

Faculté des Sciences et Ingénierie - Sorbonne Université

Master Informatique parcours ANDROIDE



P-ANDROIDE

Programmation de comportements pour le robot Pogobot-wheel

Rapport de projet

Réalisé par :

PINHO FERNANDES Enzo

DELLUC Mélanie

EL KHOUMSI Safia

BEN SALAH Adel

Encadrant :

BREDECHE Nicolas

Année 2024-25

Table des matières

Introduction	2
Présentation des Pogobots	3
Le Hardware	3
Le Software	5
Le simulateur Pogosim	5
Les comportements à fournir	6
Run and Tumble	6
Dispersion	8
Phototaxie	10
Suivi de leader	11
Déplacement en formation	15
Alignement face au mur	17
Retour sur le projet	18
Conclusion	19
Annexes	20
Liens utiles	20
Rappel du cahier des charges	20
Manuel de l'utilisateur	22
Installation des dépendances	22
Compilation	22
Utilisation de Pogosim	22
Charger un comportement sur Pogobot	23

Introduction

A l'occasion de l'UE P-ANDROIDE, nous avons choisi de nous intéresser au projet "Programmation de comportement pour le robot Pogobot-wheel", encadré par Nicolas Bredeche en lien avec l'équipe ISIR.

Ce projet s'inscrit dans le cadre du développement et de l'expérimentation de comportements collectifs sur des robots Pogobot, récemment conçus par l'ISIR. Ces robots compacts, mobiles et communicants, sont capables d'échanger localement des informations via infrarouge tout en se déplaçant de manière autonome.

L'objectif est de programmer ces robots, en langage C à l'aide d'une API dédiée, pour qu'ils accomplissent de manière distribuée différents tâches caractéristiques de la robotique collective. Tous les développements sont réalisés sur robots réels, dans l'arène multi-robots de l'ISIR. Le projet implique une adaptation du protocole de communication afin d'estimer la proximité entre robots, une composante nécessaire pour les comportements collaboratifs.

Une particularité à préciser du projet est que nous sommes le 1er groupe à se servir des Pogobots à roues ainsi que le nouveau simulateur associé. Nous avons donc joué le rôle actif de bêta-testeurs. En plus du développement fonctionnel, nous avons été amenées à identifier des limitations potentielles, suggérer des améliorations et fournir des retours à l'équipe de développement. Certaines de nos remarques ont été prises en compte et ont contribué à l'évolution du simulateur et de l'API durant la durée du projet.

Présentation des Pogobots

Le Hardware

Les Pogobots sont des petits robots mobiles spécialement conçus pour l'étude de comportements collectifs et de l'apprentissage social au sein d'essaims robotiques. D'un diamètre de 6cm et d'une architecture modulaire, chaque unité se compose de deux principaux modules : une tête (*head*) et d'un ventre (*belly*).

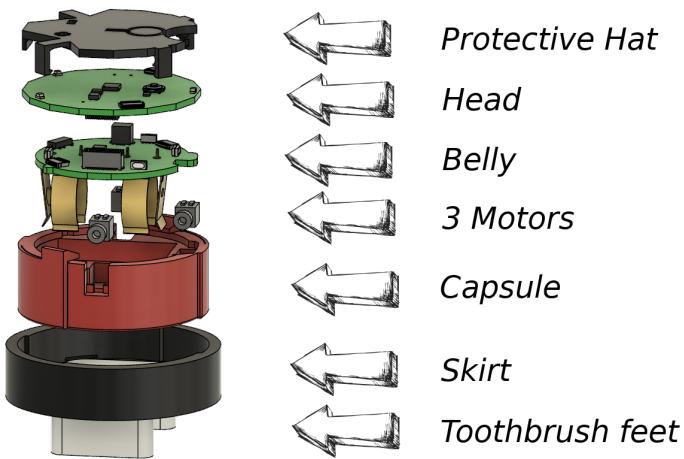


FIGURE 1 – Simplification de l'architecture d'un Pogobot

La tête intègre FPGA iCE40UP5K sur lequel est employé un softcore processor, assurant la gestion temps-réel des capteurs et de la communication. Un module inertiel (IMU) permettant de fournir des informations sur l'orientation et les accélérations subis, tandis que quatre émetteurs-récepteurs infrarouges à haute vitesse permettent l'échange d'informations. De plus, elle possède trois photorécepteurs (derrière, avant gauche et avant-droit) pour capter le niveau de lumière. Enfin, une LED frontale permettant de renseigner visuellement l'état interne du robot.

Le ventre, fixé sous la tête, abrite la partie actionneurs et énergie. On peut y trouver plusieurs LEDs sur les côtés, deux moteurs pouvant contrôler la puissance de la roue gauche, milieu et droite respectivement, et de la batterie.

Il faut savoir que le projet Pogobot est **open-hardware**. Vous pouvez consulter les modèles 3D et les fichiers KICAD sur github, les utiliser, les modifier, contribuer ou fork le projet à votre guise.

Lors du projet, nous avons eu accès à deux types de robots, les Pogobots à brosses et les Pogobots à roues.

La particularité des Pogobots à brosses est le mouvement par vibration de ces brosses sur la surface. Elles possèdent des brosses légèrement inclinées, qui en vibrant, provoqueront un mouvement. Il est assez complexe de les utiliser, notamment pour les comportements demandant une certaine précision des mouvements, pour cause de calibration et de facteurs extérieurs comme la poussière, la qualité des poils de brosses.

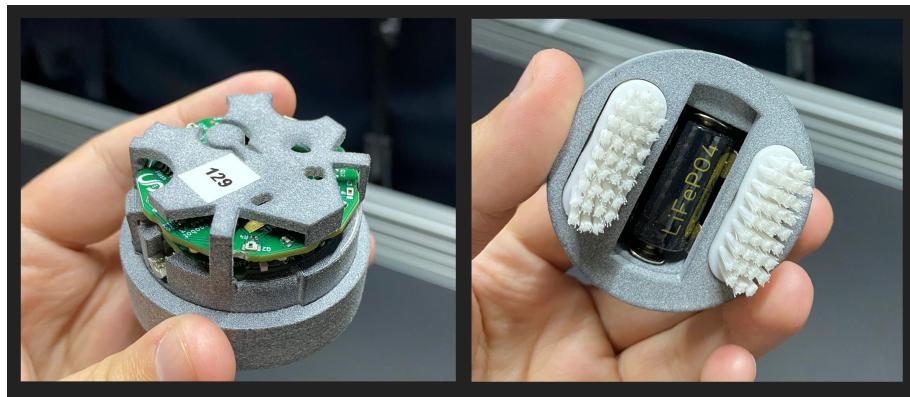


FIGURE 2 – Photos d'un Pogobot à brosses

Ensuite, nous avons eu l'accès aux Pogobots à roues, qui reste au cœur de notre projet. En plein développement, nous sommes les premiers à les essayer. Notre objectif était donc de fournir un set de comportements les utilisant, et de fournir nos observations et retours sur ces derniers, qui seront pris en compte dans leur amélioration future.

