + R_2$, où R_i sont les rayons des cercles représentatifs.

— À chaque pas de temps, les robots avancent dans la direction de leur capteur de la valeur de leur vitesse de translation, et leurs orientations tournent de la valeur de leur vitesse de rotation.

— À chaque pas de temps, si l'extrémité du rayon rouge indiquant la position du capteur est en dehors du terrain la variable sortie passe à 1 (0 sinon).

— À chaque pas de temps, si les robots sont à distance de capteur, détection passe à 1 si la demi-droite définie par le capteur intersecte le cercle du robot adverse.

— À chaque pas de temps, les fonctions *strategie_1* et *strategie_2* sont appelées. *strategie_i* modifie la valeur de *vit_trans_i*, de *vit_rot_i* et de la liste *save_i* qui contient les variables propres à la stratégie que l'élaborateur à décider de sauvegarder. Les arguments d'entrée de ces fonctions sont : *vitesse_trans_i*, *vitesse_rot_i*, *sortie_i*, *detection_i* et *save_i*.

Précisions sur les variables propres à une stratégie. On appelle variable propre à une *strategie_i* à un instant t :

— les valeurs prises par *vitesse_trans_i* aux instants précédents t ;

— les valeurs prises par *vitesse_rot_i* aux instants précédents t ;

— les résultats de fonctions qui n'ont pour arguments que les deux tirets précédents.

Les élaborateurs peuvent conserver une information propre dans la liste *save_i* pour la réutiliser dans les itérations qui suivent. Cette façon alambiquée de conserver des variables simulent la programmation du robot : quand on programme le robot en C, nous n'avons accès qu'au variables des différents moteurs et du capteur et les variables que l'on définit à partir de ces dernières (ce qui est appelé variables propres à une stratégie dans la simulation).

Précision sur l'intersection entre la demi-droite issue du capteur et le cercle du robot adverse. On rappelle que l'intersection entre la demi-droite définie par le capteur et le cercle du robot adverse est censée représenter la détection du capteur infrarouge d'un objet.

Ainsi on ne cherchera à déterminer s'il y a une intersection que si les deux cercles sont à portée de capteur l'un de l'autre.

On ajoute aussi que cette fonction de détection d'intersection appartient à la partie simulation et non à la

partie stratégie. De ce fait on s'autorise l'accès à toutes les variables du programme.

Il faut donc prouver l'existence de (x,y) tel que :

$$\begin{cases} a'y + b'x + c' &= 0 \\ (y - y_2)^2 + (x - x_2)^2 &= R_2^2 \end{cases}$$

En réécrivant le système de manière équivalente :

$$\begin{cases} a(y - y_2) + b(x - x_2) + c &= 0 \\ (y - y_2)^2 + (x - x_2)^2 &= R_2^2 \end{cases}$$

L'existence d'une intersection devient équivalente à :

$$\begin{cases} a^2(R_2^2(a^2 + b^2) - c^2) &\geq 0 \\ b^2(R_2^2(a^2 + b^2) - c^2) &\geq 0 \end{cases}$$

Qui est équivalent à :

$$R_2^2(a^2 + b^2) - c^2 \geq 0$$

Après avoir vérifié que le robot adverse est croisé par la droite du capteur, on vérifie qu'il est du bon côté (en effet la droite du capteur s'étend à l'arrière du robot). Pour cela on considère le centre de notre robot comme l'origine d'un repère orthonormé. Si le robot adverse est détecté à l'avant du robot, alors son centre est dans le même quadrant que l'avant du robot. Cela se traduit par :

$$\begin{cases} (x_2 - x_1)\cos(\theta) &\geq 0 \\ (y_2 - y_1)\sin(\theta) &\geq 0 \end{cases}$$

5 Management

5.1 Structure du groupe

Initialement, nous voulions que chaque membre du groupe soit impliqué de manière égale dans le projet et qu'il puisse travailler sur la thématique qu'il maîtrise le mieux. Pour cela, nous l'avons divisé en plusieurs sections :

- la section "méca" chargée de la construction du corps ;
- la section "tests" chargée de concevoir et réaliser les tests unitaires et globaux ;
- la section "rédac" chargée de rédiger les différents rapports ;
- et la section "programmation" chargée de programmer le robot.

