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 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.

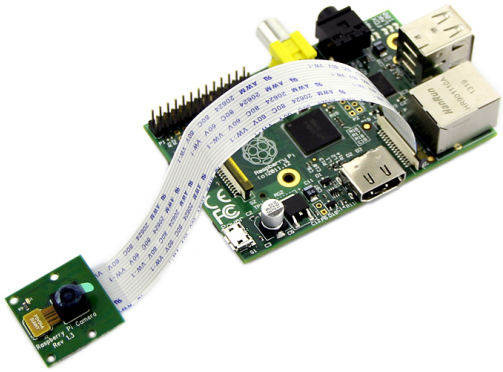


FIGURE 2 – Un Raspberry PI muni de sa caméra

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 ajouter deux commandes symétriques l'une de l'autre : **put** et **get**. La deuxième permettant aussi de supprimer des fichiers des robots.
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 remodeler de façon à ce qu'un utilisateur puisse s'y retrouver plus facilement et qu'il ne risque pas de modfier un fichier du noyau sans le vouloir.
Par la même occasion, nous avons centraliser les différents codes d'utilisateurs existant pouvant servir de librairie à importer dans de futurs expériences. Auparavant, ces fichier étaient copier/coller pour pouvoir être utiliser. Le code de Mario Viti à notamment été intégré dans ce nouveau dossier *tools*.

Fiabilisation : Cette dernière partie à de loin été celle qui nous a demandé le plus de travail. En effet, OctoPY est une plateforme composée de nombreux Thread 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 soit les changements apportés, ils ont toujours été fait dans le but de simplifié le développement des utilisateeurs. 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 Explication algos

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

Le premier algorithme consiste en (1+1)-es avec recover/reevaluate etc ...

Le deuxième algorithme consiste en ....

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.

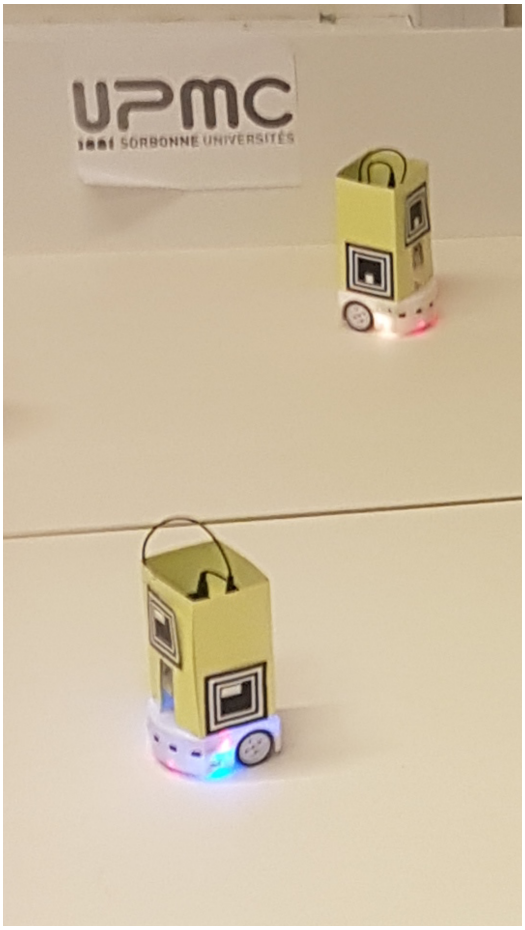


FIGURE 3 – Deux robots dans l'arène coiffés de tags

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.

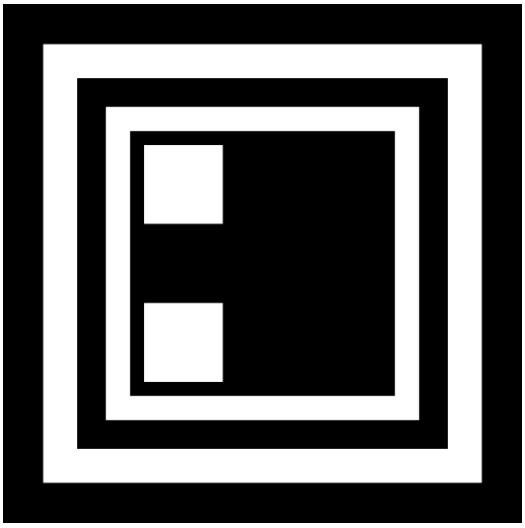


FIGURE 4 – Un tag d'ID 5

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 ceux-ci, ainsi que l'analyse qui en découle et le cheminement de pensée qui a été le nôtre.