Nous avons accès à deux moteurs contrôlant la puissance et la direction des roues gauche et droite respectivement ici.

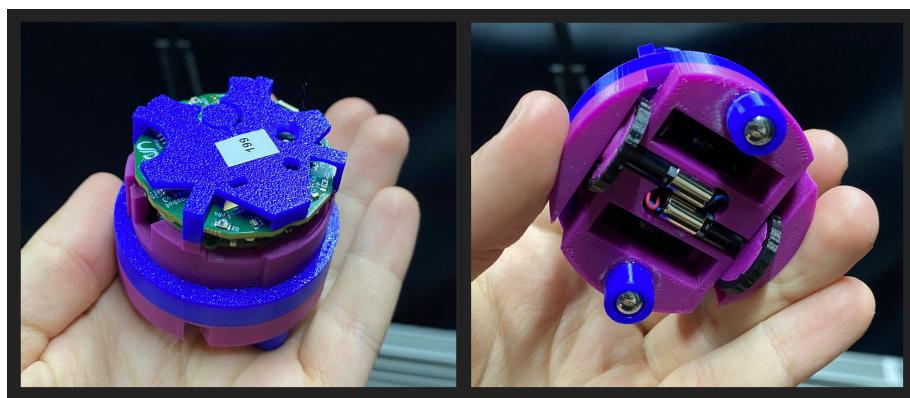


FIGURE 3 – Photos d'un Pogobot à roues

Le Software

Le logiciel des Pogobots sur un SDK **open-source**, écrit en C. Le SDK est disponible ici, et nous avons à notre disposition une documentation de l'API nous permettant d'apprendre à contrôler les différentes fonctions des Pogobots, ainsi que de certains exemples de code.

Le simulateur Pogosim

Pogosim est un **simulateur des robots Pogobots** en cours de développement par Léo Caze-nille. Il a pour ambition de reproduire l'API Pogobot utilisé sur les robots afin de pouvoir avoir un code qui fonctionne en simulation tout comme en expérimentation réelle.

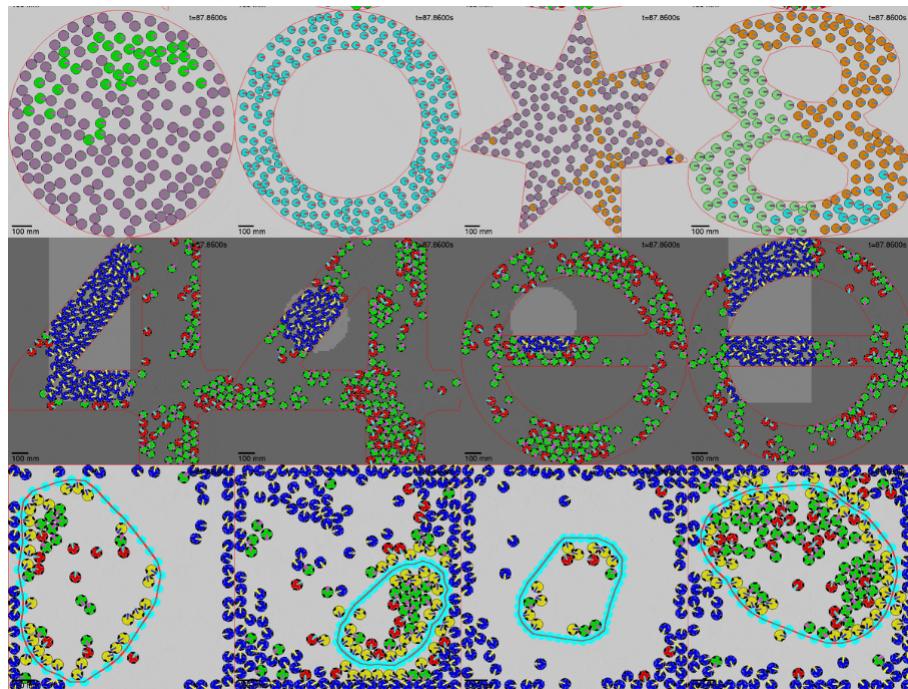


FIGURE 4 – Exemples de simulation Pogosim

Le simulateur est aujourd’hui à la version 0.10.X, la dernière avant la version stable officielle 1.0. Aujourd’hui, cette version prend en compte la création des Pogobots, de ces différentes fonctionnalités comme l’utilisation de 4 émetteurs / récepteurs infrarouges, la gestion de la puissance et direction des moteurs, l’occlusion des robots... En plus de cela, nous pouvons y rajouter d’autres objets comme des Pogowalls, étant des murs intelligents capable de communication. Ils peuvent être statiques (les murs de l’arène) ou dynamiques par membranes mobiles.

Dans le cadre du projet, nous avons connu différentes versions de Pogosim. Un de nos objectifs était de l’utiliser en parallèle de nos comportements afin de fournir des retours utiles : bugs, fonctionnalités manquantes, suggestions... Ces retours ont été pris en compte dans le développement du simulateur tout au long de notre projet, et qui ont donc permis de l’améliorer.

Les comportements à fournir

Nous avons eu à fournir un set de démos pour le projet, tout en les utilisant pour analyser et donner des retours sur les Pogobot à roues et le simulateur. Ces comportements de ces robots sont des tâches typique de la robotique autonome et collective, et nous les avons testé en continu en simulation et à l'arène multi-robots de l'ISIR sur un nombre de robots réels variant entre 10 et 20 en moyenne. Nos codes, avec des gifs de simulation et des vidéos des comportements sont disponibles ici.

Run and Tumble

Le comportement 'Run and Tumble' permet à un robot de se déplacer en ligne droite lorsqu'il n'y a pas d'obstacle (robot ou mur) et change de direction dans le cas contraire. Ce comportement cherche à reproduire le modèle de mouvements présent notamment chez certaines bactéries et agents microscopiques.

Notre stratégie pour ce dernier est simple. Etant donné que les Pogobots ne possèdent pas de capteurs à proximité, nous devions adapter le protocole d'estimation de distance. Nous avons donc créer un mini-interpréteur de règles. Chaque robot, à chaque pas de temps, envoie un ping autour de soi par message infrarouge. Ensuite, il va lire dans sa boîte de messages ce qu'il a reçu. Il retiendra uniquement par quel récepteur il les a reçu. Le Pogobot en possède 4 : à l'avant, à l'arrière, à gauche, et à droite. Pour ce comportement, on omet le capteur arrière.

Avec ces informations, il va mettre à jour sa direction en modifiant la puissance et la direction de ses moteurs afin d'éviter les obstacles.