Il est rapidement apparu que la présence d'une section "test" autonome ralentirait le processus de conception : les testeurs ne pourraient concevoir les tests unitaires des composants mécaniques et électroniques du robot, nécessaires à la vérification des exigences, qu'une fois l'architecture du robot établie.

De même pour tester la structure du robot ainsi que la faisabilité de la géométrie envisagée, il est nécessaire de le construire (ou du moins d'essayer en cas de non-faisabilité de la géométrie envisagée).

Les tests mécaniques et logiciels se faisant en fonction des avancées des équipes de construction et de programmation, il a été décidé que la section "méca" s'occuperait des tests mécaniques et que la section programmation s'occuperait des tests des capteurs, et que nous ferions les tests globaux tous ensemble. En conséquence nous avons donc dissous la section "tests".

Parallèlement nous nous sommes rendus compte que même si la section programmation était en mesure d'implémenter une stratégie donnée pour le robot, il nous manquait la conception de ladite stratégie. C'est

pourquoi nous avons fondé une section "stratégie" chargée de déterminer la stratégie à implémenter.

Finalement, il est aussi apparu que chaque section était la plus à même de parler de ce dont elle se chargait, donc l'existence de la section "rédac" n'avait pas grand sens non plus. Ainsi, il a été décidé que chaque section se chargerait de la rédaction des points dont elle avait la charge.

Ces changements ont été opérés mardi après-midi.

Pour finir, nous nous sommes aussi rendus compte que les différentes sections ne pouvaient pas être hermétiques. Beaucoup d'échanges ont été effectivement nécessaires entre les diverses sections, car elles s'appuyaient toutes sur le travail des autres.

5.2 Spécificités individuelles

Une fois ces soucis structurels réglés nous n'étions pas au bout de nos peines : selon leur présence sur le campus, ou non, les différents membres du groupe ne pouvaient pas avoir le même degré d'implication, à cause de la communication moins spontanée qu'en présentiel, et du fait que certaines étapes (comme les tests) nécessitaient d'être présent. Bien entendu, les obligations et la motivation influençaient aussi l'implication de chacun.

5.3 Problèmes inhérents à ce projet en particulier et à la nature d'un projet

Mais le plus gros frein à l'égale implication de chacun fut la taille du groupe : même à 9 il n'y aurait pas eu assez de travail pour tout le monde pour une semaine, alors à 11 ce fut encore pire.

Un autre élément qui nous a rendu la tâche plus ardue fut la différence de vision de la gestion de projet des différents membres du groupe : par exemple, quand certains voulaient concevoir un robot jusqu'à un point très avancé uniquement de façon théorique puis revoir la copie en fonction des considérations techniques, d'autres voulaient plutôt avancer avec les considérations techniques et résultats des tests comme guides, quitte à manquer de certains éléments théoriques. Cette différence de vision a eu un grande influence sur le projet, et a mis en lumière l'importance d'un manager dans une équipe afin de mettre tout le monde d'accord sur la direction à suivre.

Il y eut aussi divers imprévus "attendus" dans le planning : nous pensions avoir défini une structure et une stratégie pour notre robot assez tôt, mais au fur et à mesure des tests et des résultats de l'espionnage industriel en salle de test, nous avons dû revoir notre copie plusieurs fois. Par exemple, nous pensions utiliser les capteurs de luminosité pour détecter les bandes de délimitation du dohyo. Or, les tests ont révélé que ce serait impossible. En revanche ils ont aussi montré que ce serait possible avec les capteurs infrarouges, ce que nous ne soupçonnions pas. Ainsi nous avons dû revoir la partie "détection des lignes blanches".

