


Stage recherche fin Esial au Loria

Coordination d'un essaim de robots mobiles (30 Kilobots) par des mécanismes inspirés de l'intelligence collective observée chez les animaux sociaux.

Auteur : Thomas Moinel

Encadrant : François Charpillet

Encadrant universitaire : Gérald Oster

20 août 2012

Table des matières

Chapitre 1

Contexte

1.1 Introduction

J'ai effectué ce stage de six mois de troisième année concluant ma formation d'ingénieur à l'ESIAL. J'ai choisi un stage de recherche ayant le projet de continuer dans le domaine de la recherche académique. Le thème de ce stage est l'intelligence artificielle, et plus particulièrement les systèmes multi-agents dont les applications se retrouvent en robotique et dans l'informatique décisionnelle.

Les systèmes multi-agents sont largement inspirés du monde biologique, des animaux sociaux comme par exemple le déplacement des poissons en bancs, la recherche de nourriture par des fourmis trouvant le chemin le plus court, etc.. Le principe est que des agents tirent un bénéfice de leurs interactions, coopérations et/ou compétitions.

Plusieurs objectifs m'ont été fixé :
- prendre en main la plateforme des Kilobots,
- déterminer si cette plateforme est viable pour effectuer du flocking,
- implémenter des algorithmes inspirés d'animaux sociaux sur les kilobots, plus particulièrement du flocking.

Je vais réaliser tout d'abord une description de la plateforme des kilobots (petits robots) afin de déterminer leur capacités et leur limites.

Ensuite je présenterai le flocking et les prérequis que les kilobots n'ont pas : une méthode de synchronisation adaptée, décentralisée, robuste et efficace ainsi qu'un comparatif de plusieurs méthodes de localisation. Par ailleurs, je proposerai l'amélioration de l'une d'elle.

Enfin j'effectuerai un bilan de la plateforme et poserai des questions non abordées pouvant faire l'objet de travaux supplémentaires et de directions à prendre.

1.2 Remerciements

Je tiens à remercier tout d'abord François Charpillet, responsable de l'équipe Maia, de m'avoir proposé et adapté le sujet de ce stage, d'avoir été disponible malgré son emploi du temps chargé, d'avoir répondu à mes nombreuses questions et plus généralement pour son encadrement.

Je remercie Gérald Oster d'avoir été disponible pour le suivi de ce stage.

Je remercie également toute les personnes de la salle C104 du Loria pour l'agréable ambiance de travail qui y régnait.

Je remercie aussi le personnel technique du Loria, les chercheurs et plus particulièrement ceux de l'équipe Maia pour leur ouverture d'esprit, leur disponibilité au dialogue ainsi que leur cordialité.

Et enfin, je remercie toutes les personnes m'ayant aidé, conseillé ou relu à la rédaction de ce rapport notamment Sébastien Vandeneeckhoutte pour avoir co-écrit la présentation de l'entreprise.

1.3 Entreprise et équipe d'accueil

1.3.1 Inria

L'Inria [1] se présente comme étant le seul institut public de recherche entièrement dédié aux sciences du numérique et a pour ambition de mettre en réseau les compétences et talents de l'ensemble du dispositif de recherche français dans le domaine des sciences informatiques et mathématiques.

L'Inria en quelques chiffres (2010)

Infrastructure : 8 centres de recherche répartis dans toute la France (Rocquencourt, Rennes, Sophia Antipolis, Grenoble, **Nancy**, Bordeaux, Lille et Saclay) et un siège social à Rocquencourt près de Paris.

1. INRIA : Institut National de Recherche en Informatique et en Automatique.

Activités scientifiques :
- 208 équipes de recherche, dont 171 équipes-projets Inria
- 4 850 publications scientifiques
- 288 thèses soutenues

Relations Industrielles :
- 271 brevets actifs
- 111 logiciels déposés à l'Agence pour la Protection des Programmes
- 105 sociétés de technologie issues d'Inria

1.3.2 Loria

Le LORIA [2], est une Unité Mixte de Recherche [3] (UMR 7503) commune à plusieurs établissements : le CNRS, l'Université de Lorraine et l'Inria.

Le LORIA a pour mission la recherche fondamentale et appliquée en sciences informatiques, et s'inscrit donc logiquement dans les domaines de recherche mis en avant par l'Inria. Il est également membre de la Fédération Charles Hermite, qui regroupe les quatre principaux laboratoires de recherche en mathématiques et STIC [4] de Lorraine.

Il est l'un des plus grands laboratoires de la région lorraine, et ses travaux scientifiques sont menés au sein de 27 équipes structurées en 5 départements, dont 16 sont communes avec l'Inria, représentant un total de plus de 500 personnes.

Ces équipes sont réparties dans les cinq thèmes de recherche suivants :
- Algorithmique, calcul, image et géométrie
- Méthodes formelles
- Réseaux, systèmes et services
- Traitement automatique des langues et des connaissances
- **Systèmes complexes et intelligence artificielle**

2. Laboratoire Lorrain de Recherche en Informatique et ses Applications.

3. Entité administrative créée par la signature d'un contrat d'association d'un ou de plusieurs laboratoires de recherche, d'un établissement d'enseignement supérieur ou d'un organisme de recherche. L'ESIAL fait par exemple partie de l'UMR de l'Institut Mines-Télécom.

4. Sciences et Technologies de l'Information et de la Communication.

1.3.3 MAIA

Pour ce stage, j'ai été accueilli au sein de l'équipe Maia (MAchines Intelligentes Autonomes). Dirigée par François Charpillet, c'est l'une des plus grosses équipes du Loria comprenant une trentaine de chercheurs.

Ses centres d'intérêts en recherche sont les comportements intelligents de prise de décisions.

Les sujets traités peuvent être scindés en deux axes.

Systèmes complexes et systèmes multi-agents

Cet axe étudie les comportements émergents dans des groupes d'entités réactive (agents), avec par exemple :

- la **bio-inspiration** (colonie de fourmis, comportement en essaim, etc.)
- l'étude d'émergence dans des phénomènes physiques
- les automates cédulaires

Prise de décisions séquentielles en milieu incertain

Les problèmes spécifiques considérés par l'équipe dans cet axe comprennent :

- la coordination, prise de décision pour systèmes multi-agents
- la planification dans de grands espaces d'états de processus de décision markovien (Markov Decision Processes, MDP)
- l'apprentissage par renforcement

Chapitre 2

Kilobots

2.1 Description

Les *Kilobots* sont des petits robots conçus par une équipe de l'Université de Harvard pour être un système de robots à faible coût pour l'application de comportements collectifs à grande échelle.

Dans la recherche sur la robotique actuelle, il y a un vaste travail autour d'algorithmes et de méthodes de contrôle de groupes de robots coopérants de manière décentralisée. Ces algorithmes gèrent généralement des centaines voire des milliers de robots. Cependant, pour des raisons de coût, de temps et/ou de complexité, ces algorithmes sont généralement validés uniquement en simulation, ou tout au plus dans un groupe d'une dizaine de robots. C'est pour répondre à ce problème que des chercheurs de l'Université de Harvard ont créé les *Kilobots*.

Le plan de fabrication des Kilobots, trouvable à l'adresse `http://ssr.wikidot.com/kilobot-documents`, est sous licence libre creative commons attribution-NonCommercial-ShareAlike 3.0 (CC BY-NC-SA 3.0), c'est-à-dire que n'importe qui peut, avec le savoir-faire, en assembler librement pour une utilisation non-commerciale.

2.2 Coût

Le coût d'un Kilobot est estimé à 14$, soit la somme du prix de chaque composant. Cependant la création de la carte électronique, le soudage et l'assemblage requièrent des compétences et des machines adéquates. La société suisse K-Team, qui produit des robots très utilisés en recherche, propose de fabriquer des Kilobots pour environ 150€ l'unité. Ce prix peut paraître rédhibitoire et le qualificatif "à faible coût" ne plus correspondre. Or ce prix reste

toutefois bien inférieur à d'autres robots comme les Kheperas, aussi utilisés par l'équipe Maya, qui coûtent environ 3 000€ l'unité soit l'équivalent de 20 Kilobots.

La contre partie de ce "faible" prix est dans les capacités individuelles des robots très limitées. La puissance réside alors dans le nombre de robots. C'est un modèle grandement inspiré de la biologie avec l'exemple le plus parlant des fourmis.