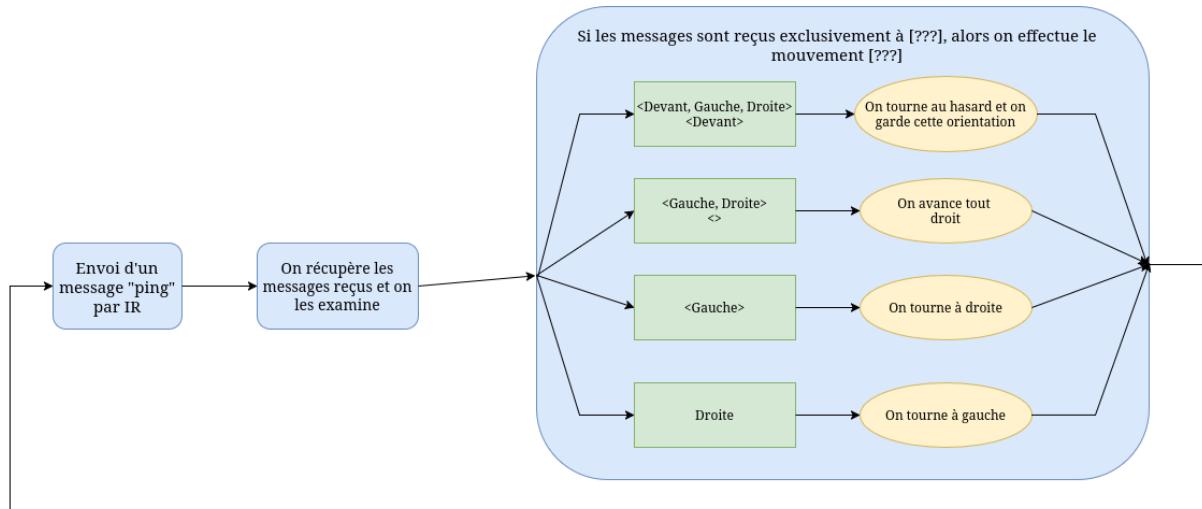


FIGURE 5 – Architecture du comportement 'Run and Tumble'

Ce comportement a été un excellent moyen de prise en main de l'API. Nous avions commencé ce comportement avec les Pogobots à brosse. Nous n'avions pas besoin de mouvements précis, mais nous avons pu constater la sensibilité des mouvements de ces derniers. Chaque un d'entre eux demandaient un calibrage très différents.

Une fois passé sur Pogobots à roues, nous n'avons eu aucun problème. Ils demandaient tout de même un léger calibrage pour avoir une marche avant stable, mais une fois qu'on avait une valeur PWM pour le moteur gauche et droit, nous pouvions l'utiliser constamment sans risque qu'un facteur comme la poussière puisse compromettre son avancé. Par ailleurs, pour une vingtaine de robots testées, nous avons conclus que pour une vitesse égale, la majorité des robots peuvent prendre une valeur de <500-550, 650-800> pour les moteurs droite et gauche respectivement.

L'API des Pogobots nous permettent d'enregistrer dans leur mémoire ces puissances, ce que nous avons donc fait lors des calibrages. Malheureusement, ce n'est pas linéaire, donc afin de pouvoir ralentir, il ne suffisait pas de garder le même ratio, d'où notre choix suivant. Au lieu d'avancer et tourner, nous avons décidé de faire tourner le robot sur lui-même en modifiant la direction d'un des moteurs. Cela évitait par ailleurs les chocs possibles entre un Pogobot et un obstacle, étant donné qu'il s'arrête immédiatement pour tourner dès qu'il détecte quelque chose.

Ce comportement ne prend pas en compte la densité de population sur chacun des côtés, ce qui fait que si l'arène est restreindre et assez complet, il peut y avoir des complications, mais il reste assez rare. De plus, le comportement n'est quasiment pas impacté par de potentiel perte de messages d'après une vingtaine de test, étant donné qu'il suffit d'un message reçu, et qu'on en envoie relativement souvent.

Côté simulation, comme nous étions au début du projet, Pogosim restait assez limité. Notamment, on ne pouvait pas recevoir ni envoyer de messages sur différents émetteurs/récepteurs, ce qui était contre notre idée. Après avoir transmis cette observation, nous avons eu accès à une nouvelle version avec ça d'implémenté. Aujourd'hui, ce comportement fonctionne plutôt bien en simulation, bien que ça reste un peu plus bruité qu'en expérience réelle, notamment avec les murs.

Dispersion

Le comportement 'Dispersion' permet aux Pogobots de s'écartier les uns aux autres pour mieux couvrir l'environnement.

Notre premier objectif était de couvrir au mieux l'environnement tout en restant à proximité d'au moins un Pogobot afin de pouvoir communiquer si besoin. Nous avons donc tenté plusieurs méthodes pour estimer la distance, comme varier la puissance des messages infrarouges 'ping' envoyés, ou bien en utilisant les photorécepteurs et les LEDs sur leurs côtés, sans réussite satisfaisante.

Au final, nous avons décidé d'utiliser le même système que 'Run and Tumble' en l'ayant légèrement modifier. Voici les deux principales différences :

- On prend en compte le récepteur IR arrière. On avance si on le peut à l'avant dès qu'on reçoit un message par là.
- On s'arrête si on ne détecte personne, pour garder cette idée de rester le plus proche possible des autres pour l'aspect groupe et communication.

Nous avons pu les tester en expérience réelle, sous plusieurs manières. Nous pouvions les disposer tous regroupés au milieu d'une arène, et observer comment ils se dispersent. Ensuite, nous les avons recueillis dans le coin d'une arène. Et enfin, nous avons testé le cas où nous les rajoutions dans l'arène petit à petit avec possiblement des obstacles (comme des piliers).

