Projet ANDROIDE :
robotique collective et apprentissage en ligne

Rapport

Parham SHAMS, Tanguy SOTO
UPMC 2017

Table des matières

1 Introduction

Avec l'augmentation de la puissance de calcul et l'intervention toujours croissante de l'informatique et de l'électronique dans notre société, les robots sont aujourd'hui un des enjeux majeurs de notre futur. Afin de permettre à ces robots de se comporter de façon toujours plus complexe et fiable, l'apprentissage est l'une des solutions les plus étudiées actuellement. Dans ce projet, nous avons étudié l'une des nombreuses méthodes existantes pour l'apprentissage : les algorithmes évolutionnistes. Dans le but final d'obtenir un comportement d'éviteur d'obstacles voire de suivi de lumière, nous verrons comment, en partant d'une implémentation dite en ligne ou embarquée, il est possible de l'étendre vers un algorithme distribué dans un essaim de robots.

Nous avons été amenés à collaborer avec Mario Viti, un étudiant du Master 1 IMA de l'UPMC. Nous détaillerons les impacts de cette collaboration dans notre projet.
Avant d'arriver à la partie purement algorithmique, nous avons dû prendre en main et apporter des modifications à la plate-forme utilisée : OctoPY. On présentera cet aspect de notre projet dans une première partie avant de nous plonger dans la partie algorithmique où nous vous présenterons les différents algorithmes codés, ainsi que les résultats associés et les problèmes rencontrés lors de leur écriture et mise en oeuvre.

2 Aspect technique et matériel

2.1 Les Robots

Les robots que nous avons utilisés, sont des robots très simples à 2 roues mesurant environ 12cm de côtés : les Thymios. Dans les grandes lignes, ils sont capables de se déplacer, émettre des sons et lumières et disposent de capteurs tels que des capteurs de proximités ou même un thermomètre.

Vous pouvez retrouver le détail des capacités d'un thymio dans l'annnexe A.

FIGURE 1 – Un Thymio

Étant seulement légèrement programmables et ne pouvant être connectés avec le monde extérieur que par l'usb, ces robots sont contrôlés par des Rapsberry PI qui ne sont rien d'autre que des micro-ordinateurs posés sur les Thymios grâces à un socle fabriqué par impression 3D. De plus, chaque Rasberry PI nous est fourni avec une caméra connectée à celui-ci.

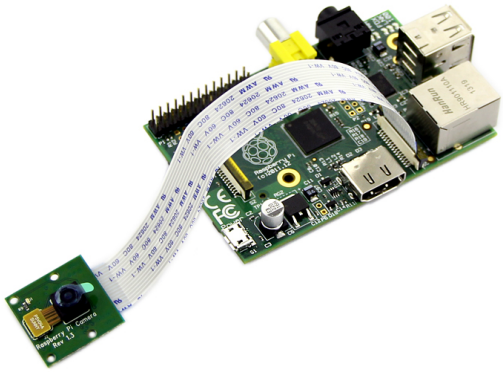


FIGURE 2 – Un Raspberry PI muni de sa caméra

Ainsi, nous disposons d'un robot en 3 parties assez complet doté d'un ordinateur UNIX qu'il est facile de programmer et d'exécuter à distance via le WIFI (intrégré au Raspberry PI). Pour la suite, nous appellerons robot cette triple association thymio-raspberry-camera.

2.2 La plate-forme OctoPY

Ces robots sont relativement bon marché et permettent donc de mener des expériences avec un nombre important d'entre eux. Notre projet s'est déroulé à l'ISIR et nous disposions donc d'une arène de $2.5m * 2.5m$ où jusqu'à

30 de ces robots peuvent évoluer sans encombre.

Afin de pouvoir développer des expériences sur un si grand nombre de ces robots sans avoir à se soucier des problèmes techniques tels que le réseau ou la parallélisation des commandes sur chaque robot, une plate-forme a été développé en Python en 1 mois environ par Arthur Bernard (ancien doctorant à l'ISIR). Cette plate-forme fournit également une API permettant d'envoyer et de recevoir des informations au Thymio en Python.

A notre arrivée, la plate-forme était juste fonctionnelle. Elle permettait effectivement de lancer des expériences depuis un ordinateur central mais ne gérait pas encore les cas d'erreur et contenait encore de nombreux bugs, laissant parfois les robots en route sans pouvoir les arrêter. Notre première tâche a donc été de continuer cette plate-forme afin qu'elle soit fiable et complète, autant pour les futurs utilisateurs que pour nous qui allions devoir l'utiliser pendant notre projet.

