

Faculté des Sciences et Ingénierie - Sorbonne université

Master Informatique parcours ANDROIDE



Rapport du Projet Androide

Les robots Pogobots pour la robotique en essaim

Réalisé par :

Maxence MAIRE

Amélie SUN

Nour Ismahane SLIMANI

Loona MACABRE

Encadrant :

Nicolas BREDECHE

Année 2022-2023

Table des matières

1	Introduction	1
2	Les Pogobots : présentation des robots	2
2.1	Architecture du Pogobot	2
2.2	Communication entre les Pogobots	3
3	Contribution	4
3.1	Étude des capacités fonctionnelles d'un Pogobot	4
3.1.1	Vitesse de communication	4
3.1.2	Taille maximale d'un message et distances maximales de transmission entre deux robots	5
3.1.3	Champ de communication et règle en plexiglas	5
3.1.4	Conversation entre plus de 2 Pogobots	5
3.2	Implémentation de comportements individuels	8
3.2.1	Le filtre de Kalman	8
3.2.2	Calibrage des robots	13
3.3	Implémentation de comportements collectifs	14
4	Conclusion	17
A	Cahier des charges	19
B	Manuel de l'utilisateur	20
B.1	Installer les dépendances	20
B.2	Charger une application sur un Pogobot	21

Table des figures

2.1	Schéma descriptif de l'architecture des pogobots	2
2.2	Représentation des composants des pogobots	3
3.1	Expériences pour déterminer la vitesse de communication (capture d'écran avec le temps)	4
3.2	Évolution du nombre de messages reçus par le Listener en fonction du temps de sonde	7
3.3	Évolution du nombre de messages reçus par le Listener en fonction de la fréquence d'émission	8
3.4	Expériences pour le filtre Kalman : trajectoires suivies par les robots . . .	9
3.5	Données de l'IMU, trajectoire en ligne droite, avant et après filtre	10
3.6	Données de l'IMU, virage à droite, avant et après filtre	11
3.7	Données de l'IMU, demi tour par la gauche, avant et après filtre	12
3.8	Photos de différents configurations avec la fonction de BorderDetection (cyan : 1 voisin, rouge : 2 voisins, vert : 3 voisins, bleu : 4 voisins)	16
B.1	Message de réussite de connexion de l'ordinateur au Pogobot	21

Introduction

Dans le cadre de l'UE P-ANDROIDE, nous avons choisi le sujet intitulé “Les robots Pogobots pour la robotique en essaim”. Ce projet a été encadré par Nicolas Bredèche et nous a introduit aux thématiques de la robotique en essaim, et plus généralement, au développement sur des systèmes embarqués. Nous étions également accompagnés par Alexandre Guerre, ingénieur logiciel participant au développement des Pogobots.

La robotique en essaim, dite aussi robotique de groupe, est un domaine qui développe la coordination des systèmes composés de nombreux robots, principalement de conception simple. Les robots travaillent ensemble pour accomplir une tâche commune. Ces agents communiquent entre eux et prennent des décisions de manière collective pour atteindre l'objectif souhaité.

Les Pogobots sont nés d'une collaboration entre l'ISIR (Institut des Systèmes Intelligents et de Robotique) et le SUMMIT (Sorbonne Université Maison des Modélisations Ingénieries et Technologies). Ils sont conçus pour la recherche expérimentale en laboratoire, notamment pour l'étude des interactions physiques entre robots, la recherche sur le comportement collectif et l'apprentissage social. Ils sont les descendants des Kilobots, des robots conçus à l'Université d'Harvard pour effectuer des tâches collectives.

Au cours du semestre, nous avons eu accès à plusieurs robots que nous avons appris à manipuler et sur lesquels nous avons implémenté de nouvelles fonctionnalités.

Deux thèmes se sont dégagés dans nos objectifs :

- Améliorer le comportement individuel d'un robot, en particulier au niveau du déplacement ;
- Implémenter des comportements en groupe, ce qui touche plus à l'aspect robotique en essaim.

Étant donné que les Pogobots sont encore en cours de conception, ce projet a également eu pour but de tester les robots et de faire part de nos impressions à M. Bredèche et M. Guerre. Ce rapport décrira d'abord plus en détail les Pogobots, ce à quoi nous ajouterons les observations que nous avons pu faire au cours de nos expériences. Nous aborderons ensuite les comportements individuels et collectifs des robots et nous détaillerons les différents problèmes que nous avons rencontré ainsi que nos contributions au code.

[Lien vers notre github.](#)

Les Pogobots : présentation des robots

2.1 Architecture du Pogobot

Comme évoqué plus haut, les Pogobots sont des robots conçus pour la recherche dans la robotique en essaim. Ce sont de petits robots d'à peu près 5 cm de haut et de diamètre, équipés d'un exosquelette imprimé en 3D.

Un pogobot est composé d'une tête (head) équipée d'un processeur, d'une IMU, de dispositifs de communication infrarouge et d'une LED (diode électroluminescente). La tête est branchée sur un "ventre" (belly) qui est composé de plusieurs LEDs, de trois moteurs (gauche (L), milieu (M), droit (R)), de la batterie et du système de régulation de la batterie. Voici un schéma descriptif de leur architecture :

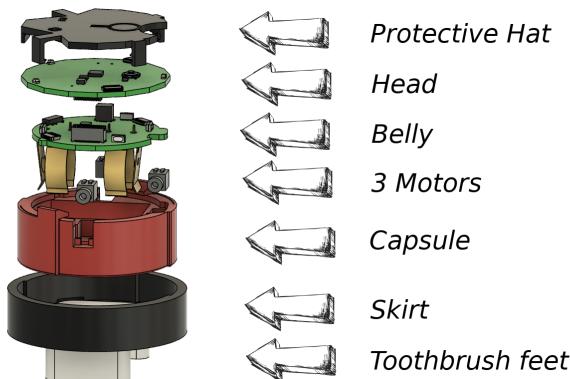


FIGURE 2.1 – Schéma descriptif de l'architecture des pogobots

Comme on peut le voir, un Pogobot n'est pas équipé de roues. Il se déplace grâce à la vibration de ses moteurs sur ses pieds en brosse à dents. Ces brosses sont d'ailleurs légèrement inclinées pour permettre le déplacement. Elles demandent une attention particulière : il est très facile d'abîmer les poils des brosses ou bien de les tordre, ce qui en impacte le déplacement. C'est un des problèmes que nous avons rencontré lors de l'étude du déplacement et de la trajectoire des robots.