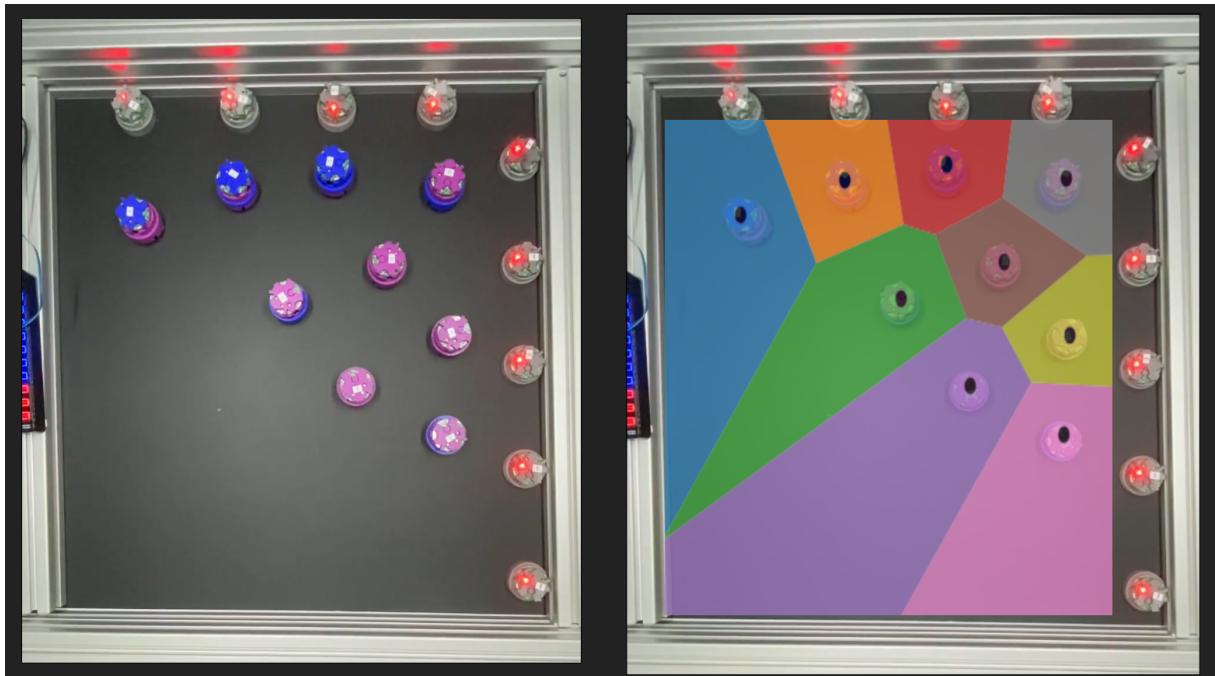


FIGURE 6 – Exemple de dispersion (début au coin) avec un diagramme de Voronoï



FIGURE 7 – Exemple de dispersion (regroupés au milieu) avec un diagramme de Voronoï

Nous avons en moyenne une belle répartition des Pogobots dans le premier cas, bien qu'ils ne prennent pas en compte la taille de arène, après 8 tests. On peut y remarquer la distance de communication possible une fois qu'ils sont à l'arrêt. L'algorithme présente ses faiblesses quand le ratio robots/superficie augmente, leur laissant moins de place. Ils peuvent continuer indéfiniment de s'ajuster sans jamais s'arrêter. Bien sûr, cet algorithme peut présenter d'autres limites, notamment suivant la forme de l'arène. Nous n'avions, physiquement, qu'une arène à la forme simple.

Heureusement, grâce à la simulation, nous avions accès à d'innombrables type d'arène, et nous pouvions aussi en créer nous-mêmes si besoin. Nous avons pu tester la dispersion dans des cas précis de couloirs étroits par exemple, ou de forme particulière comme des étoiles. Nous avons pu constater qu'il pouvait y avoir des cas où les Pogobots ne s'arrêtaient jamais de bouger ou d'avancer, notamment dans le coin d'une étoile, ou dans un couloir étroit. Ce simulateur nous a beaucoup aidé pour ce comportement quand on avait pas accès à l'arène. Les gifs présents sur notre répertoire github présentent ces cas.

Phototaxie

Ce comportement permet aux Pogobots de suivre la source lumineuse la plus forte captée par leurs photorécepteurs. Les Pogobots ont trois photorécepteurs. Un à l'arrière, et deux à l'avant-gauche et l'avant-droite respectivement. Chacun renvoie une valeur proportionnelle à la lumière captée.

Notre algorithme ici est très simple, il suffit de comparer les trois valeurs récupérées par les photorécepteurs, et de s'approcher vers le côté le plus lumineux.

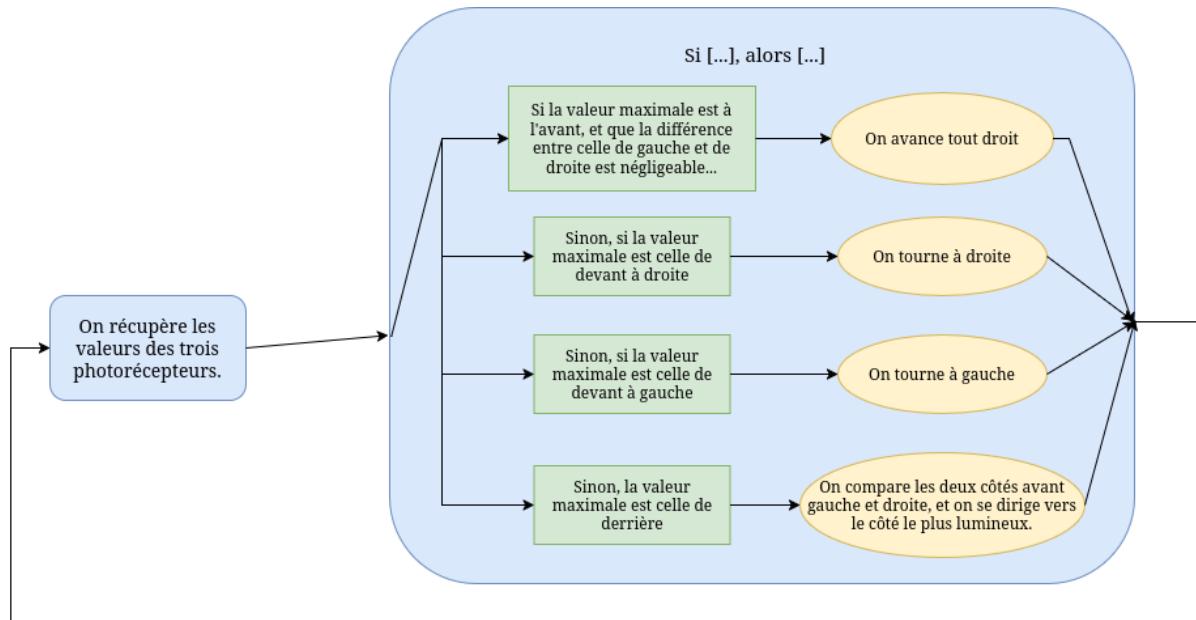


FIGURE 8 – Architecture du comportement 'Phototaxie'

En expérimentation réelle, les Pogobots répondent bien à la problématique dans nos conditions de tests. Nous avons essayé ce comportement dans un environnement sombre avec une simple lampe torche. Les Pogobots se rapprochaient bien de la lampe après quelques ajustements. Fait intéressant, ils vont parfois à la "limite du trait" entre la lampe et l'obscurité pour remonter ensuite vers son objectif. Le mouvement reste clair, et même avec plusieurs Pogobots qui suivent ce comportement sans faire attention aux autres, on a le même résultat, bien qu'ils peuvent se pousser mutuellement vers l'objectif. Nos vidéos sur le répertoire github en est un bon exemple.