Voici une liste non-exhaustive des améliorations que nous avons pu faire :

Nouvelles fonctionnalités : Parmi les commandes disponibles dans OctoPY depuis l'ordinateur, manquait cruellement la possibilité d'envoyer de nouveaux fichiers (de code ou même de configuration) vers tous les robots à la fois. De même, les expériences généraient des logs sur les Raspberry PI, mais il n'était pas possible de tous les récupérer depuis OctoPY. Nous avons donc ajouté deux commandes symétriques l'une de l'autre : **put** et **get**. La deuxième permettant aussi de supprimer des fichiers des robots (seulement lorsque le transfert est un succès).
Par ailleurs, l'API permettant de communiquer avec le Thymio autorise désormais l'utilisation de l'accéléromètre.

Changements dans l'organisation : Jusqu'à maintenant, que ce soit sur l'ordinateur central ou sur la partie embarquée dans les Raspberry d'OctoPY, il n'y avait pas de séparation claire entre les fichiers faisant partie du noyau d'OctoPY et les fichiers créés par les utilisateurs. Les dossiers ont donc été remodelés de façon à ce qu'un utilisateur puisse s'y retrouver plus facilement et qu'il ne risque pas de modifier un fichier du noyau sans le vouloir.
Par la même occasion, nous avons centralisé les différents codes d'utilisateurs existant pouvant servir de librairie à importer dans de futures expériences. Auparavant, ces fichiers étaient copiés/collés pour pouvoir être utilisés. Le code de Mario Viti a notamment été intégré dans ce nouveau dossier *tools*.

Fiabilisation : Cette dernière partie a de loin été celle qui nous a demandé le plus de travail. En effet, OctoPY est une plateforme composée de nombreux Threads tournant sur des machines différentes et il n'est pas facile dans ce cas d'identifier tous les cas d'erreur pouvant arriver. Nous avons fait de notre mieux pour synchroniser ces différents processus lors de l'arrêt de la plate-forme ou de l'apparition d'une erreur quelconque.

Quelques soient les changements apportés, ils ont toujours été fait dans le but de simplifier le développement des utilisateurs. Bien entendu ces modifications ont été accompagnées d'une documentation sous la forme de mise à jour du manuel déjà existant.

Un diagramme complet résumant le fonctionnement de la plate-forme est disponible en annexe B. En tant qu'utilisateurs cherchant à implémenter plusieurs comportements, nous serons amenés dans la partie suivante à développer 2 à 3 "simulations" selon la nomenclature OctoPY.

3 Algorithmes d'apprentissage en ligne

Nous allons maintenant vous présenter les différents algorithmes que nous avons codés. L'objectif est d'arriver, à partir d'un comportement (et donc d'un génome) aléatoire, à un comportement de suivi de lumière. Or, pour suivre la lumière on veut aussi évidemment éviter les obstacles. Les algorithmes suivants ont donc pour but de converger vers un comportement éviteur d'obstacles. Cependant, ils incluent déjà dans leur génome un gène à appliquer sur un capteur de lumière et peuvent devenir des comportements de suivi de lumière en modifiant la fonction de fitness utilisée.

3.1 Principes de nos algorithmes

Nous avons principalement codé deux algorithmes évolutionnistes. Le premier correspond à un algorithme d'apprentissage en ligne non distribué servant de témoin tandis que le deuxième est distribué. Ces deux algorithmes sont respectivement présentés dans l'article [1] et dans les articles [2] et [3]. Le principe général des algorithmes évolutionnistes est le suivant :

```
Generer une population initiale P de N individus

pour g de 1 a NB_GENERATIONS, faire :
    pour i de 1 a N, faire :
        evaluer(i) # on calcule le score (ou fitness) de i

    j = selection(P) # on selectionne k meilleurs individus
    j = mutation(j) # ces individus mutent selon certains
    parametres
    P = insertion(P,j) # on insere ces nouveaux individus dans la
     population

retourner le meilleur individu de G
```

Dans notre cas les algorithmes sont exécutés sur chaque robot et évalue le comportement du robot même sur lequel il se déplace. De ce fait, la population à évaluer est toujours de taille 1 et l'individu courant correspond aux paramètres (génome) du comportement courant à tester.
Tout comme dans [1], [2] et [3], notre génome est constitué des poids d'un réseau de neurones mono-couche. Les entrées sont :

- 7 entrées continues correspondant aux capteurs de proximité du Thymio
- 1 entrée continue servant de biais
- 1 entrée binaire associé à notre futur capteur de lumière

Les sorties quant à elles sont au nombre de 2, une pour chaque moteur du robot. On obtient 18 poids et donc un génome de 18 gènes. Par ailleurs toutes les valeurs sont normalisées entre -4 et 4 pour notre algorithme témoin et de -1 à 1 pour l'algorithme distribué. Notre fonction d'activation est la suivante, elle permet d'obtenir des sorties dans l'intervalle de vitesse de rotation des moteurs :

$$s(x) = \frac{1}{1+e^{-a*x}} * 1000 - 500$$

a pouvant varier selon l'algorithme et la nervosité souhaitée des robots.