2.3 Capacités

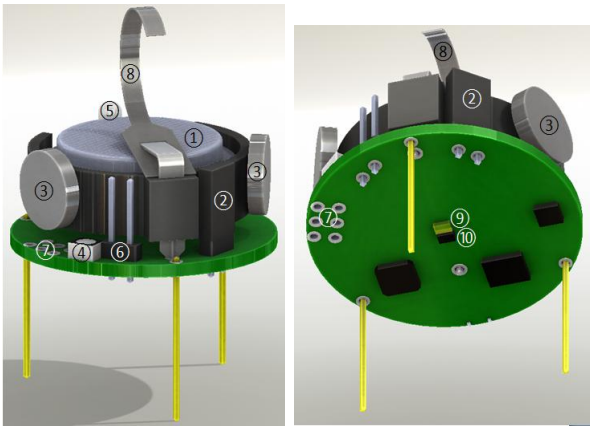

FIGURE 2.1 – Kilobot et ses différents composants.

Description des composants électroniques des kilobots, figure 2.1 :

1. Batterie 3.7V
2. Moteurs vibrants
3. Cavalier marche/arrêt
4. LED rouge/vert/bleu à trois intensités
5. Capteur de luminosité ambiante
6. Sortie serial pour le débugage
7. Socket JTag de programmation directe
8. Languette de recharge de la batterie avec les pattes
9. Emetteur infrarouge
10. Récepteur infrarouge

Les kilobots sont des petits robots circulaires de 33 mm de diamètre et 34 mm de hauteur. Ils sont composés d'une carte électronique, qui leur sert de chassie, où sont soudés les composants électriques. Au dessus se trouve une batterie Lithium-Ion. Ils se dressent sur trois pattes rigides et se déplacent à l'aide de deux moteurs vibrants. Ils disposent pour communiquer d'un émetteur/récepteur infrarouge.

2.3.1 Locomotion

Les kilobots se déplacent par la vibration de deux moteurs situés sur chaque coté. La vibration génère une force centripète qui est convertie en mouvement du fait que le centre de gravité du robot n'est pas au centre de celui-ci. Pour plus d'informations sur le déplacement des kilbots, consultez [6, 8].

Faire vibrer le moteur de droite à une intensité suffisante fait pivoter le robot dans le sens horaire, et le gauche dans le sens anti-horaire. Actionner les deux moteurs à la même intensité fait avancer le robot. Chaque moteur des kilobots doit être calibré afin qu'un programme demandant d'aller dans une direction ne soit pas différent d'un robot à l'autre. Les valeurs de calibration propres à chaque robot sont stockées dans une mémoire EEPROM [1].

Les déplacements ne peuvent s'effectuer uniquement sur des surfaces lisses.

Alors qu'un déplacement conventionnel à roue est plus précis, il est en revanche beaucoup plus couteux à mettre en place que le déplacement par vibration. Il n'y a donc pas d'odomètre [2] pour connaître leurs déplacements. Cependant les kilobots se déplacent approximativement à 1cm/sec et effectuent une rotation à 45°/sec.

Il faut alors utiliser la quantité de kilobots et la communication pour avoir un retour sur leurs déplacements et/ou positions. Ceci fait l'objet de la section 5 page 24.

2.3.2 Communication

La plupart des algorithmes de robots collectifs utilisent des communications locales, c'est-à-dire robot à robot. Pour communiquer avec leurs voisins

1. Electrically-Erasable Programmable Read-Only Memory ou mémoire morte effaçable électriquement et programmable.

2. L'odométrie est une technique permettant d'estimer la position d'un véhicule en mouvement. Cette mesure de bas niveau est présente sur quasiment tous les robots mobiles, grâce à des capteurs embarqués permettant de mesurer le déplacement du robot (de ses roues). (Wikipédia)

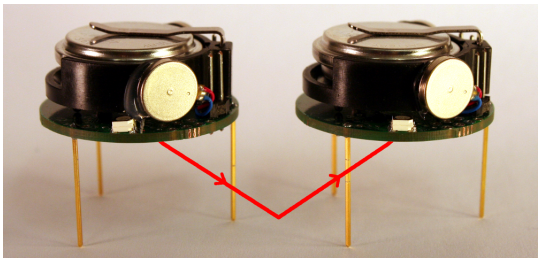

FIGURE 2.2 – Communication par infrarouge

les kilobots possèdent un émetteur et un récepteur infrarouge localisés sous la carte élécronique et dirigés vers le sol, comme le montrent les figures 2.2 et 2.1.

L'émetteur et le récepteur ont une émission et une réception de forme isotopique, ce qui signifie que les kilobots émettent et reçoivent des messages dans toutes les directions. La communication s'effectue avec un débit de 30 kb/s avec des robots éloignés de 10 cm, soit l'équivalant de 3 robots côte à côte.

Un kilobot communique avec d'autres kilobots en envoyant des messages d'une taille de 3 octets. Pour envoyer une information plus grande, il faut alors envoyer plusieurs messages.

Seulement tous les kilobots utilisent le même canal infrarouge pour communiquer. Il est donc possible et même fréquent que deux robots émettent au même moment. Pour palier à ce problème une méthode est utilisée, empruntée au wifi : CSMA/CA [3], détaillée plus loin en section 4.1.4 page 18.

Lors de la réception d'un message, le kilobot récepteur calcule la distance du kilobot émetteur en mesurant l'intensité du signal infrarouge reçu. Cette intensité décroit en fonction de la distance de l'émetteur. En pratique, l'intensité est aussi dépendante de la surface du sol (réfléchissant plus ou moins l'infrarouge) et de la luminosité ambiante. Un capteur de luminosité a donc été ajouté pour augmenter la précision du calcul de la distance menant à une justesse de ± 2mm et une précision sous 1mm.

2.3.3 Processeur

Le processeur pour les kilobots dessert deux fonctions. Premièrement, il effectue l'interface entre l'électronique bas niveau comme les moteurs, la communication, le circuit électrique et la LED. Secondement il exécute le

3. Standard Carrier Sense Multiple Access with Collision Avoidance

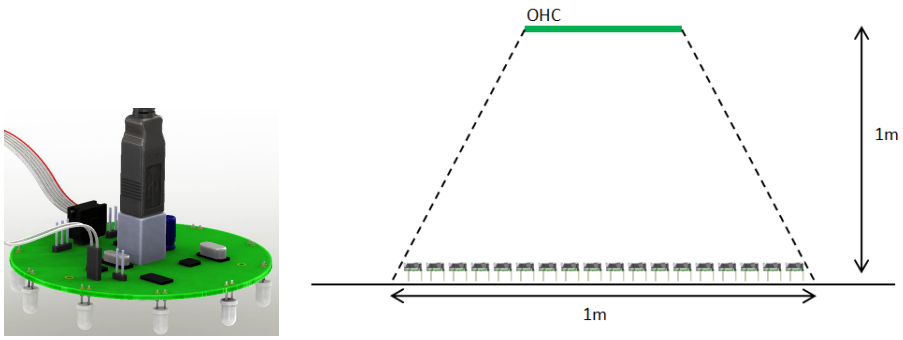

FIGURE 2.3 – Interface de commande OHC des kilobots

programme défini par l'utilisateur. Le processeur utilisé est un microprocesseur Atmega328 cadencé à 8MHz avec 32K de mémoire. Il dispose d'un mode de faible consommation permettant de mettre les kilobots en "sommeil". Il se programme en assembleur ou en C.

2.3.4 Alimentation

Les kilobots sont alimentés par une batterie 3,4 V à 160 mA lithium-ion. Cette batterie alimente un robot entre 3 et 10 heures suivant son activité et jusqu'à plusieurs semaines en mode "sommeil". La batterie se recharge via sa partie supérieure et via les pattes des kilobots. La mise en marche des kilobots s'effectue en mettant un cavalier qui fait office d'intérupteur on/off.

2.3.5 Mise à l'échelle

La force des kilobots résidant dans leur grand nombre, un accent a été mis sur la gestion et le contrôle des robots tous ensemble ou par grands groupes pour limiter les actions unitaires.

Les robots sont contrôlés par un dispositif de contrôle appelé Overhead controller (OHC) permettant de donner des ordres par infrarouge à tous les kilobots se trouvant sous sa zone d'émission, figure 2.3. Il est relié en usb à un ordinateur sous windows et commandé via un logiciel développé par les créateurs des kilobots à Harvard.

Ce logiciel, cf figure 2.4, permet de charger un programme utilisateur dans les kilobots (1 et 6), l'exécuter (4), l'arrêter (3), endormir le robot (2), le réveiller (7), le mettre en charge (9), visualiser son niveau de batterie (5) et le réinitialiser (8).

