Faculté des Sciences
Département d'Informatique

UNIVERSITE LIBRE DE BRUXELLES,
UNIVERSITE D'EUROPE

Outil automatique de placement de robots pour des expériences de Swarm Robotics

Bernard Mayeur

Directeur : Prof. Mauro Birattari

Co-superviseurs : Gianpierro Francesca, Andreagiovanni Reina

Mémoire présenté en vue de l'obtention

Du diplôme du Master en Sciences informatiques

Année académique 2013 - 2014

L'important n'est pas la destination,
mais le chemin pour y arriver.
Lao-Tseu

Remerciements

Je tiens à remercier les différentes personnes qui ont permis cette expérience enrichissante.

Tout d'abord Mauro Birattari, pour m'avoir permis d'entamer cette expérience et m'avoir soutenu dans sa continuité.

Francesca Gianpiero, pour m'avoir conseillé et poussé de l'avant tout au long du mémoire.

Je tiens également à remercier ma famille et mes amis, tout particulièrement Xavier Bodart, pour leur soutien et leurs conseils.

Enfin, je tiens à remercier Guenaelle Loncin, pour tout les encouragement qui m'ont permis d'arriver au bout de ce mémoire.

Résumé

Ce mémoire consiste en le développement d'un outil de positionnement de robots pour les expériences de Swarm Robotics. À cette fin, de nombreux outils ont dû être utilisés. C'est le cas du Tracking system concu par IRIDIA, d'ARGoS3 - un simulateur de robots développé par le laboratoire IRIDIA - et d'un système de capteurs virtuels. L'idée est d'utiliser la position des robots, récupérée à l'aide de l'analyse des images du Tracking system, afin de créer dans le simulateur une réalité simulée. Grâce à cela, une série de capteurs peuvent être implémentés en simulation, et les valeurs de ceux-ci sont envoyées aux robots. Un capteur de position et de destination permet ainsi aux robots de connaitre la destination qu'ils doivent rejoindre.

Les premiers résultats ont été encourageants, avec un potentiel très intéressant. Toutefois, certaines erreurs dans le Tracking system empêchent encore à l'heure actuelle d'utiliser le système en déploiement.

Mots-clefs : Intelligence artificielle, Swarm Robotics, Tracking system, ARGoS3, IRIDIA, capteurs virtuels, système de positionnement.

Résumé

This thesis consists in the development of a tool to position robots for Swarm Robotics experiments. Lots of already existing tools have been used. This is the case of the Tracking System, of ARGoS3 - a simulator of robots developed by the lab IRIDIA - and of a system of virtual sensors. The main idea is to use the position of the robots, recovered from the pictures of the Tracking system, to build in the simulator a virtual reality. Thanks to this simulation, some sensors could be implemented in simulation and then their value send to the robots. A sensor of position and another one of destination allow the robots to know the destination they should go.

The first results have been promising. Some errors in the Traking system that still exist delay its deployment.

Key-words : Artificial Intelligence, Swarm Robotics, Tracking system, ARGoS3, IRIDIA, virtual sensors, positionning system.

Table des matières

Chapitre 1

Introduction

Dans le cadre des expériences de Swarm Robotics, le laboratoire IRIDIA est amené à réaliser de nombreuses expériences tout au long de l'année. Elles sont chacunes composées de nombreuses petites expériences qui peuvent, suivant les résultats recherchés, durer quelques dizaines de secondes, jusqu'à plusieurs minutes. C'est dans ce cadre que vient se placer ce mémoire.

1.1 Problématique

Comme pour les ordinateurs, au fur et à mesure des années, le hardware des robots a évolué, les processeurs sont devenus de plus en plus puissants et les capteurs de plus en plus précis. Toutefois, tout n'a pas évolué aussi vite. Les batteries ont relativemet peu évolué, déséquilibrant ainsi les besoins pour tous les composants embarqués. Avec l'évolution des différents composants et la réduction de leur taille, ceux-ci ont été amenés à consommer moins d'énergie, permettant ainsi la création de nouveaux robots de plus petite taille. Ces petits robots permettent ainsi d'augmenter le nombre utilisable pour une expérience.

Toutefois, plusieurs problèmes restent toujours d'actualité. La batterie, comme évoquée ci-dessus, s'épuise rapidement : les robots utilisés par le laboratoire, une version personnalisée des e-pucks (voir section 2.2.4) possède deux batteries, une fixe et l'autre amovible. En changeant cette dernière

en cours d'expérience, la durée moyenne pendant laquelle des expériences peuvent être menées est d'environ deux heures. Même si cette autonomie semble importante, il faut considérer le fait que replacer les robots, entre chaque expérience, peut prendre un temps assez conséquent suivant la précision et le nombre de robots. Si les expériences durent une trentaine de secondes à quelques minutes, et que le placement dure environ une minute, cela représente une partie extrêmement importante du temps imparti qui est consacré à la préparation des expériences.

La deuxième grande problématique rencontrée correspond à la reproductibilité des expériences. Il est en effet difficile, voire impossible, de reproduire des expériences dans les mêmes conditions que précédemment. Au mieux, les robots peuvent être replacés dans leurs positions d'origine juste avant l'expérience. Dans le cas où nous voulons replacer les robots à leur position initiale exacte, avec la même orientation, ce placement devient une tâche extrêmement lourde et difficile. Et ceci, en ne prenant même pas en compte les imprécisions qui sont introduites par le biais de l'humain.

Ce mémoire se consacre donc à la création d'un outil permettant le placement des robots de manière automatique. Le but étant de générer de manière automatique les positions pour l'expérience, ou de laisser le soin à l'utilisateur de les choisir, et d'automatiquement y amener tous les robots. Un tel système ne demanderait donc, après configuration, plus aucune intervention de l'humain, excepté pour le changement de batterie et la décision des différentes expériences à mener. Cela permet ainsi de libérer une personne (qui était assignée à replacer les robots en place), de gagner en précision et en reproductibilité. Nous espérons également, grâce à ce système, pouvoir placer les robots en un temps réduit par rapport à ce qui était fait à la main.

1.2 Structure

Le chapitre 2 reprend une partie de l'état de l'art de la "Swarm robotics" mais aussi la présentation des différents robots et du contexte dans lequel se placent les expériences. Ce chapitre reprend aussi une description du simulateur ARGoS(simulateur de robots consacré aux larges essaims de

robots hétérogènes), utilisé par le laboratoire, et développé par Carlo Pinciroli. Celle-ci se termine par le système de suivi des robots déjà mis en place.

Le chapitre 3 quant à lui, explique la méthode utilisée, ainsi que tout ce qui a dû être développé pour y parvenir. Ce sont les modifications et corrections des logiciels existants, ainsi que le développement d'un système de positionnement centralisé dans le simulateur ARGoS.

Le chapitre 4 présente les résultats obtenus dans une expérience réalisée à l'aide de l'outil de positionnement.

La dernière partie du mémoire se terminera par les conclusions et les possibilités d'extensions de l'étude.

1.3 Suppositions

L'ensemble de ce qui a été réalisé dans le cadre de ce mémoire a été créé afin de s'interfacer avec de nombreux programmes déjà existants. Parmi ceux-ci, nous noterons ARGoS, qui en est à sa troisième version. Bien qu'elle soit encore en bêta, celle-ci a été utilisée. Pour la suite du mémoire, l'utilisation du terme ARGoS fera donc référence à la version 3. De même, le software de l'e-Puck existe en une version stable pour ARGoS2, toutefois la version 3, encore en développement, a du être utilisée afin de s'interfacer à ARGoS3. Lorsque nous parlerons par la suite de l'e-Puck, nous parlerons de sa version pour ARGoS3.

Certains termes techniques ne se prêtant que peu à une traduction française, il a été décidé de garder ceux-ci afin de garder une compréhension convenable.

1.4 Contributions

Durant la conception de ce mémoire, de nombreux éléments ont du être réalisés. Ceux-ci se composent des éléments suivants, triés dans l'ordre d'importance, du plus important au moins important.

— Système de positionnement complet (aussi flexible et facile d'utilisation que possible).
— Interface facile afin d'exécuter le positionnement entre chaque expérience.
— Système afin de changer les batteries facilement.
— Amélioration du software des e-pucks (correction de bugs en majorité).
— Amélioration du Tracking system (correction et extension).

Chapitre 2

État de l'art

De nombreuses techniques ont été utilisées, afin d'obtenir un outil de positionnement automatique. Celles-ci seront expliquées dans le chapitre suivant (chapitre 3). Ce chapitre tente au mieux de présenter les différents concepts inhérents au projet ainsi que ceux utilisés lors de l'implémentation de la solution. Beaucoup de ces concepts sont propres au laboratoire (ou produits par celui-ci ou disponible en open source). La documentation de ceux-ci, ainsi que leur état lors du commencement du projet est mis en avant ci-dessous.

2.1 La Swarm Robotics

2.1.1 Qu'est ce que la swarm robotics ?

Traduction de la définition donnée dans l'article écrit par Dorigo, M. and Birattari, M. and Brambilla, M.[6]

> Swarm robotics is the study of how to design groups of robots that operate without relying on any external infrastructure or on any form of centralized control. In a robot swarm, the collective behavior of the robots results from local interactions between the robots and between the robots and the environment in which they act. The design of robot swarms is guided by swarm intelligence principles. These principles promote the realization of systems that are fault tolerant, scalable and flexible.

> La Swarm Robotics est l'étude de la manière de créer des groupes de robots qui s'exécutent sans dépendre d'aucune infrastructure extérieure ou aucune forme de contrôle centralisé. Dans un essaim de robots, le comportement collectif des robots résulte de l'interaction locale entre les robots et l'environnement dans lequel ils interagissent. La création de ces essaims est guidée par les principes de la Swarm Intelligence. Ces principes encouragent la réalisation de systèmes tolérants aux erreurs, extensible et flexible.

L'idée principale relève du souhait d'un système non centralisé pouvant s'adapter le plus possible aux erreurs qu'il pourrait subir (erreur de capteurs, robots défectueux...). Dans ce domaine, de nombreuses techniques sont développées sur base de faits observés dans la nature sur des insectes sociaux. De nombreux exemples sont disponibles, nous pouvons citer tout particulièrement les fourmis, les lucioles, les abeilles qui sont toutes des insectes sociaux et ont inspiré de nombreux algorithmes :

Fourmis : Ant colony optimization[5] (recherche de chemins optimaux à l'aide de "phéromones"), Ant-based routing[21] (routage des packets à l'intérieur des réseaux de télécommunication)...

Lucioles : Firefly algorithm[3] (inspiré par les lucioles cherchant à clignoter toutes ensemble afin d'augmenter l'intensité totale)

Abeilles : Artificial Bee Colony[11] (ABC algorithm) (différentiation de 3 types d'individus en fonction de leur tâche : explorer, rassembler de la nourriture, et transmettre les informations)

2.1.2 Principes et propriétés désirées

2.1.2.1 Principes

Lorsque l'on cite le principe de la Swarm Intelligence, nous sous-entendons les particularités permettant de catégoriser les problèmes à résoudre comme tels. On parle d'essaim, et de ce fait, le premier principe sera le grand nombre d'individus caractérisant les expériences. Le deuxième principe correspond

à l'absence de contrôle centralisé, induisant des communications locales. Ce phénomène, combiné avec le grand nombre d'individus, implique une certaine redondance dans le système. Enfin, la coopération entre les individus, afin de parvenir à leur but commun, relève également de l'un des éléments clefs en Swarm Robotics.

Grand nombre d'individus Un des points fondamentaux en swarm robotics est la présence d'un grand nombre d'individus. Cet aspect caractérise les systèmes, à l'inverse des systèmes traditionnels où une petite quantité (voire un seul) de robots multifonctions sont amenés à résoudre leur problème. Les essaims se construisent autour de nombreux robots semblables qui interagissent afin de résoudre le problème posé. Ces essaims peuvent avoir des tailles variables en fonction du problème étudié : le nombre d'individus optimal n'est pas toujours simple à déterminer et peut nécessiter une étude approfondie[14].

Contrôle non centralisé et perception locale Par la taille et les limitations physiques des robots, qui sont supposés être les plus simples possible, il est supposé que ceux-ci ne possèdent qu'une partie des informations concernant leur environnement. Ainsi, chaque robot est livré aux données qu'il peut récolter et éventuellement aux communications locales entre les robots. Nous supposons, en effet, que les robots ont la capacité de prendre des décisions de manière autonome, c'est-à-dire en l'absence d'un système de contrôle centralisé. Il peut être concevable de posséder un système d'élection afin d'obtenir un leader pour un groupe de robots. Toutefois, dans ce cas, afin d'obtenir un système plus robuste, il est nécessaire que celui-ci ne possède aucune caractéristique différente des autres. Il convient que le choix du leader soit apparu d'une décision commune, et qu'en cas de disparition de celui-ci, un nouveau leader puisse être élu.

Ce principe nous intéresse tout particulièrement dans la construction de notre outil automatisé de placement. Il est en effet nécessaire de prendre des décisions globales telles que les destinations, l'assignation des destiations

aux robots. Afin de faire un tel choix, il est nécessaire de posséder une idée globale de l'espace, ce qui est contraire aux idées intrinsèques de la Swarm Intelligence. Il pourrait être très intéressant d'explorer l'idée d'une conscience collective de l'espace, mais cela sort du cadre de ce mémoire. En centralisant le problème, il existe alors la possibilité d'obtenir des informations globales telles que la structure de l'arène, la position et l'orientation des robots dans celle-ci. Cette dernière partie sera effectuée par le Tracking system (voir section 2.4). Par ce fait, il va de soi que le travail à part entière n'est pas une expérience de Swarm Robotics, mais bien un outil au service de celle-ci.

Redondance Dans les essaims de robots, il est souhaité de posséder une certaine homogénéité entre les individus. Il est, en effet, sous-entendu que les robots sont interchangeables et qu'une fois interchangés, ils réagissent de manière similaire. Bien que certains projets, comme le projet Swarmanoid[9], se composent de plusieurs sortes de robots, à l'intérieur de ces groupes, les robots restent homogènes. Cette homogénéité implique une certaine redondance au niveau des individus, ce qui augmente grandement l'extensibilité ainsi que la flexibilité du système. Par le fait que ces robots soient interchangeables, il est même courant que ceux-ci soient amenés à changer de rôle en cours d'expérience. Un exemple, bien exposé par le projet SWARM-BOTS[1] correspond à la division des d'individus en deux types d'interactions différentes. Alors que le premier groupe crée une chaine entre le point d'origine et la destination, le second s'occupe de transporter un objet en suivant la chaine. Nous observons donc un 'sacrifice' de certains individus, afin que l'ensemble puisse atteindre l'objectif fixé.

2.1.2.2 Propriétés souhaitées

Par ces principes, la Swarm Robotics essaye de résoudre certains problèmes liés à la robotique. Ce sont à la fois la mise à l'échelle des systèmes, une tolérance accrue aux erreurs et une flexibilité qui sont aux centres de celle-ci.

Mise à l'échelle Le nombre de robots impliqués suppose que l'ajout ou le retrait de robots ne doit pas réduire de manière significative les résultats de

l'ensemble. Cette mise à l'échelle doit également permettre de résoudre les problèmes de différentes tailles en y adaptant la tailles de Swarm, et ce, sans modification conséquente du système.

Tolérence aux erreurs Par la redondance du système, et l'absence de contrôle centralisé, on souhaite que le système soit aussi robuste que possible. Pour un système centralisé, le fait de perdre la communication entre le système centralisé et un noeud constitue un problème critique. Par opposition, ce type de problèmes n'existe pas dans le cas d'un système non centralisé. Dans le pire des cas, si un robot devient défaillant, il peut être remplacé par un autre robot. L'utilisation de capteurs ne pouvant capter que des données locales peut également mener à de mauvaises décisions lors de l'exécution des différents algorithmes. Toutefois, la redondance du système mène à ce que les erreurs se compensent, et même si un robot venait à prendre une mauvaise décision, la majorité des robots serait elle même capable de réagir convenablement pour résoudre le problème.

Flexibilité La flexibilité est la possibilité pour un essaim de robots de résoudre différentes tâches dans différents environnements. Ceci s'inscrit dans les conséquences de la redondance des systèmes. Si les mêmes robots sont utilisables pour différentes tâches et environnements, ils peuvent alors s'adapter à ceux-ci.

2.1.2.3 Avantages et inconvénients

L'ensemble de ces propriétés rend le système très particulier. Cela le rend très différent de la façon dont pensent les humains, étant donné son caractère collectif. C'est une des raisons supplémentaires pour laquelle la Swarm robotics constitue un domaine de recherche assez novateur. Il est difficile de réfléchir de manière 'sociale', c'est pourquoi les algorithmes utilisés sont inspirés des différents animaux sociaux, et plus particulièrement des insectes qui représentent l'exemple parfait de Swarm. Un des autres éléments ralentissant le développement de ces techniques représente le coût élevé et l'absence de robot convenable. En effet, bien que ces dernières années, les modèles se soient

quelque peu différenciés, ils restent peu nombreux, et ceux-ci sont en général trop limités pour pouvoir être utilisés dans des expériences très diverses. Les E-pucks et Foot-Bot, utilisés par le laboratoire IRIDIA (voir section 2.1.4), sont deux robots parmi d'autres. Néanmoins, ils ont l'avantage de posséder un ensemble de capteurs très divers. Toutefois, leur prix trop élevé limite leur nombre, comme cela serait possible avec de plus petits (plus limités en capacité) modèles. D'un autre côté, cette nouvelle approche nous ouvre des portes vers de nouvelles manières de résoudre des problèmes de manière plus efficace, ou tout simplement de résoudre des problèmes précédemment irrésolus. Nous pouvons également imaginer à l'avenir de nombreux domaines comme le déminage, la recherche de personnes dans des ruines de bâtiments, ou même des techniques de soin à l'aide de nano-robots, où une intelligence non centralisée viendrait à prendre tout son intérêt dans la vie de tous les jours.