La tête est également équipée d'un émetteur-récepteur UART permettant de connecter le robot au port USB d'un ordinateur pour échanger des informations, qu'il s'agisse du code à téléverser depuis l'ordinateur vers le robot, ou bien d'affichages sur la console concernant le robot...

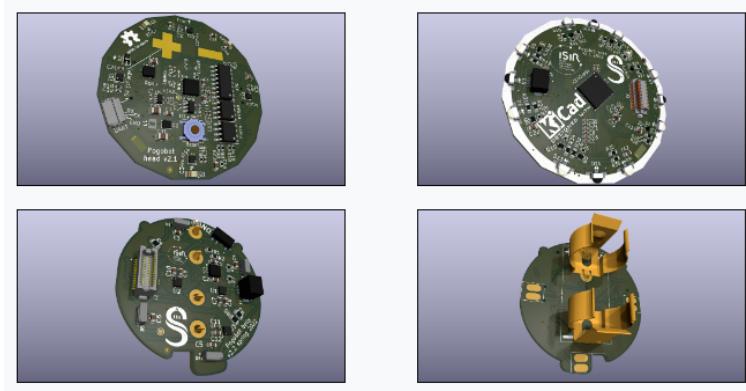


FIGURE 2.2 – Représentation des composants des pogobots

Au niveau logiciel, le robot est développé en langage C. Il implémente un nombre assez limité de bibliothèques pour ne pas saturer sa mémoire. On dispose d'une documentation présentant l'API ([ici](#)) du robot et permettant de comprendre comment utiliser ses fonctionnalités.

2.2 Communication entre les Pogobots

Un message infrarouge se compose d'un payload (contenu du message) et d'un en-tête qui contient des informations telles que la taille du message et l'identifiant du Pogobot expéditeur. Chaque Pogobot est équipé de 4 récepteurs infrarouges et 4 émetteurs infrarouges, correspondant aux 4 directions (front - 0, right - 1, back - 2, left - 3) du robot. De plus, il y a également 4 puissances de transmission d'un message (IR power level values : max - 3, twoThird - 2, oneThird - 1, null - 0). Chaque message capté par un récepteur est enregistré dans la "pile" du robot (message queue). Par conséquent, il existe une fonction pour vérifier si la "pile" contient des messages, une fonction pour retirer un message de la "pile" et une fonction pour vider complètement "la pile".

Contribution

3.1 Étude des capacités fonctionnelles d'un Pogobot

Dans le cadre de développement d'un robot autonome, il est essentiel de comprendre les capacités fonctionnelles du robot dans différentes conditions.

3.1.1 Vitesse de communication

Pour déterminer la vitesse de communication, nous avons filmé 2 Pogobots. Le sender allume sa LED en rouge dès qu'il envoie son message (taille de 384 octets) et le receiver allume sa LED en vert dès qu'il reçoit ce message. Nous avons itéré ce processus en utilisant les différentes puissances de transmission (IR power level) et en les plaçant à leur distance maximale respective (cf section 3.1.2, page 5). Une vitesse de communication approximative de 1280 octets par seconde. Malgré le manque de précision de ce calcul par rapport à l'expérience empirique, il permet tout de même d'obtenir une estimation de la vitesse de communication.

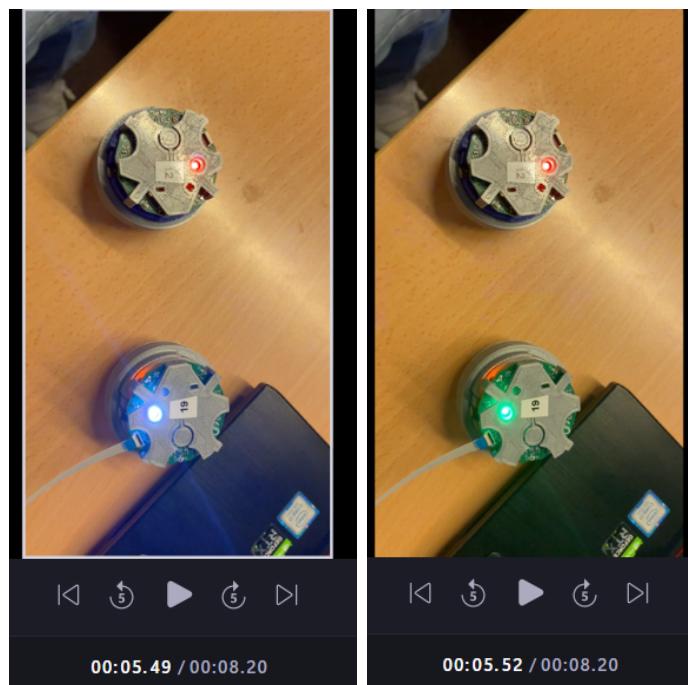


FIGURE 3.1 – Expériences pour déterminer la vitesse de communication (capture d'écran avec le temps)

3.1.2 Taille maximale d'un message et distances maximales de transmission entre deux robots

Pour la partie communication entre deux robots, nous avons essayé de déterminer la taille maximale d'un message tout en garantissant sa réception. Nous avons adopté une approche empirique en manipulant la taille du message émis par le Pogobot expéditeur (sender) et en observant si le Pogobot destinataire (receiver) recevait effectivement le message. La taille maximale du message trouvée est de 384 octets.

De plus, nous avons déterminé empiriquement les distances maximales pour chaque puissance d'envoi (IR power level) à l'aide d'une règle. Pour une puissance de 3, la distance maximale est d'environ 9 cm, pour une puissance de 2, elle est de 7 à 7,5 cm, et pour une puissance de 1, elle est d'environ 6 cm.

3.1.3 Champ de communication et règle en plexiglas

Au cours de nos observations, nous avons constaté une particularité intéressante : la transmission d'un message infrarouge à travers une règle en plexiglas. Cette constatation est pertinente, car elle suggère que l'utilisation d'une règle en plexiglas peut potentiellement étendre la distance de transmission des Pogobots, qui ont une portée maximale définie. En exploitant la transparence du plexiglas à la lumière infrarouge, il est envisageable d'utiliser une règle en plexiglas comme guide pour diriger le faisceau infrarouge entre les Pogobots, permettant ainsi une meilleure propagation du signal sur de plus grandes distances. Cela pourrait être particulièrement utile lorsque la portée des Pogobots est limitée et qu'une communication fiable sur de longues distances est nécessaire. En utilisant une règle en plexiglas pour guider le signal infrarouge, il serait possible d'amplifier la portée effective des Pogobots, facilitant ainsi leur communication sur de plus grandes distances.