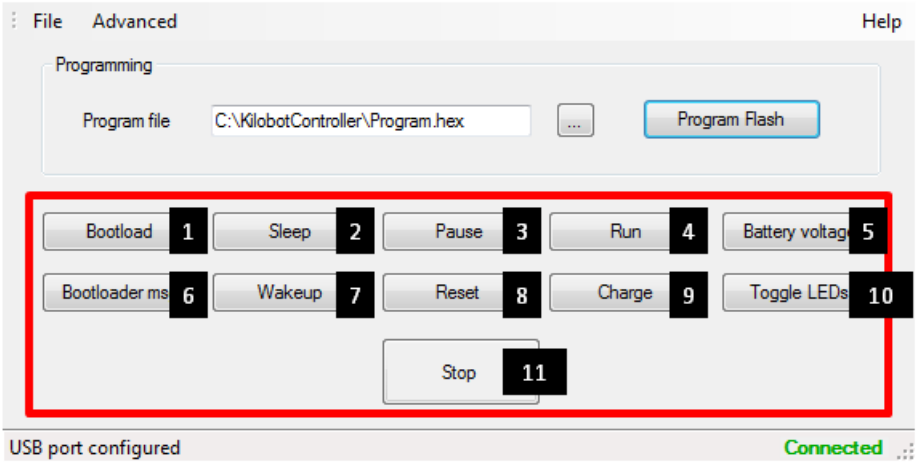

FIGURE 2.4 – Interface de commande des kilobots

2.3.6 Chaine de programmation

La programmation des kilobots s'effectue de la manière suivante : On écrit d'abord un programme en C en utilisant une bibliothèque de fonctions fournie, dédiée au contrôle des kilobots. Ensuite on cross-compile le programme sur ordinateur. Puis on le charge sur l' OHC. Et enfin on l'envoie sur les kilobots par infrarouge.

2.4 Réseau de capteurs mobiles

Les kilobots, de par leur communication, peuvent être considérés comme un réseau de capteurs mobiles. De nombreuses recherches ont été effectuées sur les réseaux de capteurs, souvent fixes. Les kilobots peuvent alors en bénéficier à quelques adaptations près pour intégrer la notion de mobilité des capteurs. La plupart des algorithmes présentés ici sont issus de travaux sur les réseaux de capteurs.

Chapitre 3

Flocking

3.1 Description

Le *flocking*[1] est un comportement de déplacement collectif par nuée, observé chez les oiseaux. Il est généralisé dans le monde biologique, notamment les bancs de poissons souvent utilisé en exemple, les bactéries, les essaims d'insectes, et tout les animaux au comportement grégaire. Plus généralement le flocking décrit comment les individus d'un groupe peuvent agir ensemble sans direction prévue.

Un groupe d'oiseaux manoeuvre gracieusement, avec chaque individu se déplaçant en parallèle dans la même direction. Lorsque la nuée change soudainement de direction, tous ses membres répondent rapidement à ce changement, avec cohésion, presque à l'unisson et aussi parfaitement que s'il faisait partie d'un unique organe.

Ces comportements suggèrent que la nuée possède des propriétés spéciales au niveau du groupe. Une des propriétés est le transfert rapide d'informations à travers le groupe qui permet d'exécuté par le groupe en entier une manoeuvre d'évitement de prédateur par exemple. L'hypothèse de la cohésion du groupe est basée sur des observations individuelles des oiseaux voisins et en réagissant à temps à ces observations. L'information se propage alors par vague dans le groupe à la manière d'une holà de spectateurs dans des gradins où chaque personne observe ses voisins pour savoir s'il doit ou non lever les bras.

Pour comprendre le comportement du flocking nous devons comprendre comment un individu, un oiseau par exemple, coordonne son comportement avec ceux des autres membres du groupe. On dit alors que le flocking est

1. D'autres termes sont aussi utilisés dans la littérature anglophone comme le swarming, boids, herding, shoaling ou schooling

l'émergence du comportement individuel des oiseaux et donc décentralisé.

L'émergence : L'émergence désigne l'apparition de nouvelles caractéristiques à un certain degré de complexité. On peut définir l'émergence par deux caractéristiques :

- L'ensemble fait plus que la somme de ses parties. Ceci signifie qu'on ne peut pas forcément prédire le comportement de l'ensemble par la seule analyse de ses parties.
- L'ensemble adopte un comportement caractérisable sur lequel la connaissance détaillée de ses parties ne renseigne pas complètement.

Un modèle du flocking, présenté dans le livre *Self-Organisation in Biological System* [2], s'est construit sur des observations biologiques et validé en simulation.

3.2 Définition

Le modèle de flocking présenté ici fait l'hypothèse que chaque individu peut observer chez ses voisins, à une distance limitée ou non, leur direction et la distance qui les sépare. Chaque individu du groupe est identique [2] et réagit de la même manière à la même situation, il n'y a pas de leader.

Selon Craig Reynolds, le flocking peux se définir comme l'émergence résultant de l'application de trois règles individuelles, qui sont :

- Se déplacer dans la même direction que ses voisins
- Rester suffisamment proche de ses voisins
- Eviter les collisions avec ses voisins

Plus concrètement, ces règles peuvent être implémentées de la manière suivante avec la figure 3.1 en support :

Alignement : on calcule la direction moyenne des voisins situés dans la zone d'alignement.

Répulsion : on calcule la direction pour s'éloigner des voisins les plus proches.

Attraction : on calcule la direction pour se rapprocher des voisins les plus éloignés.

La nouvelle direction que l'individu doit prendre est alors la moyenne entre les directions calculées issues des trois règles et de sa propre direction.

2. Il peut être intéressant d'introduire un ou plusieurs d'individus différents et d'observer l'évolution du comportement du groupe, mais ce n'est pas l'objet de ce stage.

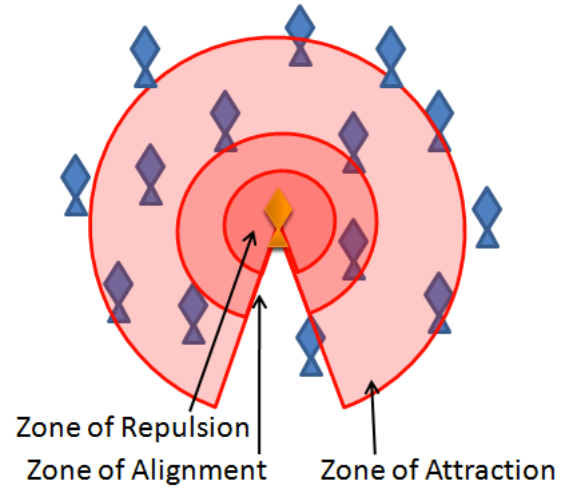

FIGURE 3.1 – Représentation des trois règles du flocking.

3.3 Application aux Kilobots

L'implémentation de ce modèle sur les kilobots pose un certain nombre de problèmes. En effet le flocking fait l'hypothèse que chaque individu, robot, peut observer immédiatement la direction dans laquelle ses voisins se dirigent par rapport à lui, ce qui n'est pas le cas nativement.

Il faut, avant de pouvoir faire du flocking avec les kilobots, obtenir la direction des kilobots voisins. Se dessine alors deux problèmes sous-jacents à l'obtention de la direction : la synchronisation et la localisation.

Si le problème de localisation des robots, détaillé dans le chapitre 5 page 24, vient du calcul simple de la direction par l'obtention de deux positions successives, celle de la synchronisation en est tout autre.

Obtenir deux positions successives implique une notion de temporalité en ces deux instants. Seulement les kilobots étant indépendants, ils se déplacent et localisent leurs voisins en même temps. Et même si ces deux tâches sont effectuées séparément et à la suite, les autres robots n'exécuteront pas forcement la même tâche au même moment. Cette désynchronisation complexifie grandement l'implémentation du flocking. C'est donc pour une raison de simplification que j'ai cherché un algorithme simple, décentralisé et robuste de synchronisation pour les kilobots dont nous allons détailler le fonctionnement dans le chapitre 4 ci-après.

Chapitre 4

Synchronisation inspirée des lucioles

4.1 Modèle historique

4.1.1 Définition

Mirollo et Strogatz [5] ont proposé en 1990 un modèle mathématique de synchronisation décentralisée s'inspirant de la manière dont les neurones ou certaines espèces de lucioles [1] se synchronisent spontanément. Ils ont prouvé qu'un comportement simple de nœuds réactifs va toujours converger vers une synchronisation et ce peu importe le nombre de nœuds et leurs dates de départ. Les nœuds forment un réseau dont la topologie est un graphe complet, i.e. chaque nœud est connecté à tous les autres (cf figure 4.1).

En 2004, Lucarelli and Wang [4] ont démontré que ce résultat était valable pour une topologie multi-hop (relaxation de la complétude du graphe, exemple figure 5.3 page 28). Cette contribution rend alors possible l'élaboration d'un algorithme décentralisé de synchronisation pour les réseaux de capteurs.