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".

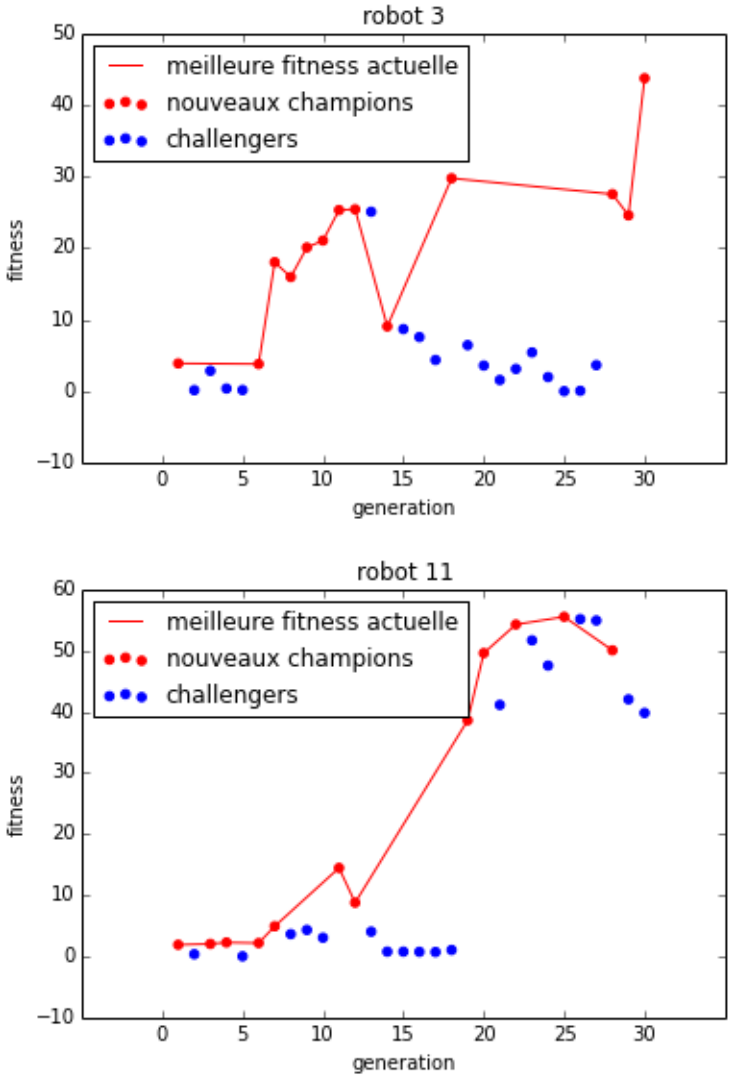


FIGURE 5 – Évolution de la fitness de 2 robots durant notre algorthme embarqué non-distribué

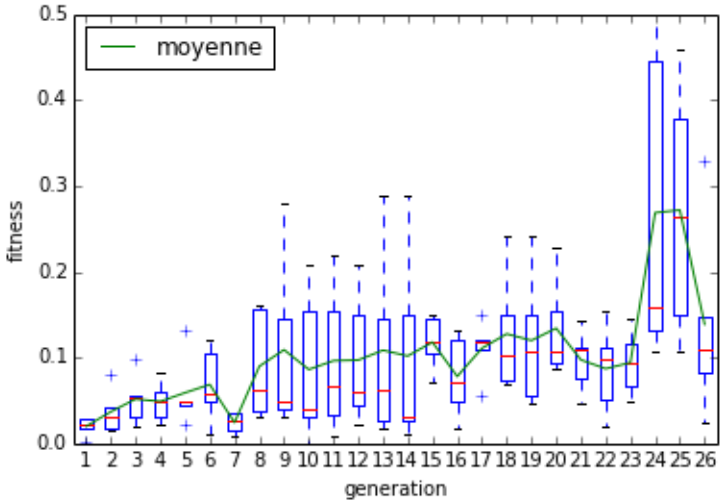


FIGURE 6 – Évolution de la fitness sur 5 robots durant notre algorithme embarqué distribué