Par rapport à l'algorithme décrit dans [1], nous n'avons pas effectué de modification sur la sélection ni la mutation qui se fait bien par sigma adaptatif. De même les phases de *recover* et de *reévaluation* sont bien présentes. Notre fitness est aussi :

$$\sum_i f(x) \qquad \forall i \in 0, 1, ..., evalTime$$

où $f \in [0, 1]$ et sera décrite dans la partie suivante

Enfin, pour notre algorithme distribué, la sélection ce fait par un tournoi de taille k parmi les génomes reçus des autres robots comme précisé dans

[2]. L'envoi d'un génome aux robots voisins est réalisé selon le modèle PGTA décrit dans [3], c'est-à-dire avec une probabilité p proportionnelle à la fitness du robot. Ainsi la fitness est ici la moyenne des w dernières valeurs $f(x)$ calculées à chaque itération d'une génération afin d'obtenir une valeur entre 0 et 1. La mutation est réalisée par mutation gaussienne avec un sigma fixe sur le génome sélectionné ou sur le génome courant si aucun n'a été reçu.

3.2 Mise en place

Pour mettre en place ces algorithmes nous avons créé des simulations via la plate-forme OctoPY. La première simulation se nomme *FollowLightGenOnline* tandis que la deuxième se nomme *FollowLightGen*. Nous avons aussi créé une simulation *ApplyEvolvedGenome* pour nous permettre de tester les génomes évolué obtenus.

En ce qui concerne la partie codage des algorithmes (et des simulations), nous n'avons rencontré aucun problème particulier car toutes les fonctionnalités nécessaires à leur implémentation sont présentes en Python. Les problèmes rencontrés résident donc principalement sur la manière de déterminer la fitness à utiliser pour notre problème et la manière de permettre à un robot de communiquer avec ses voisins seulement.

Enfin, dans l'éventualité de chercher à aboutir à un comportement suiveur de lumière et non seulement éviteur d'obstacles, nous avons été amenés à créer par le biais de la caméra, un capteur de lumière. Celui-ci nous permet de savoir si le robot perçoit plus de lumière à gauche ou à droite.

Fitness : Pour les deux algorithmes, nous avons d'abord pensé utiliser la même fitness suivante :

$$f = v_{translation} * (1 - v_{rotation}) * (1 - mean(senseurs))$$

Elle permet de maximiser la vitesse de translation, minimiser la vitesse de rotation et enfin éviter les obstacles. La vitesse de translation nous a cependant posé problème car nous ne pouvons qu'utiliser la vitesse de rotation des moteurs du Thymio. Or, nous avons remarqué que le robot pouvait de retrouver "collé" dans un mur, complètement immobile, mais tout en ayant ses roues qui tournent à grande vitesse en patinant. Ainsi, la vitesse de translation dans une telle situation était quasiment à son maximum au lieu d'être nulle et cela avait de graves conséquences sur la fitness.
Pour pallier à ce problème, nous avons alors recherché la possibilité d'avoir une réelle valeur de la vitesse du robot. Pour cela, nous avons eu l'idée d'utiliser l'accéléromètre du Thymio afin d'avoir une estimation de sa vitesse. C'est

pour cette raison que nous avons ajouté cette fonctionnalité à la plate-forme OctoPY. Malheureusement, cette méthode a été un échec car l'accéléromètre du Thymio est loin d'être assez fin pour faire ce genre d'estimations. D'après nos recherche cet accéléromètre est plutôt utilisé pour détecter des pentes. Nous avons finalement fini par changer notre fitness en :

$$f = v_{translation} * (1 - v_{rotation}) * (1 - max(senseurs))$$

En effet, cela permet de résoudre le cas qui nous dérangeait car dès qu'un capteur de proximité du Thymio est activé sa fitness s'en trouve annulée et la valeur faussement élevée de la vitesse de translation n'a alors aucun impact sur la fitness.