Vidéo d'une simulation entre un Pogobot qui envoie un message (rouge) de puissance 1 (donc distance maximale égale à 6cm) et l'autre Pogobot qui reçoit le message (vert) grâce à la règle à une distance de 8cm.

3.1.4 Conversation entre plus de 2 Pogobots

Notre encadrant nous a expliqué que suivant la quantité de messages et d'émetteurs, les messages pouvaient entrer en collision et ne jamais être reçus. Nous avons voulu observer cette situation avec plusieurs robots. Nous avons pris 4 robots qui exécutent un code consistant simplement à dépiler les messages qu'il reçoit et envoyer un message en moyenne

1 fois sur n. Nous avons désigné un "Listener" qui exécute le code durant un temps de sonde et qui rend compte du nombre de messages qu'il a reçu ainsi que du nombre de voisins différents dont il a reçu au moins un message, à la différence des "Talkers" qui exécutent exactement le même code indéfiniment mais n'enregistrent pas ces informations.

Algorithm 1 Listener

```

fréquenceTick ← 30Hz
fréquenceEmission ← 50%
tempsSonde ← 5 secondes
nombreMessagesReçus ← 0
neighborsID ← [-1, -1, -1, -1]
t_DébutExperience ← Now
while temps écoulé depuis t_DébutExperience < tempsSonde do
    t_DébutTick ← Now
    while il reste des messages dans la pile do
        senderID ← ID de l'auteur du message
        if senderID ≠ neighborsID ET il reste des -1 dans neighborsID then
            On remplace un -1 dans neighborsID par senderID
        end if
        nombreMessagesReçus ← nombreMessagesReçus + 1
    end while
    random ← tir aléatoire entre 1 et 100
    if random < fréquenceEmission then
        Envoyer un message en broadcast
    end if
    Vider la pile
    t_finCode ← secondes écoulées depuis t_DébutTick
    Dormir  $\frac{1}{fréquenceTick} - t\_finCode$ 
end while
nombreVoisins ← nombre d'éléments de neighborsID différents de -1
Afficher nombreVoisins, nombreMessagesReçus
```

L'algorithme exécuté par le Listener est donné en page 6. *fréquenceEmission* est le pourcentage de messages envoyés en moyenne : si il vaut 50, alors on envoie en moyenne 1 message sur 2. Les messages envoyés lors de cet algorithme sont tous les mêmes et ont une taille de 100 octets. La variable *fréquenceTick* décrit la fréquence à laquelle le robot exécute la boucle, et vaut 30Hz par défaut, on ne change pas ce paramètre. En revanche, on fait varier *fréquenceEmission* et *tempsSonde* pour observer :

- l'influence de la fréquence d'émission de messages sur la fluidité de l'échange. En particulier, à partir de quel moment les collisions sont trop nombreuses et empêchent un grand nombre de messages d'être délivrés ;
- le temps de sonde nécessaire pour identifier tous ses voisins.

Variation du temps de sonde

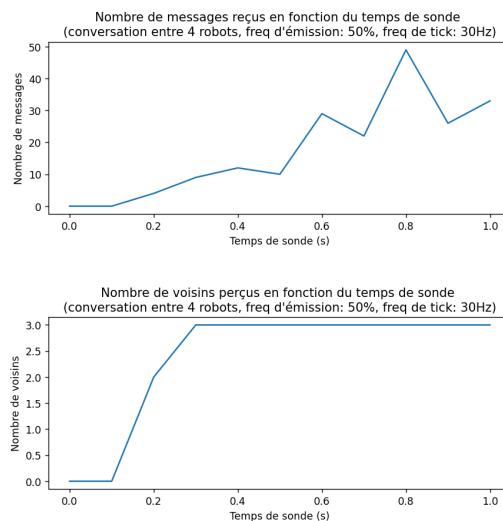


FIGURE 3.2 – Évolution du nombre de messages reçus par le Listener en fonction du temps de sonde

Concernant le temps de sonde, nous avons fait varier le paramètre de 0 à 1 seconde avec un pas de 0.1 seconde. Nous avons fixé arbitrairement la fréquence d'émission de messages à 50%. On observe les résultats de la figure 3.2. On constate que, sur cet exemple, il a suffi de 0.2 seconde au Listener pour identifier tous ses voisins.

Variation de la fréquence d'émission

Pour la variation de la fréquence d'émission, on a fixé le temps de sonde à 5 secondes. L'allure de la courbe montre un certain équilibre si la fréquence est aux alentours de 60%. C'est intéressant car ça nous permet de ne pas noyer les capteurs des robots avec des signaux infrarouge. Il aurait été intéressant également de compter le nombre de voisins identifiés en faisant varier cette fréquence d'émission pour un temps de sonde plus petit, de l'ordre du tick (soit $\frac{1}{30}$ seconde), mais faute de temps, cette expérience n'a pas pu aboutir.

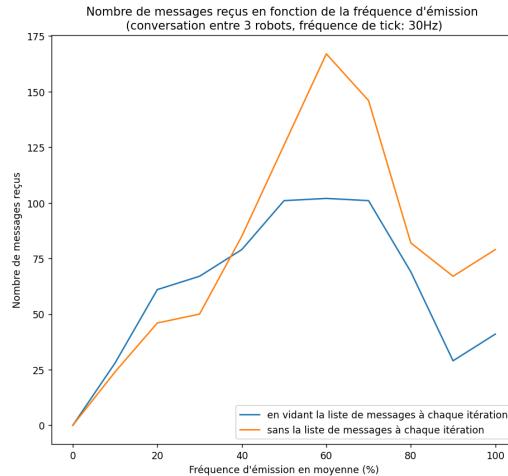


FIGURE 3.3 – Évolution du nombre de messages reçus par le Listener en fonction de la fréquence d'émission

3.2 Implémentation de comportements individuels

Outre les différentes observations que nous avons effectuées sur les robots, un des objectifs majeurs de ce projet a été de contribuer à la correction de la trajectoire et améliorer le déplacement des Pogobots.

Nous avons observé dès le début que les robots ont beaucoup de mal à se déplacer en ligne droite. Malgré le fait que les deux moteurs gauche et droite sont à la même puissance, les robots dévient plus ou moins fortement et se retrouvent à tourner sur eux-mêmes. Il y avait deux alternatives pour corriger cette trajectoire :

- soit de manière dynamique et autonome par les robots en détectant qu'on s'éloigne d'une ligne droite via l'IMU ;
- soit en calibrant à la main les valeurs moteurs des robots.