La synchronisation inspirée des lucioles apporte des fonctionnalités attractives pour les réseaux de capteurs : les nœuds effectuent des calculs et des interactions simples, ils ne gardent pas l'état des voisins ou la topologie du réseau. L'algorithme peut être considéré comme robuste dans le sens où il s'adapte aux changements de topologie, que ce soit à la perte ou à l'ajout d'un nœud dans le réseau [4].

1. Firefly dans la littérature anglo-saxonne.

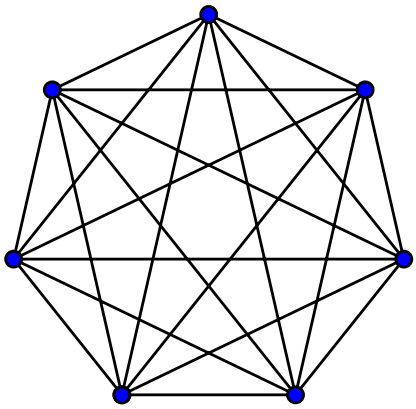

FIGURE 4.1 – Exemple de graphe complet, `http://commons.wikimedia.org/wiki/File:Complete_graph_K7.svg`

Selon Mirollo et Strogatz [5] chaque noeud (luciole, kilobot, etc) est un oscillateur couplé avec une période fixe T. Un noeud possède un timer interne (ou phase) t qui commence à 0, s'incrémente au cours du temps jusqu'à $t = T$. A ce point le noeud émet un "flash" et réinitialise son timer à 0.

Sans nœud dans le voisinage (à portée de communication), il flash simplement à $t = T$ (figure 4.2(a)). Lorsqu'un nœud observe un autre nœud flasher, il réagit en avançant son timer (sa phase) afin de se rapprocher de son prochain flash (figure 4.2(b)). La quantité d'ajustement est déterminée par une fonction $\Delta(t)$ en fonction de l'état de son timer interne. On se rapproche de son propre flash et ce plus rapidement quand on arrive vers la fin de notre période (Δ est continue et croissante).

4.1.2 Suppositions garantissant la convergence :

- Lorsqu'un nœud flash, ses voisins observent instantanément le flash.
- Les nœuds peuvent calculer $\Delta(t)$ instantanément.
- Tous les nœuds ont exactement la même période T.
- Chaque nœud observe tous les flashs de ses voisins (pas de perte).

4.1.3 Implémentation pour les robots :

Lors du flash d'un robot, celui-ci envoi un message indiquant qu'il flash. Or la synchronisation des robots par l'envoi de message est la pire situation dans la communication des robots implémentant une simplification de l'algorithme de communication CSMA/CA (Carrier sense multiple access

with collision avoidance), aussi utilisé pour les réseaux Wifi, afin d'éviter les congestions du canal de communication.

4.1.4 CSMA/CA :

Le robot voulant émettre écoute le réseau en envoyant un signal. Si le réseau est encombré, i.e. le signal perçu n'est pas le même que celui envoyé à cause d'une collision avec un signal d'un autre robot, la transmission est différée d'un délai aléatoire. Dans le cas contraire, si le canal est libre, alors le robot peut émettre. Tant qu'une collision est détectée lors de l'émission du message, il le réémet immédiatement.

4.2 Reachback Firefly Algorithm (RFA)

Pour palier à tous ces inconvénients, G. Warner-Allen et al [9] ont proposé quatre améliorations pour une implémentation réaliste.

4.2.1 Timestamping message

Timestamping message consiste à prendre en compte le temps de transmission des messages.

4.2.2 Pre-emptive message staggering

Cela consiste à échelonner les messages de flash. Lorsqu'un robots flash il envoie un message indiquant qu'il flash avec un retard aléatoire afin d'éviter la congestion du réseaux (vu en 4.1.4). Il indique dans son message le retard afin que les robots qui reçoivent le message puissent le prendre en compte et calculer le moment effectif du flash par rapport à leur horloge interne (phase).

En conséquence de ce retard aléatoire, des flashs peuvent être reçus chronologiquement désordonnés. Par exemple, dans le cadre théorique selon Mirollo et Strogatz [5], lorsqu'un noeud B observe un flash a à $t = 30$ il calcule instantanément le saut $\Delta(30) = 5$ et l'ajoute au timer $t = 30 + \Delta(30) = 35$. Dix pas de temps après ($t = 45$) , un flash, b, est observé. A nouveau le noeud calcule le saut $\Delta(45) = 10$ et effectue le saut $t = 45 + \Delta(45) = 55$, etc.

Seulement avec le *pre-emptive message staggering* le flash b peut arriver avant le flash a avec, par exemple, des délais aléatoires respectifs de $d_b = 2$ et $d_a = 15$. Le noeud B va observer le flash b à $t = 42$ et donc $t = (42 - d_b) + \Delta(42 - d_b) = 40 + \Delta(40) = 50$. Puis il va observer le flash a à $t = 53$ et appliquer le même calcul : $t = 53 - d_a + \Delta(53 - d_a) = 38 + \Delta(38) = 48$, etc.

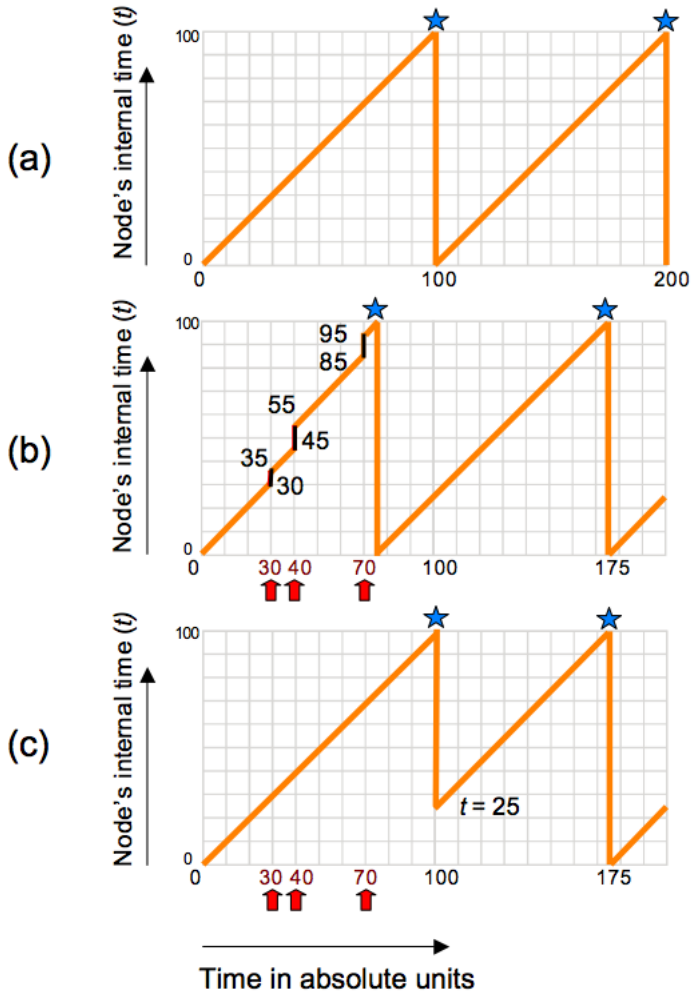

FIGURE 4.2 – Algorithme de synchronisation ($T = 100$). (a) Robot flash sans voisin. (b) Algorithme idéal théorique. (c) ReachBack Firefly Algorithm [9].

On remarque donc que la valeur du timer t est différente de ce qu'elle devrait être à cause du retard aléatoire dans l'émission des flashs. Pour pallier à ce désagrément RFA propose l'amélioration suivante.

4.2.3 Reachback response

Au lieu de réagir immédiatement à la réception d'un message (indication qu'un voisin flash), les messages sont stockés avec leur moment de réception (retranché du retard aléatoire) dans une liste triée sans pour autant appliquer de saut dans le timer de $\Delta(t)$. Un unique saut est effectué au moment du flash, somme des sauts que le timer aurait dû effectuer. Le robot se synchronise au cycle suivant, et non immédiatement à la réception, en réinitialisant son timer avec M flashs reçus au cours du cycle : $t = \sum_{i=0}^{M} \Delta(t_i)$ au lieu de $t = 0$, figure 4.2 (c).

4.2.4 Simplified Firing Function

$\Delta(t)$ doit être simple et rapide à calculer, une fonction linéaire type

$$\Delta(t) = \epsilon.t$$

Epsilon est le paramètre qui affecte le plus le comportement du système, à savoir la vitesse de synchronisation.