En simulation, nous n'avions malheureusement pas la possibilité de placer des sources lumineuses au début du projet. Mais après cette demande, la version 0.10.0 a rajouté la possibilité de créer des zones lumineuses (statiques) dans l'arène. Nous n'avons pas eu le temps de faire une mise à jour pour cette version, mais sur le github de Pogosim, il existe un gif et un exemple de code de phototaxie montrant qu'on peut bien en simuler simuler une !

Suivi de leader

Pour ce comportement, il s'agit de placer les robots dans une arène en file indienne. Un robot suit un comportement de déplacement au hasard, devenant alors le leader de file, et les autres suivent leur prédécesseur dans la file.

Nous sommes partis du principe que le sens de la file indienne sera établie selon l'orientation physique initiale des robots. Tous les robots doivent donc être alignés suivant la même direction, et celui positionné en tête de file, sans aucun prédécesseur détecté face à lui, sera alors élu leader.

Lors de l'implémentation et de la mise en pratique de ce comportement, nous avons rencontrés de nombreux défis principalement liés à la synchronisation. Il est en effet difficile, voire impossible, de placer tous les robots simultanément en file. Par conséquent, ces derniers doivent attendre que la file indienne soit complètement formée avant d'initier le déplacement collectif.

Pour gérer cela, nous avons segmenté le déroulement du comportement en plusieurs phases successives, contrôlées à l'aide de minuteurs, comme décrit dans l'algorithme ci-dessous. Dans cet algorithme, les robots échangent leurs identifiants par messages, qu'ils retransmettent à leur tour aux autres afin de propager l'information à l'ensemble du groupe. Cela permet à chaque robot de déterminer si la file est au complet en comparant le nombre total de robots connus au nombre total attendu, défini manuellement (correspondant à `NB_MAX_ROBOTS`). Ce n'est qu'une fois la file constituée que le processus d'élection du leader peut commencer, suivi de l'activation du déplacement collectif.

Ensuite, un minuteur de 10 secondes est déclenché pour permettre au dernier robot ajouté à la file de recenser à son tour l'ensemble des robots présents, tout en laissant à chacun le temps d'identifier son prédécesseur et son successeur. À l'issue de ce délai, les rôles de leader et de suiveur sont attribués.

Cependant, ce minuteur ne démarre pas simultanément sur tous les robots, ce qui peut entraîner un léger décalage entre eux. Pour compenser cette désynchronisation, un second minuteur est utilisé : déclenché uniquement par le leader, il impose un délai supplémentaire de 20 secondes pendant lequel tous les robots sont à l'arrêt, attendant le signal du leader qui n'est envoyé qu'à l'expiration du minuteur. Ce minuteur permet donc d'assurer un déplacement collectif synchronisé.

Pendant le déplacement, le leader se déplace de manière aléatoire tout en évitant les obstacles rencontrés, qu'il s'agisse de murs ou de la file elle-même. Lorsqu'un robot ne détecte plus son successeur (sauf pour le dernier de la file, qui n'en a pas), il interrompt son déplacement et attend de le retrouver dans son champ de détection pour reprendre son mouvement. Les robots suiveurs ajustent leur trajectoire en fonction de la direction depuis laquelle ils reçoivent les messages émis par leur prédécesseur.

```

1 NB_MAX_ROBOTS = n ;
2 counter_robots = 1 ;
3 id_received = [] ;
4 successor_id  $\leftarrow$  none ;
5 predecessor_id  $\leftarrow$  none ;
6 leader_elected  $\leftarrow$  false ;
7 if leader_elected  $\leftarrow$  false then
8   while counter_robots < NB_MAX_ROBOTS do
9     ENVOYER(my_id) ;
10    if MESSAGE REÇU then
11      sender_id  $\leftarrow$  message[id] ;
12      if sender_id  $\neq$  my_id ET not sender_id in id_received then
13        AJOUTER sender_id dans id_received ;
14        counter_robots++ ;
15      end
16      ENVOYER(sender_id) ;
17    end
18  end
19  if counter_robots = NB_MAX_ROBOTS then
20    TIMER 10 secondes lancé
21    Continue d'envoyer son id et celui des messages reçus
22    if MESSAGE REÇU then
23      sender_id  $\leftarrow$  message[id] ;
24      if message reçu de devant then
25        | predecessor_id = sender_id ;
26      end
27      if message reçu de derrière then
28        | successor_id = sender_id ;
29      end
30    end
31    if TIMER terminé then
32      if predecessor_id  $\leftarrow$  none then
33        | Robot élu LEADER
34      end
35      sinon Robot devient FOLLOWER
36      leader_elected  $\leftarrow$  true ;
37    end
38  end
39 end
40 TIMER 20 secondes lancé par le LEADER une seule fois
41 if TIMER terminé then
42  Détection d'obstacle
43  Mouvement aléatoire si LEADER
44  Suivi du prédecesseur si FOLLOWER
45 end

```

Algorithme 1 : Pseudo-code du comportement de suivi de leader

Le critère de réussite est le suivant : après avoir correctement identifié leur rôle, les robots doivent parvenir à se déplacer de manière coordonnée, fluide et continue tout en maintenant la structure de la file indienne.

Les tests en simulation ne sont pas totalement possibles avec plusieurs robots en raison de l'impossibilité à placer initialement les robots en file indienne. Toutefois, la simulation a été d'une grande aide pour tester le comportement avec deux robots lorsque ces derniers étaient placés côte à côte. De plus, nous avons réalisés cinq tests en expérience réelle avec deux, puis trois et enfin quatre pogobots. Tous les robots ont été calibrés au mieux au préalable.