Plus intéressante mais aussi plus compliquée, la première option a occupé la majeure partie de notre projet. En effet, les données de l'IMU étant très bruitées, impossible de les exploiter telles quelles pour détecter un problème dans la trajectoire. C'est pourquoi nous avons d'abord dû nous pencher sur l'implémentation d'un filtre de Kalman.

3.2.1 Le filtre de Kalman

Le filtre de Kalman est une méthode consistant à estimer les états d'un système à partir d'une série de mesures incomplètes ou bruitées. Il fonctionne en deux étapes : la prédiction et la correction. Durant l'étape de prédiction, le filtre de Kalman essaie de deviner le prochain état du système à partir de ses estimations précédentes. Ce n'est pas suffisant puisque l'estimation à un instant t hérite de toutes les erreurs faites lors des

estimations précédentes. C'est pourquoi, lors de l'étape de correction, le filtre utilise les mesures bruitées des capteurs de l'IMU pour améliorer son estimation.

Dans notre cas, nous avons utilisé un filtre de Kalman étendu qui est approprié pour les modèles non linéaires. En effet, on ne peut pas modéliser l'évolution des données de l'accéléromètre et du gyroscope par une droite, le filtre de Kalman n'est donc pas utilisable tel quel. L'implémentation de ce filtre a été une étape assez longue. Comme nous sommes tous novices dans les domaines de la robotique et du développement sur des systèmes embarqués, nous avons eu des difficultés concernant les points suivants :

- Estimer les facteurs responsables du bruit et à quel point ils influençaient les erreurs de l'IMU : les moteurs, les pattes en brosses à dents, le biais de l'IMU ;
- Comprendre le principe d'un filtre de Kalman (standard et étendu) ainsi que trouver des exemples d'utilisation de ce filtre sur des problèmes pas trop éloignés du nôtre ;
- Trouver un moyen de l'implémenter sans faire appel à de nombreuses librairies utiles en C pour ne pas avoir un code trop lourd...

Après avoir beaucoup étudié le filtre de Kalman étendu sur différents exemples tels que le filtre de données sur un drône, sur un avion, sur un robot tondeuse... et lu de nombreuses implémentations proposées sur le web, nous avons fini par implémenter le filtre de manière simple en Python, puis nous l'avons traduit en C pour que les robots puissent l'exécuter. Nous avons évité les implémentations traditionnelles du filtre utilisant des fonctions trigonométriques, celles-ci n'étant pas incluses dans la bibliothèque de fonctions des Pogobots, et avons préféré une approche utilisant du calcul matriciel.

Expérience

Pour estimer au mieux les matrices de covariance du filtre de Kalman, nous avons effectué plusieurs observations des valeurs de l'IMU avant et après passage dans le filtre. Nous avons décidé de contrôler le robot à la main pour effectuer des déplacements précis : une ligne droite, un virage de 90° vers la droite, et un demi tour en prenant à gauche.

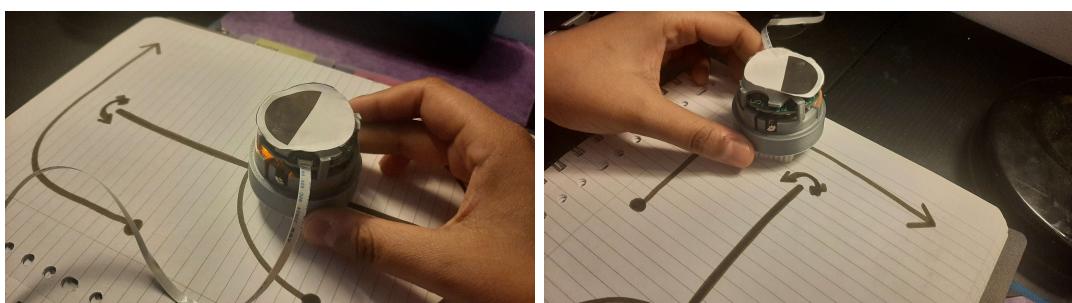


FIGURE 3.4 – Expériences pour le filtre Kalman : trajectoires suivies par les robots

Nous avons répété ces chemins à trois reprises : une pour bouger sans moteur, une avec les moteurs gauche et droite à 512 chacun, une à 1023 chacun. Nous pouvons voir les résultats sur les figures 3.5, 3.6 et 3.7.