2.1.3 Méthodes de design de Swarms

Cette section essaye d'expliquer les différentes méthodes utilisées afin de créer et d'analyser les comportements de Swarm Intelligence. Il se base sur l'article de Manuele Brambilla et al[2], essayant de décrire les techniques les plus communes.

Bien que certaines techniques existent pour décrire des systèmes, il n'existe pas encore d'outil formel qui décrit des comportements individuels afin d'obtenir un comportement global. L'humain reste encore l'élément clef de ce développement. Deux grandes catégories coexistent pour la création des comportements.

Méthodes comportementales La première consiste en l'analyse et l'implémentation de manière itérative d'une solution se basant sur des comportements d'animaux sociaux. Elle consiste principalement en des machines finies à états probabilistes et des forces physiques virtuelles. Les machines finies à états probabilistes contiennent les tâches, sous forme d'état, et les changements de tâches, sous forme de probabilités. De nombreux comporte-

ments de Swarm tels que l'agrégation et la formation de chaines ont pu être créés de cette manière. Les forces physiques virtuelles correspondent à des champs de potentiel artificiels, dans lesquels chaque robot est une molécule soumise au champ. Cette méthode à l'avantage d'utiliser des propriétés physiques possédant un bagage mathématique conséquent. De plus, tel que dans la physique classique, les différents champs peuvent s'additionner. D'autres méthodes existent, mais sont encore peu répandues.

Méthodes automatiques Les autres méthodes font appel à des outils automatiques de génération de comportement qui se composent de deux parties : la première utilise les techniques de Reinforcement learning, tandis que la deuxième emploie les techniques génétiques. Chacune d'entre elles utilise des essais erreurs pour résoudre le problème. Le Reinforcement learning apprend grâce à un feedback positif ou négatif associé aux diverses actions qu'il entreprend. En essayant d'optimiser la récompense obtenue, le système tente ainsi d'obtenir le comportement idéal. Ce système toutefois reste extrêmement complexe, attribuable à la complexité individuelle et le nombre de robots entrant en jeu. De plus, il existe un autre problème, nommé le 'spacial credit assignment' qui correspond à la répartition du retour du système à chaque robot de manière individuelle. De l'autre côté, les algorithmes génétiques utilisent les techniques classiques de cross-over (échange de morceaux de séquence) et de mutation (modification de certains paramètres) afin de trouver les paramètres d'un réseau de neurones artificiels. Les réseaux de neurones ont l'inconvénient d'être une boite noire, très difficile à comprendre, et dont la convergence ne peut être certaine.

Développé dans le laboratoire par Gianpiero Francesca dans le cadre de sa thèse de doctorat, AutoMoDe[8] est une approche comparable à celles utilisées en Reinforcement learning. Celui-ci tente de construire une machine probabilistique à états finis à partir d'une série d'instructions atomiques. Chaque état de la machine représente une instruction atomique à exécuter. L'algorithme s'occupe à la fois de choisir les instructions dans un ensemble qui lui est fourni, de les combiner, et choissisant les probabilités. Bien qu'il soit encore en développement, celui-ci donne des résultats de bonne qualité

dans les expériences où il a déjà été testé.

2.1.4 IRIDIA

L'Institut de Recherches interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA) fait partie des différents laboratoires de l'ULB et se divise en deux parties. La première s'occupe d'algorithmique en intelligence artificielle, alors que la deuxième se concentre sur la Swarm Intelligence. Le laboratoire est à la pointe dans le domaine et possède de nombreuses figures de la Swarm Intelligence. Nous noterons tout particulièrement Marco Dorigo et Mauro Birattari qui possèdent un grand nombre de publications sur les sujets d'intelligence artificielle et de Swarm intelligence. Ce mémoire se place dans cet environnement très particulier, et est développé pour les infrastructures existantes. Celles-ci consistent en une arène dans laquelle sont faits les expériences, une série de robots, et le simulateur ARGoS développé par Carlo Pinciroli. Dans les sections suivantes, nous développerons l'ensemble de ces points en y décrivant les particularités ajoutées par le laboratoire.

2.2 Robots

2.2.1 Introduction

Au fil du temps, les différents robots utilisés par le laboratoire ont évolué. À l'heure actuelle, les différents modèles employés par le laboratoire comprennent : les e-Pucks[4], les Foot-bots, Hand-bots et Eye-bots[9] et enfin les Kilobots[20]. Le premier modèle correspond à un projet éducatif créé par l'École Polytechnique Fédérale de Lausanne qui a vu naitre la version actuelle des robots en 2006. Les trois suivants font partie du projet Swarmanoid, financé par la commission européenne et commencé en 2006. Ce projet, géré par Marco Dorigo, fut mené à terme en septembre 2010. Enfin, le Kilobot est un robot dont la description technique a été publiée en 2011. Il a été conçu de manière à n'atteindre qu'un coût d'une quinzaine de dollars de construction, de manière à pouvoir être utilisé en grandes quantités.

2.2.2 Swarmanoid

Le projet Swarmanoid s'inscrit dans la succession du projet Swarm-bots. La particularité des trois robots composant l'essaim, concerne leur réalisation spécialement conçue pour se diviser les tâches. L'essaim se compose des Foot-bots (voir figure 2.1), faisant office de pieds, qui peuvent se déplacer sur deux dimensions à l'aide de roues. Ils peuvent également agripper d'autres modules (d'autres Foot-bots ou des Hand-bots). Les Hand-bots (voir figure 2.2), quant à eux, ne possèdent pas de roues, mais un harpon leur permettant de s'accrocher sur un plafond et de grimper le long d'un câble. Ils disposent également de bras articulés leur permettant de saisir différents objets. Enfin, les Eye-bots (voir figure 2.3) bénéficient d'hélices leurs permettant de se mouvoir dans les trois dimensions. Ils jouissent également d'un système de caméras leur permettant de détecter des objets. Chacun d'entre eux est équipé d'une série de capteurs. Une description plus complète des robots se retrouve dans les articles suivants : Swarmanoid[7], Hand-bot[15], Eye-bot[10].

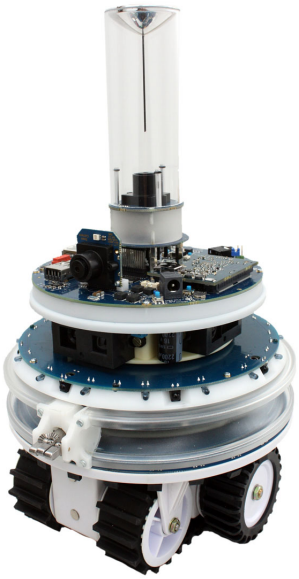

FIGURE 2.1 – Version finale du Foot-bot. Extrait de http ://www.swarmanoid.org/swarmanoid_hardware.php

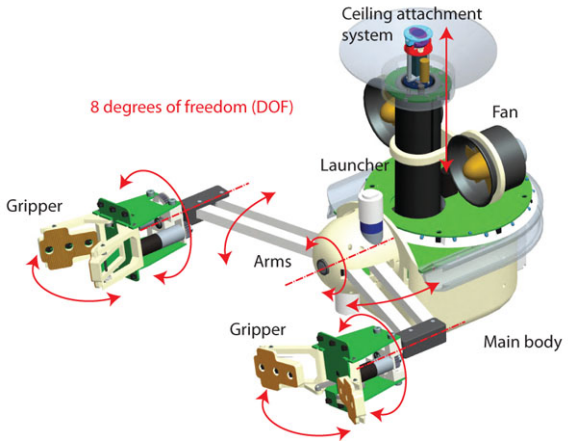

FIGURE 2.2 – Modélisation 3D du Hand-bot. Extrait de http ://www.swarmanoid.org/swarmanoid_hardware.php

FIGURE 2.3 – Photo d'un Eye-bot. Extrait de http ://www.swarmanoid.org/swarmanoid_hardware.php

2.2.3 Kilobot

Le kilobot est un robot prévu pour la Swarm, sa description complète se situe dans la description technique publiée par Harvard[20]. La simplicité du

modèle représente son atout principal. Il ne possède, en effet pas de roues, mais se déplace à l'aide de vibrations(A). À part leur capacité de mouvement, il est doté d'une batterie (B), d'un capteur/recepteur infrarouge (D), d'une led tricouleur (RGB) (E) et d'un capteur de lumière.

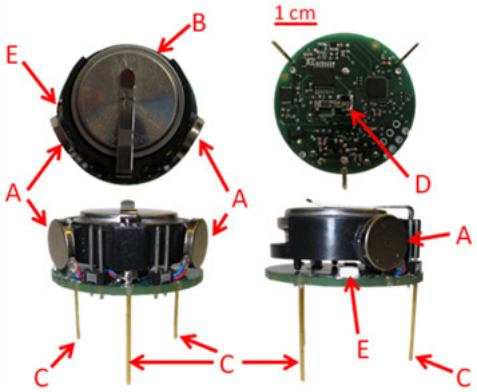

FIGURE 2.4 – Photo d'un Kilobot. Extrait de http ://science.howstuffworks.com

2.2.4 e-Puck

Bien que les e-Pucks aient été créés en 2006, ce qui correspond au plus ancien modèle de robots actuellement utilisés, ils n'en reste pas moins de très bonne qualité. Suivant les différentes versions, ils possèdent en effet de nombreux capteurs et actionneurs. La version basique comporte une batterie, deux roues, un anneau de 8 LEDs, un micro et une petite enceinte, une carte Bluetooth, une caméra à l'avant et des capteurs de proximité pouvant également servir comme capteurs de lumière. De nombreuses extensions sont également disponibles laissant la liberté d'y ajouter d'autres capteurs. C'est le cas des capteurs/émetteurs infrarouge, servant également de moyen de communication entre les robots (Range and Bearing sensor), du capteur de sol pouvant capter les niveaux de gris, et de la caméra omnidirectionnelle, montée à l'intérieur de la tourelle. Une deuxième batterie, fixe, ainsi qu'un module WiFi ont également été ajoutés parmi les différents composants. Avec tous ces ajouts, il possède pratiquement toutes les fonctionnalités des Foot-

bots, à l'exception de la possibilité de s'agripper aux objets. La principale différence entre ces deux-ci est la précision des différents capteurs qui est plus élevée dans le cas du Foot-bot : le Foot-bot, vu sa taille, est capable d'embarquer du matériel plus précis, ainsi qu'une batterie beaucoup plus importante. Il a, de plus, été construit plus tardivement et a donc pu profiter de capteurs de dernière génération.

Dû à leur large gamme de capteurs, les expériences qui peuvent être menées avec ceux-ci peuvent être très variées. Les e-Pucks sont avec les Foot-bots les candidats idéaux pour l'outil de positionnement. Le projet a été pensé afin de placer des e-Pucks, mais il est toutefois préférable de le réaliser de la façon la plus générique possible afin de pouvoir, dans un futur proche, l'utiliser avec d'autres types de robots qui seraient utilisé par le laboratoire.

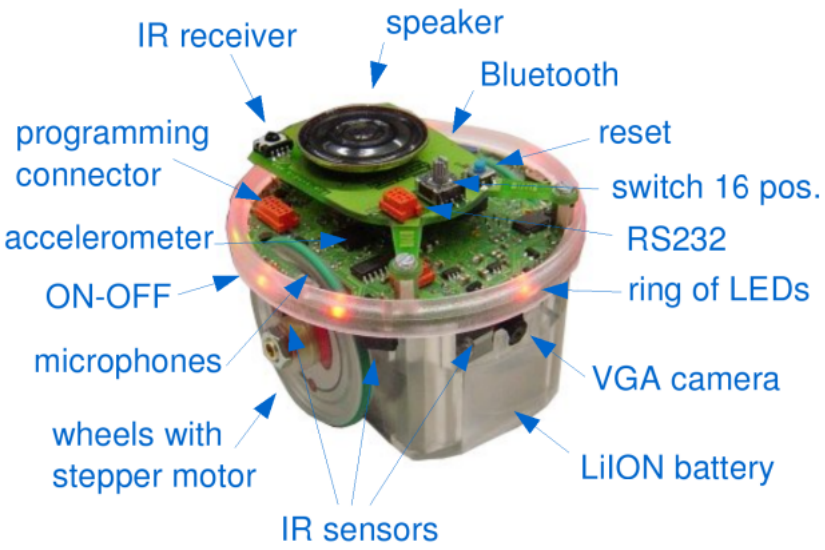

FIGURE 2.5 – Photo d'un e-Puck dans sa version de base. Extrait de http ://www.generationrobots.com

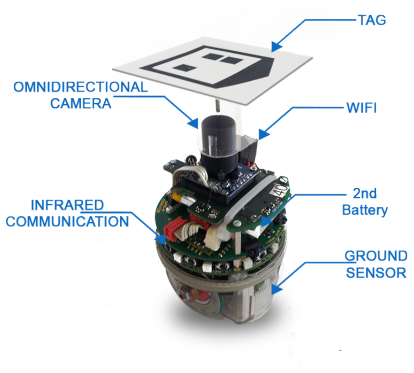

FIGURE 2.6 – Photo d'un e-Puck avec les extensions du laboratoire.

2.2.5 Utilité des robots

La présence des simulateurs (voir section 2.3) permet de simplifier la recherche scientifique en robotique en permettant de faire des expériences en simulation. Toutefois, les robots restent un élément essentiel, ils sont en effet l'élément nécessaire pour confirmer les suppositions faites en simulateur. Par exemple : si un nouvel algorithme est testé en simulateur, rien ne prouve que celui-ci sera aussi efficace en réalité, suite aux imprécisions des capteurs.

2.2.6 Problèmes

Alors que les robots, par leurs capteurs de plus en plus précis, permettent d'obtenir des valeurs de plus en plus fines, ceux-ci restent tout de même extrêmement sensibles au bruit. Ce phénomène est facilement observable. En effet, si un même robot utilise un capteur de lumière dans deux environnements différents, sans les avoir recalibrés, il est fort probable que la lumière ne soit pas détectable, ou alors que le capteur soit saturé. Afin de pallier au maximum ce genre de problèmes, les capteurs doivent être recalibrés entre chaque changement de milieu d'expérience. La méthode utilisée lors des expériences est de calibrer les capteurs dans un cadre bien défini et connu. Afin que le cadre de l'expérience soit bien défini, on essaye d'isoler celle-ci de tous les éléments extérieurs. C'est particulièrement le cas de

ceux qui pourraient varier au cours d'une expérience, par exemple la lumière ambiante, qui varie au cours de la journée. L'arène est donc isolée le plus possible du milieu extérieur pendant les expériences.

Un autre élément qui a déjà été abordé est la consommation énergétique des robots. En effet, celle-ci évolue en fonction du hardware. Malheureusement, les moyens de stockage d'énergie n'évoluent pas aussi rapidement. Bien que l'autonomie générale augmente, les batteries ne peuvent pas non plus être disproportionnées par rapport à la taille des robots. Avec les deux batteries, et en interchangeant la batterie amovible entre deux instances d'une expérience, les e-Pucks ont la possibilité de tenir environ deux heures. Ceci reste très limité. L'intérêt que porterait une solution afin de pouvoir recharger les robots pendant les expériences semble d'ailleurs très grand auprès des membres du laboratoire.

2.3 ARGoS

2.3.1 Qu'est ce qu'ARGoS ?

ARGoS[19][18] (Autonomous Robots Go Swarming), est un simulateur de robots créé par Carlo Pinciroli [1] pendant le projet Swarmanoid. Il est développé dans le laboratoire IRIDIA, et différents modules tels que l'ajout de e-Puck dans la liste des robots (ceux du projet Swarmanoid) y sont également développés. Le but de ce projet était de développer un outil flexible et efficace afin de simuler des expériences complexes impliquant des essaims de robots de différents types. La simulation est en effet un outil indispensable afin de créer des solutions pour les systèmes multi-robots, et ce, sans prendre de risques pour les robots et autres objets. ARGoS s'inscrit dans une optique aussi flexible qu'efficace. Il se veut en effet être flexible grâce à la modularité introduite dans celui-ci : ses modules, vus comme des plug-ins peuvent être configurés, étendus, interchangés suivant les nécessités des différentes expériences. Différentes implémentations des mêmes composants peuvent même exister afin d'obtenir différents niveaux de précisions, et leurs coûts correspondants. D'un autre côté, l'efficacité se retrouve dans le fait de ne calculer que les composants nécessaires, tout en créant une architecture parallèle afin de pouvoir faire usage des processeurs multicoeurs. ARGoS est un simulateur disponible de manière libre et gratuite et est utilisé par de plus en plus de laboratoires à travers le monde.

2.3.2 Unique ?

L'ensemble des simulateurs recontrés sont centrés sur la physique et en temps discret, ce qui signifie qu'ils sont régis par des lois physiques calculées par des équations et que celles-ci sont calculées de manière synchrone. Par son architecture, ARGoS se veux être plus efficace et plus flexible que les autres simulateurs. On retrouve en général dans les simulateurs l'un ou l'autre, mais une modularité possède en général un coût qui empêche une efficacité

1. Postdoctorant chez IRIDIA. Page personnelle : http ://iridia.ulb.ac.be/ cpinciroli/

conséquente. Il existe donc de nombreux simulateurs qui se concentrent sur l'un ou l'autre, mais pas les deux en même temps.

Flexibilité Du point de vue de la flexibilité, on nomme tout particulièrement Webots, USARSim [2], Gazebo [3]. Le moteur utilisé pour le rendu est soit libre dans le cas de Webots et Gabezo, soit commercial, dans le cas de USARSim, mais ces deux derniers ne possèdent pas de réelles possibilités de modifier les modèles inhérants à ceux-ci, ce qui limite malgré tout leur modularité. Webots quant à lui n'offre pas de possibilité de facilement modifier les capteurs et actionneurs. MuRoSimF quant à lui, un simulateur plus récent, offre la possibilité d'une précision modulable avec la possibilité de distribuer les ressources de calcul dans les différents aspects plus importants de la simulation.