Comme $t < T$, l'incrément maximum que le nœud appliquera sur son horloge interne est donc de $\Delta(T) = \epsilon.T$. Si $\epsilon = \frac{1}{100}$ alors un nœud réagira au plus à $\frac{T}{100}$. Choisir epsilon plus grand signifie effectuer des sauts plus grands en réponse aux flashs, ce qui mène à une convergence plus rapide et donc à une synchronisation plus rapide. Cependant choisir epsilon trop grand entraine un sur-flash. En effet si en 3 flashs le nœud atteint sa période T, il va flasher en quasi-permanence et donc ne pas converger. Au contraire, un epsilon trop petit va garantir la convergence mais en augmenter l'échéance. Warner-Allen [9] prouve que le temps de convergence est proportionnel à $1/\epsilon$.

4.3 Résultats

4.3.1 Implémentation

J'ai implémenté l'algorithme RFA sur les kilobots en une boucle appelée toutes les millisecondes où je récupère les messages arrivés. Je les traite, i.e. les mémorise dans une pile triée par leur moment d'arrivé $t_i = t - retard_i$, puis

j'incrémente l'horloge interne t, si $t = T$ alors j'émet un signal lumineux[2], je réinitialise t avec les sauts à effectuer mémorisés dans la pile et enfin je tire aléatoirement un retard entre 0 et D pour l'émission de mon flash. On répète cette boucle à l'infini.

```
Var : t=0;        // Timer, horloge interne
      list_flash; // Liste triée de flash

TOUTE les millisecondes FAIRE
    Si nouveau message ALORS
        ajouter un flash à t_i = t - retard_i dans list_flash
    FIN SI
    incrémenter l'horloge interne $t$
    SI t à atteint la période T ALORS
        émettre un message avec un délai tiré aléatoirement entre 0 et D
        rénitialiser t en appliquant les sauts contenus dans list_flash
        vider list_flash
    FIN SI
FIN FAIRE
```

J'ai fixé une période de $T = 1000ms$, un retard aléatoire maximal de $D = 250ms$ et epsilon à $\epsilon = \frac{1}{50}$.

J'ai considéré que le temps d'émission/réception du message est négligeable par rapport à la période, donc nul.

On peut discuter mes choix des paramètres, surtout pour le retard maximal D qui pourrait être très inférieur à la valeur choisie, cependant cela n'affecte que très peu le résultat, voir [9].

4.3.2 Interprétation

Pour des soucis de temps évidents, je n'ai répété que 26 fois l'expérience pour chaque nombre de kilobots. De plus j'ai déterminé les temps en chronométrant manuellement et jugeant à l'œil nu de la synchronisation, en conséquence les valeurs trouvées sont approximatives. En effet, l'estimation de la synchronisation d'un grand nombre de kilobots est difficile à établir avec précision. La moyenne lissée par courbes de Bézier donne une tendance mais ne vaut évidemment pas celle qu'on obtiendrait avec un nombre plus important d'expériences.

2. Le signal lumineux sert uniquement de retour visuel pour l'utilisateur et n'est pas perçu par les autres kilobots.

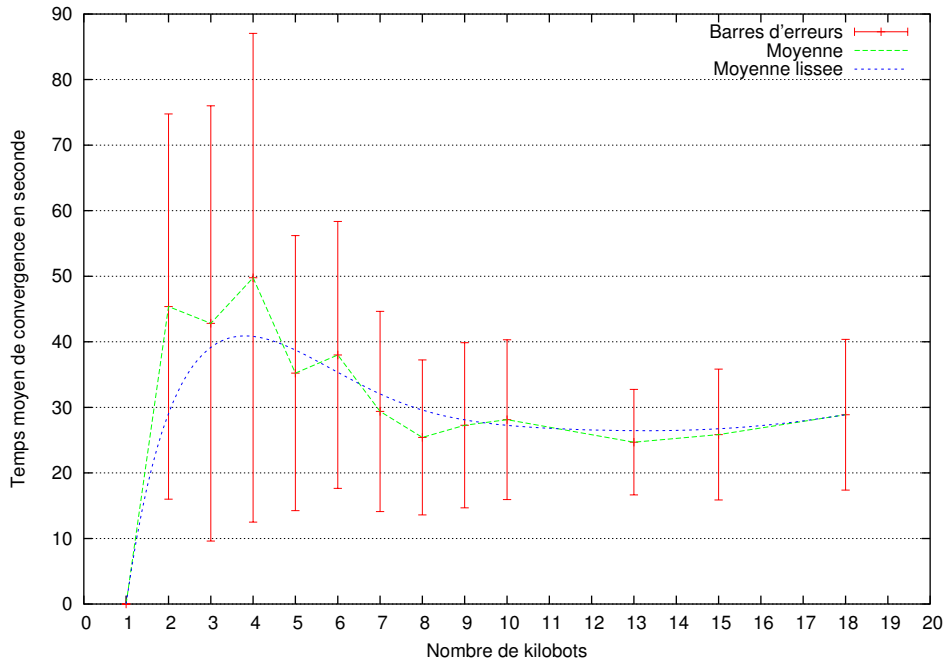

FIGURE 4.3 – Temps de synchronisation moyen en fonction du nombre de robots (26 expérimentations par nombre de robots)

Ces résultats ne sont donc pas à prendre tel quel, mais indiquent seulement une tendance. Pour améliorer la précision des expériences il faudrait avoir recours à la vidéo et à l'analyse d'image.

On remarque que le temps moyen de convergence diminue avec le nombre de kilobots. Il semble se stabiliser autour de 28 secondes. Il est intéressant de noter que l'écart type, représenté par les barres d'erreurs, diminue également avec le nombre de kilobot. On peut noter qu'à partir de sept robots l'algorithme RFA donne des temps de synchronisation stables et constants.

Diminuer le temps d'une période aura pour effet d'accélérer le temps de synchronisation mais saturera plus facilement le canal de communication des kilobots. En effet si la synchronisation n'est pas l'unique tâche, il faut laisser de la bande passante du canal libre pour d'autres tâches ayant besoin de communication.

Ces résultats de montrent pas la robustesse de cet algorithme au changement de topologie. On peut cependant faire l'hypothèse qu'un changement de topologie mineur est au moins aussi rapide, si ce n'est plus, que la resynchronisation à partir de zéro. En effet la faible vitesse de déplacement des kilobots entraîne un faible changement de topologie. De plus la localisation des kil-

boots voisins ne change en rien la synchronisation, seule la communication importe.

L'algorithme RFA est un algorithme de synchronisation décentralisé, robuste et perpétuel. Ses qualités en font l'algorithme de synchronisation de choix pour les kilobots.

Chapitre 5

Localisation

Le but de la localisation est de déterminer les coordonnées physiques d'un groupe de nœuds.

Dans un réseau de capteurs, il existe plusieurs formes de localisation. Il y a tout d'abord la localisation dans un repère global absolu commun et partagé par tous les noeuds du réseau, comme le système GPS [1]. Il y a ensuite la localisation relative, où chaque nœud localise ses voisins, voire l'ensemble du réseau, dans son propre repère. Ou enfin une localisation hybride qui dans un premier temps repère les voisins les uns par rapport aux autres, puis convient d'un repère commun global.

Bachrach et Taylor [1] passent en revue differents types d'algorithmes de localisation. La décentralisation étant au coeur même des réseaux de capteurs, nous allons nous concentrer uniquement sur ceux-ci et laisser de côté les algorithmes centralisés tel que : "SemiDefinite Programming" [1, 3] ou "MDS-MAP" [1, 7].

5.1 Localisation globale

La localisation globale absolue peut être définie comme une localisation relative à des noeuds dont on connait la position absolue dans un repère global, on les appellera alors des *balises* [2].

Il faut au moins 3 balises non-colinéaires pour une localisation en 2 dimensions, 4 balises non-coplanaires pour 3 dimensions (cf. [1]).

Le placement de ces balises peut avoir un impact important sur la précision de la localisation des autres noeuds du réseau. La plupart des algorithmes

1. Global Positioning System
2. Beacon ou anchor dans la litérature anglaise

sont plus efficaces si les balises sont placées selon une forme convexe autour du réseau.

Les avantages de l'utilisation de balises sont triviaux. La présence de nœuds pré-localisés simplifie grandement la tâche de localisation des autres nœuds. Cependant ils ajoutent un coût car chaque balise est un robot inutilisable pour d'autres tâches, il faut donc faire attention au nombre de nœuds balises par rapport au nombre de nœuds utiles.

