3 janvier 2014

Table des matières

Abstract

The goal of this project was to design a swarm-intelligent behaviour for virtual robots using the ARGoS software. These robots had to explore an unknown environment with obstacles in order to ultimately loop between spots marked on the ground and a starting area. The fundamental paradigm was that a single set of rules would be followed independently by every robot, which would allow a swarm-intelligent behaviour to emerge through the robot interactions prescribed by these rules. For that to work, exploration, shortest path-finding and obstacle-avoidance algorithms were needed, along with elementary automated decision making, communication and odometry. These concepts were implemented using the ARGoS loop approach, which means that the same sequence of actions takes place at every step, while only events occurring during that step can influence these actions. However, a Dijkstra path-finding algorithm, which would only execute once before every trip while always providing the shortest trajectory, was also considered. A first working solution was produced using the Lua language, then put to the test and could quickly be enhanced accordingly thanks to the flexible framework. This allowed robots to loop between a starting area and spots of known position while avoiding collisions, and information was gathered on parameters meaningful for the experiment. Future development should be focused on optimizing these parameters and enabling robots to explore a fully unknown environment.

Résumé

Le but de ce projet était de créer une intelligence en essaim pour des robots modélisés dans le simulateur ARGoS. Ceux-ci devaient explorer un environnement inconnu qui comportait des obstacles afin d'ensuite faire des allers-retours entre des zones marquées aux sols et leur nid de départ. Le principe de base était que le même ensemble de règles devrait être suivi de manière indépendante par chaque robot ; la caractéristique intelligente de l'ensemble de robots devant émerger à travers les interactions entre robots prescrites par ces règles. Pour cela, des procédures d'exploration, de recherche du plus court chemin et d'évitement furent nécessaires ainsi que des principes de base de prise de décision, d'odométrie et de communication. Ces concepts furent mis en pratique en utilisant l'approche loop d'ARGoS, qui implique que le robot exécute la même séquence d'opérations à chaque pas. Cependant, une recherche du plus court chemin Dijkstra, qui ne serait faite qu'une seule fois avant chaque trajet d'un robot, tout en trouvant toujours le chemin le plus court, fut aussi considéré. Une première solution fonctionnelle en Lua fut construite, testée et ensuite améliorée en conséquence grâce au turnaround loop très court offert par ARGoS. Cette solution permet aujourd'hui au robot de faire des allers-retours entre leur nid et une ou plusieurs ressources dont ils connaissent la position à l'avance, tout en évitant les collisions dans un environnement inconnu. De plus, des informations ont été collectées sur des paramètres influençant l'expérience. La deuxième partie du quadrimestre devrait être consacrée à l'optimisation de cette solution et à permettre au robot d'explorer un environnement afin de trouver la position des sources à exploiter.

Chapitre 1

Introduction

1.1 Description générale

Le but du projet est d'établir un comportement en essaim à des robots afin qu'ils puissent explorer un environnement non connu à l'avance, et naviguer entre un nid et une ressource, qu'ils exploiteront. Le nid sera représenté par une zone peinte au sol et les ressources par une lumière dans l'arène. Le nid sera la zone où les robots pourront recharger leur batterie. Effectivement, le fait que les robots possèdent une batterie, limitée à un aller-retour du centre à l'extrémité de l'arène, devra également être pris en compte. Des obstacles seront également présents dans l'arène. Les robots devront donc également être capable d'éviter tout type d'obstacles. ARGoS, un simulateur développé par le laboratoire IRIDIA, sera utilisé afin de simuler le comportement des robots dans l'arène.

1.2 Intérêt du projet

L'intérêt principal du projet est la compréhension du fonctionnement d'un comportement en essaim. La conception d'un comportement en essaim performant permettrait de remplacer les êtres humains par des robots dans des tâches dangereuses et où l'utilisation de ceux-ci serait trop coûteuse. Par exemple, l'exploration de la surface terrestre de Mars peut se faire à l'aide d'un essaim de robots ce qui enlèverait les problématiques de l'envoie et du rapatriement des astronautes à la fin de leur mission.