Efficacité D'un autre côté, pour ce qui est de l'efficacité, les simulateurs cités précédemment deviennent lents (plus lent que si l'expérience devait être faite en réalité) dès qu'il excèdent quelques dizaines de robots. Il n'existe toutefois pas encore d'étude d'efficacité complète sur MuroSimF. Le simulateur connu pour pouvoir simuler des milliers de robots est Stage qui impose toutefois de nombreuses limitations au design. Il n'est en effet possible que de simuler des modèles 2D régis par des équations de cinématique en 2D. De plus, tout cela ne peut se faire que sans bruit. Ces limitations empêchent l'utilisation de ce simulateur pour de vraies expériences où l'on peut avoir besoin de s'agripper, pousser des objets ou de s'assembler.

ARGoS ARGoS essaye quant à lui d'allier les deux. À la fois la flexibilité grâce à une modularité de tous les composants, une efficacité grâce à une parallélisation du travail. Il existe également la possibilité de diviser l'espace en différents moteurs physiques, afin de gagner aussi bien en flexibilité qu'en efficacité. L'ensemble des éléments permettant à ARGoS d'atteindre ce but est expliqué dans la section suivante.

2. http ://usarsim.sourceforge.net
3. http ://playerstage.sourceforge.net

2.3.3 Structure

2.3.3.1 Modularité

Comme dans les programmes les plus complexes, ARGoS découpe son architecture en de nombreux modules interagissant. C'est en effet le moyen le plus répandu afin de permettre à l'utilisateur de choisir les modules à utiliser, et éventuellement de modifier ceux existants, ou d'en ajouter de nouveau. L'avantage de cette modularisation est la possibilité de permettre l'activation d'uniquement les modules qui sont utiles pour la simulation, et ainsi de ne dépenser que les ressources nécessaires à la simulation. Un exemple concret serait l'utilisation d'un modèle physique 2D pour calculer les interactions entre robots cloués au sol, alors que si certains des robots volent, alors un modèle 3D prend tout son sens. Cette modularité peut s'observer sur la figure 2.7 où l'on peut observer que le robot (entité), les capteurs et actionneurs, le Controller (logique du robot) sont des modules configurables par l'utilisateur, mais également le moteur physique, la visualisation et la 'loop function'(logique supplémentaire pouvant accéder à tous les éléments de la simulation).

2.3.3.2 Division de l'espace

ARGoS a ce souhait de modularité également à travers la possibilité de diviser l'espace en divers moteurs physiques. L'existence de ceux-ci complique grandement les interactions des entités comprises entre les différents moteurs physiques. Il existe en effet deux types d'entités pour ARGoS, les fixes, et les amovibles. Ces deux-ci interagissent de deux manières différentes dans l'espace. Les fixes ont la particularité de ne pas changer d'états tout au long de la simulation, et représentent des murs ou des objets inamovibles. Ils peuvent être assez conséquents et peuvent s'étendre sur plusieurs moteurs physiques. Par ce fait, ces objets peuvent exister à la fois dans plusieurs moteurs, et cela afin de garder une cohésion au niveau de la simulation. D'un autre côté, les objets amovibles sont considérés ne pouvant s'étendre à la fois sur deux moteurs physiques. Cela implique une imprécision à la limite des espaces définis par les moteurs physiques, qui empêche la vérification

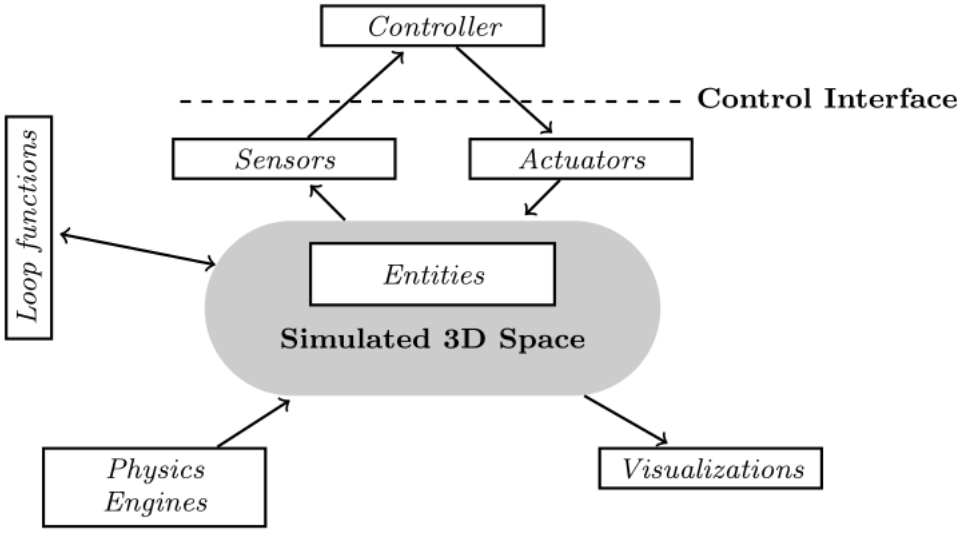

FIGURE 2.7 – Architecture d'ARGoS. Chaque boite blanche correspondant à un module définissable par l'utilisateur. source : [19]

d'une collision entre eux. Pour pallier en partie ce problème, la vérification des intersections avec les rayons, utilisée dans le cas des capteurs, peut être vérifiée à la fois sur tout les moteurs physiques.

2.3.3.3 Multi-thread

Afin de maximiser la vitesse de la simulation, l'utilisation des techniques de parallélisation est très présente. L'exécution de la boucle principale d'ARGoS est ainsi divisée en 3 grandes parties à l'intérieur desquelles les calculs peuvent être parallélisés. Cette subdivision a en effet été réalisée de manière à éviter l'utilisation des sémaphores et mutex, très lourds d'un point de vue de la performance. Ces trois phases se décomposent comme suit :

— L'actualisation des capteurs par l'état de la simulation présent dans le moteur physique. L'exécution du Controller comprenant la logique du robot, et le stockage des actions à effectuer dans les actionneurs. L'ensemble de cette première partie ne peut poser de problème de synchronisation grâce au fait qu'il ne fait que des lectures sur les composants autres que les actionneurs du robot traité.

— La deuxième phase consiste en la mise à jour des composants de chaque robot en fonction de ce qui a été prévu lors de la première phase. Cela correspond à la mise à jour des composants en fonction des actionneurs. De manière similaire, aucun problème de parallélisation ne peux se produire grâce au fait que chaque composant est lié à maximum un actionneur et chacun d'entre eux est lié à strictement un composant.
— Enfin, pendant la dernière phase, les différents moteurs physiques sont mis à jour. Ceux-ci n'étant pas interdépendants, il n'y a aucun risque lié aux différents threads.

2.3.4 Simulation et realité

ARGoS possède encore une autre particularité afin d'aider les chercheurs. L'ensemble des composants des différents robots est divisé en trois parties : les capteurs, les actionneurs et le Controller (la logique du comportement des robots). Ces deux premiers possèdent une logique différente suivant qu'ils soient utilisés en simulateur ou en réalité, mais de manière pratique ils demandent d'implémenter une interface commune. De cette manière, lorsque le Controller utilise l'interface commune, il fait appel à l'implémentation correspondant au milieu dans lequel il se trouve. En ayant recourt à ce procédé, le Controller peut être construit de manière générique aussi bien pour la simulation que pour les robots réels. L'implémentation au niveau du robot ne nécessite donc aucun changement de la part de l'utilisateur.

2.4 IRIDIA Tracking System (ITS)

2.4.1 Introduction

Dans le laboratoire IRIDIA, une des pièces, appelée par les chercheurs du laboratoire comme l'arène d'expérience, est consacrée aux expériences de robotiques. Celle-ci se compose principalement d'un grand espace ainsi qu'une grille de caméra installée au plafond afin de pouvoir capturer des vidéos des différentes expériences menées. Dans ce cadre là, un software[22] a été

développé afin de déterminer la position des différents robots dans l'arène en temps réel. Celui-ci a été nommé ATS pour "Arena Tracking System" ou, système de traçage de l'arène. Celui-ci capture à la fois les images afin de pouvoir en faire une vidéo, et les analyse en temps réel afin de pouvoir déterminer la position des robots. De plus, une connexion avec le simulateur ARGoS permet au simulateur de récupérer les informations de position. La fusion de ces deux entités a été nommée ITS pour IRIDIA Tracking System. Cette intégration permet au simulateur de posséder les informations de positions des différents robots, ce qui permet alors de simuler de nouveaux capteurs grâce aux connaissances du simulateur, et d'envoyer les valeurs de ces capteurs aux différents robots correspondants. Lorsque l'on parle de ces capteurs simulés, on parle de Virtual Sensors, ou capteurs virtuels.

Les sections suivantes sont constituées de la sorte : tout d'abord, une brève explication concernant le matériel utilisé, suivi par une section par modules constituant l'ITS. Cela consiste en l'Arena Tracking System, la fusion de celui-ci avec ARGoS (ITS) et enfin les capteurs virtuels.

2.4.2 Matériel

L'arène est une pièce d'environ 10m sur 7m. Les 16 caméras sont réparties dans cette pièce suivant un positionnement en matrice de 4x4 caméras afin de couvrir toute la superficie de celle-ci. Elles sont toutes reliées par une connexion Ethernet à un serveur se situant dans une autre pièce. Celui-ci possède un processeur 16 coeurs avec l'hyperthreading permettant la gestion en parallèle de toutes les caméras. Il abrite également l'interface de programmation servant à l'enregistrement des expériences. Il est possible d'exécuter une expérience à travers toute l'arène, ou même plusieurs en même temps en découpant l'arène en deux zones distinctes.

2.4.3 Arena Tracking system (ATS)

Le Tracking system est composé de trois couches. La première, en partant du plus bas niveau, consiste en la librairie Halcon[16], une librairie commerciale utilisée pour les applications d'analyse d'image, et est interfacée avec

le framework QT. Celui-ci fournit de nombreux outils afin de permettre une construction rapide des programmes pour la calibration et les outils de configuration des caméras.

Au-dessus vient se greffer le coeur du Tracking system, fournissant les structures afin d'extraire les informations des images, et le Tracking system à proprement parler, s'occupant de récupérer les informations, de remplir les structures de données, fournissant une interface de programmation afin de configurer les différents aspects du Tracking system, et de faire également l'analyse des images à l'aide des outils de la première couche. La capture d'image peut se paramétrer à l'aide de fichiers XML dans lesquels il est possible de changer le grain des caméras (au plus il est élevé, au plus la caméra donnera des images claires, mais au plus il y aura de bruit) et le temps d'exposition de celles-ci (également plus claires, mais si l'exposition est trop longue, les images risquent de devenir floues). Il est également possible de configurer le Tracking system de la même manière, en choisissant les caméras à utiliser, s'il faut ou non en enregistrer les images, et les tags à utiliser. Les tags sont des marqueurs servant à la reconnaissance des robots (voir Figure 2.8 et 2.9 pour un exemple). Ils sont recherchés à travers les images prises par les caméras toutes les 100 millisecondes. Il est également possible de configurer le Tracking system afin de choisir la périodicité des images clefs (voir Figure 2.10 : toute l'image est analysée). En effet, pour gagner en efficacité, la majorité des images ne sont analysées que dans les alentours des précédentes positions des robots (voir Figure 2.11). La taille de la zone déterminée comme les alentours est également un paramètre qui doit être déterminé en fonction du mouvement maximal que peut parcourir un robot durant l'intervalle entre deux images.

Enfin, au sommet, se situe les deux applications pour l'utilisation de ceux-ci. C'est le cas du viewer, une application permettant de voir en direct la vision des différentes caméras, et d'éventuellement demander à celles-ci d'enregistrer. La deuxième application est un serveur, attendant une connexion d'ARGoS, qui une fois connecté, s'occupe d'envoyer les données de positionnement et d'orientation à travers le réseau.

FIGURE 2.8 – Exemple de marqueur. Source : [22]

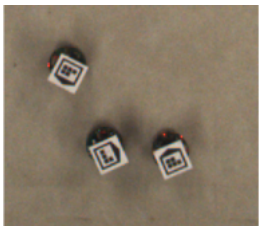

FIGURE 2.9 – Exemple de marqueurs sur des e-Pucks. Source : [22]

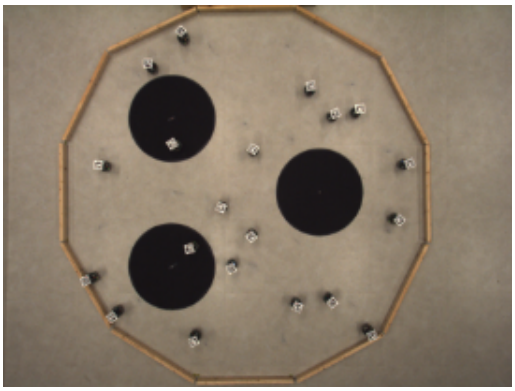

FIGURE 2.10 – Exemple d'image clef à analyser. Source : [22]

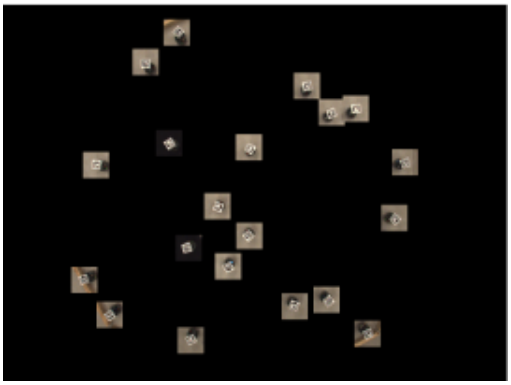

FIGURE 2.11 – Exemple d'image non-clef à analyser. Source : [22]

2.4.4 Combinaison avec ARGoS (ITS)

La combinaison avec ARGoS se fait à l'aide d'un moteur physique personnalisé. Les moteurs physiques ont comme rôle de calculer les interactions entre les différents robots afin de déterminer les positions de ceux-ci à chaque moment de la simulation. Dans ce cas-ci, le but étant de récupérer les données de positionnement et d'orientation des robots, le moteur physique va se connecter au serveur du Tracking system. Toutes les 100ms, celui-ci lui enverra l'ensemble des positions et orientation des différents robots dont il a pu trouver le tag dans les images. Le moteur physique possède également comme rôle celui d'envoyer les valeurs des différents capteurs dernièrement actualisés, si ceux-ci l'ont demandé. Lors de l'initialisation de celui-ci, il crée donc un thread acceptant les connexions afin que les robots puissent venir

s'y connecter, et un thread pour recevoir les données du Tracking system.

En ce qui concerne des robots, l'échange se fait également avec un thread qui écoute les informations que lui envoie ARGoS. Ceci comprend les données des différents capteurs à actualiser. Une fois récupérées, elles sont redistribuées à chacun des capteurs afin de mettre à jour les données de celui-ci.

2.4.5 capteurs virtuels

Les capteurs virtuels sont un moyen technique afin de créer une réalité augmentée pour les robots. Celle-ci se compose en deux parties. Premièrement, il est possible de virtualiser l'environnement réel, en contenant en simulation des obstacles n'existant pas dans l'arène, ou un environnement pouvant évoluer au fur et à mesure de la simulation. Nous pouvons noter par exemple l'utilisation de phéromones, qui serait par exemple très difficiles à créer en réalité, du à la présence de substances consommables et des différents capteurs que cela impliquerait. Avec un système tel que celui des capteurs virtuels, il suffirait au robot de signifier par un actionneur le dépôt de phéromones. Il suffirait alors à ARGoS de garder cela en mémoire afin de pouvoir calculer les valeurs des capteurs associés. Celles-ci pourraient alors être envoyées aux différents robots.

Ces capteurs virtuels permettent également de prototyper de nouveaux capteurs avant la construction de ceux-ci. Ils peuvent en effet être testés sur de vrais robots au lieu de devoir se contenter d'une simulation. En simulant du bruit sur ceux-ci, il est également possible de déterminer la qualité minimale dont ceux-ci doivent faire preuve afin de satisfaire au mieux les exigences.

2.5 Path planning

Le path planning, ou planification de trajectoire, est une branche très développée de la recherche. Pour des raisons pratiques d'utilisations, le terme path planning ne sera pas traduit dans la suite de ce document, celui-ci étant très répandu dans la littérature. De nombreuses recherches ont été menées, toutefois ce travail ne se prétend pas égaler celles-ci. Il se contente

de s'inspirer de certains principes. Pour plus de consistances, une approche assez générique sur les différents problèmes, ainsi qu'un point un peu plus conséquent sur le problème particulier qui nous préoccupe, sera développée ci-dessous. La thèse de Jur Pieter van den Berg, "Path Planning in Dynamic Environments" [23] et le livre de Steven M. LaValle, "Planning algorithm" [13], on servi d'inspiration.

2.5.1 Environnement statique

Le cas le plus simple de path planning est celui dans lequel un robot évolue dans un environnement statique. La solution la plus répandue est alors la constitution de routes, de telle manière à ce que chaque point de l'espace soit connectable à celle-ci de manière triviale. Celle-ci est généralement créée à partir d'une décomposition de l'espace en reliant toutes les extrémités des objets afin de créer des portions d'espaces dans lesquelles l'on doit posséder un morceau de route. Cette technique à l'avantage d'être simple, et de pouvoir majoritairement être précalculée.

Toutefois dès que l'environnement devient dynamique, ou lorsque le path planning doit être résolu une fois, le plus rapidement possible, d'autres techniques sont utilisées. Elles sont alors plutôt des méthodes se concentrant sur des champs virtuels physiques tels que le champ de potentiel. La destination est considérée comme un point d'attraction, et les obstacles comme des éléments répulsifs, le robot se contente alors de suivre le champ. Le principal problème de cette technique est la présence éventuelle de minima locaux pouvant bloquer le robot. Toutefois, des techniques existent afin d'éviter d'y rester bloquer, mais nous ne les aborderons pas. Un autre technique est la création de deux arbres aléatoires (un à l'origine, et l'autre à la destination) dont les feuilles sont étendues de manière aléatoire jusqu'à ce que ceux-ci se rejoignent, déterminant ainsi le chemin à suivre.