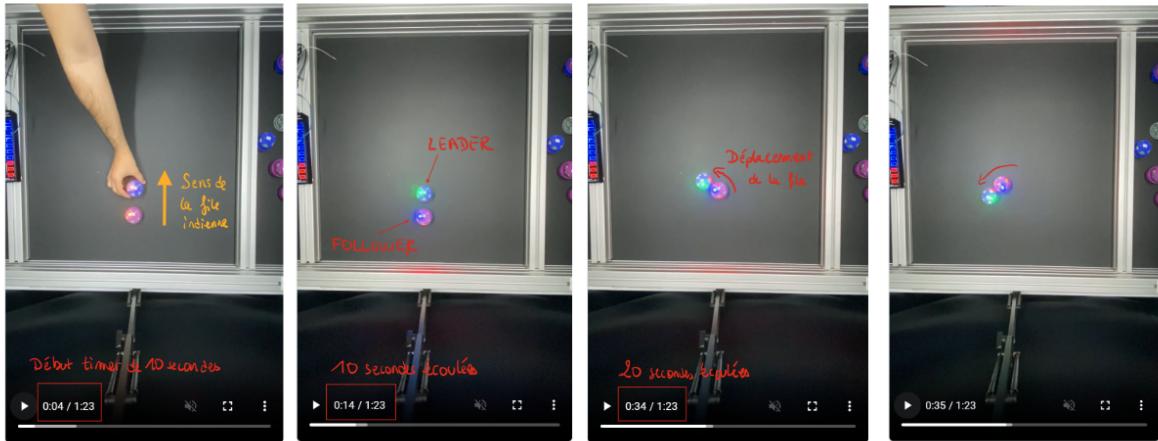


FIGURE 9 – Etapes du suivi de leader avec deux Pogobots

Nous observons que le système d'établissement des rôles à l'aide des minuteurs fonctionne bien pour deux robots. La LED du leader élu s'allume en vert et celles du suiveur en bleu (image 1 en partant de la gauche). Dans le cas de trois ou quatre robots, le succès de l'élection devient moins systématique, selon l'état des capteurs infrarouges détectant les messages, la perte ou le brouillage des messages envoyés en grande quantité, ou bien la durée insuffisante des minuteurs par rapport au nombre total de robots croissant.

Concernant le déplacement, nous observons que, dans le cas de deux robots, le mécanisme d'arrêt du leader lorsqu'il ne détecte plus son successeur fonctionne correctement. Le suiveur suit bien le leader tant que celui-ci se déplace en ligne droite. Dès que le leader amorce un virage, le suiveur a tendance à perdre sa trace parfois. Cela semble s'expliquer par une rotation trop prolongée du suiveur, qui le fait dévier de la trajectoire du leader sans parvenir à le réaligner correctement. Ce problème n'a pas pu être résolu par manque de temps.

Pour améliorer ce comportement, il serait pertinent de trouver une alternative au système de minuteurs, car celui-ci dépend fortement du nombre total de robots dans la file et introduit des délais arbitraires. À la place du minuteur de 20 secondes, il serait par exemple possible d'utiliser une source lumineuse comme signal de départ collectif. Ce type de synchronisation, plus réactif et indépendant du nombre de robots, est actuellement en cours d'implémentation par l'ISIR.

Enfin, en complément du système d'arrêt déjà mis en place lorsque la liaison est rompue, nous avons envisagé un mécanisme de ralentissement progressif. Celui-ci permettrait aux ro-

bots d'adapter leur vitesse en fonction de la distance qui les sépare de leur prédecesseur. Une estimation approximative de cette distance pourrait être obtenue à partir de la puissance des signaux infrarouges reçus par exemple.

Déplacement en formation

Chaque Pogobot se déplace en ligne droite en l'absence de voisin et évite les murs si nécessaire. En présence d'un voisin, un Pogobot modifie sa direction pour s'aligner sur celle de son (ses) voisin(s).

Les Pogobots utilisent initialement le comportement du déplacement au hasard pour éviter les murs. Puis, lorsque l'un des Pogobots détecte un ou plusieurs Pogobots, il se dirige vers la formation contenant le plus de membres et décide alors de suivre celui dont l'identifiant est le plus petit et le plus proche de son propre identifiant.

Nous avons décidé de prendre en compte le nombre de membres par formation détectée pour que les Pogobots ne suivent pas un leader défini mais privilégie plus le fait de rejoindre la plus grande formation détectée, pour ensuite suivre un des membres bien précis de cette formation. Le choix du membre suivi a été décidé ainsi afin d'éviter qu'un même Pogobot se fasse suivre par tous les autres, pouvant ainsi le bloquer entre plusieurs Pogobots.

Lors de la conception de ce comportement, nous avons dû faire face à plusieurs défis comme les différences de temps de transmission et réception, empêchant parfois le Pogobot courant de bien détecter tous ses voisins et d'ainsi choisir la meilleure formation. De plus, la différence de vitesse entre les Pogobots peut mener à certains d'entre eux de ne plus être en mesure de détecter son leader et donc de quitter la formation.

Le critère de réussite est le suivant : réussir à maintenir un mouvement de formation sans que les membres ne divergent de celle-ci.

Nos tests en expérience réelle révèlent que notre Pogobot s'adapte bien au mouvement de son voisin, y compris lors de l'évitement des murs, mais au bout d'un moment, on observe que le Pogobot suiveur perd la détection de son voisin suivi et diverge ainsi de la formation.

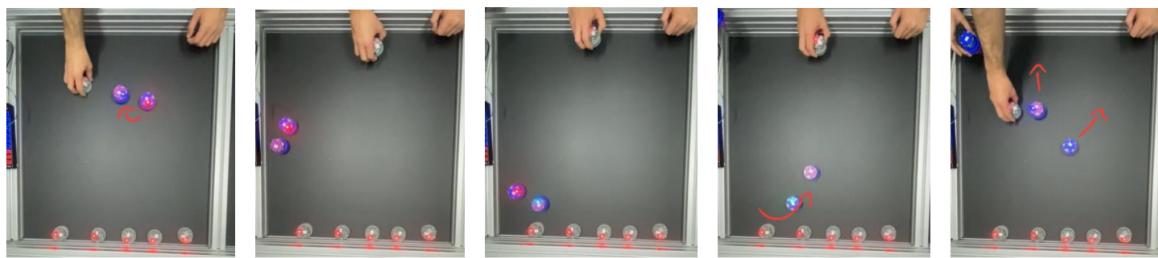


FIGURE 10 – Etapes du déplacement en formation avec deux Pogobots

1. Les deux Pogobots se détectent et celui avec la LED rouge s'adapte aux déplacements de celui avec la LED bleue.
2. Le Pogobot suiveur effectue les mêmes déplacements que son leader.
3. Le Pogobot non suiveur détecte un mur.
4. Ce dernier évite bien le mur et le Pogobot suiveur le suit toujours.
5. Le Pogobot suiveur ne détecte plus son leader donc se remet à effectuer le déplacement au hasard.

Voici les améliorations possibles que nous suggérons :

- Trouver un moyen d'évaluer la distance entre Pogobots pour éviter de diverger de la formation.
- Trouver un moyen de réceptionner tous les messages des Pogobots à proximité, même un peu tardifs, pour trouver les meilleures formations à suivre.