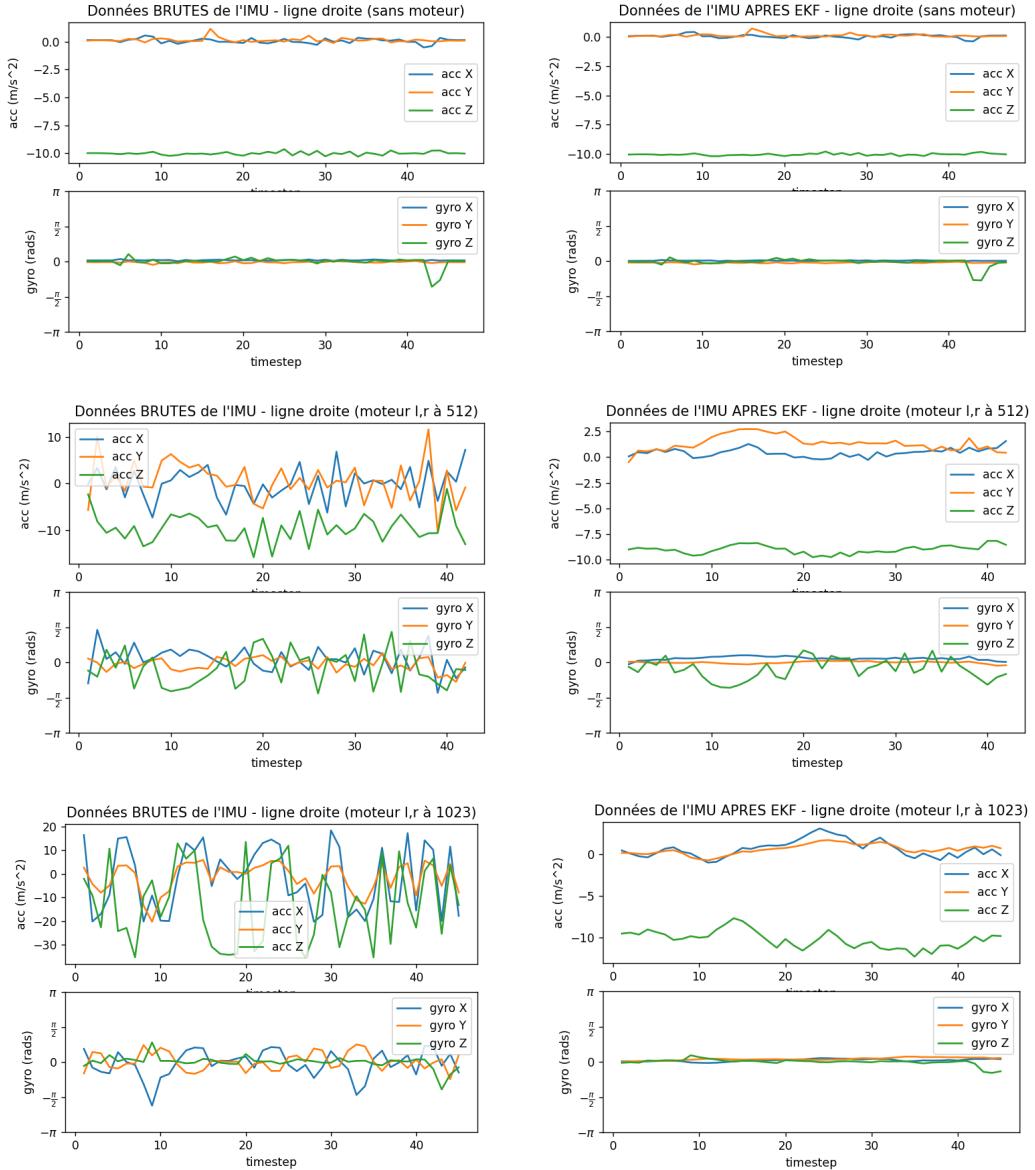


FIGURE 3.5 – Données de l'IMU, trajectoire en ligne droite, avant et après filtre