5.2 Problèmes inhérents à la conception d'algorithmes de localisation

Les réseaux de capteurs, comme un réseau formé par les kilobots ont des ressources en calcul et en mémoires limitées. Les limites de portée des communications jouent aussi sur les limites du nombre de connexions possibles dans le réseau i.e dépend de la densité du réseau. La forme (non-convexe) du réseau accroit la difficulté de nombreux algorithmes, ainsi les nœuds se trouvant en bord de réseau sont souvent moins bien localisés. Les irrégularités de l'environnement et les obstacles sont aussi un frein à l'efficacité des algorithmes.

5.3 Algorithmes distribués avec balises

Cette catégorie d'algorithmes de localisation permet d'extrapoler la position de nœuds d'après la position des balises. La localisation est alors globale pour tout le réseau relativement aux balises. Nous allons présenter trois de ces algorithmes : "diffusion", "bounding box" et "gradient multilateration".

5.3.1 Diffusion

L'algorithme de diffusion provient d'une idée simple : la plupart des positions des nœuds est le centre de ses voisins.

```
Initialiser la position de tout les nœuds non-balise à (0,0)
Tant que la position n'a pas convergé (ou répéter à l'infini)
    Calculer la position du nœud au centre de celle de ses voisins.
```

Cet algorithme est peu efficace lorsque la densité de balises est faible, voir comparatif figure 5.5.

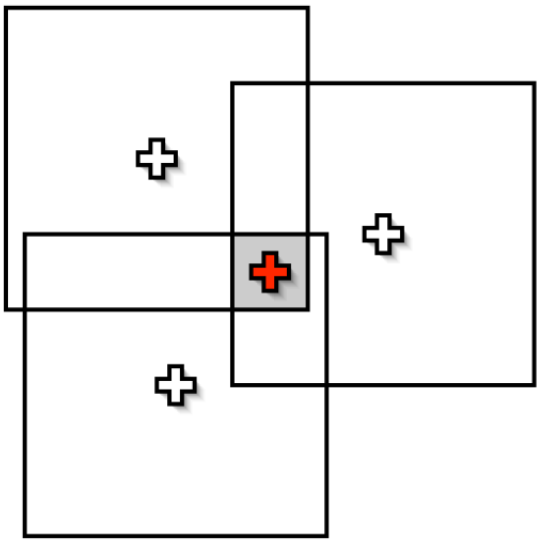

FIGURE 5.1 – Exemple d'intersection de bounding box. Le centre de l'intersection est utilisé comme position estimée.

5.3.2 Bounding box

L'algorithme bounding box est une méthode simple de localisation dont l'idée est d'approcher la position en affinant une boîte englobante. C'est une simplification de la multilatération (voir Annexe A) dont la zone d'intersection est un rectangle. La bounding box est l'intersection des rectangles centrés (x_b, y_b) sur une balise b et de taille $2d_b$, où d_b est la distance euclidienne entre la balise et le nœud à localiser.

Le calcul de la bounding box se fait de la manière suivante pour n balises :

$$\begin{cases} \max(x_i - d_i) & \leqslant & x & \leqslant & \min(x_i + d_i) \\ \max(y_i - d_i) & \leqslant & y & \leqslant & \min(y_i + d_i) \end{cases} i = 1...n \tag{5.1}$$

On utilise le centre de cette zone comme position estimée pour le nœud à localiser (voir figure 5.1). Plus la surface d'intersection est petite plus on augmente la précision de l'estimation de la position.

5.3.3 Multilateration par descente de gradients

La *multilatération* est une généralisation de la trilitération à plus de 3 distances pour une meilleure approximation de la localisation. C'est un problème s'approchant de la triangulation, mais qui utilise des distances à la place des angles, cf. figure 5.2.

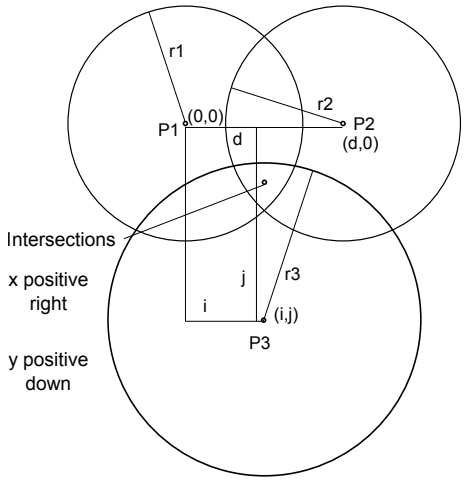

FIGURE 5.2 – Exemple de trilatération. Source : `http://commons.wikimedia.org/wiki/File:3spheres.svg`

Je ne pas implémenter le calcule de la multilatération, car elle demande énormément de ressources en calculs et en mémoires. Pour cause, il faudrait embarquer sur les kilobots un solveur de moindres-carrés. Une explication mathématique de la multilatération est toutefois donnée en annexe A. Je présente tout de même le principe de la descente de gradient que j'utiliserai par la suite.

Formellement, étant donné m balises dont on connait les positions cartésiennes absolues $b_i, i = 1...m$, la multilatération permet de trouver la position d'un nœud à partir de leur distance.

L'évaluation de la distance entre les nœuds est donc une composante essentielle de la multilatération et de la localisation plus généralement.

Dans les réseaux sans fils comme celui formé par les kilobots, la question est comment la communication peut aider à la localisation ? Deux techniques sont utilisées pour y répondre, le calcul de la distance entre 2 nœuds voisins et la propagation du gradient de l'information dans le réseau.

Le calcul de la distance entre 2 kilobots est détaillé à la section 5.3.3.

S'agissant de la propagation du gradient, il faut faire appel à la théorie des graphes. Chaque nœud propage soit le nombre de sauts dans le graphe

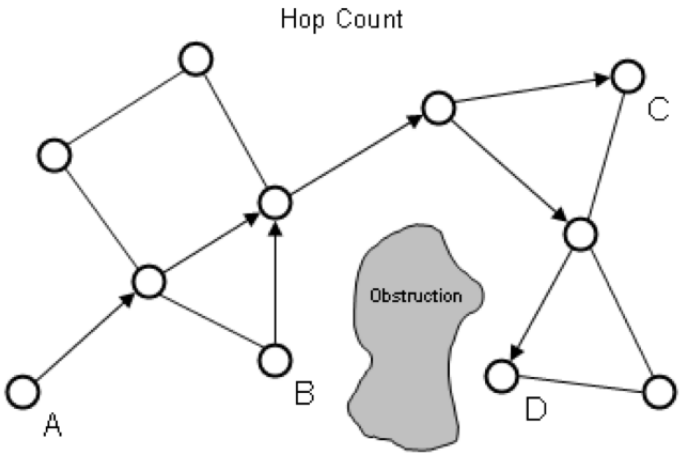

FIGURE 5.3 – Exemple de réseau de capteur [1]

pour relier 2 nœuds soit directement la somme des distances pour atteindre le nœud.

Chaque balise propage son gradient à travers le réseau, ce qui est l'équivalent distribué du calcul du plus court chemin entre toutes les balises et tous les nœuds à localiser. La propagation du gradient s'effectue selon l'algorithme suivant :

Pour chaque nœud j et balise k, initialiser d_{jk} (la distance entre j et j) à 0 si $j = k$ et ∞ sinon.
Sur chaque nœud j répéter
Pour chaque balise k et voisin i,
Récupérer la distance d_{ik} de i puis mettre à jour la distance : $d_{jk} = min(d_{ik} + \hat{r}_{ij}, d_{jk}$ où $\hat{r}_{ij}$ est la distance estimée entre les nœuds i et j.

Après un certain temps, chaque valeur d_{jk} va être la longueur du plus court chemin entre le nœud j et la balises k.

La figure 5.3, montre bien les limites de cette technique qui n'est qu'une approximation de la distance et dont l'efficacité est proportionnelle avec la densité du réseau. La distance calculée entre les nœuds A et C est proche de la distance euclidienne. En revanche la distance calculée entre les nœuds B et C est largement supérieure à la distance euclidienne. Cette distance calculée est donc une majoration de la distance euclidienne aux aléas près de la certitude des distances inter-nœuds.

5.4 Amélioration proposée

L'algorithme bounding box demande peu de ressources en calcul et donne des positions d'une précision tout à fait acceptables. Cependant il émet l'hypothèse que chaque nœud à localiser connait sa distance exacte par rapport aux différentes balises. Dans le cadre des kilobots il faudrait quadriller l'environnement de kilobots balises du à la faible portée de communication des robots. La densité de balises par rapport à celle de robots utiles est très élevée, ce qui en fait une méthode non viable pour les kilobots.