2.5.2 Environnement dynamique

Dans le cas de la robotique, si l'on considère les autres robots comme des obstacles à part entière, ceux-ci se meuvent également dans l'espace, et

il est donc clair que l'on se trouve alors dans un environnement dynamique. Pour généraliser, nous parlerons donc d'objets et d'obstacles également pour des robots. La supposition faite derrière ceci est que la position des obstacles est connue au fur et à mesure du temps. La solution consiste alors à résoudre non plus un espace en deux dimensions, mais bien en trois dimensions. La dimension ajoutée est le temps, ce qui implique une série de contraintes supplémentaires comme la vitesse de mouvement et la linéarité du temps (on ne peut ni reculer, ni stopper le temps). La majorité des algorithmes précédents restent toutefois applicables, toutefois, si la vitesse du robot est limitée et que certains objets bougent plus rapidement que ceux-ci, le problème prend alors un temps d'exécution exponentiel au nombre d'objets mouvants.

En introduisant la notion de temps, il faut également commencer à prendre en compte le temps de calcul de l'algorithme de path planning. Autant celui-ci ne pose pas de problèmes si le path planning est calculé à l'avance, autant si celui-ci est calculé au moment même, le temps d'exécution devient très important. Il va en effet de soi que le temps d'exécution n'étant jamais nul, le path planning exécuté sera toujours dépassé au moment ou il sera exécuté. Il devient alors nécessaire de prévoir la position des obstacles au moment où sera réellement exécuté le path planning, ou de considérer des temps suffisamment petits, afin que l'environnement soit fort semblable. Dans ce dernier cas survient alors un autre problème, il n'est en effet que peu probable de pouvoir parcourir l'ensemble des solutions dans le temps imparti. Pour palier à cela, l'approche est de choisir le meilleur chemin trouvé dans le temps imparti, et de l'exécuter.

2.5.3 Multiple robots

Tous les cas abordés précédemment concernent un path planning pour un seul robot. Dans le cas où de multiples robots sont impliqués, deux approches existent. La première, centralisée, consiste à faire le path planning des différentes entités dans un espace dynamique, mais dont l'état du système peut être connu pour tous les instants à venir. La résolution du

problème consiste en la résolution de tous les systèmes temporels en fonction des autres. Cela correspond à un produit cartésien des différents espaces temporels. L'avantage est la complétude de la solution, mais son coût en temps de calcul la rend peu réalisable pour des cas complexes. La seconde approche consiste à résoudre les différents systèmes indépendamment des autres systèmes, puis de faire appel à un algorithme de coordination afin de coordonner les mouvements. Cette approche peut toutefois produire des solutions très lointaines de l'optimale. En pratique, des méthodes essayant de faire un compromis seront plutôt utilisées.

Bien que peu étudiée, une autre approche est souvent utilisée. L'idée est de donner un ordre de priorité à chaque robot. De cette manière, chaque robot se voit à son tour calculer son path planning en fonction des autres robots, qui sont vu comme des obstacles mobiles. Le problème se voit donc réduit à plusieurs problèmes dans un environnement dynamique entièrement connu.

En simulations, toutes ces techniques permettent de résoudre les différents systèmes qui ont été décrits, toutefois, une fois appliqués à une expérience réelle, d'autres facteurs doivent rentrer en compte. Les robots sont en effet amenés à se déplacer de manière imparfaite, ce qui va les amener à s'éloigner du chemin prévu. Certains algorithmes tentent de limiter les perturbations, ou à les intégrer dans le chemin restant à faire.

Chapitre 3

Outil automatique de placement de robots

Dans le cadre de ce mémoire, il a été demandé d'implémenter une solution de placement automatique pour aider les chercheurs dans leurs expériences en Swarm Robotics. Pour cela, une solution a été conçue à partir des différents éléments exposés dans l'état de l'art. Cette section tentera d'expliquer tout d'abord la raison d'une telle demande, la structure et l'agencement des différents composants afin de créer cet outil. Une description plus précise de chacun des composants sera ensuite effectuée.

Problème, Arène	Algorithme	Conditions initiales, Positions initiales	Performance, Résultat
Agrégation	Algo 1	A	P_{1A}
Agrégation	Algo 2	A	P_{2A}
Agrégation	Algo 2	B	P_{2B}
Agrégation	Algo 1	B	P_{1B}
Agrégation	Algo 2	C	P_{2C}
Agrégation	Algo 1	C	P_{1C}

TABLE 3.1 – Exemple de table d'expérience à mener.

La table ci-dessus est un exemple du genre d'expériences qui peuvent être menées. L'idée est d'obtenir des statistiques sur la résolution des problèmes

par les différents algorithmes. À cette fin, en général, un problème et une arène d'expérience sera choisie à la fois, et une série d'algorithmes y sera exécutés dans diverses positions. Ceux-ci sont en général choisis afin de pouvoir être comparés avec des résultats précédents. Les résultats seront alors calculés à partir d'une bonne résolution ou non du problème. Jusqu'à présent, les conditions initiales de chaque expérience étaient une position aléatoire dans l'arène, plus ou moins répartie de manière uniforme. Il n'était pas contre pas possible d'un point de vue pratique d'exécuter plusieurs fois de suite différents algorithmes dans la même position initiale, celle-ci dépendant de facteurs aléatoires tel que le placement fait par l'homme et l'algorithme de dispersion qui était exécuté. Le but de cet outil est de rendre cela possible en un minimum de configuration. Une fois configuré et lancé, il devra être capable de positionner les robots sans intervention humaine. De plus, par la configuration, les positions choisies pourront être réutilisées, même des mois plus tard, et ce, même si elles sont aléatoires, le nombre ayant servi à leur génération étant affiché, et pouvant être réutilisé comme entrée du système.

3.1 Structure

L'ensemble des éléments servant à la création de cet outil a été énuméré lors de l'état de l'art. Celui-ci est composé d'une pièce d'arène, munie d'un système de caméras récupérant les images de l'expérience en cours. Celles-ci sont envoyées au serveur de l'arène qui les traite afin de récupérer les positions relatives des robots par rapport à la pièce de l'arène. Les coordonnées sont alors envoyées à ARGoS par l'interface entre le serveur du Tracking system et le client à l'intérieur d'ARGoS.

À l'intérieur de celui-ci, les positions sont mises à jour en fonction du système de coordonnées de ce dernier, ce qui permet de mettre à jour les deux capteurs virtuels créés dans le cadre du projet. Toutefois, bien que ceci soit suffisant pour mettre à jour le capteur GPS, le capteur de destination, lui, doit se mettre à jour en fonction de décisions allant bien au-delà d'un simple capteur. C'est la structure existante de la Loop function qui s'occupe de ce rôle. Celle-ci s'occupe de distribuer les destinations aux robots, de vérifier

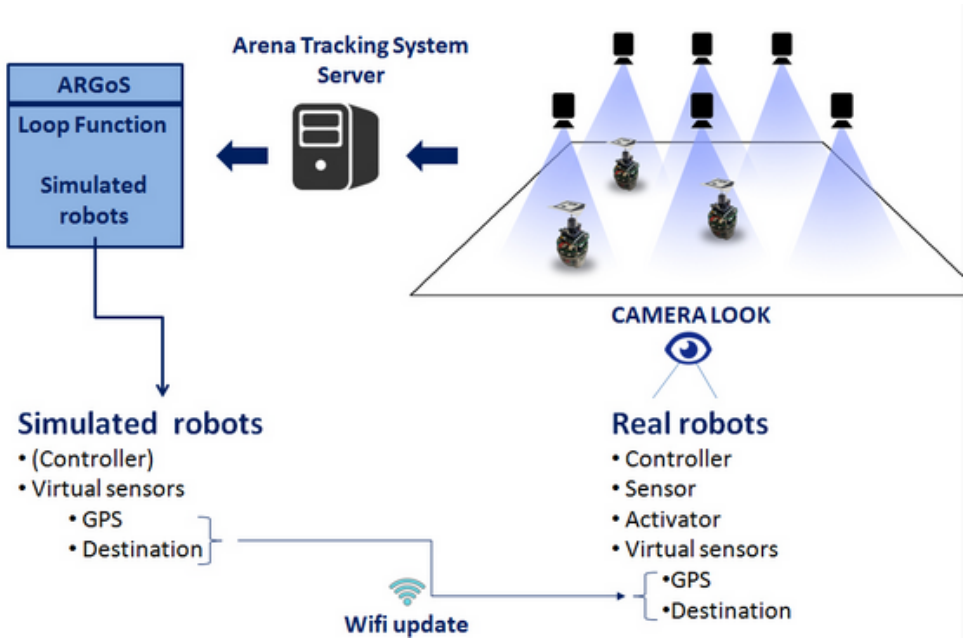

FIGURE 3.1 – Structure de l'implémentation de l'outil de positionnement.

le suivi des robots le long du chemin prévu, et de stopper le positionnement une fois les robots arrivés à destination.

Une fois les capteurs mis à jour, le serveur de capteur virtuel envoie les nouvelles valeurs à chaque robot de manière indépendante. Une fois ceux-ci reçus, les robots utilisent leurs capteurs virtuels comme s'il s'agissait de capteurs normaux. Le Controller possède donc la logique nécessaire à l'utilisation de ces capteurs afin de se déplacer à la position souhaitée. D'un point de vue des robots, ils ne sont pas au courant de l'existence de ce système et agissent sans violer la contrainte de non-centralisation. Après un certain temps, la caméra reprend une image de l'état de l'arène, et les nouvelles positions sont ainsi relevées. Cette boucle est ainsi exécutée jusqu'au placement complet ou partiel de tous les robots, suivant les conditions de configuration.

Chaque partie du système, le Tracking system, ARGoS, et le Controller du robot, sont exécutés de manière parallèle à trois niveaux différents. Le Tracking system, au niveau du serveur, dédié à cette tâche. ARGoS, au niveau de l'ordinateur de l'utilisateur, et le Controller une fois par robot sur

ce dernier. Au niveau d'ARGoS, suite à la structure de celui-ci, un Controller doit également y être exécuté, toutefois celui-ci ne peux avoir aucune influence directe sur la position, celle-ci étant forcée par le système de Tracking system, il peut toutefois utiliser les autres actionneurs, mais la Loop function permet de faire tout cela, et même plus. Au vu de la non-pertinence du Controller pour la simulation, il est réduit à sa plus simple expression, c'est-à-dire qu'il est vide.

3.2 Travail préliminaire

Afin de mettre en place ce système, une série de modifications ont dû être exécutées sur les logiciels déjà existants. Ceux-ci consistent principalement en une série de corrections et d'ajouts dont voici la liste.

3.2.1 E-Pucks

Malheureusement, au début du mémoire, le statut de la version de l'e-Puck pour ARGoS 3 était instable. Afin de pouvoir développer un logiciel pour ceux-ci, la première étape était donc de tester tous les composants afin d'en éliminer les erreurs. Si toutefois il n'était pas possible de corriger de suite celles-ci, elles devaient au moins être reportées afin de pouvoir être corrigées à postériori. De nombreuses erreurs ont été corrigées et la logique du capteur de lumière a été entièrement refaite. Un nouveau capteur, le capteur de batterie, ainsi qu'une erreur dans le firmware du robot, ont pu ainsi être mises à jour et corrigées par d'autres membres du laboratoire. La version ainsi obtenue peut être considérée comme utilisable, mais est toujours sous test intensif.

3.2.2 capteurs virtuels

Comme expliqué précédemment, les robots exécutent une expérience en faisant abstraction de l'existence du Tracking system et de ARGoS. Toutefois, afin de rendre cela possible, la création des différents capteurs virtuels était nécessaire. Pour cela, une structure a été mise en place afin de forcer

l'implémentation des méthodes de sérialisation et dé-sérialisation. De plus, la classe CByteArray de ARGoS, servant à la transformation de tout type de variable en un vecteur de bytes a également été ré-implémentée en une classe CNetworkByteArray, permettant l'envoi de nombres réels de manière sécurisée à travers le réseau. Cette dernière a été créée afin d'être utilisée lors de la sérialisation et dé-sérialisation des données des capteurs virtuels. Avec ces prérequis, les deux capteurs virtuels ont pu être implémentés en suivant la structure des capteurs virtuels.

En simulation, ceux-ci se mettent à jour grâce aux données du simulateur. Le GPS se base sur la position du robot comprenant le capteur. Le capteur de destination quant à lui n'est pas capable de calculer par lui-même la destination du robot, une variable a donc été ajoutée, et celle-ci comprend la prochaine valeur du capteur, qui est mise à jour lors de l'exécution de la Loop function (voir 3.3.3). Une fois les valeurs actualisées, les capteurs se chargent de donner les valeurs des capteurs si celles-ci ont été modifiées. Elles seront toutes envoyées en un bloc lors de l'exécution du moteur physique comprenant les connexions avec les robots.

Sur les robots, le thread recevant les données s'occupe lui-même de mettre à jour les données du capteur. Il le fait en redivisant le paquet reçu afin de pouvoir dé-sérialiser les données. De cette manière, lors de la prochaine lecture des données du capteur, celui-ci aura été mis à jour.

3.2.3 Tracking system engine

La version du moteur physique construit pour le Tracking system a été faite au plus simple, et ce, au vu du fait qu'il n'était pas utile de gérer les collisions dans un modèle où la position est déterminée par de vrais robots gérant la collision de facto. Toutefois, afin de permettre le path planning à l'intérieur d'ARGoS, il était nécessaire d'avoir un modèle cohérent, pouvant gérer les collisions. Celle-ci a donc dû être implémentée. Celle des obstacles a été activée, toutefois, pour des raisons d'optimalité, celle-ci étant chère d'un point de vue puissance de calcul, la collision entre robots a été désactivée. Une autre modification a été apportée, jusqu'à présent, seul un ordre di-

rect d'ARGoS, annonçant la fin de l'expérience, permettait de mettre fin à l'exécution du Controller sur les robots. Il a été ajouté la possibilité de pouvoir décider, dans le Controller, de mettre fin à l'exécution de celui-ci. Cela a dans notre cas été fait des fins d'utilisations du capteur de batterie, comme le sera expliqué dans la section 3.3.2, mais pourrait être utilisé à d'autres fins.

3.3 Positionnement

3.3.1 Partitionnement

Avec l'ensemble de ces modifications, l'implémentation du système à part entière peut enfin débuter. Celui-ci est divisé en deux grandes parties : ARGoS, de manière centralisée, prend les décisions. Les robots, eux, se contentent de suivre la logique de leurs capteurs. Nous exposerons donc séparément les deux parties.

Cette séparation a été décidée de manière à garder l'esprit de la Swarm Intelligence, et de permettre aux robots d'agir comme tel. Il aurait en effet été possible de créer un capteur qui donne directement des ordres au robot. Toutefois, cela ne respecterait pas la répartition des tâches. La logique implémentée ici dans le Controller peut être modifiée à tout moment, et les capteurs développés pourraient être utilisés dans d'autres situations.

3.3.2 Controller

Le controleur exécuté sur le robot comprend la logique interne du robot. Son rôle est de donner les valeurs aux actionneurs et ce, à partir des différents capteurs auxquels il a accès. C'est la manière dont le robot peut interagir avec le monde extérieur, on peut considérer les capteurs comme les entrées d'une boite noire, et les actionneurs comme les sorties de celle-ci. À l'intérieur de celle-ci, la logique implémentée est celle présentée par l'algorithme 1.

Logique Le robot avance en direction de sa destination (qui peut varier), et si sa batterie devient trop faible, celui-ci se dirige à la place vers une position

prédéfinie, où il se coupe. Dans le cas où le capteur GPS ne serait pas mis à jour pendant une durée trop longue, on suppose que le tag du robot n'est plus reconnu, et dans ce cas-là, ne pouvant se fier à la position en mémoire, on force le robot à se déplacer de manière aléatoire en espérant trouver un nouvel endroit où celui-ci sera à nouveau détecté.

Caractéristiques L'évitement des obstacles se fait à l'aide des capteurs de proximité. Une force inversement proportionnelle à la proximité, et dans la direction inverse de celle-ci est ajoutée à la force de la direction à prendre. De cette manière, un obstacle détecté créera une force dans la direction opposée. Une fois ajoutée à la direction, celle-ci amènera le robot à éviter l'obstacle.

Afin de pouvoir obtenir une meilleure précision, il a été décidé que la vitesse du robot devait être inversement proportionnelle à la distance le séparant de sa destination. La fonction est visible dans la figure 3.2. Celle-ci est bornée entre 3cm/s et 14cm/s, 14cm/s étant la limite supérieure de vitesse de l'e-Puck. Au-dessus de celle-ci, un risque physique existe pour celui-ci. 3cm/s correspond à la limite inférieure choisie, elle a été choisie en fonction des caractéristiques du système. À une vitesse de 3cm/s, sachant que le temps nécessaire à l'obtention de la position entre le Tracking system et le robot est de maximum 300ms théoriques, cette vitesse implique que le robot parcourra moins d'un centimètre entre la position réelle et celle donnée par le capteur.

Afin de se mouvoir vers une position, le robot n'emprunte pas une direction rectiligne, en effet celui-ci utilise des trajectoires elliptiques afin d'éviter les rotations sur place. Ceci rend le système beaucoup plus résistant en corrigeant directement la trajectoire, gardant la majorité de l'effort à avancer, alors que si une trajectoire rectiligne était utilisée, l'angle devrait être corrigé en faisant des rotations sur place, ce qui pourrait créer de l'instabilité par le fait du délai entre le Tracking system et les capteurs, et pourrait impliquer de nombreuses corrections. Ces problèmes pourraient être fort réduits par la présence d'un seuil au-dessous duquel l'erreur est jugée acceptable, mais consituerait toutefois une solution moins efficace, de par le fait qu'une partie du travail serait perdu dans ces imprécisions.