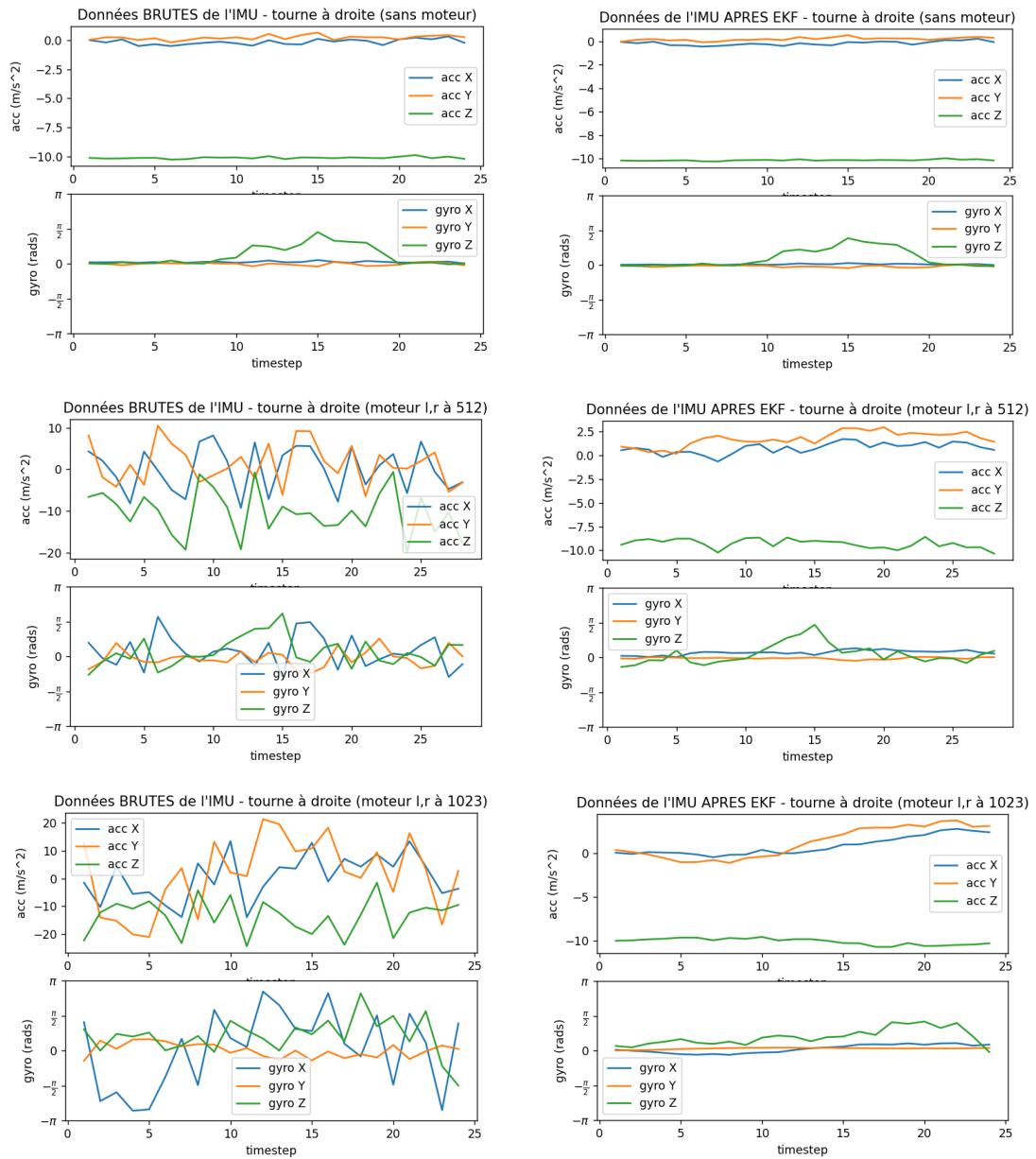


FIGURE 3.6 – Données de l'IMU, virage à droite, avant et après filtre

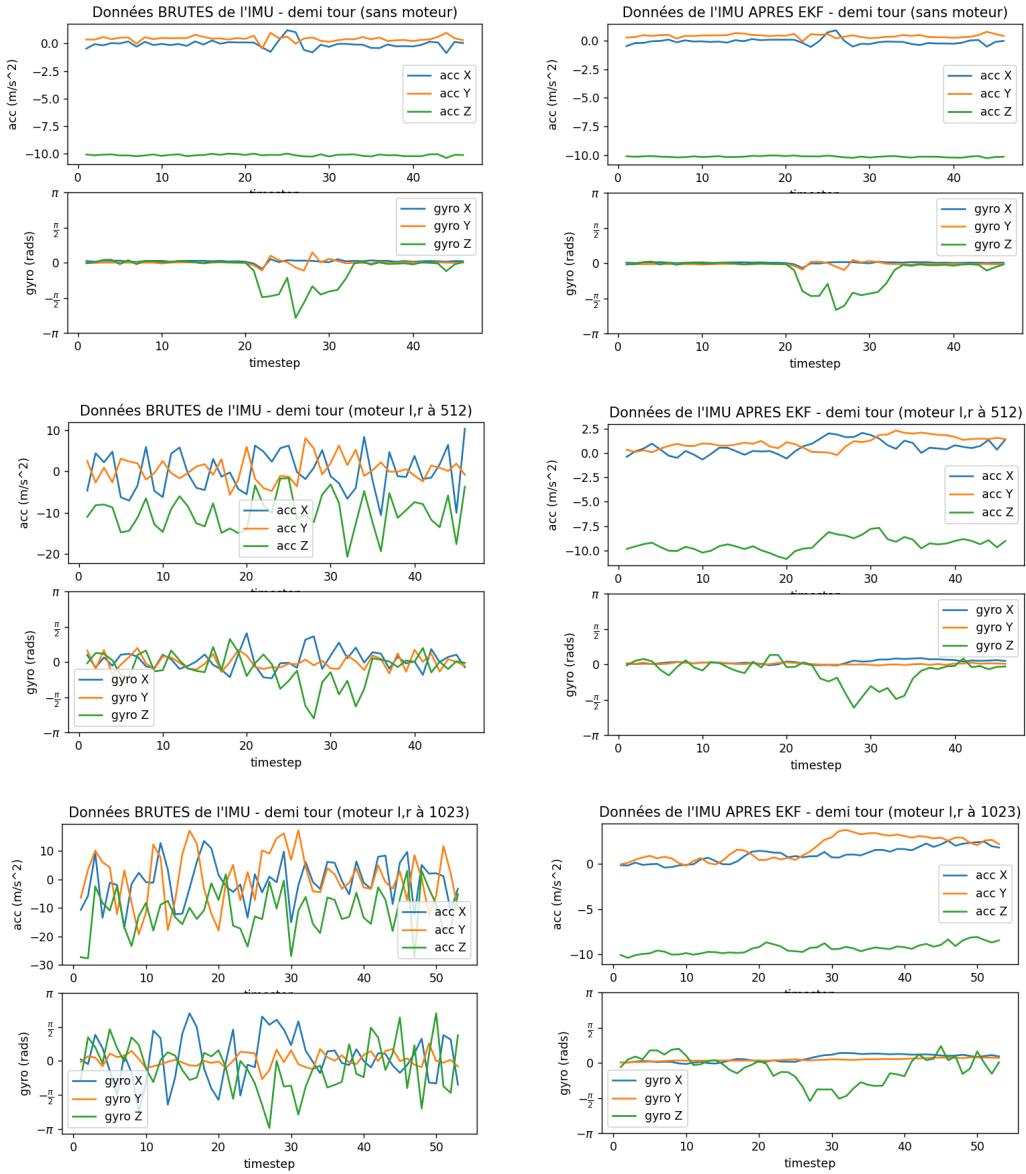


FIGURE 3.7 – Données de l'IMU, demi tour par la gauche, avant et après filtre

Grâce au choix des trajectoires testées, il était facile de prédire le comportement des courbes, en particulier celles décrivant l'accélération ou l'orientation sur l'axe z. Par exemple, pour le virage de 90° vers la droite (figure 3.6), on s'attend à voir un maximum global des mesures du gyroscope en z à $\frac{\pi}{2}$.

Ces graphes confirment que les moteurs apportent un bruit très important, à mesure que la puissance augmente. Nous avons avancé à tâtons en modifiant les matrices du filtre de Kalman d'après l'allure des courbes post filtre. Le tout était d'appliquer un filtre assez efficace pour rendre des courbes propres et proches de nos attentes, sans être trop fort et masquer des changements d'allure importants. Nous avons d'ailleurs fait le choix d'ignorer pour l'instant le cadre où le robot se déplace sur des surfaces en pente : on aplani au possible les valeurs sur les axes x et y, pour permettre en priorité les changements de

valeurs sur l'axe z.

On notera néanmoins qu'en contrôlant à la main la trajectoire du robot, on n'est pas tout à fait dans des conditions réelles de déplacement et on apporte à notre tour une certaine incertitude et un poids qui ne sont pas là d'ordinaire. Ces biais sont un peu difficile à quantifier et ont sans doute en partie rendu le calibrage plus difficile à effectuer.

3.2.2 Calibrage des robots

Après application du filtre de Kalman sur les données de l'IMU, nous avons pu établir un algorithme simple de calibrage des Pogobots. Cet algorithme permet de trouver par expérimentation automatique les puissances à attribuer au moteur droit et au moteur gauche afin que le robot se déplace en ligne droite, pour une vitesse (entre 0 et 1024) souhaitée, définie par l'utilisateur. Le fonctionnement de cet algorithme ci-dessous.

On fait démarrer les moteurs du robot pendant une courte période, puis l'on arrête le robot et l'on ajuste les valeurs moteurs pour minimiser la rotation mesurée pendant le test. On répète ensuite ce même test plusieurs fois pour se rapprocher le plus possible d'une rotation finale nulle.

Limites : les robots ont tous un comportement différent. Alors qu'un robot va avoir une trajectoire pas si mauvaise en ligne droite (jamais parfaite cela dit), un autre va immédiatement se bloquer dans un comportement de toupie, et ce pour les mêmes valeurs moteurs. De plus, certains robots ont les moteurs montés à l'envers. Il est donc difficile de prédire si l'étape de calibrage va fonctionner, et en général on a observé que c'était très dépendant du robot utilisé. Enfin, la calibration se fait une seule fois. L'algorithme ne prend donc pas en compte les problèmes liés au terrain ou aux événements qui pourraient perturber le comportement du robot après la calibration.