En revanche, l'algorithme de multilatération par descente de gradient demande beaucoup plus de puissance de calcul, à cause de l'implémentation d'un solveur de moindre carré expliqué en annexe A. Cependant il n'a besoin que de 3 balises minimum et la connectivité directe des balises par rapport aux nœuds à localiser n'est pas obligatoire. La distance estimée est majorée par propagation d'un gradient.

L'idée est alors simple, du *gradient bounding box* en utilisant l'algorithme bounding box pour sa simplicité en remplaçant la distance euclidienne par une estimation propagée par gradient dans le réseau. Les bounding box calculées englobent alors les bounding box à distance euclidienne. En conséquence de quoi, nous perdrons en précision troquée pour une plus grande flexibilité.

5.5 Comparatifs

J'ai sélectionné cinq topologies pour comparer les algorithmes présentés ci-dessus.

La figure 5.5 se présente sous la forme d'un tableau, où chaque ligne correspond à une topologie différentes. La première colonne est un aperçu des positions réelles des nœuds. Les autres colonnes correspondent à la localisation calculée en fonction des algorithmes.

L'erreur d'un nœuds est obtenue en calculant la distance euclidienne (en millimètre) entre la position réelle et celle calculée par l'algorithme. Les valeurs présentées sur la figure 5.4 sont les moyennes des erreurs de nœuds pour chaque topologie et chaque algorithme.

5.5.1 Choix des topologies

Topologie n°1 : Cas le de trilitération le plus simple, un nœuds et trois balises à portées.

Topologie n°2 : Trois nœuds reliés entre eux et chacun relié à une balises différentes, sans symétrie.

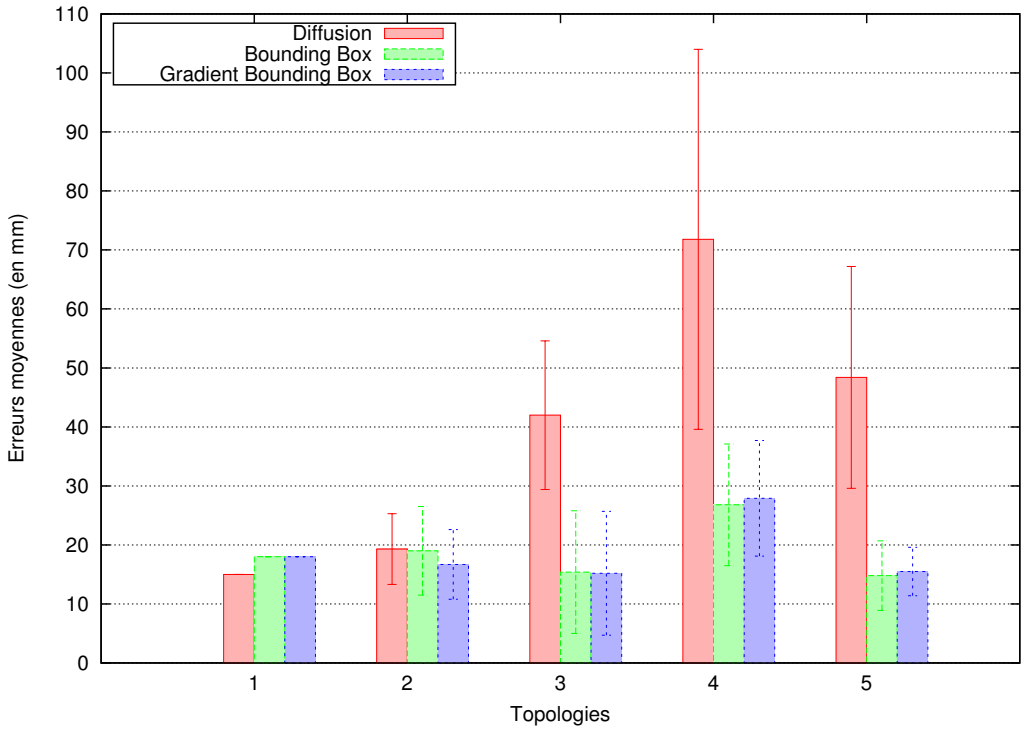

FIGURE 5.4 – Comparatif de l'erreur moyenne des algorithmes de localisation sur différentes topologies.

Topologie n°3 : Neuf nœuds répartis de façon homogène dans la forme convexe formée par quatre balises.

Topologie n°4 : Douze nœuds répartis de façon homogène dans la forme convexe formée par trois balises.

Topologie n°5 : Ajout de trois balise autour des nœuds de la Topologie n°4

5.5.2 Analyse

Sur les topologie 1 et 2, les trois algorithmes se comportent de la même manière avec une faible erreur.

Alors que sur les topologie 3, 4 et 5 l'algorithme de diffusion voit sont erreur augmenter fortement, les deux algorithmes à base de bounding box restent à une erreur moyenne plutôt faible et même meilleur que la topologie 1 le plus simple. On note que l'écart type inférieur de l'algorithme de diffusion est à chaque fois supérieur à l'écart type supérieur des algorithmes bounding box.

L'ajout de balises entre la topologie 4 et 5 améliore entre 30 et 50% la localisation. La densité de balises joue bien un rôle dans la précision de la localisation. Il faut trouver le compromis entre le nombre de balises et le nombre de nœuds dit "utiles".

L'algorithme à base de gradient bounding box est tout aussi efficace que le bounding box. Il est donc très intéressant car le bounding box suppose une liaison directe entre les nœuds à localiser et les balises au contraire de la méthode par descente de gradient qui est plus fidèle au fonctionnement des kilobots.

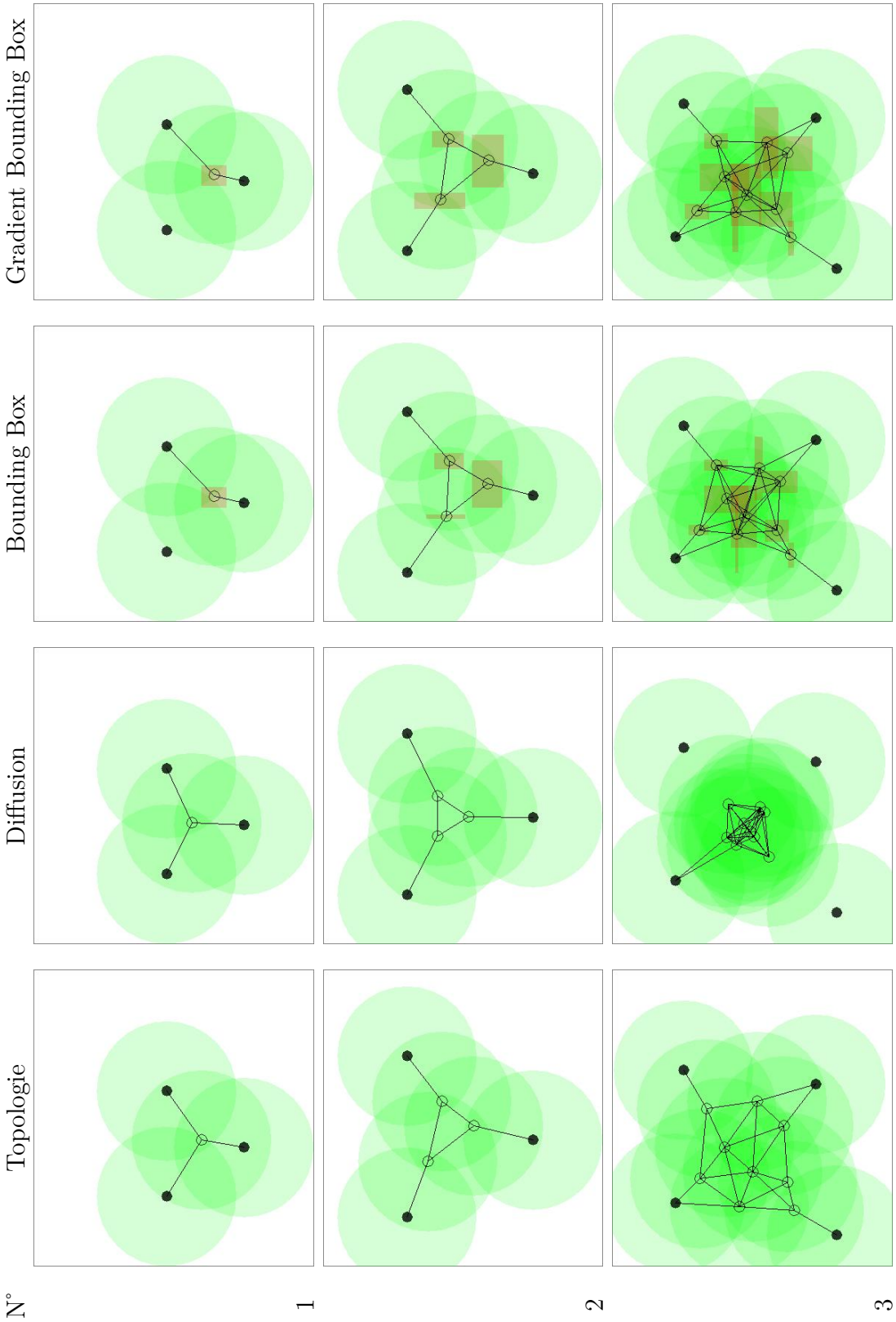
Gradient Bounding Box
Bounding Box
Diffusion
Topologie
N°
1
2
3