Voisinage d'un robot : Un deuxième enjeu pour nous fut l'implémentation de la reconnaissance par un robot de ses voisins. En effet, l'algorithme distribué stipule qu'un robot ne doit transmettre son génome qu'à ses voisins et non à toute la population. C'est dans cette optique que notre encadrant Nicolas Bredeche nous a proposé de collaborer avec Mario Viti. Comme mentionné précédemment, il s'agit d'un étudiant en Master 1 IMA de l'UPMC réalisant son projet sur la reconnaissance de tags. L'idée était donc, une fois son travail terminé et fonctionnel, d'utiliser cela pour qu'un robot communique seulement avec ses voisins. De plus, c'était l'occasion d'intégrer son travail dans les *tools* de la plate-forme afin de rendre cette fonctionnalité facilement disponible à l'avenir.

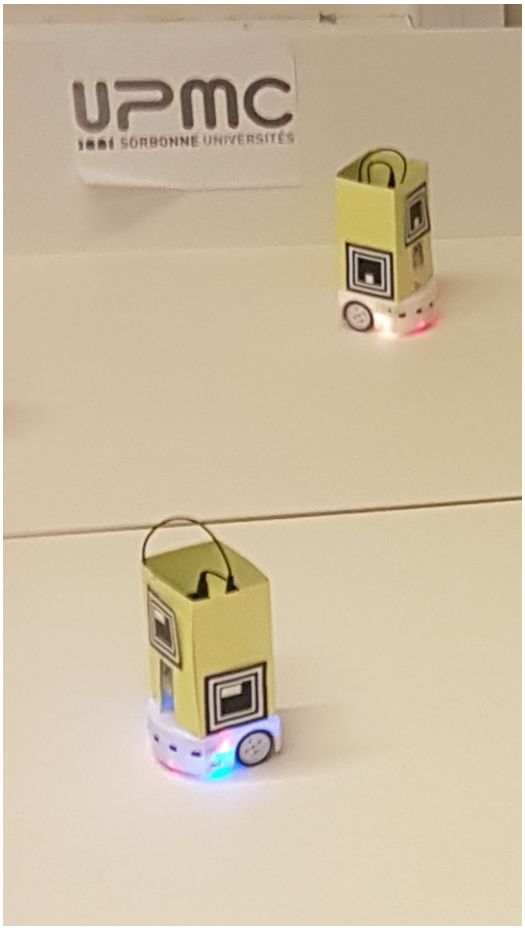


FIGURE 3 – Deux robots dans l'arène coiffés de tags

Le projet de Mario Viti permet entre autres de reconnaître un tag en nous renvoyant une ID, la distance à celui-ci et son orientation.

Ainsi, avec l'aide de Mario nous avons coiffé les robots des 4 tags ayant une même ID. Un robot envoie donc des messages aux robots correspondant aux tags qu'il voit. Sachant que la distance de reconnaissance d'un tag est d'environ 1m50, cela représente bien le voisin d'un robot. Enfin, nous avons évidemment utilisé des tags permettant à chaque robot d'avoir une ID unique à laquelle nous avons associé adresses IP des Raspberry PI. Les tags ont une plage d'ID allant de 0 à 511 car l'ID est codée sur 9 bits. Néanmoins, ce n'est qu'une approximation de voisinage puisque deux robots peuvent être géographiquement voisins mais ne pas s'envoyer de messages car ils ne se voient pas via la caméra.

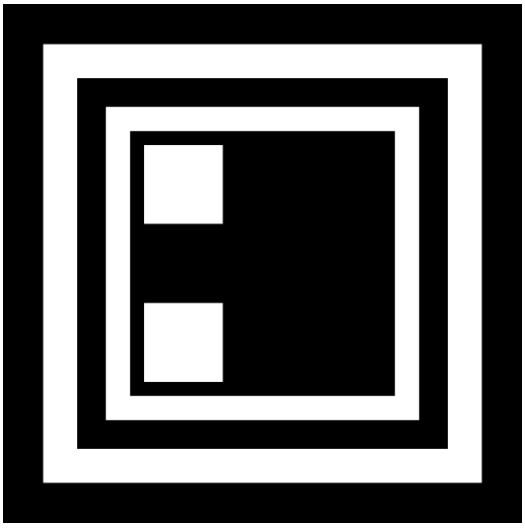


FIGURE 4 – Un tag d'ID 5

3.3 Résultats

Une fois tous ces problèmes de mise en place réglés, nous avons pu passer aux expérimentations réelles et nous allons vous présenter ici les résultats de celles-ci, ainsi que l'analyse qui en découle et le cheminement de pensée qui a été le nôtre. Notons par ailleurs que nos expériences ne sont que rarement très longues car nous nous sommes heurtés à quelques difficultés. En effet, il n'était pas rare d'avoir un robot qui tombait à la renverse pendant une expérience ou tout simplement un problème réseau au milieu d'une expérience ou encore des problématiques de batterie. Cependant, les expériences ont duré assez longtemps afin de nous faire une idée précise de ce que l'on essayait de montrer.