Algorithm 2 PogobotCalibrate(*power*, *startupDuration*, *tryDuration*,

numberOfTries, *correction*)

Require: $0 \leq power \leq 1024$

- 1: *leftMotorVal* \leftarrow *power*
 - 2: *rightMotorVal* \leftarrow *power*
 - 3: **for** *i* $\leftarrow 1$ to *numberOfTries* **do**
 - 4: *registeredRotation* $\leftarrow 0$
 - 5: INITIALISER LE FILTRE DE KALMAN
 - 6: DÉMARRER LES MOTEURS À *leftMotorVal* ET *rightMotorVal*
 - 7: ATTENDRE *startupDuration*
 - 8: RÉINITIALISER *timer*
-

```

9:   while timer  $\leq$  tryDuration do
10:    registeredRotation  $\leftarrow$  rotation (gyro Z) du robot, filtrée avec Kalman
11:    timer  $\leftarrow$  elapsedTime
12:   end while
13:   ARRÊTER LES MOTEURS
14:   c  $\leftarrow$  registerRotation  $\times$  correction
15:   if leftMotorVal - c  $\geq$  1024 then
16:     rightMotorVal  $\leftarrow$  rightMotorVal + c
17:   else
18:     leftMotorVal  $\leftarrow$  leftMotorVal - c
19:   end if
20:   ATTENDRE 170 ms
21: end for
22: Return leftMotorVal, rightMotorVal

```

3.3 Implémentation de comportements collectifs

Nous avons eu le temps d'implémenter un comportement collectif sur les pogobots : la détections des voisins d'un robot au sein d'un groupe.

Dans l'algorithme suivant, les robots s'envoient mutuellement des messages, et changent de couleur en fonction du nombre de voisins qu'ils ont.

Border detection : [vidéo d'une simulation](#)

Algorithm 3 BorderDetection(*F*, *probingTime*)

```

1: neighboursMissing = [0, 0, 0, 0]
2: neighboursHere = [MISSING, MISSING, MISSING, MISSING]
3: while True do
4:   neighboursIDs  $\leftarrow$  []
5:   tStartExp  $\leftarrow$  Now
6:   tEndExp  $\leftarrow$  0
7:   if MESSAGE REÇU then
8:     while MESSAGE REÇU do
9:       sender  $\leftarrow$  ID DE LA SOURCE DU MESSAGE
10:      if not sender in neighboursIDs then
11:        append sender to neighboursIDs
12:      end if

```

```

13:      DÉPILER LE MESSAGE
14:      end while
15:      end if
16:      for  $i \leftarrow 1$  to 4 do
17:          if  $neighboursIDs[i] \neq -1$  then
18:               $neighboursMissing[i] \leftarrow 0$ 
19:               $neighboursHere[i] \leftarrow HERE$ 
20:          else
21:               $neighboursMissing[i] \leftarrow neighboursMissing[i] + 1$ 
22:              if  $neighboursMissing[i] \geq 100$  then
23:                   $neighboursHere[i] \leftarrow MISSING$ 
24:              end if
25:          end if
26:      end for
27:       $tEndExp \leftarrow$  TEMPS ACTUEL  $- tStartExp$ 
28:       $i \leftarrow$  NOMBRE DE "HERE"
29:      if  $i = 0$  then ETEINDRE LA LED DU POGOBOT
30:      else if  $i = 1$  then ALLUMER LA LED DU POGOBOT EN cyan
31:      else if  $i = 2$  then ALLUMER LA LED DU POGOBOT EN rouge
32:      else if  $i = 3$  then ALLUMER LA LED DU POGOBOT EN vert
33:      else if  $i = 4$  then ALLUMER LA LED DU POGOBOT EN bleu
34:      end if
35:      if  $genRandomInt[1, 100] \leq 50$  then
36:          ENVOYER UN MESSAGE
37:      end if
38:       $t1 \leftarrow$  NOW  $- t0$ 
39:      ATTENDRE  $((1000000/F - t1)/1000)$  ms
40:  end while

```



FIGURE 3.8 – Photos de différents configurations avec la fonction de BorderDetection (cyan : 1 voisin, rouge : 2 voisins, vert : 3 voisins, bleu : 4 voisins)

Conclusion

Utilité du projet et continuité possible

Les robots Pogobots sont très récents, et toujours en développement. Ce projet P-ANDROIDE a donc été une occasion d'en tester le hardware comme le software, et d'expérimenter sur leurs capacités et limites.

Les tests des capacités fonctionnelles des Pogobots ont permis de présenter des relevés expérimentaux formels à l'équipe de développement. Les tests des capacités de communication en particulier seront utilisés pour étudier et comparer des variantes possibles de l'utilisation du système infrarouge, en faisant par exemple un sacrifice de vitesse pour envoyer les messages sur de plus grandes distances.

L'implémentation du comportement de communication (nombre de voisins) a permis d'effectuer des tests expérimentaux du système de communication dans des situations avec un groupe de robots, et d'étudier les collisions entre messages en fonction du nombre de Pogobots. Nous avons donc pu développer un algorithme fonctionnel dans ces conditions. Si nous disposions de plus de temps, nous aurions souhaité approfondir notre étude en développant des programmes sur les comportements collectifs tels que la phototaxie (la capacité à se déplacer vers une source de lumière) ou le suivi de leader (la capacité à suivre un robot leader dans ses déplacements).

Ces comportements collectifs auraient permis d'explorer les capacités de coordination et de collaboration entre plusieurs Pogobots, et d'observer comment ils interagissent ensemble pour atteindre un objectif commun. Ces expériences auraient également permis d'étudier les mécanismes de communication et de prise de décision au sein d'un groupe de robots.

Malheureusement, faute de temps, nous n'avons pas pu approfondir ces aspects, mais ils représentent des pistes intéressantes à explorer pour de futures recherches sur les Pogobots et leur comportement collectif.

Enfin, le développement d'un filtre de Kalman a permis de rendre les données très bruitées de l'IMU des Pogobots bien plus utilisables, et l'algorithme de calibration des robots devrait permettre de gagner beaucoup de temps sur les expériences réalisées avec les Pogobots à partir de maintenant, puisqu'une calibration manuelle individuelle n'est plus nécessaire. À terme, les valeurs optimales des moteurs devraient pouvoir être sauvegardées dans la ROM des robots pour qu'elles puissent être conservées entre plusieurs utilisations.