6 Conclusion

Nous étions satisfaits de notre robot, il répondait aux exigences et contraintes précédemment définies. Il a bien réalisé les manoeuvres décrites dans notre stratégie.

Un des problèmes que nous avons rencontré lors des différents tests avec la simulation est que la taille initiale que nous avions envisagée était nettement inférieure à sa taille réelle. De plus, de prime abord, nous avions schématiquement représenté les robots par des cercles mais cela s'est vite révélé être une limite de notre simulation car on ne prenait alors pas en considération l'avantage que représentaient les bras de notre robot. Il a donc fallu adapter la représentation du robot dans la simulation pour essayer de mieux visualiser la stratégie de combat. Par ailleurs, a posteriori, nous n'avions pas pris en compte la forme que pourrait avoir le robot ennemi. En effet, les robots ennemis avaient souvent une taille bien plus imposante que ce que nous avions prévu. Cette faiblesse de l'étude nous a d'ailleurs coûté cher lors des combats réels.

Le plus difficile a été de gérer la répartition du travail dans le groupe et de se mettre d'accord à plusieurs. Il y a peu de chance pour que nous ayons tous la même vision d'un problème et donc de la solution à adopter. On ne peut pas, non plus, être d'accord avec tous les choix mais il faut travailler en équipe et avancer dans le projet. Ainsi, ce genre d'exercice nous apprend à prendre du recul et à nous mettre en retrait quelques fois, quand il faut faire passer l'avis général avant le sien. Ce qui est quelque chose d'indispensable dans le monde professionnel.

L'Ingénierie Système permet d'avoir une ligne de conduite tout au long de la conception et de la mise en oeuvre du système. De plus, cette ligne de conduite est construite de façon à ce que l'avancement suive un ordre très précis permettant de toujours s'assurer qu'on ne dévie pas des objectifs initiaux (répondre aux besoins

du client). En effet, les exigences et contraintes à satisfaire ainsi que leur analyse, pour aboutir aux meilleures solutions techniques possibles, sont au coeur de cette démarche d'ingénierie système.