Pour toutes les expériences suivantes nous avons utilisé cinq robots évoluant dans une arène rectangulaire de $2.5m * 1.5m$.

Algorithme en ligne Nous avons commencé par tester l'algorithme en ligne qui sert d'algorithme "témoin". En effet, le but est de savoir si l'on arrive à converger vers le comportement souhaité (ici éviteur d'obstacles) sans se soucier de la partie communication entre robots. Voici les résultats obtenus lors de cette expérience. Les résultats étant assez similaires sur les cinq robots ayant pris part à l'expérience, nous avons fait le choix de seulement mettre en avant deux de ces robots qui paraissaient les plus évolués :

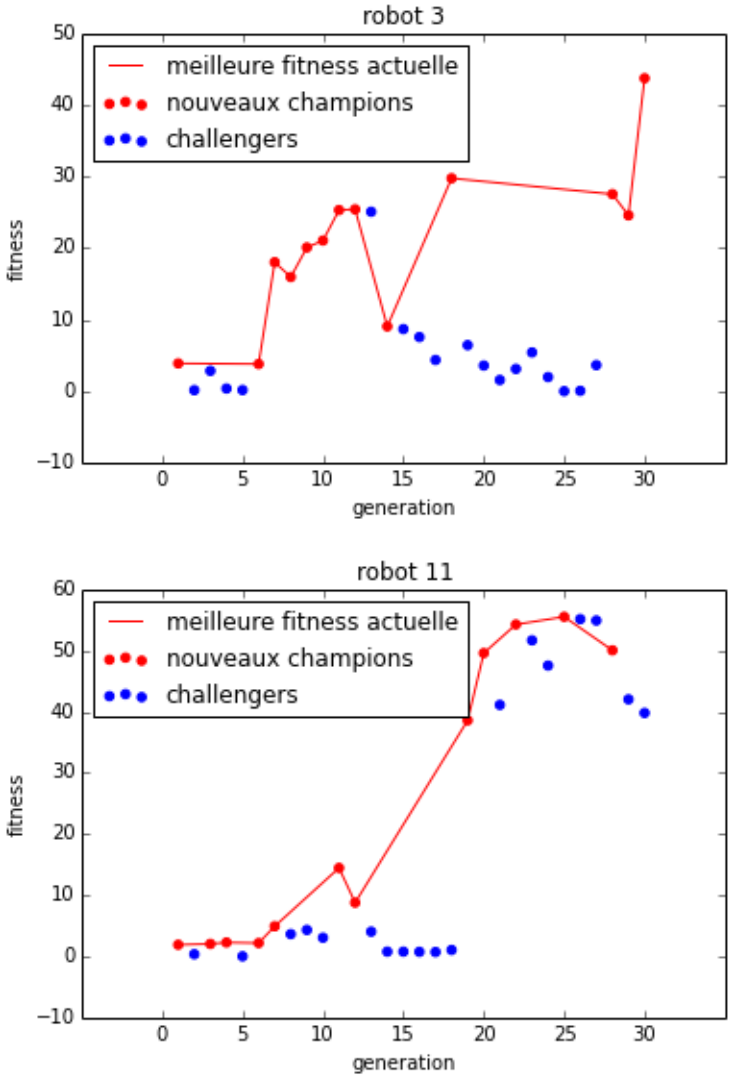


FIGURE 5 – Évolution de la fitness de 2 robots durant notre algorithme embarqué non-distribué ($p_{reevaluate} = 0.2$, $min_{sigma} = 0.1$, $max_{sigma} = 4$)

L'évolution de la fitness des deux robots au cours des générations est globalement croissante et atteint environ 50 pour les deux robots après être parti d'une fitness quasiment nulle. Les moments où le nouveau champion obtient une fitness plus basse que précédemment correspondent aux reévaluations d'un même champion. En plus de cela, nous avons remarqué que les robots agissaient déjà comme un bon éviteur d'obstacles et nous considérons alors que l'expérience est un succès et que cela valide notre simulation. Ainsi, nous pouvons maintenant passer à la deuxième simulation qui correspond à

l'algorithme distribué.