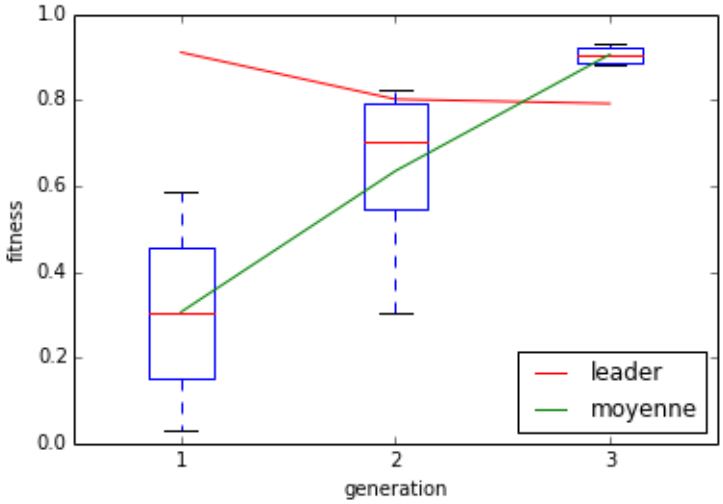


FIGURE 7 – Évolution de la fitness sur 4 robots durant notre algorithme embarqué distribué avec présence d'un leader déjà évolué et avec envoi du génome de chaque robot à tous les autres

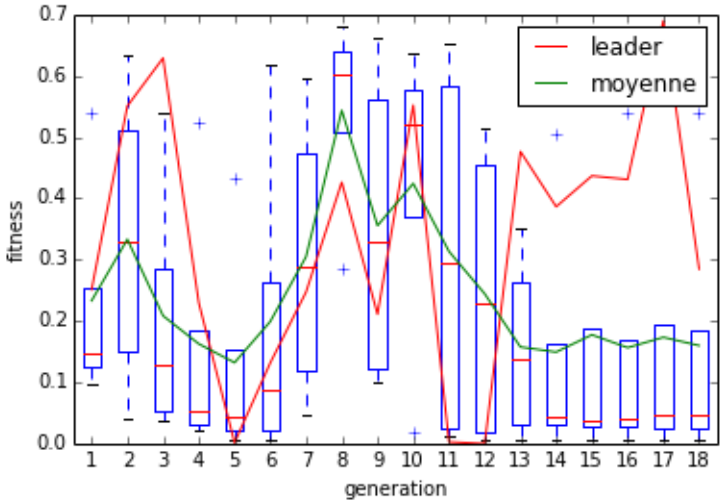


FIGURE 8 – Évolution de la fitness sur 4 robots durant notre algorithme embarqué distribué avec présence d'un leader déjà évolué et avec envoi du génome à ses voisins uniquement