Configuration Afin de permettre une meilleure gestion de l'algorithme, le Controller prend une série de paramètres non obligatoires. S'ils ne sont pas spécifiés, les valeurs par défaut sont utilisées. Celles-ci doivent être placées dans le fichier de configuration de l'expérience envoyé au robot, dans la section argos-configuration > controllers > path_planning_controller > params. Si une valeur par défaut existe, elle est précisée et le paramètre est non obligatoire. Si une unité existe, elle est également spécifiée. En gras, le code de paramètre utilisé, il sera utilisé dans la section expérience (4) afin de clarifier les paramètres utilisés lors d'expériences. Ceux-ci suivent la notation suivante : **P**aramètre **C**ontrôleur avec le premier chiffre pour rassembler les paramètres en sections. La section 1 concerne le placement, la 2 le capteur de proximité, la 3 le rechargement de la batterie, et le 4 le problème de mise à jour de capteurs.

Liste des paramètres :

- **PC1.1** Précision de placement : 0.03 (mètres) (maximum : 0.05)
- **PC1.2** Précision de l'angle de placement : 1 (degrés)
- **PC2.1** Valeur minimale du capteur de proximité afn de considérer une présence d'obstacle : 0.4
- **PC2.2** Force de la répulsion due à un obstacle : 8
- **PC2.3** Bruit ajouté à la répulsion : 4
- **PC3.1** Temps en batterie faible avant de recharger : 10 (x0.1 sec)
- **PC3.2** Destination de recharge
- **PC4** Temps de non mise à jour du capteur GPS avant de se déplacer de manière aléatoire : 0 (désactivé) (x0.1 sec)

3.3.3 Loop function

La Loop function contient quant à elle la majorité de la logique au niveau d'ARGoS, c'est en effet elle qui a la tâche de générer les positions, choisir les destinations des robots, ... En fait, elle se charge de mettre à jour le capteur de destination, et pour cela doit exécuter toute une série d'algorithmes. La complexité étant plus haute que le Controller, cette section commencera par une explication générale de la structure, puis une explication plus en

```
Data: GPS, Destination, Battery & Proximity sensors
initialization;
while no close message from ARGoS do
    if battery too low then
        if in recharge position then
            quit the controller for battery change;
        else
            go to the recharge position while avoiding obstacles;
        end
    else
        move to destination position while avoiding obstacles;
    end
    if no GPS update for a while then
        random walk
    end
end
```

Algorithm 1: Pseudo-code du Controller présent sur les robots

profondeur de chacun de ces points.

3.3.3.1 Structure

ARGoS définit les Loop functions à travers 3 actions qui s'exécutent à différents moments de la simulation.

L'initialisation se fait une fois au lancement d'ARGoS, elle s'occupe en général de lire les paramètres. Afin de pouvoir positionner les robots, on va avoir besoin de prendre conscience des différents obstacles placés dans l'arène, pour cela, un graphe sera construit. Celui-ci consiste en une création de routes sur lesquelles le robot peut se déplacer afin d'atteindre tous les points de l'espace. Il sera utile afin de créer les chemins entre deux points de l'espace. Ce graphe ne dépendant que des objets statiques de l'espace, il restera constant tout au long des simulations. Il sera donc précalculé à l'initialisation afin de gagner du temps lors de l'exécution.

La deuxième partie consiste en la logique qui est exécutée avant l'exécution des Controllers des robots. Celle-ci s'occupe de générer la liste des destinations si celle-ci n'existe pas, et de stocker la liste des robots en mémoire (afin de leur donner un ordre). Une fois ordonnés en mémoire et que les destina-

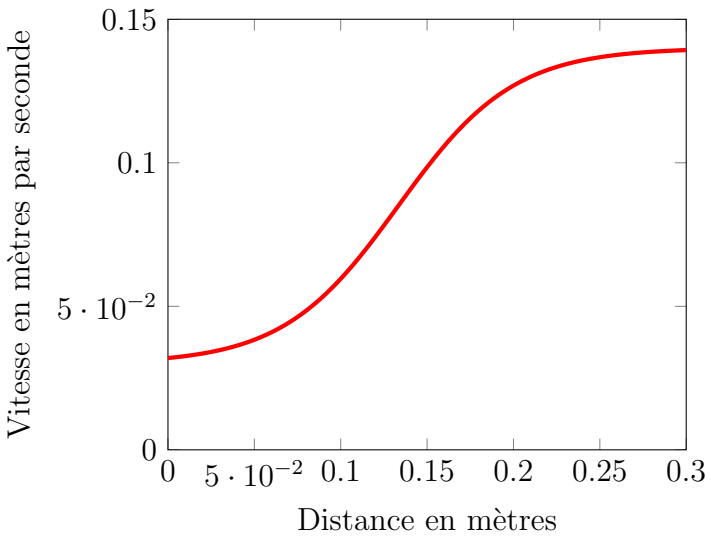

FIGURE 3.2 – Fonction de la vitesse en fonction de la distance

tions existent, ceux-ci seront assignés à une destination, et le chemin afin de l'atteindre sera calculé à l'aide du graphe précédemment construit.

La dernière partie s'exécute après le Controller des robots. Bien que celui-ci soit vide, et donc que d'un point de vue simulation, ces actions auraient pu être exécutées avant le Controller, la logique se veut de les mettre à la fin. En effet, cette partie-ci gère l'arrivée à une destination intermédiaire (afin d'éviter un obstacle), et l'arrivée de l'ensemble des robots à leur position finale. L'arrivée à une destination intermédiaire change également la destination du robot et le capteur de destination. L'ensemble des robots arrive à destination lorsqu'ils sont tous à moins de 5 cm de leur destination, dans ce cas, un temps fixe (et configurable) leur est laissé pour se placer exactement avant que l'expérience de positionnement ne se termine.

3.3.3.2 Initialisation du graphe

Afin de comprendre les différentes simplifications qui ont été menées, il faut se rappeler du cas dans lequel nous nous trouvons. Nous nous trouvons dans un cas où de multiples robots se doivent d'aller à de multiples positions. L'espace est connu pour ce qui est des objets statiques, mais inconnu quant aux objets mouvants (les autres robots). De plus, nous nous trouvons

dans un système imparfait par le fait qu'il implique des robots réels, sensibles aux frottements, aux dysfonctionnement,... Comme exposé dans l'état de l'art dans la section 2.5, ce cas correspond aux problèmes les plus compliqués, nécessitant un temps exponentiel au nombre de robots afin de trouver une solution complète. Deux simplifications seront donc faites. La première consiste en une assignation séquentielle des robots, qui sera traitée dans la section suivante, et la deuxième consiste en une simplification de l'espace.

La simplification se fait à deux niveaux. Tout d'abord, les robots étant amovibles, on pourrait considérer ceux-ci comme des obstacles se déplaçant dans l'espace, ce qui signifie une planification en trois dimensions, c'est-à-dire en ajoutant comme dimension le temps. Le problème de l'ajout de celui-ci est la complexité introduite. Il faut que celle-ci vaille la peine. D'un point de vue de l'efficacité, on souhaite un système qui soit le plus réactif possible, il n'est en effet pas possible de précalculer la planification au vu du fait que chaque expérience commence dans une situation inconnue à l'avance. Chaque robot possède de plus des capteurs de proximités qui leur permettent d'éviter les autres robots tout en continuant leur course, comme expliqué dans la section Controller (3.3.2).

La deuxième simplification de l'espace se base sur le fait que l'ensemble des objets utilisables en simulation (et donc servant à représenter les objets utilisés dans les expériences réelles) sont convexes (on travaille dans un espace à deux dimensions). Les objets disponibles sont en effet des boites et des cylindres qui, quels que soient leurs orientations, une fois projetés sur le plan, donnent des figures connexes. Les objets plus complexes sont des compositions de ces deux objets de base, ils seront donc décomposés et traités de manière séparés. L'idée est que chacun de ces deux objets peut être contenu dans un rectangle. En se baladant le long de celui-ci, tous les points de l'espace sont alors accessibles. La simplification consiste donc à récupérer les 4 coins, de s'éloigner de la figure d'une taille d'au moins une fois le rayon du robot (afin que celui-ci ne rentre pas en collision avec les murs). Nous avons choisi de nous éloigner d'une fois et demi le rayon afin de laisser un peu de marge. Il existe toutefois un risque pour les objets complexes de petite taille (dont les pièces désassemblées sont plus petites qu'une fois et demi la

taille du robot). Dans ce cas, ce choix de points de navigation ne pourrait pas convenir. Il serait alors nécessaire de ré-implémenter cette méthode en gérant ce cas particulier. Le résultat obtenu peut être observé sur les figures 3.3 et 3.4, celui-ci correspond aux 4 points verts autour des objets.

L'ensemble de ces points de navigation potentiels est alors simplifié. Pour cela, les points générés à l'intérieur d'autres obstacles sont alors enlevés, les doubles sont également simplifiés, et enfin les noeuds proches les uns des autres (en dessous de la taille d'un robot) sont fusionnés.

Cet ensemble constitue alors l'ensemble des points de navigation, représentés dans le graphe comme des noeuds. Les arêtes sont alors construites en faisant appel au moteur physique. Celui-ci est interrogé afin de savoir si un obstacle existe entre deux points de navigation. Si ce n'est pas le cas, une arête est ajoutée entre les deux noeuds correspondants, et la pondération de celle-ci est la distance séparant les deux noeuds. Les arêtes sont représentées sur les figures 3.3, 3.4 et 3.5 comme les lignes vertes.

Ce système n'est pas parfait, car certains noeuds pourraient encore être simplifiés tout en gardant la complétude de l'espace. À l'inverse, certaines arêtes passent proche de murs, ce qui pourrait impliquer une collision avec celui-ci si ce chemin était utilisé. Une extension de cet outil de positionnement serait, de prendre en compte, tout les types d'objets (en incorporant les objets complexes de petite taille) et une meilleure gestion des noeuds et arêtes. En pratique, ces imperfections ne posent pas de problèmes grâce aux capteurs de proximité qui permettent d'éviter les obstacles.

3.3.4 Génération de destinations

Trois modes existent pour le choix des positions de destination, ce choix doit être fait avant le lancement de l'application et est fixe pendant toute la durée des expériences. Il doit être choisi dans le fichier de configuration. La première possibilité, la plus simple, consiste à laisser la possibilité à l'utilisateur de choisir ces positions. Il est supposé que l'utilisateur sait ce qu'il fait, et les coordonnées sont simplement stockées sans modifications.

Les deux autres possibilités consistent à générer des positions aléatoirement,

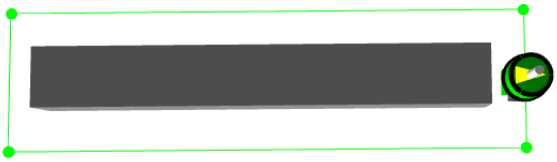

FIGURE 3.3 – Construction de graphe pour une boite

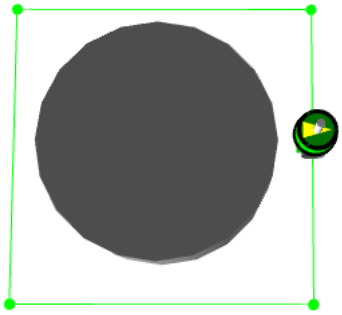

FIGURE 3.4 – Construction de graphe pour un cylindre

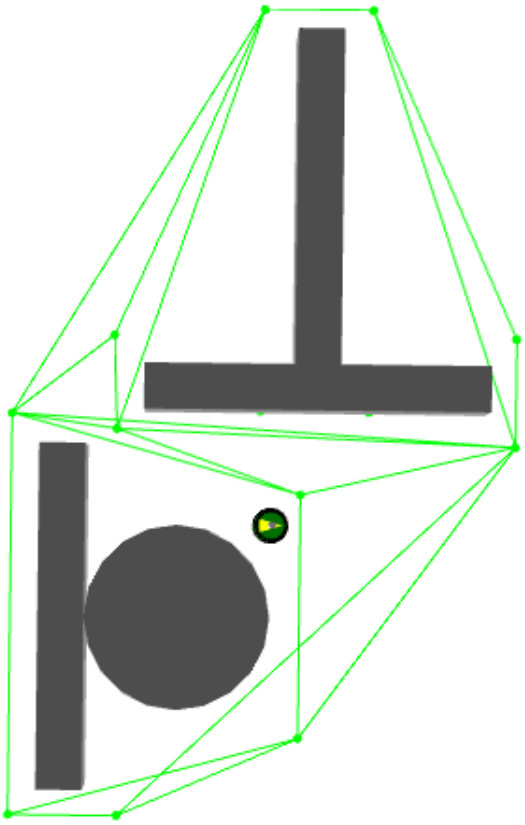

FIGURE 3.5 – Construction de graphe pour de multiples objets

pour cela, les algorithmes de générations prévus dans ARGoS sont utilisés, et la vérification de superposition de positions est faite à l'aide des cubes englobant les robots. Il est également prévu de ne pas générer de positions à l'intérieur d'obstacles statiques. Les deux possibilités se différencient par l'utilisation de différents seeds afin de générer les positions. Dans le premier cas, l'utilisateur laisse libre choix au programme d'utiliser des seeds aléatoires, dans l'autre cas, l'utilisateur fournit la liste des seeds qui seront utilisés pendant les différentes instances de positionnement. Cette possibilité lui donne la possibilité de positionner plusieurs fois de suite les robots à la même position afin de pouvoir y tester de multiples algorithmes.

3.3.5 Assignation

L'assignation correspond à choisir un robot, et à lui associer une destination finale. Suivant la configuration choisie, les robots seront directement assigné à une position précise, ou n'importe quel robot pourra aller en n'importe quelle position. Ce choix doit être fait par l'utilisateur avant le lancement. La résolution de l'assignation est triviale dans le premier cas, les robots et les destinations étant ordonnées, le $i^{ième}$ robot sera assigné à la $i^{ième}$ destination. Dans le cas où les destinations sont partagées, nous nous retrouvons dans une situation beaucoup plus compliquée, nous avons en effet un nombre de robots, et le même nombre de destinations. Nous cherchons donc à faire une bijection d'un ensemble vers l'autre. Nous cherchons donc parmi ces solutions, celle qui sera la meilleure.

Jusqu'à présent, il n'existait pas de méthode afin d'ordonner les solutions. Après analyse, deux possibilités sont ressorties : trier en fonction de la somme des coûts (représenté par les distances à parcourir), afin de minimiser le déplacement total, ou trier en fonction de la distance maximale minimum, c'est-à-dire tenter d'obtenir la plus petite distance maximale. La première méthode aura tendance à privilégier le fait que les robots déjà placés restent à leur position, tandis que la seconde privilégiera de nombreux petits mouvements, quitte à déplacer tous les robots qui seraient déjà en place. Souhaitant privilégier une minimisation du temps nécessaire au placement des robots,

c'est la seconde solution qui a été retenue, toutefois, il serait intéressant dans une extension de l'outil, d'implémenter également la première méthode et de laisser le choix à l'utilisateur de la méthode souhaitée. Le problème ainsi défini correspond au 'Linear bottleneck problem'[17] dans la littérature.

Afin de résoudre ce problème, un des algorithmes les plus répandus est le "Threshold algorithm". La version de base sera utilisée ici afin de trouver le seuil permettant d'obtenir un système résolvable. Résolvable signifie qu'une assignation existe de manière à ce que toutes les distances entre les robots et leur distination soient plus petite ou égales au seuil. L'algorithme utilisé pour la résolvabilité d'un système sera exposée dans la section 3.3.5.1.

L'algorithme a été légèrement modifié afin d'essayer de trouver la solution optimale. Le pseudo-code de la version modifiée est celui décrit par l'algorithme 2. Alors que l'algorithme du seuil se contente normalement d'essayer de trouver une solution faisable telle que la distance maximale soit plus petite ou égale au seuil, la solution implémentée génère une seule fois la matrice des distances, mais résout de multiple fois le système en le simplifiant à chaque nouvelle assignation. Le robot et la destination sont donc enlevés de la matrice, réduisant ainsi l'espace de recherche. D'un point de vue pratique, cela correspond à assigner la plus grande distance minimale, et à rechercher dans le système restant la plus grande distance minimale restante, cela, jusqu'à la résolution complète du système. Cette résolution permet d'obtenir d'excellents résultats, bien que ceux-ci dépendent directement de l'algorithme déterminant la résolvabilité du système.

```
Generate the matrix of distance robot - destination;
while some robot not assigned do
    Find the limit value of the threshold s.t. the system is resolvable;
    Assign the robot to the corresponding destination;
    Remove the assigned robot & destination from the matrix;
end
```

Algorithm 2: Pseudo-code de l'algorithme d'assignation

3.3.5.1 Résolvabilité

Afin de trouver une réponse à la résolvabilité, une solution basée sur le 'Hungarian algorithm'[12] a été construite. Celui-ci est simplifié afin de considérer le seuil actuel du Threshold algorithm, à la place de l'algorithme de base qui construit normalement lui-même les positions zéros dans la matrice. En pratique l'algorithme utilisé est décrit par le pseudo-code suivant : voir l'algorithme 3.

L'initialisation choisit la position de base des zéros suivant le seuil à tester. Si une des lignes ou colonnes ne possède pas de zéros, il est alors trivial que le système ne soit pas résolvable. Dans le cas contraire, en se basant sur le fait que si un zéro est seul dans une ligne ou colonne, le seul moyen de pouvoir résoudre le problème est d'assigner celui-ci, on l'assigne. En faisant cela de manière récursive, cela signifie que toutes les lignes et colonnes restantes possèdent aucun zéros ou au moins deux. S'il n'y a pas de zéros, cela signifie qu'il n'existe pas de possibilités d'assigner cette ligne, le système n'est donc pas résolvable.