Algorithme en ligne distribué Maintenant que nous avons validé l'algorithme témoin, nous allons tester si nous arrivons à mettre en place le même comportement via l'algorithme distribué. Nous testons donc la convergence de notre simulation. Pour ce test, un robot n'envoie son génome qu'à ses voisins (grâce à la méthode décrite précédemment). Voici les résultats de ce test :

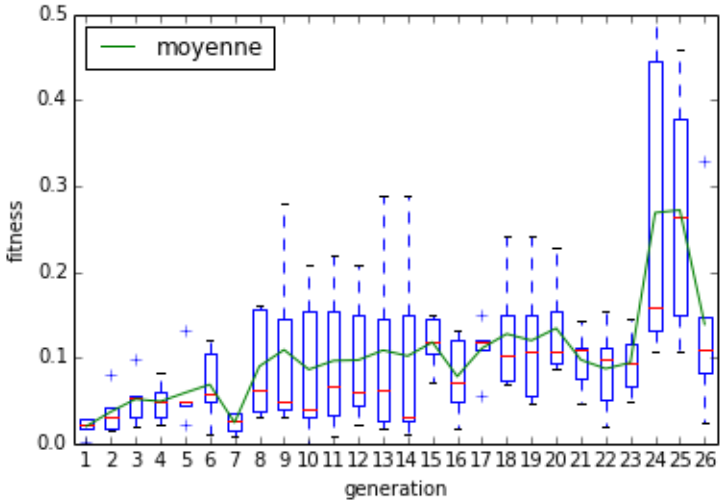


FIGURE 6 – Évolution de la fitness sur 5 robots durant notre algorithme embarqué distribué ($sigma = 0.1$)

Nous voyons très clairement que le comportement des robots ne converge pas. Cette impression est bien corroborée par la courbe de résultats. En effet, hormis certains "pics" à 0.3, la fitness moyenne des robot plafonne à 0.1. Devant cet "échec", nous avons retenté l'expérience plusieurs fois en changeant certains paramètres tels que le sigma, la durée d'une génération ou encore la méthode de diffusion (ne pas envoyer qu'à ses voisins). Or, le résultat obtenu était toujours le même.
Nous donc avons conjecturé l'éventualite d'un nombre trop faible de robots engendrant une diversité génétique trop faible ne permettant pas de converger vers le comportement voulu. Cependant, cela ne restait qu'une cause possible parmi beaucoup d'autres. Afin d'écarter des hypothèses, nous avons d'abord fait un test qui nous a permis de déterminer si un "bon" comportement (c'est-à-dire avec une bonne fitness) se propageait correctement dans la population. Ainsi, nous avons utilisé un génome correspondant à un génome évolué grâce

au premier algorithme présenté (témoin) et qui est un bon comportement éviteur d'obstacles. Ce génome est assigné à l'un (et un seul) des robots, ce robot sera désigné comme le leader. Voici les résultats du test :

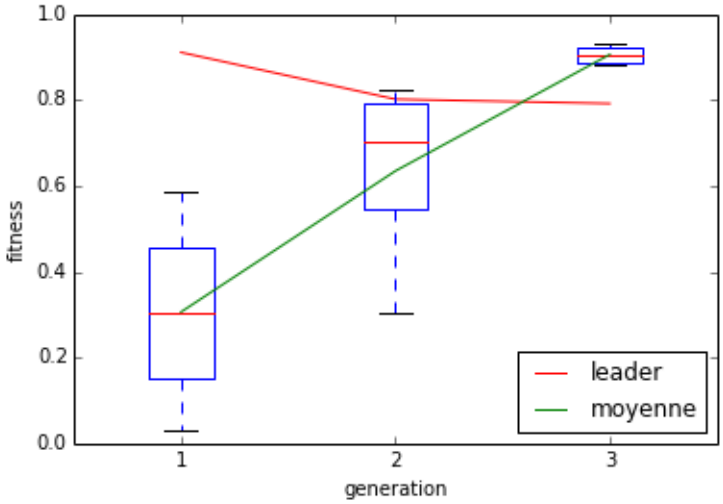


FIGURE 7 – Évolution de la fitness sur 4 robots durant notre algorithme embarqué distribué avec présence d'un leader déjà évolué et avec envoi du génome de chaque robot à tous les autres

Les résultats sont vraiment positifs, en effet les robots non leader commencent avec une fitness moyenne de 0.2 pour finir au bout de 3 générations déjà à une fitness de 0.9. De plus, du point de vue de l'expérience nous avons pu clairement constater la transmission par le leader de son comportement. Cela nous permet donc d'écarter l'hypothèse selon laquelle la non-convergence est dûe à la mauvaise propagation d'un comportement. Mais ceci est vrai seulement dans le cas où un robot communique avec tout le monde. Nous décidons donc de faire le même test pour voir si notre voisinage fonctionne correctement.