1.3 Résultat attendus

L'objectif premier est de trouver un comportement qui permet aux robots de survivre et d'exploiter une ressource de manière autonome dans un environnement non connu à l'avance.

Dans un premier temps, les robots seront considérés comme omniscient et connaîtront l'environnement à explorer. Dans un second temps, la comportement en essaim devra prendre en compte la non connaissance de l'environnement à explorer et inclura, donc, une mémoire permettant de stocker les données qui s'accumuleront. Une communication entre les robots pourra être envisagée par la suite. Même si la

tâche à accomplir est au niveau de l'essaim, ce dernier ne sera jamais programmé directement. Il devra être fait au travers du comportement individuel des robots. Des interactions locales entre robots, physiques ou non, émergera un comportement global qui devra être étudié et être rendu prévisible.

Des outils de mesure, afin de calculer la qualité du comportement en essaim, devront être élaborés. Ils devront établir la performance des solutions proposées en fonction des objectifs initiaux.

Pour résoudre la problématique donnée, le projet a été décomposé en trois grandes parties : l'intelligence artificielle, la trajectoire des robots et la prise de décision.

Chapitre 2

L'intelligence artificielle

2.1 Général

Afin de mener à bien l'objectif énoncé dans l'introduction, les robots doivent se comporter de manière intelligente. Le plus important est l'interaction avec l'environnement. En effet, le choix de la bonne action est cruciale. Il est cependant délicat de définir qu'une seule bonne action. Ceci est développé plus bas, en introduisant un «nouveau concept» qui est la rationalité. De manière générale, un robot peut être assimilé à un agent qui reçoit des «percepts» par l'intermédiaire de ses capteurs. L'agent réagit alors en exerçant une action sur l'environnement grâce à ses effecteurs. La description des différents capteurs et effecteurs du robot est faite

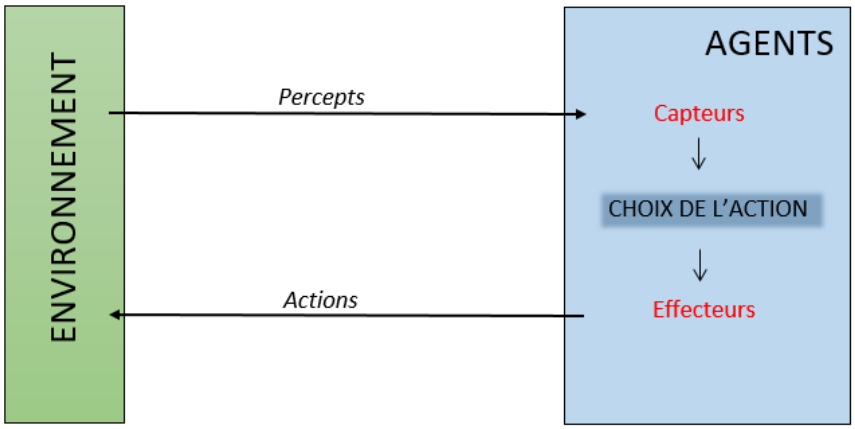

FIGURE 2.1 – Cycle d'interaction entre l'environnement et les agents

ci-dessous dans le chapitre 3.

Comme énoncé précédemment la rationalité d'un agent peut être délicate à mesurer. D'après la traduction française du livre «Artificial Intelligence : A Modern Approach » [1] : «La rationalité n'est pas synonyme de perfection, la rationalité maximise la performance espérée tandis que la perfection maximise la performance

réelle».

En effet, dans un groupe composé de plusieurs agents, un choix d'action peut s'avérer bénéfique pour un agent mais mauvais pour l'ensemble du groupe. C'est pourquoi il est préférable de concevoir les mesures de performance en fonction de ce que l'on souhaite obtenir dans l'environnement et non en fonction de la façon dont devrait se comporter un agent.