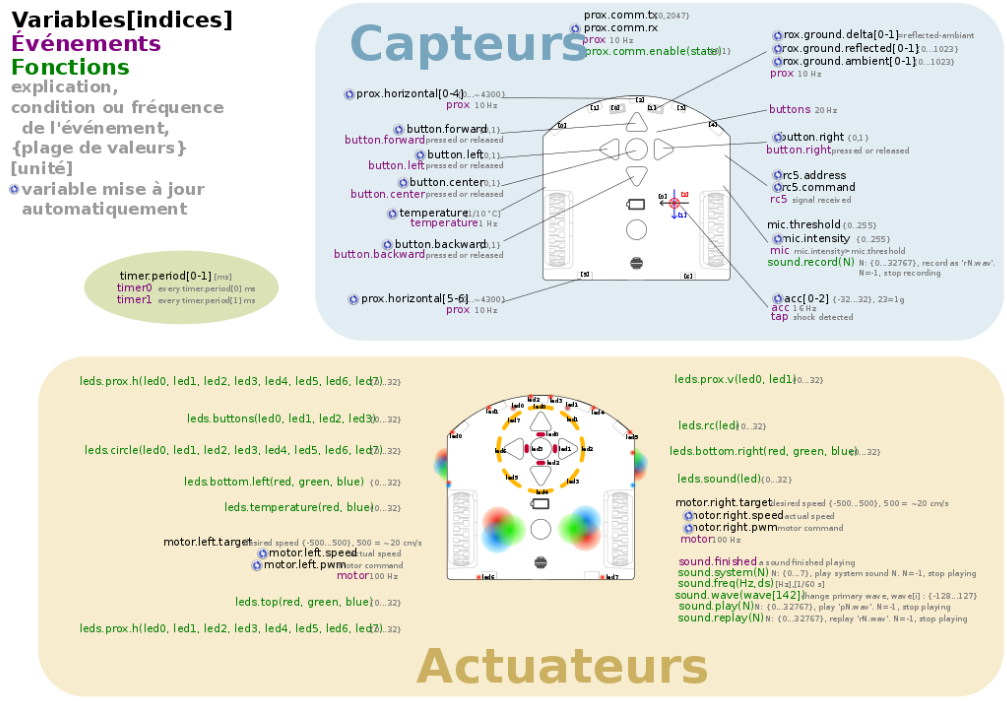
4 Conclusion

Au travers de ce projet, nous avons pu un aperçu des principaux enjeux et difficulté 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 amener 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 une domaine aussi vaste que la robotique.

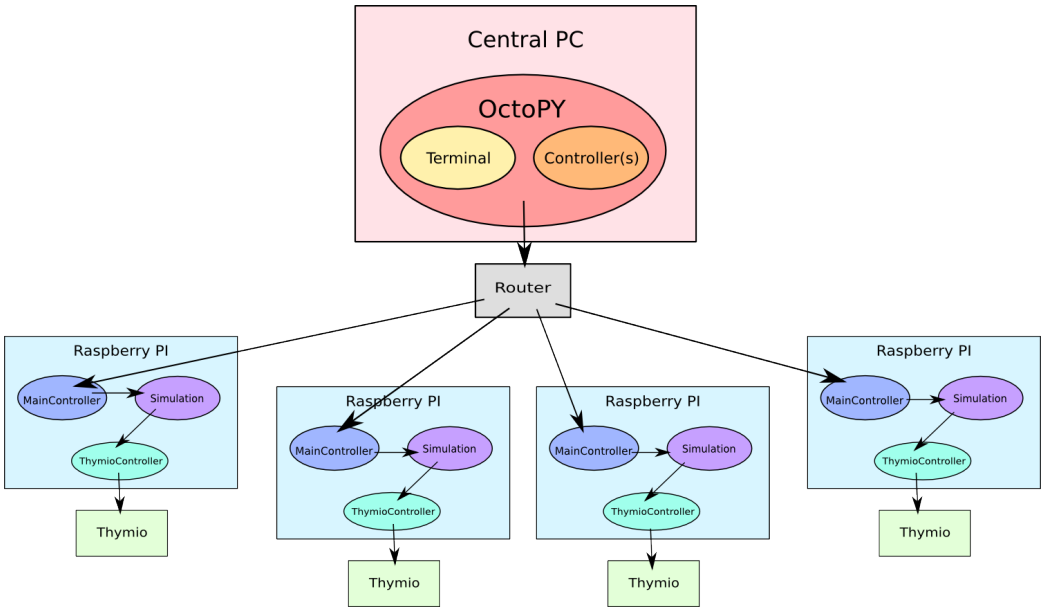
Dans la continuïté de notre travail, il pourrait être intéressant de pousser nos recherches et nos tests afin aboutir à un comportement de suivi de lumière et ou même de transport collectif.

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.