Dans le cas où il ne reste plus que des lignes et colonnes avec au moins deux zéros, à moins d'avoir un cas trivial, il n'est pas possible de déterminer si le système est résolvable. On va donc assigner de manière aléatoire un des zéros. Après cela, on va recommencer à essayer d'assigner les robots de la même manière que précédemment. Toutefois, une absence de zéros sur une ligne ou une colonne n'implique pas que le système n'est pas résolvable. Il est en effet possible que le choix aléatoire n'ait pas été correct. D'un autre côté si tous les robots ont été assignés, le choix aléatoire était alors correct, et le système possède au moins une solution, il est donc résolvable.

Dans le cas où le choix aléatoire du zéro n'était pas correct, il n'est malheureusement pas possible de tester toutes les possibilités, l'ensemble des permutations pouvant être énorme (factorielle de la taille restante). Le problème de savoir si le système est assignable est donc difficile. Le 'Hungarian algorithm' nous apporte la solution à ce problème.

On sélectionne l'ensemble des robots non assignés, et on sélectionne l'ensemble des destinations qui ont été assignées à d'autres robots, mais qui au-

raient pu être assignées aux robots non assignés. Par après, on va sélectionner les robots qui ont été assignés à ces destinations sélectionnées, et les destinations auxquelles ils auraient pu être assignés. Cela de manière récursive, jusqu'à la stabilité des ensembles. On se retrouve avec un ensemble de robots et l'ensemble des destinations auxquelles ils peuvent être assignés. Si ces deux-ci sont de tailles différentes cela signifie que l'assignation ne peux être faite, et donc que le système n'est pas résolvable. Dans le cas contraire, le système est donc résolvable.

3.3.5.2 Création des chemins

Afin de créer un chemin entre deux points de l'espace, deux possibilités existent. S'il n'y a pas d'obstacles entre les deux points, le chemin est trivial. Dans le cas contraire, il est nécessaire d'utiliser l'algorithme de Dijkstra, en ayant préalablement connecté le point de départ et le point d'arrivée au graphe qui avait précédemment été construit.

Pour cela, on teste pour chaque noeud du graphe si un obstacle existe entre celui-ci et l'autre point considéré (départ ou arrivée). En pratique, nous ne modifierons pas le graphe directement, mais nous garderons les liens avec le départ et l'arrivée sur le côté pour des questions d'optimalité. Le pseudo-code de l'algorithme de Dijkstra est décrit dans l'algorithme 4. L'idée derrière celui-ci est de parcourir le graphe en étendant le noeud le moins loin du départ, et ce, jusqu'à ce que le noeud à étendre soit la destination.

En pratique, cela se fait en initialisant une liste de noeuds à visiter par le noeud de départ, et en étendant le noeud avec la plus petite distance. L'extension se fait en ajoutant à la liste des noeuds à visiter les noeuds connectés au noeud sélectionné. Si celui-ci a déjà été visité, cela signifie que la distance la plus courte pour l'atteindre était plus petite que la distance actuelle. S'il est déjà dans la liste des noeuds à visiter et que le noeud actuel permet d'y accéder de manière plus rapide, il faut alors mettre celui-ci à jour. Une fois que le noeud à visiter est la destination, vu que les noeuds restants se situent plus loin, on est sûr d'avoir obtenu la distance minimale vers celui-ci.

La récupération de la solution ce fait alors à rebours, en regardant le noeud précédant en partant de la fin, et ce jusqu'à arriver au noeud de départ.

3.3.5.3 Configuration

De manière similaire à ce qui a été fait pour le Controller, la Loop fonction possède également de nombreux paramètres devant être configurés dans le fichier de configuration de l'expérience. Cette configuration doit être choisie pour la configuration exécutée sur la machine servant de serveur de communication entre le Tracking system et les robots. Ceux-ci se situent dans la section argos-configuration > loop_function > path_planning. Si une valeur par défaut existe, elle est précisée et le paramètre est non obligatoire. Si une unité existe, elle est également spécifiée. En gras, le code de paramètre utilisé, il sera utilisé dans la section expérience (4) afin de clarifier les paramètres utilisés lors d'expériences. Ceux-ci suivent la notation suivante : **P**aramètre **L**oop function avec le premier chiffre pour rassembler les paramètres en sections. La section 1 concerne la distribution des destinations, et la 2 le budget alloué au placement (les conditions d'arrêt).

Liste des paramètres :

- **PL1.1** Distribution aléatoire des robots
- **PL1.2** Destinations partagées ou fixes
- **PL1.3** Nombre de séquences de positionnement avant de finir l'expérience
- **PL1.4** Nombre de robots
- **PL1.5** Type de robots
- **PL2.1** Nombre de temps en place avant de changer de séquence (ou de terminer l'expérience de placement) : 20 (x100 ms)
- **PL2.2** Nombre de temps maximal pour placer les robots : 0 (désactivé par défaut)
- **PL2.3** Pourcentage minimal nécessaire de robots en place : 1 (tous)
- **PL2.4** Temps maximum afin de laisser les dernier robots se mettre en place après le pourcentage atteint : 0 (x100 ms)

Si la distribution des robots est aléatoire :

— **PL1.6** Fichier où trouver la liste des seeds à utiliser
— **PL1.7** Type de distribution
— **PL1.8** Zone de placement et orientation
— **PL1.9** Nombre maximal d'essais de génération

Si la distribution des robots n'est pas aléatoire :
— **PL1.10** Liste de destinations et orientations

La majorité des paramètres parlent d'eux-mêmes, à l'exception du paramètre du nombre de séquences, celui-ci est surtout utile dans le cas où les destinations sont déterminées par l'utilisateur, et que celui-ci aimerait que les robots atteignent pas à pas une série de destinations. Une séquence correspond à une liste de destinations (une par robot). Une fois que tous les robots sont en place, ceux-ci entament alors la seconde séquence, et ce jusqu'à arriver à la dernière.

3.3.6 Utilisation pratique

D'un point de vue pratique, il est nécessaire que cet outil puisse être utilisé par l'ensemble des chercheurs, et ce, avec un minimum d'apprentissage, il se doit donc d'être simple et pratique. Toutefois, il se doit également d'être configurable afin de pouvoir s'adapter à l'ensemble des expériences. Enfin le Tracking system, lors de son développement, avait apporté la possibilité aux chercheurs de faire des statistiques sur les expériences réelles en temps réel. Le Tracking system n'acceptant pas la connection de deux instances d'ARGoS pour la même caméra, et que le positionnement se fait sur les mêmes caméras que l'expérience, il a été nécessaire d'offrir la possibilité de fusionner les expériences de positionnement et les statistiques que voudraient faire l'utilisateur sur des expériences réelles, et ce, manière simple.

Pour cela, il a été ajouté une série de paramètres permettant de charger une deuxième bibliothèque contenant la Loop function de l'utilisateur, et la classe de Loop function correspondante. Une nouvelle section a été également ajoutée dans le fichier de configuration de l'expérience afin de permettre l'envoi de tous les paramètres nécessaires à celle-ci comme si celle-ci était unique.

La Loop function utilisateur n'a d'ailleurs pas conscience de l'existance d'une autre Loop function.

L'idée, telle qu'illustrée sur la figure 3.6, est qu'un placement raté des robots implique de recommencer celui-ci avec le même seed. Une expérience ratée, quant à elle, recommence le placement d'avant l'expérience, et recommence la même expérience (bien que cela dépende de la Loop function de l'utilisateur). Si le placement et l'expérience réussissent, le prochain seed est chargé, et les nouvelles destinations de positionnement sont générées.

La définition d'une expérience réussie est laissée à l'utilisateur dans sa Loop function. Pour ce qui est de l'expérience de positionnement, la réussite peut être configurée à l'aide des paramètres fournis via le fichier de configuration. Par défaut, un placement est considéré comme réussi si tous les robots sont situés à la position souhaitée. Toutefois, en diminuant le pourcentage nécessaire de robots en place, celle-ci peut s'arrêter une fois qu'un pourcentage des robots sont placés. Cela permet d'assurer une certaine qualité de placement afin de permettre une certaine distribution dans l'arène. Le deuxième paramètre influençant directement la qualité du placement est le temps maximum alloué au placement des robots. Par défaut celui-ci est désactivé, mais pour des questions de rendement, on pourrait souhaiter limiter le temps imparti. À la fin de cette période, le Controller se coupe automatiquement comme si les robots étaient en place, et ils sont prêts à exécuter la prochaine expérience.

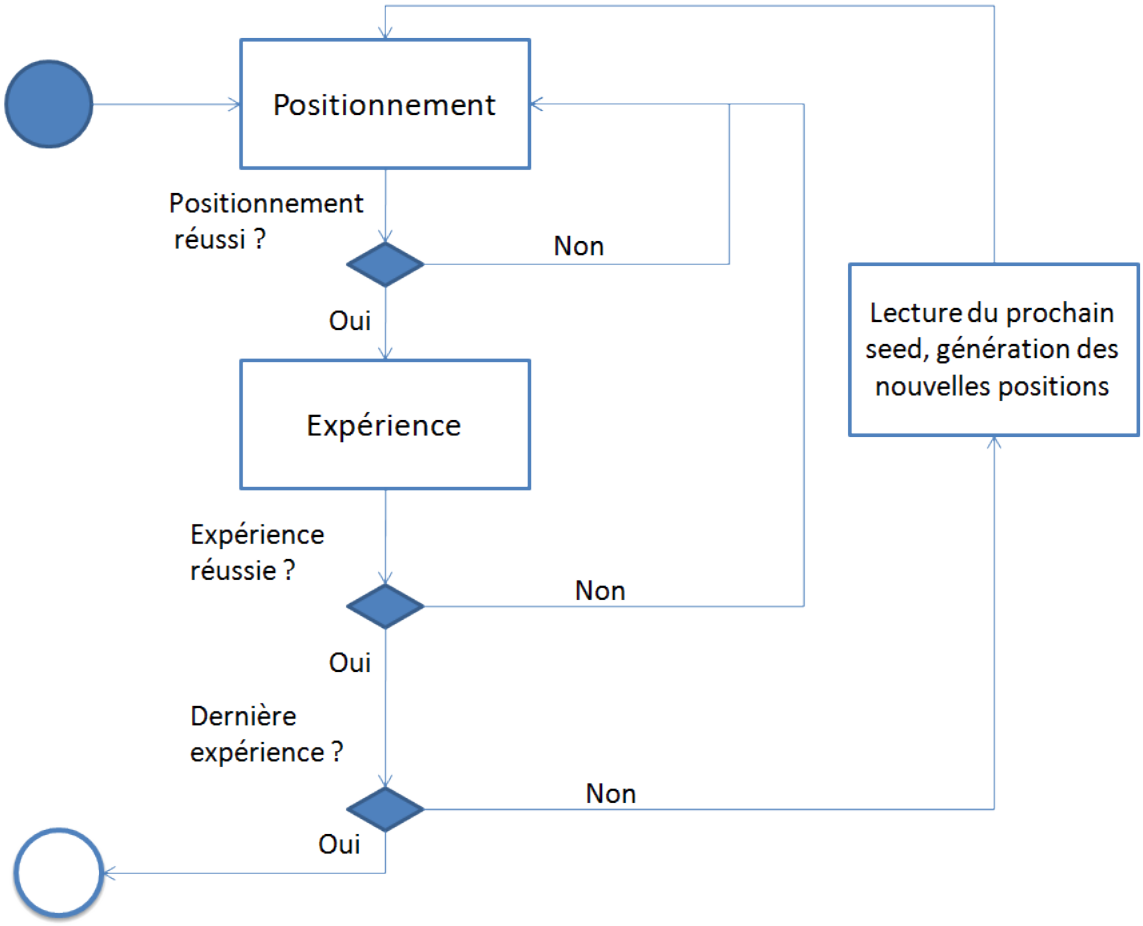

FIGURE 3.6 – Machine a états finis utilisée pour la gestion des Loop function à exécuter

```
Data: Matrix of distance robot - destination
Data: Threshold value
Consider all value in matrix under the threshold as zeros;
System is deterministic;
if No zero in a line or column then
    return system not resolvable;
end
while some robot not assigned do
    if Only one zero in a line or column then
        Assign the robot(line) to the destination(column);
        Remove the corresponding line & column from the matrix;
    else if Zero zeros in a line or column & system deterministic then
        return system not resolvable
    else if No line/column with less than two zeros then
        Randomly choose one zero and assign it;
        Remove the corresponding line & column from the matrix;
        System is not deterministic anymore;
    else if No zeros remaining & system non deterministic then
        In base matrix;
        Select all unassigned lines;
        while something is added do
            Select all columns in which there is a zero in a selected line;
            Select all lines that has been assigned in a selected columns;
        end
        if #selected columns + #unselected lines == #robots then
            return system resolvable
        else
            return system not resolvable
        end
    end
end
All robots assigned, system is resolvable;
```

Algorithm 3: Pseudo-code de l'algorithme de résolvabilité

```
Data: Graph with length between nodes
Data: List of nodes linked to start position and their distance
Data: List of nodes linked to goal position and their distance
for all nodes do
    if node connected to start then
        distance[node] ← length between node and start;
        previous[node] ← start;
        add node to the visitable_nodes group;
    else
        distance[node] ← infinity;
        previous[node] ← NULL;
    end
end
while there is a node in the visitable group and distance[node] <
minimal distance found to goal do
    selected_node ← min ( distance[visitable_nodes] ) for all nodes
    connected to selected_node do
        if not already visited & distance[selected_node] +
        length(selected_node,node) < distance[node] then
            distance[node] ←
            distance[selected_node]+length(selected_node,node);
            previous[node] ← selected_node;
            add node to the visitable group;
        end
    end
    if selected node connected to goal then
        if actual minimum distance to goal > distance[selected node] +
        length(selected_node,goal) then
            minimum distance to goal ← distance[selected node] +
            length(selected_node,goal);
            previous[goal] ← selected node;
        end
    end
end
listNodes[0] ← (goal);
while listNodes[0] != start do
    listNodes ← (previous[listNodes[0]], listNodes);
end
return listNodes;
```

Algorithm 4: Pseudo-code de l'algorithme de Dijkstra

Chapitre 4

Expérience

Afin de tester le système ainsi construit, une série d'expériences a été réalisée dans différentes conditions. Cette section tentera de décrire chacune d'entres-elles. Parmi celles-ci, malheureusement seulement une a pu être faite dans des conditions réelles. Il était en effet nécessaire d'avoir une expérience programmée afin de pouvoir tester l'outil de positionnement sur celle-ci. Les autres ont été créées afin de tester l'outil de positionnement dans différents cas.

4.1 Première expérience : Agrégation

Cette expérience a été menée dans le cadre du développement du Tracking System (section 2.4) par Mattia Salvaro. Afin d'incorporer les résultats des capteurs virtuels, une expérience d'agrégation a été menée à l'aide d'un Controller généré par le système AutoMoDe[8]. L'expérience consiste en l'exécution du même Controller en utilisant une fois les capteurs réels du robot, et l'autre fois les capteurs virtuels. Il était espéré que les capteurs virtuels donneraient de meilleurs résultats que les capteurs réels.

Entre chaque instance d'expérience, les robots ont été mis en place à l'aide de l'outil de positionnement. Chaque paire d'expériences (une virtuelle et une réelle) était positionnée à l'aide du même seed, c'est-à-dire à des positions identiques. 20 seeds différents ont étés utilisés, soit 40 positionnements à

faire. Voici les paramètres qui ont été utilisés pour l'expérience.

Code	Paramètres	Valeur
Placement		
PC1.1	placementPrecision	0.03
PC1.2	anglePrecision	5
PL2.1	timestepInPosition	20
capteur de proximité		
PC2.1	proximitySignificativeThreshold	0.4
PC2.2	proximityRepulsionForce	12
PC2.3	proximityRepulsionForceNoise	4
Batterie		
PC3.1	batteryTimeStepThreshold	60
PC3.2	recharge_destination	(0,-1)
Erreur de détection		
PC4	forceRandomWalkAfterNoUpdateFor	0
Destinations		
PL1.1	randomDistribution	TRUE
PL1.2	sharedDestination	TRUE
PL1.7	distribution	uniform
Destination generation		
PL1.3	sequence	1
PL1.4	robots	20
PL1.5	robotType	epuck
PL1.8	zone of placement	(-1,-1) to (1,1)
PL1.8	orientation of placement	0 to 360 °
PL1.9	max_trials	100
Conditions d'arrêt		
PL2.2	tickBudget	600
PL2.3	minProcentPlacementBudget	0.8
PL2.4	maxTimeAfterBudgetPlacement	30

TABLE 4.1 – Paramètres de l'expérience 1

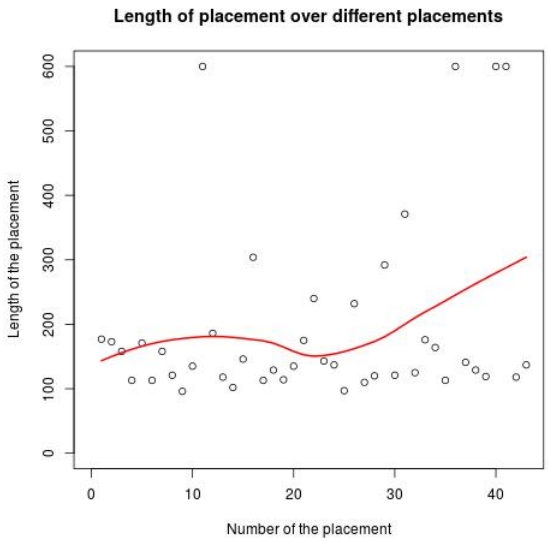

FIGURE 4.1 – Distribution des temps de placement en fonction de l'itération de placement

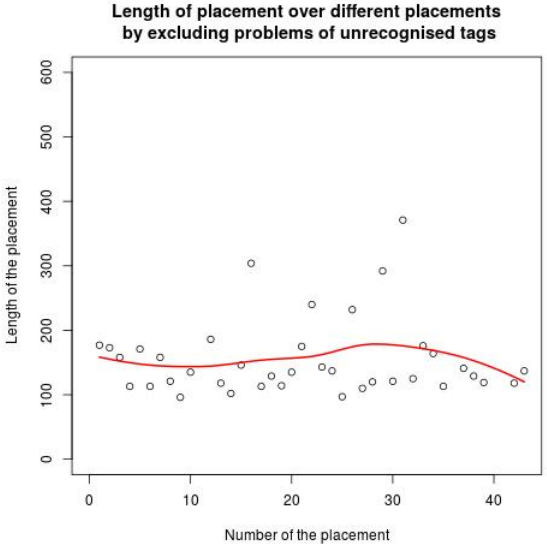

FIGURE 4.2 – Distribution des temps de placement en fonction de l'itération de placement en excluant celles où trop de tags sont non reconnus

Au fur et à mesure des expériences, les images prises par le Tracking system ainsi que les statistiques ont été enregistrées. Les tables comprenant l'ensemble des résultats sont disponibles en annexe (6.1). Celles-ci consistent en la liste des algorithmes exécutés, l'ordre, les résultats de l'expérience réelle et les erreurs rencontrées au fur et à mesure des exécutions.