FIGURE 5.5 – Comparatif des positions sur 5 topologies des 3 algorithmes de localisation avec balises. Les points noirs représentent les balises, les cercles des kilobots, le vert le champ de vision et en rouge les bounding box

Chapitre 6

Bilan

6.1 Kilobot une plateforme pour le flocking ?

Les Kilobots sont une plateforme à faible coût dont le but est l'application d'algorithmes collectifs. En conséquence les robots ont des capacités très limitées : déplacements imprécis, pas d'informations de direction/localisation des autres kilobots. La gestion d'une grande quantité de kilobots est facilitée par leurs concepteurs et permet de se concerter sur la conception et l'implémentation de ces algorithmes.

L'incertitude des déplacements par vibration est un frein à l'adoption des kilobots mais avec un nombre plus important de robots cette limite peut être outrepassée par la localisation (en corrigeant le déplacement avec le retour des autres robots).

Il faut encore du travail pour implémenter du flocking sur les kilbots mais les sous tâches que sont la synchronisation et la localisation sont en partie résolues.

6.2 Questions non-abordées

Il y a beaucoup d'autres algorithmes inspirés des animaux sociaux qui pourraient être implémentés sur les kilobots, le livre Self-organisation in Biological Sysem [2] fait un tour d'horizon des ces algorithmes avec par exemple :

- La formation de structure des bactéries
- Le foraging, inspiré des fourmis, consiste à trouver des sources de nourriture, ramener le contenu jusqu'au nid (la fourrière).
- La formation de sentiers par les fourmis
- La sélection de sources de nectar par les abeilles.
- La création de structures inspirées des termites.

– etc.

Les kilobots peuvent servir aussi de plateforme d'implémentation d'algorithme génétique, dont les principes sont tirés des théories de l'évolution.

On peut même s'imaginer implémenter des machines de Breitenberg.

6.3 Apports du stage

Outre l'application de savoirs abordés durant ma formation, j'ai beaucoup appris durant ce stage notamment sur le processus de recherche.

J'y ai fait des rencontres enrichissantes, entre autre en assistant à des conférences. J'ai pu y observer les thèmes de recherche actuels et futurs, dans l'intelligence artificielle et la robotique. Ces deux sujet me passionnent et cette expérience me conforte dans l'idée de continuer sur cette voie par le biais d'une thèse.

Annexe A

Calcul de la multilatération

La multilatération est une technique simple mais les mathématiques spécifiques à son implémentation varient largement, tout comme son application aux réseaux de capteurs.

Bacharch [1] explique que la multilatération est obtenue en minimisant l'erreur quadratique entre les distances observées aux balises r_i et la distance prédite calculée $\|s - b_i\|$:

$$\begin{aligned} s &= \underset{s}{\operatorname{argmin}}\ E(s) \\ E(s) &= \sum_{i=1}^{m} (\|s - b_i\| - r_i)^2 \end{aligned} \tag{A.1}$$

Le problème de minimisation peut être résolu en utilisant une méthode des moindres-carrés de Newton-Raphson. Il faut d'abord approximiser la fonction d'erreur $e(s, b_i) = \|s - b_i\| - r_i$ de l'équation (A.1) avec le premier ordre de la série de Taylor de s_o :

$$\begin{aligned} e(s, b_i) &\approx e(s, b_i) + \nabla e(s, b_i)(s - s_0) \\ &= \nabla e(s_0, b_i)s - (-e(s_0, b_i) + \nabla e(s_0, b_i)s_0) \\ \nabla e(s, b_i) &= \frac{s - b_i}{\|s - b_i\|} \end{aligned}$$

En utilisant cette approximation dans l'équation (A.1) :

$$s \approx \underset{s}{\operatorname{argmin}} \sum_{i=1}^{m} (\nabla e(s_0, b_i)s - (-e(s_0, b_i) + \nabla e(s_0, b_i)s_0))^2$$

Mis sous forme vectorielle :

$$s \approx \underset{s}{\operatorname{argmin}} \|As - b\|^2 \tag{A.2}$$

$$A = \begin{bmatrix} \nabla e(s_0, b_1) \\ \nabla e(s_0, b_2) \\ \vdots \\ \nabla e(s_0, b_m) \end{bmatrix} \tag{A.3}$$

$$b = \begin{bmatrix} -e(s_0, b_1) + \nabla e(s_0, b_1) s_0 \\ -e(s_0, b_2) + \nabla e(s_0, b_2) s_0 \\ \vdots \\ -e(s_0, b_m) + \nabla e(s_0, b_m) s_0 \end{bmatrix} \tag{A.4}$$

La partie droite de l'équation (A.2) est sous une forme qui permet d'être résolue par un solveur de moindres carrés itératifs. La position s résultant est une bonne estimation de la position du nœuds à localiser à partir de b_i et r_i.

Pour résumer la méthode de multilatération :

Etape 1 : Choisir s_0 pour être le point d'entrée de l'optimisation. Le choix est arbitraire, le centre de gravité $\bar{b}$ est un bon choix :

$$\bar{b} = \frac{1}{m} \sum_{i=1}^{m} b_i$$

Etape 2 : Calculer A et b avec s_0 et les équations (A.3) et (A.4).

Etape 3 : Calculer $s'_0 = \underset{x}{\operatorname{argmin}} \|Ax - b\|^2$ avec un solveur de moindres carrés.

Etape 4 : Si $E(s_0) - E(s'_0) < \epsilon$, alors s'_0 est la solution, sinon mettre $s_0 = s'_0$ et réitérer à l'Etape 2.

Il y a d'autres manières de résoudre le problème de la multilatération. Celle présentée ici est équivalente à une descente sur la fonction d'erreur E pour Newton-Raphson (équation (A.1). La plupart des méthodes alternatives tentent également de minimiser l'erreur quadratique avec des formes d'optimisations itératives.

Table des figures

Bibliographie

[1] J. Bachrach and C. Taylor. 1 localization in sensor networks.

[2] S. Camazine, J.-L. Deneubourg, and N. R. F. . [et al.]. *Self-organization in biological systems*. Princeton studies in complexity. Princeton, N.J. Princeton University Press, 2003.

[3] L. Doherty, K. pister, and L. El Ghaoui. Convex position estimation in wireless sensor networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1655 –1663 vol.3, 2001.

[4] D. Lucarelli and I. jeng Wang. Decentralized synchronization protocols with nearest neighbor communication. In *In SenSys '04 : Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 62–68. ACM, 2004.

[5] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6) :pp. 1645–1662, 1990.

[6] M. Rubenstein, N. Hoff, and R. Nagpal. Kilobot : A low cost scalable robot system for collective behaviors. 06 2011.

[7] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from mere connectivity. In *MobiHoc'03*, pages 201–212, 2003.

[8] P. Vartholomeos and E. Papadopoulos. Analysis, design and control of a planar micro-robot driven by two centripetal-force actuators. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 649 –654, may 2006.

[9] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 142–153, New York, NY, USA, 2005. ACM.

Résumé

Les systèmes multi-agents sont largement inspirés du monde biologique, des animaux sociaux comme par exemple le déplacement des poissons en bancs, la recherche de nourriture par des fourmis trouvant le chemin le plus court, etc.. Le principe est que des agents, ici des robots, tirent un bénéfice de leurs interactions, coopérations et/ou compétitions.

Nous nous intéresserons plus particulièrement au flocking et son implémentation sur des robots. Le flocking est le comportement social du déplacement à la manière des nuées d'oiseaux ou des bancs de poissons.

J'ai pris en main une plateforme de petits robots aux capacités réduites nommés *Kilobots*. J'ai effectué une étude de faisabilité du flocking sur ces kilobots.

Je présente le principe du flocking et ses obstacles pour les kilobots. J'étudie une méthode de synchronisation inspirée des lucioles et passe en revue trois méthodes de localisation dont je propose une amélioration.

En conclusion, la plateforme des kilobots bien que très limitée en capacités est viable pour faire du flocking. Cependant il faut d'abord développer un arsenal de méthodes dont un début est présenté ici.