Limites du P-ANDROIDE

Ce projet avait toutefois ses limites. En particulier, le peu de matériel à disposition a ralenti la vitesse de nos recherches : nous avions un nombre assez réduit de robots et de programmateurs, et il arrivait régulièrement que ces derniers se cassent ou ne fonctionnent pas correctement. Le fait que ces robots soient extrêmement récents signifie également que beaucoup de tâtonnements et d'expérimentation ont dû être nécessaires à chaque étape du projet, ce qui a pris un temps considérable du projet. Pour finir, les capacités techniques limitées des Pogobots nous ont forcés à développer nos algorithmes en très bas niveau, ce qui était intéressant mais a aussi ralenti l'implémentation de ces algorithmes.

Remerciements

Nous souhaiterions remercier notre encadrant de projet Nicolas Bredèche pour son aide et ses conseils tout au long du projet, Alexandre Guerre qui nous a accompagnés et guidés dans nos expérimentations sur les pogobots, et Lyna Saoucha pour ses encouragements et son soutien amical.

Cahier des charges

Les objectifs de ce projet (pas forcément à réaliser en entier) sont :

1. Réaliser une étude des capacités fonctionnelles d'un pogobot (quantifier la vitesse de déplacement et la vitesse de communication)
2. Implémenter des comportements individuels simples. Plus précisément, implémenter un filtre de Kalman sur les données de l'IMU, correction de la trajectoire en utilisant l'IMU (étude de faisabilité) et détection de collision avec un obstacle en comparant les données de l'IMU et les valeurs moteurs
3. Implémenter des comportements collectifs simples :
 - Exploration au hasard (chaque pogobot explore indépendamment des autres)
 - Détection de bordure (les pogobots n'ayant pas de voisins sur au moins un côté changent de couleur)
 - Phototaxie (agrégation autour d'une source lumineuse, qu'on pourra déplacer au cours du temps)
 - Aggrégation (se rapprocher les uns des autres)
 - Suivi de leader (un pogobot explore, les autres suivent en fil indienne)
 - Boucle infinie (les pogobots forment une boucle, puis se suivent en fil indienne - il s'agit d'une variation du précédent)
 - Déplacement en ligne frontale (les pogobots avancent en ligne et corrige leur trajectoire pour rester aligné)
 - Dispersion (les pogobots s'écartent les uns des autres)

Manuel de l'utilisateur

Un kit de développement logiciel (SDK) nous a été fourni afin de compiler et charger du code sur les pogobots. Ce SDK marche uniquement sur Ubuntu. Il est donc nécessaire de télécharger d'installer Ubuntu sur son ordinateur pour pouvoir travailler sur les Pogobots. Une fois le système d'exploitation installé, il faudra effectuer les étapes suivantes sur le terminal :

B.1 Installer les dépendances

1- Créer un dossier de dépendances :

```
$ mkdir -p dependencies
```

2- Installer les system dependencies :

```
$ sudo apt-get -y install wget bison build-essential clang clang-format
cmake flex gawk git graphviz libboost-all-dev libboost-dev
libboost-filesystem-dev libboost-iostreams-dev libboost-program-options-dev
libboost-python-dev libboost-system-dev libboost-thread-dev libeigen3-dev
libffi-dev libftdi-dev libreadline-dev mercurial pkg-config python python3
python3-dev python3-pip python3-setuptools qt5-default tcl-dev xdot zlib1g-dev
$ sudo pip3 install meson ninja
$ echo "export PATH=\"$PATH:~/local/bin\"" >> ~/.bashrc
```

3- Installer un cross-compiler gcc :

```
$ cd tools
$ wget https://static.dev.sifive.com/dev-tools/freedom-tools/v2020.08/
riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux-ubuntu14.tar.gz
$ tar xvf riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux
-ubuntu14.tar.gz
```

4- Ajouter le compiler au path :

```
$ export GCC_RISCV=$PWD/riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64
-linu
$ echo "export PATH=\"$PATH:$GCC_RISCV\"" >> ~/.bashrc
```

5- Ajouter l'utilisateur au "Dialout Group". C'est-à-dire que l'utilisateur aura un accès complet et direct aux ports série :

```
$ sudo adduser $USER dialout
```

6- Redémarrer la machine.

B.2 Charger une application sur un Pogobot

Une fois les étapes précédentes exécutées avec réussite, votre machine est prête à charger du code sur les Pogobots. Pour cela, il vous suffit de créer un nouveau dossier dans lequel vous mettrez le fichier de votre code "main.c", ainsi que le fichier Makefile qui vous a été fourni. Attention, vérifiez que le path du SDK dans le Makefile est correct, sinon insérez le chemin approprié. Ensuite, ouvrez le terminal dans ce dossier et executez les commandes suivantes : 1- Compiler votre code :

```
$ make all
```

2- Un programmateur et un cable micro-USB vous sont fournis. Ces derniers permettent de connecter les Pogobots à votre machine. Un programmateur est une Program Board relié à une longue bande blanche. Branchez le coté du programmateur ayant une bande bleue au pogobot, puis branchez le coté micro du cable à l'autre coté du programmateur. Enfin branchez l'USB à votre ordinateur.

Il suffit à présent de connecter le Pogobot à votre machine avec la commande suivante :

```
$ make connect TTY=/dev/ttyUSB0
```

Si tout se passe bien, ceci s'affichera sur votre terminal :

```
maxence@maxence-VirtualBox:~/pogobot-sdk-main/examples/demo_d2_1_mono$ make connect TTY=/dev/ttyUSB0
.../..../tools/litex_term.py --serial-boot --kernel ./build/firmware.bin --kernel-addr 0x260000 --safe /
/dev/ttyUSB0
/home/maxence/pogobot-sdk-main/examples/demo_d2_1_mono/.../tools/litex_term.py:575: DeprecationWarning:
setDaemon() is deprecated, set the daemon attribute instead
    self.reader_thread.setDaemon(True)
/home/maxence/pogobot-sdk-main/examples/demo_d2_1_mono/.../tools/litex_term.py:604: DeprecationWarning:
setDaemon() is deprecated, set the daemon attribute instead
    self.writer_thread.setDaemon(True)
□
```

FIGURE B.1 – Message de réussite de connexion de l'ordinateur au Pogobot

Appuyez ensuite sur la touche "entrée". Vous êtes à présent dans le prompt du Pogobot (l'invite de commandes du robot). Enfin, insérez la commande suivante :

```
$ serialboot
```

Félicitations ! Votre code est maintenant chargé sur le Pogobot et va s'exécuter dans quelques secondes. Vous pouvez débrancher le Pogobot une fois le programme chargé dessus, sauf si votre code contient des print dont vous avez besoin..