De manière globale, l'expérience s'est très bien déroulée au début, puis au fur et à mesure des expériences, les problèmes techniques se sont montrés de plus en plus nombreux. Alors que pour la première moitié, une seule erreur ayant nécessité le redémarrage complet du système a pu être observée, dès que le niveau des batteries est devenu critique, le système est devenu instable. Pour la réalisation des quatre dernières expériences, de nombreux redémarrages ont été nécessaires. À un tel point que les deux dernières n'ont pas été faites.

En observant la figure 4.1, nous pouvons observer la répartition des temps nécessaires au placement (en dixième de secondes), on voit quelques cas anor-

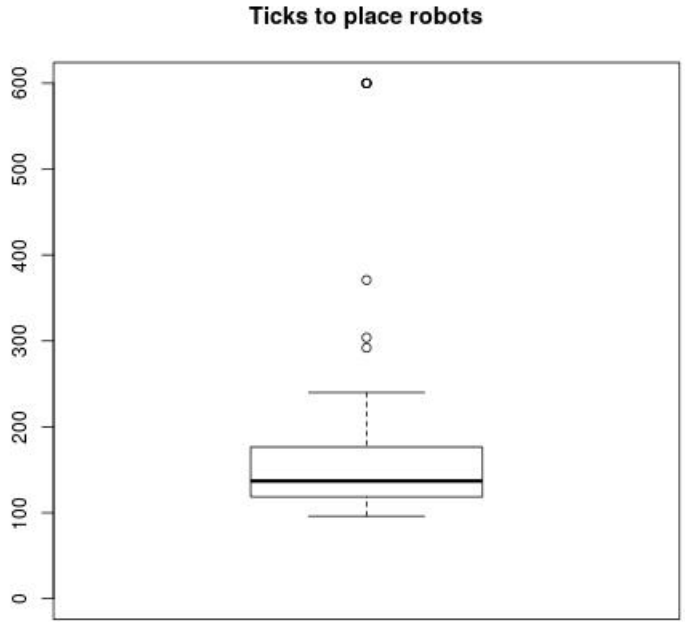

FIGURE 4.3 – Répartition des temps de placements

maux pour lesquels le placement a duré 60 secondes, ce sont des placements où les tags étaient mal reconnus, ou pour lesquels le robot avait un problème technique. En excluant ces anomalies, on obtient la figure 4.2, où l'on peut observer que le placement dure en moyenne moins de vingt secondes, et est réparti de manière plus ou moins uniforme. Sur la boite de Tukey(boxplot : figure 4.3), on peut également observer que plus des trois quarts des placements se situent en dessous de 20 secondes. Ceci est un excellent résultat sachant que celui-ci est réalisé avec 20 robots, qui nécessiteraient entre une minute et une minute trente à placer si ceci devait être fait à la main.

Ce résultat est malheureusement à contrebalancer avec un problème technique rencontré lors des expériences. Dans presque tout les cas, il a en effet été nécessaire de redémarrer le Linux embarqué de quelques robots. Cela est dû au fait que le Controller ne se coupait pas correctement. Il est fort probable que l'implémentation du Tracking System en soit la cause. Cette

erreur devrait être corrigée au plus vite afin d'obtenir un système réellement exploitable. Elle amène en effet à perdre une à deux minutes à chaque erreur, ce qui amène à de mauvais résultats en pratique. Ceux-ci deviennent même moins bons qu'un placement manuel.

Avec cette correction, nous pourrions amener le système à placer les robots de manière systématique en moins de 30 secondes si les tags sont reconnus correctement. Ètant donné que le problème de reconnaissance est en général lié à des tags courbés, il faudrait s'arranger pour soit les rendre plus résistants, soit diminuer légèrement leur taille, afin que quand deux robots se touchent, ceux-ci ne puissent rentrer en contact.

Une remarque peut également être faite sur le rechargement. A cause des paramètres de l'expérience, le placement se coupe après le positionnement de 80% des robots, ce qui signifie que si moins de 4 robots ont besoin de se recharger, il est difficile d'observer ce comportement. Il a été observé une seule fois, et les batteries ont dû être changées normalement tout au long des expériences.

4.2 Deuxième expérience : Influence du budget alloué au placement (PL2.3)

L'expérience est réalisée afin de réaliser l'impact de la précision demandée (pourcentage de robots placés nécessaire afin de considérer un placement comme correct). L'expérience consiste à demander aux robots de prendre des positions aléatoires définies, et une fois le pourcentage requis atteint, générer une nouvelle liste de positions à atteindre. Les mêmes positions sont utilisées pour tester les différentes valeurs du paramètre. Le temps moyen est calculé sur le placement de 10 positions. Les autres paramètres sont définis par défaut.

Robots minimum en position	10	12	14	16	18	20
Temps	3.58 s	4.61 s	4.90 s	5.22	5.71 s	6.6 s

TABLE 4.2 – Variation du temps en fonction du nombre de robots minimum à placer

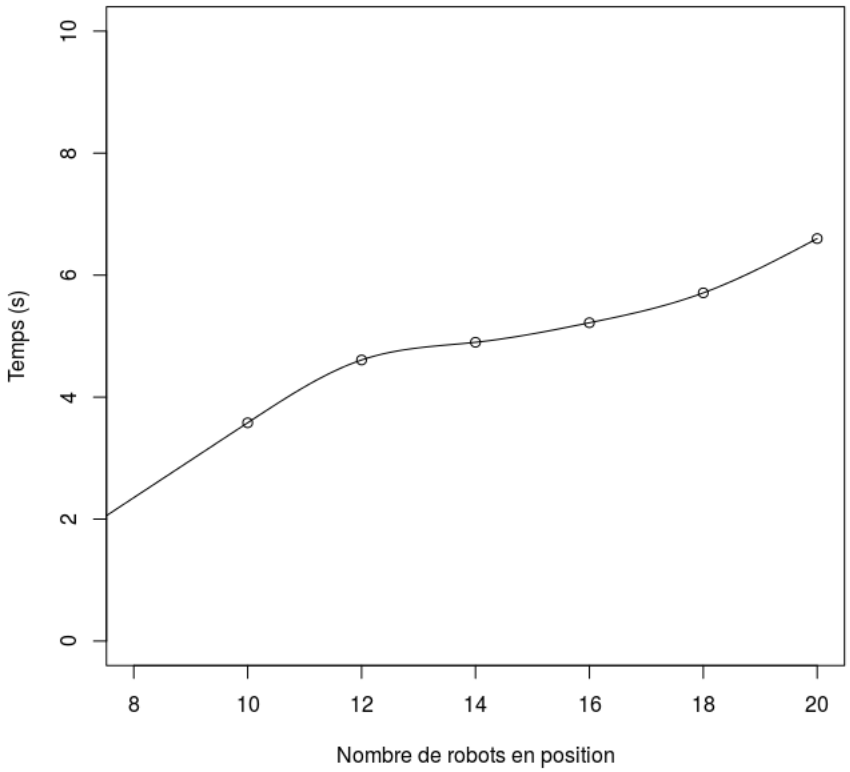

FIGURE 4.4 – Temps en fonction du nombre de robots devant être en place

On peut observer ici que les résultats sont bien meilleurs que ce à quoi l'on pourrait s'attendre, c'est en effet causé par la répartition des robots qui est déjà répartie de manière uniforme. Le but de cette expérience n'est pas de déterminer le bon fonctionnement de l'algorithme de positionnement, mais d'essayer de comprendre le temps perdu par le fait d'augmenter le nombre minimum de robots devant aller en place.

On peut se demander le comportement dans le cas d'expériences réelles, mais on peut s'attendre à une courbe de la même allure. Cette expérience a en

effet tenté de supprimer une majorité des facteurs aléatoires en remettant les robots dans la même position initiale et en les faisant parcourir les mêmes destinations. Dans une expérience ou les robots ne seraient pas distribués équitablement, on peut imaginer que le temps global sera plus long, et que les robots plus lointains seront ceux qui ne seront pas placés. Si nous faisons une approximation de la courbe par une droite, on peut donc supposer que le coefficient angulaire dépend de la distance entre les robots et leur destination.

4.3 Troisième expérience : Gestion des obstacles

La troisième expérience a pour but de tester la robustesse du système, en le mettant dans un cas extrême. Pour cela, l'arène est divisée en deux zones par un mur ayant une forme crochet. La figure 4.5 montre l'arène telle qu'utilisée pour l'expérience. Les robots ont servi à définir les objets dans ARGoS, tel qu'observé sur la figure 4.6. L'ensemble des robots est alors placé du même côté du crochet, comme exposé sur les figures 4.7 et 4.8. L'algorithme de positionnement est alors lancé, les destinations sont des points aléatoires de l'autre côté de l'arène. Sur la figure 4.9 représentant ce que peut voir l'utilisateur sur ARGoS, on peut observer en vert le graphe construit pour le path planning, en rouge la destination actuelle, et en bleu les destinations suivantes, qui seront considérées une fois arrivés à la destination courante.

L'expérience a été lancée trois fois et mène à des résultats similaires : une partie des robots part de chaque côté, mais là où la grosse majorité va, il y a un phénomène de bouchon à l'endroit où ils doivent tourner. Ils doivent en effet tous en même temps passer à un point précis. Dans le cas où un robot arriverait au point et qu'il est entouré d'autres robots, après que la force qui le tirait vers l'avant s'estompe, il est repoussé violemment vers l'arrière suite aux capteurs de proximité. Dans ce cas-là, s'il part du mauvais côté, il y a de grandes chances qu'il reste bloqué dans le crochet, comme observé sur la figure 4.10. Le reste du placement se fait correctement, pour le résultat final, comme exposé sur la figure 4.11. Lors de cette expérience particulière, un

robot n'a pas bougé, il s'est malheureusement redémarré lors du lancement de l'expérience.

Dû à l'implémentation actuelle de l'algorithme, il y avait de grandes chances à ce que cela puisse ce produire. Afin de résoudre ce problème, diverses options sont possible. La première serait de vérifier à intervalle régulier la possibilité pour le robot d'atteindre non pas sa destination actuelle, mais la suivante. De cette manière, le robot serait capable de sauter une destination si la suivante est atteignable, ce qui pourrait résoudre le problème. Toutefois, un robot qui se serait retrouvé bloqué comme exposé si dessus ne pourrait pas se retrouver. Il devrait y avoir également la possibilité d'ajouter un noeud si la destination n'est plus atteignable. On pourrait également imaginer changer le path planning en recalculant le Dijkstra de manière régulière, ce qui résoudrait le problème, mais demanderait du temps de calcul plus important. Ces deux solutions sont envisageables et pourraient être également vues comme un choix laissé à l'utilisateur.