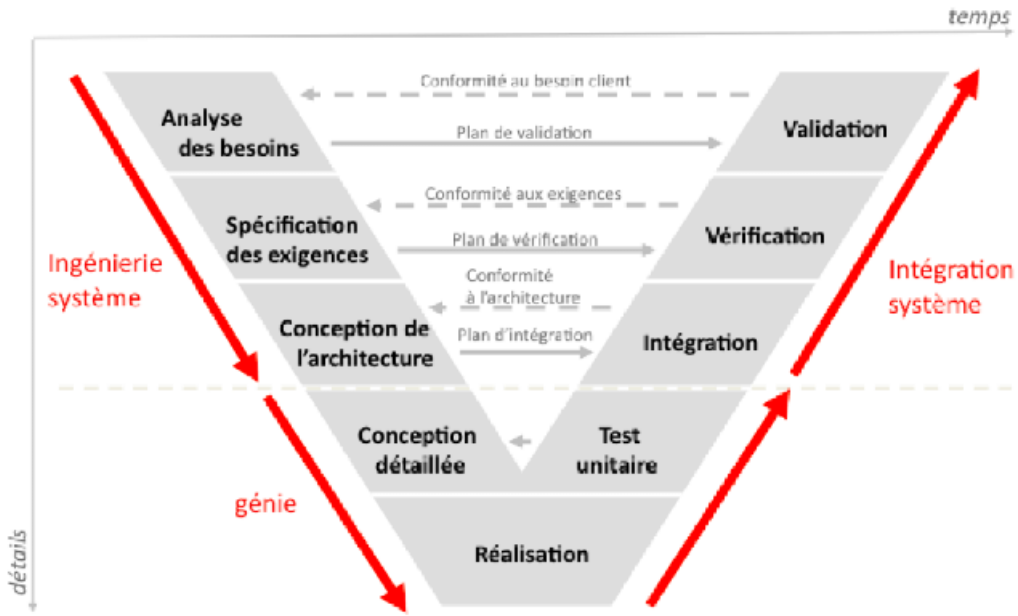


FIGURE 7 – Cycle en V

7 Annexe

7.1 Vidéos

- Test du code de musique : https://www.youtube.com/watch?v=HsWfcnilrF4
- Combat 1v1 : https://www.youtube.com/watch?v=AA8Quw4eTsI
- Simulation : https://www.youtube.com/watch?v=cJpirU2b9k8

7.2 Inventaire

L'inventaire final de la boîte "Bioloid Beginner Kit" a révélé l'absence de deux pièces :

- 1 pièce F3 ;
- 1 pièce WA.

7.3 Code source

Le code est assez long, alors pour ne pas alourdir le document, il est disponible ici : https://github.com/gpotter2/sumo-meca/blob/master/APP/src/main.c