Pour créer ce comportement, nous avons beaucoup travaillé sur la première version du simulateur mais étant donné que nous avions besoin de détecter les directions en fonction des capteurs de nos Pogobots, les mouvements selon le suivi de voisin ne pouvaient pas être réalisées. Cependant, la détection de voisin et le choix du voisin à suivre selon les identifiants fonctionnait bien.

```
1 NB_MAX_ROBOTS = n ;
2 NB_DIRECTIONS = 5;
3 VAL_MAX = m;
4 for i = 0 à NB_DIRECTIONS – 1 do
5   | cpt_robots_par_direction[i] ← 0 ;
6 end
7 for i = 0 à NB_DIRECTIONS – 1 do
8   | for j = 0 à NB_ROBOTS – 1 do
9     |   | id_robots_par_direction[i][j] ← VAL_MAX;
10    | end
11 end
12 ENVOYER("robots");
13 if MESSAGE REÇU et MESSAGE REÇU ← "robots" then
14   |   | id_robots_par_direction[ir_receiver][cpt_robots_par_direction[ir_receiver]] ←
15   |   |   | id_sender ;
16   |   |   | cpt_robots_par_direction[ir_receiver] ++;
17 end
18 if voisin_detecte ← true then
19   | if cpt_robots_par_direction[i] ← meilleure_formation then
20     |   | if id_robots_par_direction[i][j] ← id_meilleur_voisin then
21       |   |   | direction_suivie ← i;
22       |   |   | move_follow(direction_suivie);
23     |   | end
24   | end
25   | Else : move();
26 end
26 Else : move();
```

Algorithme 2 : Pseudo-code du comportement de déplacement en formation

Alignment face au mur

Lors de ce comportement, les Pogobots se rapprochent des murs et s'arrêtent. L'état final consiste en un alignement des Pogobots face au mur.

La stratégie pour laquelle nous avons opté est la suivante. Les pogobots avancent tout droit en l'absence de mur. Une fois un mur détecté, un timer est lancé pour évaluer quand le Pogobot doit s'arrêter. Dans le cas des Pogobots à roues, ils doivent se mettre dos au mur et reculer jusqu'à son arrêt total. Dans le cas des Pogobots à brosses, ceux-ci avancent vers le mur jusqu'à leur arrêt. Cette stratégie a été décidée ainsi afin d'utiliser les capacités des Pogobots à roues telles que pouvoir reculer et tourner sur eux-mêmes, afin de pouvoir "se garer" et ainsi être prêts à repartir aisément dans le bon sens.

Nous avions rencontrés plusieurs défis. Dans un premier temps, nous voulions établir une stratégie où l'on calculait la distance des Pogobots au mur en se basant sur le temps de réception des messages provenant de ce-dernier pour savoir quand les Pogobots devaient ralentir et s'arrêter. Cependant, les temps de réception ne différaient pas selon la distance, donc nous sommes plutôt partis sur l'idée d'un timer pour déclencher un décompte jusqu'à l'arrêt des Pogobots.

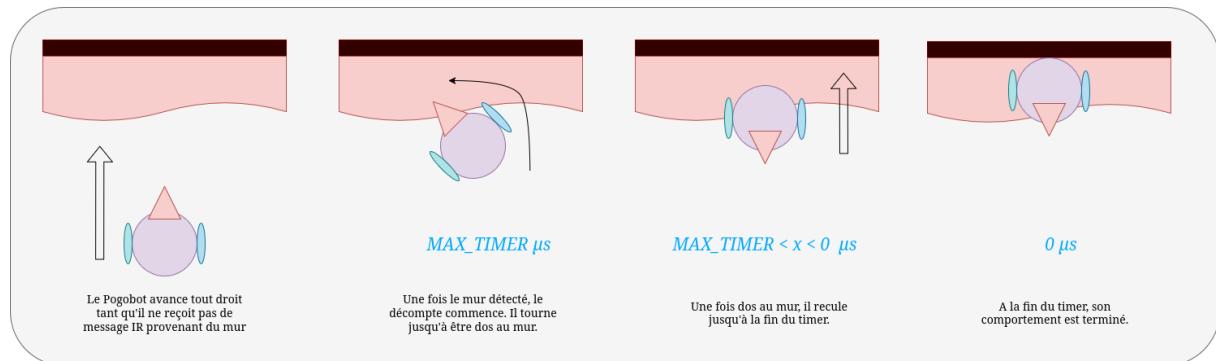


FIGURE 11 – Schéma descriptif du comportement 'Alignment face au mur'

Pour que ce comportement soit réussi, tous les Pogobots doivent s'arrêter à une distance d'au moins 5mm du mur, avec une orientation correcte pour repartir.

On observe que nos Pogobots avancent approximativement droit dû au calibrage des Pogobots. Dès que les Pogobots détectent un mur, on voit qu'ils se retournent bien dos au mur et reculent jusqu'à leur arrêt total. Une amélioration possible à laquelle nous pensons est de trouver un moyen d'évaluer la distance au mur pour ralentir petit à petit jusqu'à l'arrêt total du Pogobot.

La première version du simulateur ne permettait pas de réellement simuler les pogowalls. Donc pour effectuer nos tests nous avons simulé le mur avec les Pogobots et ainsi observer que le comportement marchait plutôt bien, les Pogobots détectaient leurs congénères et au bout d'un certain temps s'arrêtaient. Cependant, nous ne pouvions pas simuler la marche arrière, mais ce ne fut pas un problème, nous l'avons juste adapté et utilisé le cas des Pogobots à brosses pour la simulation, donc continuer d'avancer jusqu'à arrêt. Dans la version actuelle du simulateur, nous pouvons bel et bien simuler ce comportement !

Retour sur le projet

Les Pogobots à roues ont été récemment créés, et sont toujours en cours de développement. Nous sommes les premiers testeurs de ces derniers, et nous avons pu les comparer à grands frères, les Pogobots à brosses.

D'après nos observations, les Pogobots à roues sont très intéressants. Simple d'utilisation, des légères différences de calibrage sur les roues sans que ce soit dérangeant, nous n'avons pas réellement eu des difficultés avec si on omet l'estimation de la distance entre deux objets. Par conséquent, il devient très facile et rapidement de programmer des comportements, et de les tester avec peu de matériel. Le fait que le projet soit open-hardware et open-source est un grand plus.

En ce qui concerne le simulateur, nous l'avons vu se développer au fil du projet. Nous avons pu participer activement à son développement avec de nombreuses propositions, suggestions, observations fournis à Léo Cazenille, qui ont été pris en compte lors des mise à jour de version. Aujourd'hui, nous considérons le simulateur très agréable à utiliser, avec une affichage simple, propre et assez paramétrable. Le lien avec l'API Pogobot fonctionne de manière très intuitive, et nous avions accès à assez d'exemples pour la prise en main.

Nous avons quelques suggestions d'améliorations encore qui, selon nous, pourrait apporter une aide aux utilisateurs de Pogosim :

- La possibilité de déplacer ou repositionner les Pogobots avec la souris pendant la simulation.
- Un éditeur d'arène afin de pouvoir tester plusieurs scénarios sans le relancer.
- Une documentation un peu plus détaillée, notamment sur les paramètres de simulation / le fichier yaml.