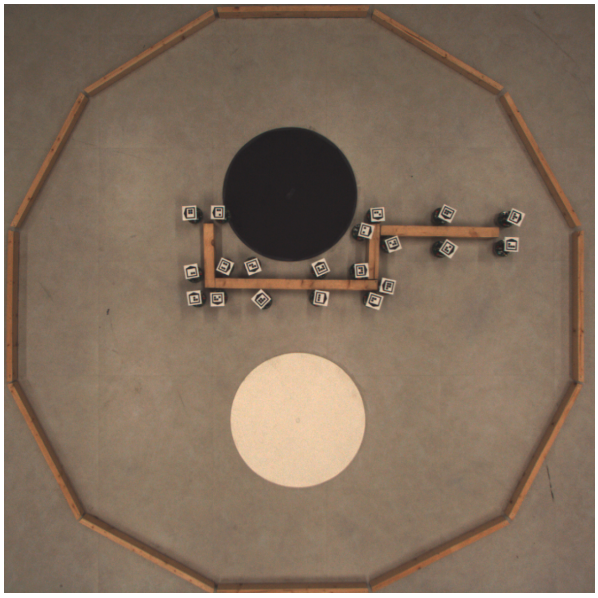

FIGURE 4.5 – Positionnement de l'obstacle dans l'arène

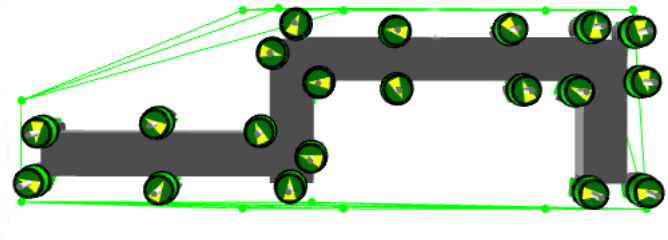

FIGURE 4.6 – Création de l'obstacle dans ARGoS

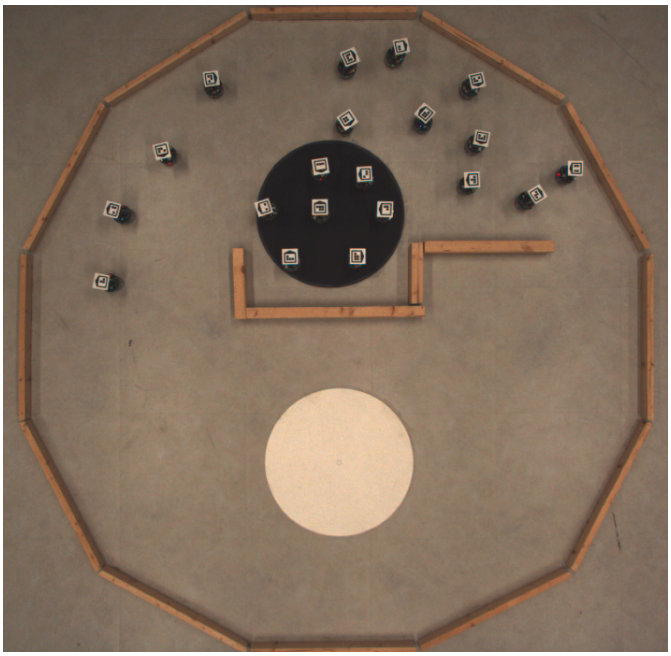

FIGURE 4.7 – Positionnement initial des robots dans l'arène

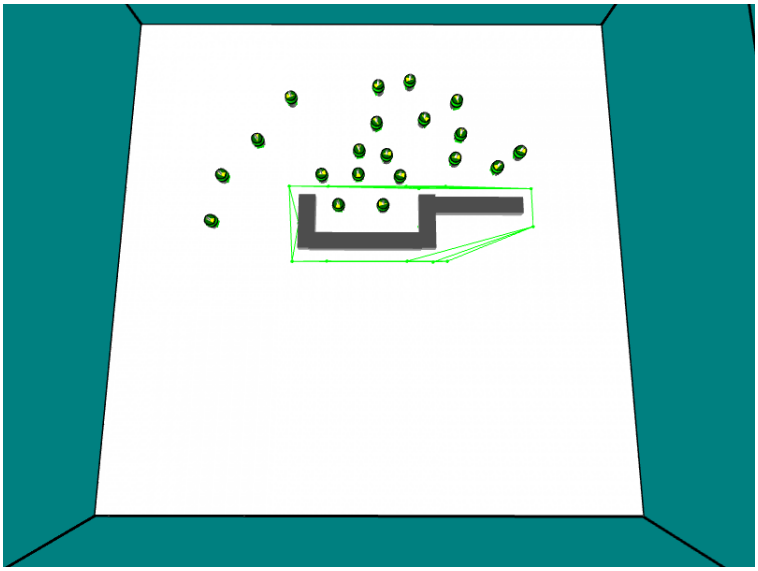

FIGURE 4.8 – Positionnement initial des robots dans ARGoS

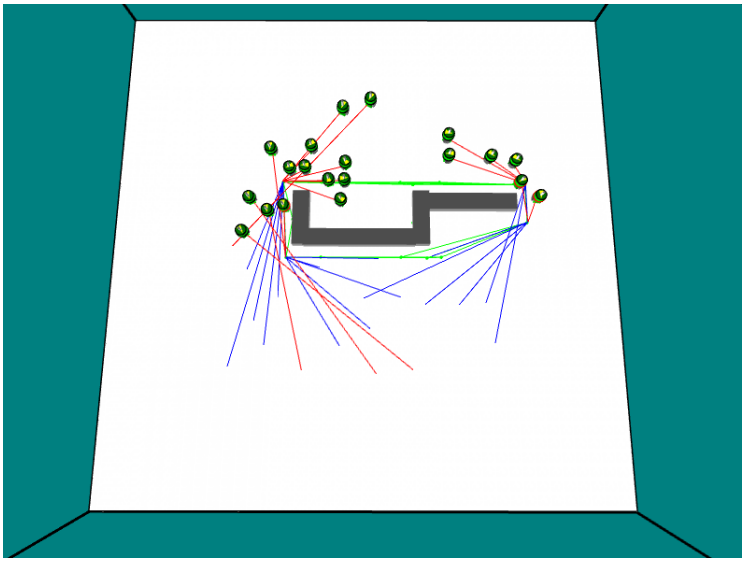

FIGURE 4.9 – Path planning des robots

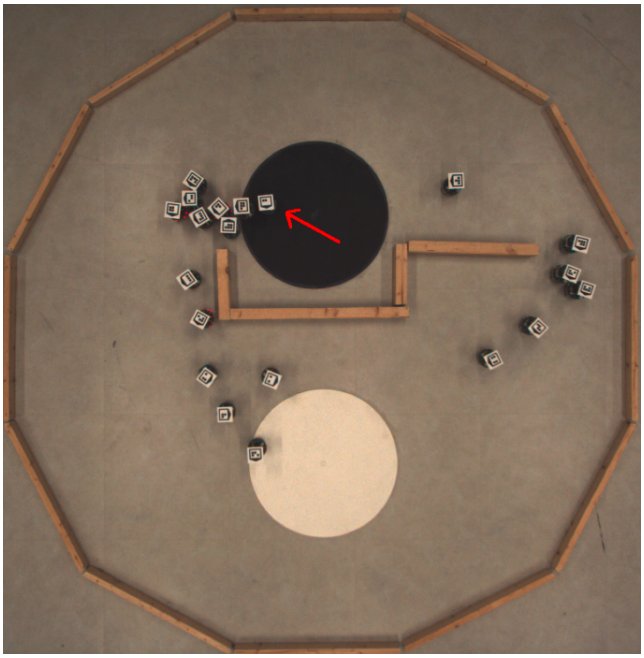

FIGURE 4.10 – Robot bloqué suite à un retour en arrière

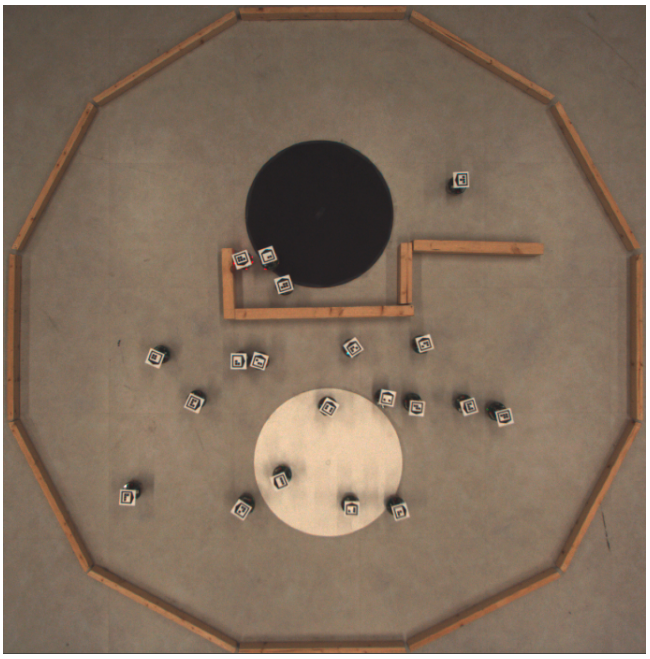

FIGURE 4.11 – Positionnement des robots après 30 secondes

4.4 Extensions possibles

Grâce aux expériences, nous avons pu mettre en avant certains petits défauts dont la correction permettrait de passer d'une version utilisable à une version déployable. Ceux-ci consistent principalement en la correction du Controller des robots, dont la version spécifique au Tracking system plante de manière trop fréquente lorsque le programme devrait être quitté.

Également lors d'expériences, il a été remarqué que parfois les tags qui se retrouvaient au bord de l'espace géré par la caméra n'étaient plus reconnus, et que ARGoS recevait parfois pour ceux-ci des coordonnées aberrantes. Dans ce cas, il est impossible de récupérer le robot en simulation, et la seule solution trouvée à l'heure actuelle est de redémarrer le serveur du Tracking system. Corriger cette erreur devrait également permettre d'améliorer le système de manière significative.

Une amélioration importante serait celle d'empêcher que les robots puissent se retrouver bloqués suite au phénomène expliqué lors de la troisième expérience (section 4.3). Des pistes ont été introduites afin de solutionner ce problème.

Rendre les tags plus résistants ou s'arranger pour que ceux-ci ne puissent se toucher (en réduisant légèrement leur taille) pourrait permettre que ceux-ci ne se plient pas légèrement tel que cela peut se passer actuellement. Cela éviterait la diminution de l'efficacité du Tracking sytem.

Le système a été conçu afin de pouvoir être adapté aux Foot-bots aisément, il serait intéressant de faire les modifications nécessaires afin de permettre l'utilisation par de multiples robots. Cela démontrerait la robustesse et permettrait l'ajout éventuel d'autres robots de manière aisée.

Actuellement, la définition des chemins pour le path planning ne peut prendre en compte de petits objets complexes (voir section 3.3.3.2), il serait intéressant de posséder une manière facile de faire des extensions d'objets.

L'assignation des destinations se fait actuellement à l'aide de la résolution du 'linear bottleneck problem', l'implémentation de la seconde solution cherchant à minimiser la distance totale, comme vue dans la section 3.3.5, et laisser le choix à l'utilisateur de l'algorithme à utiliser serait un plus.

Afin de rendre l'outil plus pratique, l'introduction de zones circulaires

pour la génération des positions permettrait de couvrir l'arène des expériences entièrement au lieu de devoir définir une zone rectangulaire incluse dans celle-ci.

Un outil de génération de fichier de configuration ARGoS pourrait simplifier l'utilisation de l'outil de positionnement en le configurant à l'aide d'une interface graphique. Cela consisterait en la liste des paramètres, mais également la liste des robots avec leurs tags. On pourrait également superposer le placement des objets (boites et cylindres) aux images obtenues par le Tracking system.

La prochaine étape dans la réalisation d'un outil automatique serait de permettre à celui-ci de lancer par lui-même les Controllers sur les robots. Cela serait réalisable à l'aide des outils existants tels que ssh. Cela impliquerait par contre une perte totale du contrôle de l'humain sur les expériences. Au vu de la fiabilité actuelle des robots, une telle solution semble toutefois pour l'instant peu envisageable.

Chapitre 5

Conclusion

Dans ce mémoire, nous avons développé un outil de positionnement pour les chercheurs en Swarm intelligence. Il en résulte un outil utilisable, mais pas encore entièrement déployable, pour ce faire, quelques corrections devraient encore être réalisées. Afin de le finaliser, les problèmes sur les programmes dépendants dvraient être résolus.

Les expériences sur le système que j'ai eu l'occasion de développer nous ont montré l'efficacité potentielle que celui-ci pourrait développer une fois les corrections du Tracking system effectuées, et les quelques légères modifications qui pourraient également être apportées si le besoin ce fait sentir. Il est toutefois prêt pour des tests à plus grande échelle qui pourraient mettre à la lumière d'éventuelles améliorations qui pourraient encore lui être apportées.

De plus la structure mise en place devrait pouvoir permettre à l'avenir de nombreuses mises à jour et extensions, tout particulièrement des capteurs. Les Foot-bots devraient d'ailleurs être les premiers à pouvoir en bénéficier.

Chapitre 6

Annexes

6.1 Expérience 1 : Résultats de l'expérience

# run	Time (x100ms)	Comment
001	3	Buffer update
002	177	Path Planning
003	1200	Real Sensors
004	173	Path Planning
005	1200	Virtual Sensors
006	158	Path Planning
007	1200	Virtual Sensors
008	113	Path Planning
009	1200	Real Sensors
010	171	Path Planning
011	1200	Virtual Sensors
012	113	Path Planning
013	1200	Real Sensors
014	158	Path Planning
015	1200	Virtual Sensors
016	121	Path Planning
017	1200	Real Sensors
018	584	Too much robot reboot needed

# run	Time (x100ms)	Comment
019	167	Still some reboot needed
020	96	Path Planning
021	1200	Virtual Sensors
022	135	Path Planning
023	1200	Real Sensors
024	600	Path Planning
025	1200	Real Sensors
026	186	Path Planning
027	1200	Virtual Sensors
028	118	Path Planning
029	1200	Virtual Sensors
030	102	Path Planning
031	1200	Real Sensors
032	6	Forgot to connect the robots
033	146	Path Planning
034	1200	Real Sensors
035	304	Path Planning
036	1200	Virtual Sensors
037	113	Path Planning
038	1200	Real Sensors
039	129	Path Planning
040	1200	Virtual Sensors
041	114	Path Planning
042	1200	Real Sensors
043	135	Path Planning
044	1200	Virtual Sensors
045	175	Path Planning
046	1200	Real Sensors
047	310	Robot errors -> REBOOT ?
048	158	Robot errors -> REBOOT ?
049	306	Robot errors -> REBOOT ?
050	51	Robot errors -> REBOOT ?

# run	Time (x100ms)	Comment
051	240	Path Planning
052	4	Changing battery
053	1	Confirmation ok
054	1200	Virtual sensors
055	143	Path Planning
056		EXPERIMENT FAIL
		REBOOT SYSTEM
057	3	Buffer update
058	137	Path Planning
059	1200	Virtual Sensors
060	97	Path Planning
061	1200	Real Sensors
062	232	Path Planning
063	1200	Virtual Sensors
064	110	Path Planning
065	642	EXPERIMENT FAIL
066	120	Path Planning
067	1200	Real Sensors
068	292	Path Planning
069	1200	Virtual Sensors
070	121	Path Planning
071	1200	Real Sensors
072	371	Path Planning
073	1200	Virtual Sensors
074	125	Path Planning
075	1200	Real Sensors
076	176	Path Planning
077		EXPERIMENT FAIL
		REBOOT SYSTEM
078	2	Buffer update
079	164	Path Planning

# run	Time (x100ms)	Comment
080	143	Forgot the light for Real Sensors
081	113	Path Planning
082	1200	Real Sensors
083	600	Path Planning
084	1200	Virtual Sensors
085		PATH PLANNING FAIL
		REBOOT SYSTEM
086	2	Buffer update
087	141	Path Planning
088	1200	Virtual Sensors
089	129	Path Planning
090	1200	Real Sensors
091	119	Path Planning
092	1200	Real Sensors
093	600	Path Planning
094	1200	Virtual Sensors
095	600	Path Planning
096		EXPERIMENT FAIL
		REBOOT SYSTEM
097	5	Buffer update
098	5	Buffer update
099	118	Path Planning
100	1200	Real Sensors
101	137	Path Planning
102	1200	Virtual Sensors
103		REBOOT SYSTEM

TABLE 6.1 – Ensemble de toutes les instances de l'expérience et répartition

# Algorithm	Result	Time	Run
RealSensors	0.000134644	1200	003
VirtualSensors	0.00015024	1200	005
VirtualSensors	0.000112082	1200	007
RealSensors	8.43668e-05	1200	009
VirtualSensors	9.20895e-05	1200	011
RealSensors	6.88563e-05	1200	013
VirtualSensors	7.44934e-05	1200	015
RealSensors	0.000197511	1200	017
VirtualSensors	0.00012282	1200	021
RealSensors	8.89442e-05	1200	023
RealSensors	7.16589e-05	1200	025
VirtualSensors	0.000178063	1200	027
VirtualSensors	0.000111982	1200	029
RealSensors	7.88457e-05	1200	031
RealSensors	0.000104373	1200	034
VirtualSensors	0.000117178	1200	036
RealSensors	7.7682e-05	1200	038
VirtualSensors	9.68992e-05	1200	040
RealSensors	9.11162e-05	1200	042
VirtualSensors	7.77786e-05	1200	044

# Algorithm	Result	Time	Run
RealSensors	0.000197006	1200	046
VirtualSensors	9.5229e-05	1200	054
VirtualSensors	0.00010866	1200	059
RealSensors	7.38607e-05	1200	061
VirtualSensors	0.000137061	1200	063
RealSensors	6.80967e-05	1200	067
VirtualSensors	0.00011241	1200	069
RealSensors	0.000118751	1200	071
VirtualSensors	0.000140944	1200	073
RealSensors	0.000134372	1200	075
RealSensors	0.00042337	1200	082
VirtualSensors	0.000123839	1200	084
VirtualSensors	0.000169463	1200	088
RealSensors	0.000147341	1200	090
RealSensors	0.000165317	1200	092
VirtualSensors	0.000148987	1200	094
RealSensors	8.2021e-05	1200	100
VirtualSensors	0.000208507	1200	102

TABLE 6.2 – Résultats de l'expérience d'agrégation

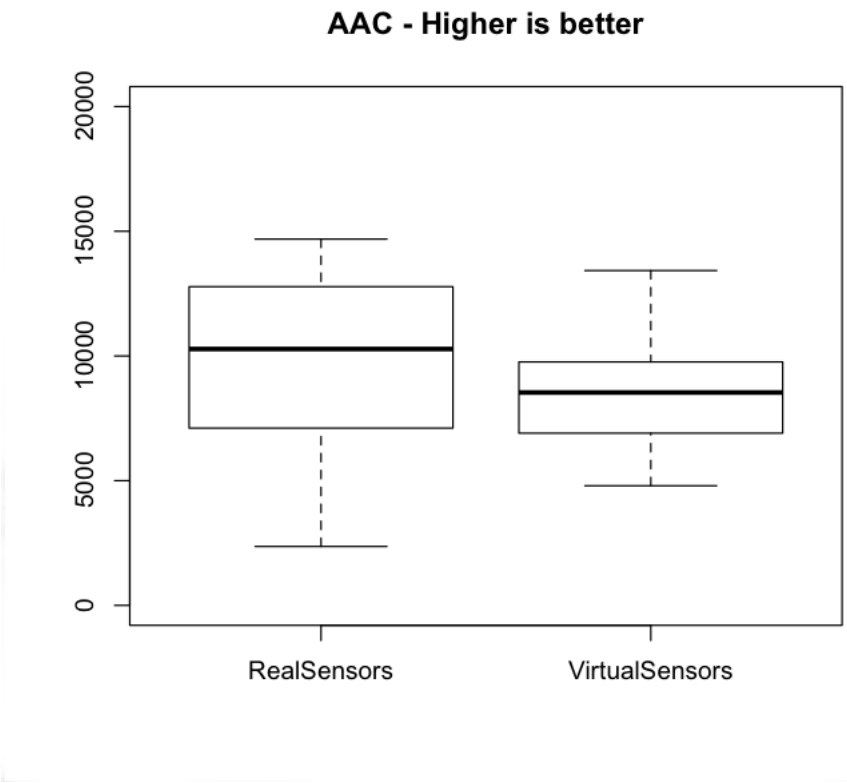

FIGURE 6.1 – Boxplot des résultats de l'expérience d'agrégation

Bibliographie

[1] Dorigo & al. The swarm-bots project. *Swarm Robotics, SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers*, pages 31–44, 2005.

[2] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics : A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1) :1–41, 2013.

[3] Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo. From fireflies to fault-tolerant swarms of robots. *IEEE Trans. Evolutionary Computation*, 13(4) :754–766, 2009.

[4] Ecole Polytechnique Fédérale de Lausanne. E-puck project. `http://www.e-puck.org/`, February 2014. [Site web ; consulté le 17-août-2014].

[5] M. Dorigo. Ant colony optimization. *Scholarpedia*, 2(3) :1461, 2007. revision 90969.

[6] M. Dorigo, M. Birattari, and M. Brambilla. Swarm robotics. *Scholarpedia*, 9(1) :1463, 2014. revision 138643.

[7] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. Guzzi, V. Longchamp, S. Magnenat, J. Martinez Gonzales, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid : A novel concept for the study of heterogeneous

robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4) :60–71, 2013.

[8] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. Automode : A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2) :89–112, 2014.

[9] IRIDIA. Swarmanoid project. `http://www.swarmanoid.org/`, August 2011. [Site web ; consulté le 17-août-2014].

[10] J. Zufferey D. Floreano J. Roberts, T. Stirling. Quadrotor using minimal sensing for autonomous indoor flight. In *European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, September 2007.

[11] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A comprehensive survey : artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42(1) :21–57, 2014.

[12] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2 :83–97, 1955.

[13] Steven M LaValle. Planning algorithms, 2004.

[14] Kristina Lerman and Aram Galstyan. Mathematical model of foraging in a group of robots : Effect of interference. *Autonomous Robots*, 13(2) :127–141, 2002.

[15] P. Rétornaz F. Mondada M. Bonani, S. Magnenat. The hand-bot, a robot design for simultaneous climbing and manipulation. In M. Xie et al., editor, *Proceedings of the Second International Conference on Intelligent Robotics and Applications (ICIRA2009)*, LNAI, pages 11–22. Springer-Verlag, 2009.

[16] MVTech. Mvtech software gmbh. halcon library website. `http://www.mvtec.com/`, August 2014. [Site web ; consulté le 17-août-2014].

[17] Ulrich Pferschy. Solution methods and computational investigations for the linear bottleneck assignment problem. *Computing*, 59(3) :237–258, 1997.

[18] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS : a modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA, September 2011.

[19] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS : a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4) :271–295, 2012.

[20] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot : A low cost scalable robot system for collective behaviors. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3293–3298. IEEE, 2012.

[21] Ruud Schoonderwoerd, Janet L. Bruten, Owen E. Holland, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adapt. Behav.*, 5(2) :169–207, September 1996.

[22] A. Stranieri, A.E. Turgut, M. Salvaro,
G. Francesca, A. Reina, M. Dorigo, and M. Birattari. Iridia's arena tracking system. Technical Report TR/IRIDIA/2013-013, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, July 2013.

[23] Jur van den Berg. *Path Planning in Dynamic Environments*. PhD thesis, Utrecht University, The Netherlands, 2007.