7.4 Code simulation

Ce code étant moins long, le voici :

```
import pygame as pg
import math as mt

pi=mt.pi
sin=mt.sin
cos=mt.cos
tan=mt.tan
sqrt=mt.sqrt



Blanc=(255,255,255)
Noir=(0,0,0)
Bleu=(0,0,255)
Rouge=(255,0,0)

###paramètres de la sim
pas_de_temps=100#en ms
vitesse_trans= 9.47*4.8*pas_de_temps*0.001#en pixel par pas_de_temps (pensez que 480 pixels
    =100cm)
vitesse_rot = 2*pi/5.5*pas_de_temps*0.001#en radian par pas_de_temps
portée_capteur=20*4.8
###fin des paramètres



def f(x,y,theta,R):
  return (round(R*cos(theta)+x),round(R*sin(theta)+y))

def distance(p1,p2):
  x=p1[0]
  y=p1[1]
  w=p2[0]
  z=p2[1]

  return sqrt((x-w)**2+(y-z)**2)

def param_droite(x,y,theta):
  return (cos(theta),-sin(theta),sin(theta)*x-cos(theta)*y)


def inter_cercle_droite(R,a,b,c,x1,y1):

  c_bis=c+a*y1+b*x1
  delta=R**2*(b**2+a**2)-c_bis**2

  return delta>=0
```

```
47
48 #####Les stratégies à étudier
49 ## V_t est la vitesse de translation actuelle dans la direction du capteur;
50 ## V_r est la vitesse de rotation actuelle (orienté sens horaire)
51 ##detection=1 si le capteur détecte l'autre robot dans sa portée, 0 sinon.
52 ##sortie =1 si le capteur est en dehors du terrain
53 ##save est un tableau où on peut stocker les variables de la stratégie
54 ##Les fonctions renvoient la nouvelle vitesse de translation et de rotation
55 ## et les variables propres à la sim stocker dans save.
56
57 def strategie1(V_t,V_r,detection,sortie,save):
58
59
60
61     if len(save)==0:
62       save=[0,0,0]
63
64
65     if save[2]==0:##s'aligner au coté
66       save[1]+=vitesse_rot
67       if save[1]>pi/4-vitesse_rot:
68         save[2]=1
69         save[1]=0
70       return 0,vitesse_rot,save
71
72     if save[2]==1:##avancer jusqu'à la moitié
73       if detection:
74         return 1.7*vitesse_trans,0,[0,0,4]
75
76       save[0]+=vitesse_trans
77
78       if save[0]>120-vitesse_trans:###moitié du terrain à partir du depart ; (92,92) est la
     position de départ du robot
79         save[2]=2
80         save[0]=0
81       return vitesse_trans,0,save
82
83     if save[2]==2:##pivoter vers le centre
84       if detection:
85         return 1.7*vitesse_trans,0,[0,0,4]
86
87       save[1]-=vitesse_rot
88       if save[1]<-pi/2+vitesse_rot:
89         save[2]=3
90         save[1]=0
91       return 0,-vitesse_rot,save
92
93
94     if save[2]==3:##avancer jusqu'au centre
95       if detection:
96         return 1.7*vitesse_trans,0,[0,0,4]
97
98       save[0]+=vitesse_trans
99       if save[0]>107-vitesse_trans:#idem a partir de l'ordonnée de depart
100        save[2]=4
101
102
103      if detection:
104        save[2]=4
105        save[0]=0
106
107
108      return vitesse_trans,0,save##on garde la mesure de la distance parcourue pour que dans
     la phase assault on passe tout de suite au balayage
109
110
111    if save[2]==4:##assault
112
113
114      if sortie:
115        return 0,0,[0,0,7]
116
117      if detection:
118        return 1.7*vitesse_trans,0,[0,0,4]
119
120
```

```
121
122       if save[0]<portée_capteur*3/2-1.7*vitesse_trans:
123         save[0]+=1.7*vitesse_trans
124         return 1.7*vitesse_trans,0,save
125
126
127       return 0,0,[0,0,5]
128
129
130
131
132     if save[2]==5:##balayage gauche
133       if detection:
134         return 0,0,[0,0,4]
135
136       if save[1]>-pi/2+vitesse_rot:
137         save[1]-=vitesse_rot
138
139       else:
140         save[2]=6
141         save[1]=0
142
143       return 0,-vitesse_rot,save
144
145
146     if save[2]==6:#balayage droite
147       if detection:
148         return 0,0,[0,0,4]
149
150       if save[1]<0.87*pi-vitesse_rot:######
151         save[1]+=vitesse_rot
152       else:
153         save[2]=4
154         save[1]=0
155
156       return 0,vitesse_rot,save
157
158     if save[2]==7:##sortie
159       if save[0]>-portée_capteur/2+vitesse_trans:
160         save[0]-=vitesse_trans
161         return -vitesse_trans,0,save
162
163       if save[1]>-pi+vitesse_rot:
164
165         if detection:
166           return vitesse_trans,0,[0,0,4]
167
168         save[1]-=vitesse_rot
169         return 0,-vitesse_rot,save
170
171
172
173
174
175
176       return 0,0,[0,0,5]
177
178
179
180 def strategie2(V_t,V_r,detection,sortie,save):
181   return 1.7*vitesse_trans,0,save
182
183
184 ##############
185 ##############
186 ##############
187
188
189 def sim(R1=41,R2=41):
190
191   pg.init()
192
193   screen = pg.display.set_mode((480,480)) #480=100cm
194   clock_prece= pg.time.get_ticks()
195   clock=clock_prece
196
```

```
continuer=True

posX1=round(130/sqrt(2))
posY1=round(130/sqrt(2))
vX1=0
vY1=0

posX2=480-round(130/sqrt(2))
posY2=480-round(130/sqrt(2))
vX2=0
vY2=0

or1=pi/4 #attention en python ça tourne dans le sens  horaire
or2=-3*pi/4

detection1=0
detection2=0
sortie1=0
sortie2=0

### save robot
save1=[]
save2=[]

#déclaration des vitesses

vit_tr_1=0
vit_tr_2=0
vit_rot_1=0
vit_rot_2=0

#condition de contact

pause=False
while(continuer):



  for event in pg.event.get():

    if event.type== pg.QUIT:
      continuer=False

    if event.type== pg.KEYDOWN:
      if event.key==pg.K_p:
        pause=not(pause)

  while(pause):

    for event in pg.event.get():

      if event.type== pg.QUIT:
        continuer=False


      if event.type== pg.KEYDOWN:
        if event.key==pg.K_p:
          pause=not(pause)




  if distance((posX1,posY1),(posX2,posY2))<=R1+R2:

    print("\ncontact\n")
    while(continuer):

      for event in pg.event.get():

        if event.type== pg.QUIT:
          continuer=False
```

```
    #condition de sortie
    if posX1+R1<0 or posX1-R1>480 or posY1+R1<0 or posY1-R1>480:


        print("\nR1 perd\n")
        while(continuer):

          for event in pg.event.get():

            if event.type== pg.QUIT:
              continuer=False


    if posX2+R2<0 or posX2-R2>480 or posY2+R2<0 or posY2-R2>480:


        print("\nR2 perd\n")
        while(continuer):

          for event in pg.event.get():

            if event.type== pg.QUIT:
              continuer=False


    ###detection de l'autre
    if distance((posX1,posY1),(posX2,posY2))<=R1+R2+portée_capteur:
      a1,b1,c1 = param_droite(posX1,posY1,or1)
      a2,b2,c2 = param_droite(posX2,posY2,or2)


      if(inter_cercle_droite(R2,a1,b1,c1,posX2,posY2) and (cos(or1)*(posX2-posX1)>=0 and sin(
    or1)*(posY2-posY1)>=0 ) ):
        detection1=1

      else:
        detection1=0

      if(inter_cercle_droite(R1,a2,b2,c2,posX1,posY1) and (cos(or2)*(posX1-posX2)>=0 and sin(
    or2)*(posY1-posY2)>=0 ) ):
        detection2=1

      else:
        detection2=0

    else:
        detection1=0
        detection2=0


    ##.

    ##detection de sortie
    if posX1+R1*cos(or1)<0 or posX1+R1*cos(or1)>480 or posY1+R1*sin(or1)<0 or posY1+R1*sin(or1
    )>480:

        sortie1=1
    else:
        sortie1=0


    if posX2+R2*cos(or2)<0 or posX2+R2*cos(or2)>480 or posY2+R2*sin(or2)<0 or posY2+R2*sin(or2
    )>480:

        sortie2=1
    else:
        sortie2=0
    ##



    clock=pg.time.get_ticks()

    if(clock-clock_prece>pas_de_temps):
      clock_prece=clock