La simulation nous a grandement aidé. En effet, nous avions peu de matériel à disposition, et un accès restreint à l'arène multi-robots (distance, réservations...). Par conséquent, l'expérimentation réelle sur robots n'était pas toujours possible, mais le simulateur nous a permis de tous travailler sur le projet, de manière efficace sans contrainte de matériel ni de temps. Il est vite devenu un atout majeur dans ce projet.

Conclusion

Ce sujet de projet nous a offert une belle introduction au développement sur systèmes embarqués et la robotique en essaim. De plus, il nous a mis en situation expérimentale réelle, ce qui nous a fait prendre conscience du potentiel fossé entre la théorie, la pratique, et la simulation.

En effet, il arrivait que la théorie était fiable, mais que par des facteurs extérieurs (comme avec les Pogobots à brosses), la pratique était toute différente. Parfois, il subsistait une différence assez marquante entre simulateur et réalité, ce qui nous a bien appris de ne pas se fier qu'à l'un et l'autre, et d'essayer de comprendre ces différences.

Remerciements

Nous remercions grandement notre encadrant de projet Nicolas Bredeche, et également Leo Cazenille, Loona Macabre et Alessia Loi pour leur gentillesse, tout le soutien apporté lors de nos interrogations, et leur réactivité.

Annexes

Cahier des charges

Voici les différents liens utiles pour le projet :

- Notre projet - Github
- Le site officiel Pogobot
- SDK Pogobot - Github
- Pogosim - Github
- La documentation Pogobot

Cahier des charges

L'objectif principal de ce projet est de programmer ces robots pour réaliser plusieurs tâches typiques de la robotique autonome et collective. Afin de réaliser ces comportements, le groupe devra prendre en main le Pogobot, programmable en langage C avec son API dédiée, et adapter le protocole de communication existant pour estimer la distance entre robots (en particulier pour les comportements de suivi, formation, alignement et dispersion). Les étudiant·es auront accès à l'arène multi-robots de l'ISIR et l'ensemble du développement et des démos devra être réalisé sur robots réels. Pour chaque tâche, entre 10 et 20 robots sont placés dans une arène.

Déplacement au hasard

Chaque robot se déplace en ligne droite lorsqu'il n'y a pas d'obstacle (robot ou mur) et change de direction dans le cas contraire.

Suivi de leader

Les robots sont placés dans une arène en file indienne. Un robot suit un comportement de déplacement au hasard, les autres suivent leur prédécesseur. Test

Déplacement en formation

Chaque robot se déplace en ligne droite en l'absence de voisin et évite les murs si besoin. En présence d'un voisin, un robot modifie sa direction pour s'aligner sur celle de son (ses) voisin(s).

Alignment face au mur

Les robots se rapprochent des murs et s'arrêtent. L'état final est d'obtenir un rangement en ligne des robots face au mur.

Phototaxie

Les robots se déplacent vers la source lumineuse.

Dispersion

Les robots s'écartent les uns des autres pour couvrir au mieux l'environnement.

En complément de l'implémentation de ces comportements, il nous est également demandé de fournir un retour d'expérience détaillé, à la fois sur le projet dans son ensemble, et plus spécifiquement sur le nouveau simulateur des Pogobots, nommé Pogosim (cf. <https://github.com/Adacoma/pogosim>). Ce retour devra couvrir les aspects suivants.

Retour sur l'installation

Évaluation de la facilité de prise en main, clarté de la documentation, compatibilité avec différents systèmes (Linux, macOS, etc.), dépendances requises, éventuels problèmes rencontrés à cette étape.

Retour sur l'utilisation

Ergonomie générale, capacité à simuler efficacement les comportements souhaités, fidélité par rapport aux comportements observés sur des robots physiques.

Retour sur les problèmes et bugs rencontrés

Identification des points bloquants ou instables (crashes, incohérences de simulation, comportements non reproductibles), suggestions d'amélioration ou de nouvelles.

Manuel de l'utilisateur

Installation des dépendances

Afin d'utiliser notre projet, vous aurez besoin du kit de développement logiciel (SDK) des Pogobots mais également du simulateur Pogosim. La méthode d'installation est précisée sur leur répertoire respectif. Officiellement, ce projet n'est supporté que pour Linux Debian/Ubuntu (WSL comprise) et MacOS. Mais en trifouillant un peu, il est possible de l'utiliser également sur ArchLinux.

L'installation du projet se fait de cette manière :

```
$ git clone https://github.com/Stalkyyy/P-Androide___Comportements_Robots_Pogobot_Wheel
$ cd P-Androide___Comportements_Robots_Pogobot_Wheel/
$ mkdir -p dependencies
$ cd dependencies

# SDK : Suivez les étapes de dépendances sur leur répertoire
$ git clone https://github.com/nekonaute/pogobot-sdk.git

# SIMULATEUR : Suivez les étapes de dépendances sur leur répertoire
$ git clone https://github.com/Adacoma/pogosim.git
```

Si vous souhaitez placer les dépendances autre part, il faudra penser à modifier chaque fichier Makefile et y placez le nouveau path du SDK et du simulateur !

Compilation

Une fois cela fait, vous pouvez compiler nos comportements. Pour cela, rendez vous dans le répertoire associé au comportement souhaité, et tapez l'une des trois commandes suivantes selon votre choix.

```
$ cd dispersion    # Exemple de répertoire de comportement
$ make clean sim  # Pour compiler uniquement la simulation
$ make clean bin  # Pour compiler uniquement le binaire pour les Pogobots
$ make clean all   # Pour compiler les deux à la fois.
```

Utilisation de Pogosim

Pour lancer une simulation, toujours dans le répertoire du comportement compilé, il vous suffira de taper cela :

```
./dispersion -c conf/test.yaml    # Exemple pour dispersion
```

Si vous voulez modifier les paramètres de la simulation, vous pouvez modifier *test.yaml* ou bien utiliser un autre fichier. Si vous regardez le terminal au lancement, il vous donnera des commandes utiles si vous souhaitez faire apparaître les réseaux de communications, ralentir la simulation, l'accélérer...

Charger un comportement sur Pogobot

Une fois compilé, vous aurez besoin d'un appareil USB vers UART entre votre ordinateur et le connecteur FFC/FPC présent sur la tête du Pogobot. Une fois branché, tapez la commande suivante :

```
$ make connect TTY=/dev/ttyUSB0    # Si besoin, mettez le bon chemin.
```

Si tout se passe bien, il suffira de taper sur 'Entrée', et vous arrivez sur l'invite de commandes du Pogobot. Vous pouvez taper '**help**' pour avoir la liste des commandes disponibles, mais les deux principales restent :

```
$ serialboot    # Charge votre code dans le Pogobot.  
$ run          # Lance le Pogobot, avec son code.
```

Enfin, vous pouvez débrancher le Pogobot si vous n'avez pas besoin d'une affichage.