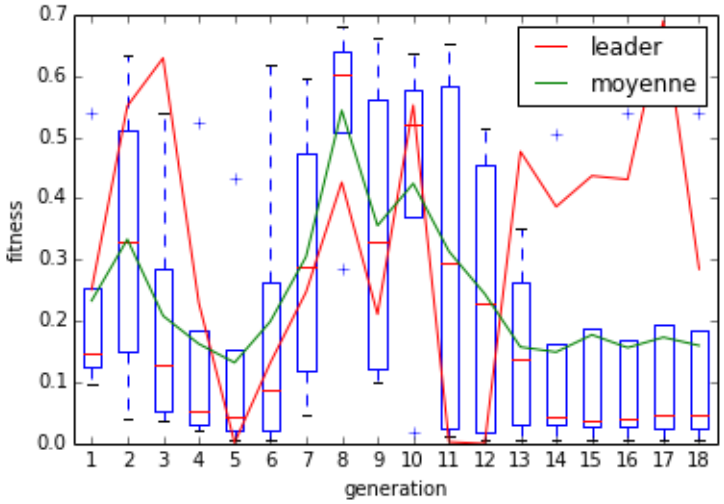


FIGURE 8 – Évolution de la fitness sur 4 robots durant notre algorithme embarqué distribué avec présence d'un leader déjà évolué et avec envoi du génome à ses voisins uniquement

Du point de vue de l'expérience, les résultats sont très bons. En effet, on observe en particulier deux robots fonçant au début tout droit et s'encastrant dans un mur avant d'être "débloqué" par le leader lorsqu'il les voit et reconnaît via la caméra. On remarquera des passages par des fitness très basses (même pour le leader) alors que nous constatons un très bon comportement. Cela est dû au fait que la fitness s'annule lorsqu'un robot a un de ses capteurs activé au maximum. Ainsi, si un robot a un comportement éviteur d'obstacle mais qu'il croise beaucoup de robots et de murs sa fitness restera tout de même basse.

Nous avons donc réussi à partir d'un leader à propager un comportement éviteur d'obstacles dans le cadre de l'algorithme. Cependant, sans présence de leader, il ne semble pas y avoir de convergence. Cela peut être tout d'abord dû à un mauvais jeu de paramètres de notre part ne permettant de converger que très lentement. Cela peut aussi être dû au nombre trop peu important de robots dont nous disposons ou au fait que nos expériences n'ont pas tourné suffisament longtemps.

Nous avons aussi tenté de faire converger les robots vers une comportement de suivi de lumière en multipliant notre fitness par la quantité de lumière perçue par le robot mais les résultats n'ont rien donné de probant. La fonction fitness employée semble trop complexe pour amener à une convergence. Une solution pourrait être d'utiliser une fonction multi-critères : un

pour l'évitement d'obstacle puis un pour la quantité de lumière perçue. On calculerait ensuite les optima par tri lexicographique ou en choisissant parmi les Pareto-optimaux.