```

```
345         vit_tr_1, vit_rot_1,save1= strategie1(vit_tr_1, vit_rot_1,detection1,sortie1,save1)
346         vit_tr_2, vit_rot_2,save2= strategie2(vit_tr_2, vit_rot_2,detection2,sortie2,save2)
347 
348         vX1=vit_tr_1*cos(or1)
349         vY1=vit_tr_1*sin(or1)
350         vX2=vit_tr_2*cos(or2)
351         vY2=vit_tr_2*sin(or2)
352 
353 
354         posX1+=vX1
355         posY1+=vY1
356         posX2+=vX2
357         posY2+=vY2
358 
359         or1+=vit_rot_1
360         or1=or1%(2*pi)
361 
362         or2+=vit_rot_2
363         or2=or2%(2*pi)
364 
365 
366 
367     screen.fill(Blanc)
368     pg.draw.circle(screen,Bleu,(round(posX1),round(posY1)) ,R1) ##robot circulaire de 10cm de
    diam
369     pg.draw.line(screen,Rouge,(round(posX1),round(posY1)), f(posX1,posY1,or1,R1))
370     pg.draw.line(screen,Rouge,(round(posX1)+R1*cos(or1),round(posY1)+R1*sin(or1)), f(posX1,
    posY1,or1+pi/4,96))
371     pg.draw.line(screen,Rouge,(round(posX1)+R1*cos(or1),round(posY1)+R1*sin(or1)), f(posX1,
    posY1,or1-pi/4,96))
372 
373 
374     pg.draw.circle(screen,Noir,(round(posX2),round(posY2)),R2) ##robot circulaire de 10cm de
    diam
375     pg.draw.line(screen,Rouge,(round(posX2),round(posY2)), f(posX2,posY2,or2,R2))
376     pg.display.flip()
377 
378 
379   pg.quit()
380 
381 sim()
```