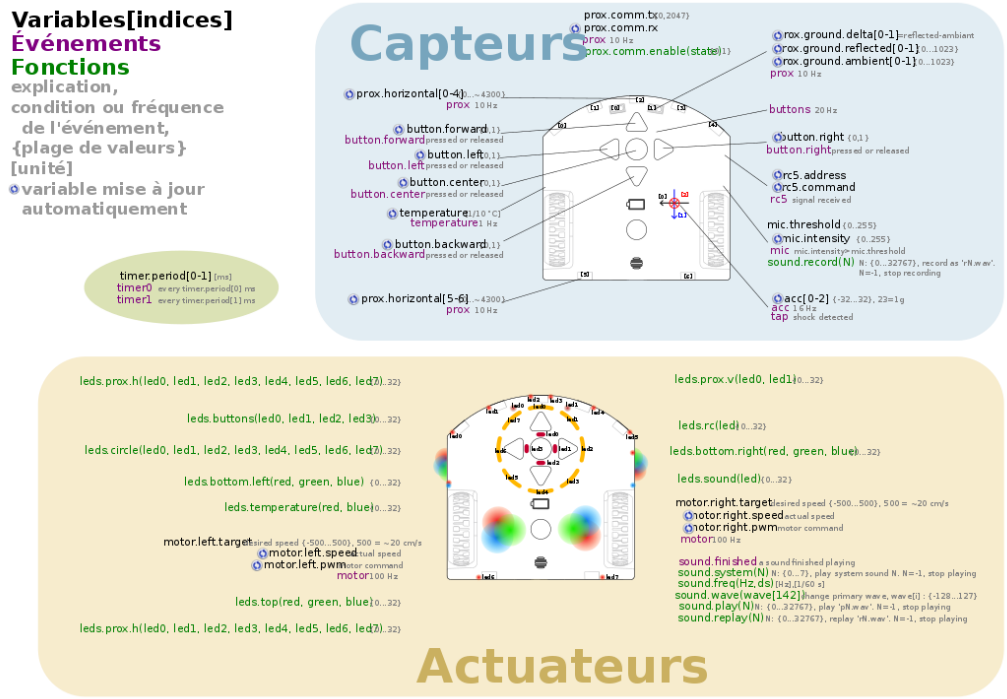
4 Conclusion

Au travers de ce projet, nous avons eu un aperçu des principaux enjeux et difficultés liés à la robotique collective et l'apprentissage en ligne d'un comportement. Que ce soit sur le plan technique ou théorique nous avons été amenés à trouver des solutions aux problèmes que avons rencontrés, que ce soit par nous même ou avec l'aide de notre encadrant. La collaboration avec Mario Viti nous a permis d'acquérir un certain recul sur notre projet et d'augmenter notre champ de possibilités, ce qui est un atout majeur dans un domaine aussi vaste que la robotique.

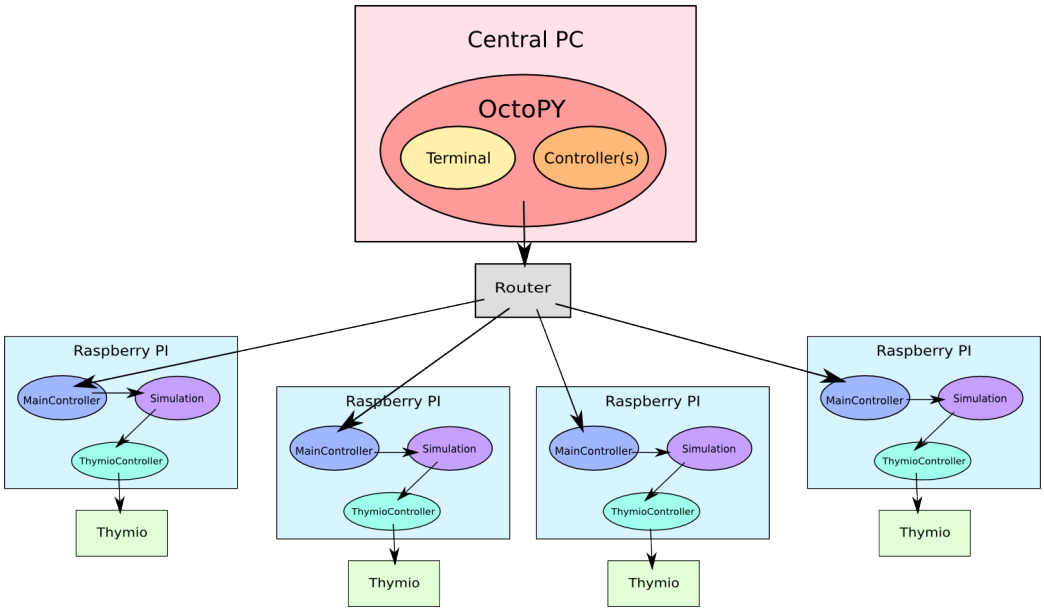
Dans la continuité de notre travail, il pourrait être intéressant de pousser nos recherches et nos tests afin d'aboutir à un comportement de suivi de lumière ou même de transport collectif.

Annexes

A Spécifications du Thymio



B Diagramme de fonctionnment d'OctoPY



C Références

[1] N. Bredeche, E. Haasdijk, and A. E. Eiben, "On-line, on-board evolution of robot controllers," *Lecture Notes on Computer Science*, vol. 5975, pp. 110–121, 2010.

[2] J.-M. Montanier, S. Carrignon, and N. Bredeche, "Behavioural specialisation in embodied evolutionary robotics : Why so difficult ?," *Frontiers in Robotics and AI*, vol. 3, no. 38, 2016.

[3] R. A. Watson, S. G. Ficici, and J. B. Pollack, "Embodied evolution : Distributing an evolutionary algorithm in a population of robots," *Robotics and Autonomous Systems*, vol. 38, pp. 1–18, 2002.