A partir de cela, tout problème peut être formellement défini par cinq composantes. Tout d'abord l'état initial dans lequel commence l'agent, puis la description de ses différentes actions, c'est à dire toutes les actions possibles dans un état donné. Ensuite, vient le modèle de transition, il décrit ce que chaque action réalise. Ces trois premières composantes définissent l'espace des états du système, c'est à dire l'ensemble de tous les états accessibles par une séquence d'action à partir de l'état initial.

Dès lors l'espace d'état peut être interprété sous forme d'arbre où les nœuds représentent des états et les branches des séquences d'action. Il en découle la notion de chemin représentant une séquence d'états reliés par une séquence d'action.

La quatrième composante correspond au test but. Celle-ci détermine si un état donné est un état but. Enfin vient la cinquième et dernière composante, le coût du chemin. Elle permet d'attribuer une valeur numérique à un chemin en accord avec la mesure de performance imposée. Maintenant qu'une présentation générale de l'intelligence artificielle a été faite, le comportement d'essaim d'agents intelligents peut être étudié. Pour ce faire, il s'est avéré très intéressant d'en comprendre le comportement à partir d'exemples se trouvant dans la nature. Les fourmis et les abeilles illustrent bien cela. Deux algorithmes mettant en avant leur comportement ont été développé ci-dessous.

2.2 Algorithme des fourmis

Cet algorithme est basé sur le comportement des fourmis dont une des particularités est la communication au travers de l'environnement par dépôts de phéromones.

L'algorithme se présente de la manière suivante. Pour commencer, une exploration de l'environnement est faite par les fourmis. Si l'une d'entre elles trouve une source, elle déposera, lors de son retour au nid, des phéromones tout au long du chemin qu'elle emprunte. Dès lors, lorsque que d'autres fourmis partiront à la recherche de nourriture, elles auront tendance à suivre le chemin marqué de phéromones. Et à leur retour à la colonie, elles renforceront cette piste.[2]

Cet algorithme illustre une communication indirecte d'un essaim et la manière dont ce dernier est influencé [3]. L'autre algorithme, illustrant le comportement d'une structure organisée, est l'algorithme des abeilles.

2.3 Algorithme des abeilles

L'algorithme des abeilles est un «algorithme d'optimisation basée sur un comportement intelligent particulier des essaims d'abeilles».CITATION ?

Dans cet algorithme, tout comme dans celui des fourmis, les abeilles explorent le milieu environnant le nid. Par contre, ces dernières partagent des informations de manière directe. En effet, si une source est trouvée, celle-ci débutera une danse qui

aura pour but d'avertir les autres abeilles. Plus la source sera importante, plus la danse sera complexe. [4]

Comme observé dans ces algorithmes, la communication est indispensable à une bonne organisation d'un collectif d'individus. Dans le cadre de ce projet, le moyen dont les robots vont se partager les informations peut s'y inspirer.

D'une part, en s'inspirant de la communication des abeilles, les robots pourraient communiquer l'emplacement des sources de manière directe. Pour cela, celles-ci allumeraient leur LED pour communiquer l'emplacement d'une ressource. Ou d'autre part, pour communiquer indirectement, les robots pourraient déplacer les objets à l'aide de ses effecteurs. Par exemple, ils déplaceraient les obstacles dans son environnement de telle manière à créer un chemin qui mènerait du nid à la source.

Le prise en charge de la communication des robots ne sera traitée qu'au second quadrimestre. Mais tout ceci est concevable car les robots disposent d'effecteurs capables d'interagir sur l'environnement et de capteurs capables de cerner les différentes modifications de ce dernier.

Chapitre 3

Présentation du simulateur ARGoS et des footbots

Dans le projet, on a à notre disposition des robots bien spécifiques (figure se trouvant ci-dessous). Ceux-ci possèdent des capteurs (des caméras, senseurs, scanners) et des effecteurs (des roues motrices, une balise et une pince). Les capteurs leurs permettent de recevoir des informations émanant de l'environnement et les effecteurs leurs donnent la possibilité d'interagir avec cet environnement.

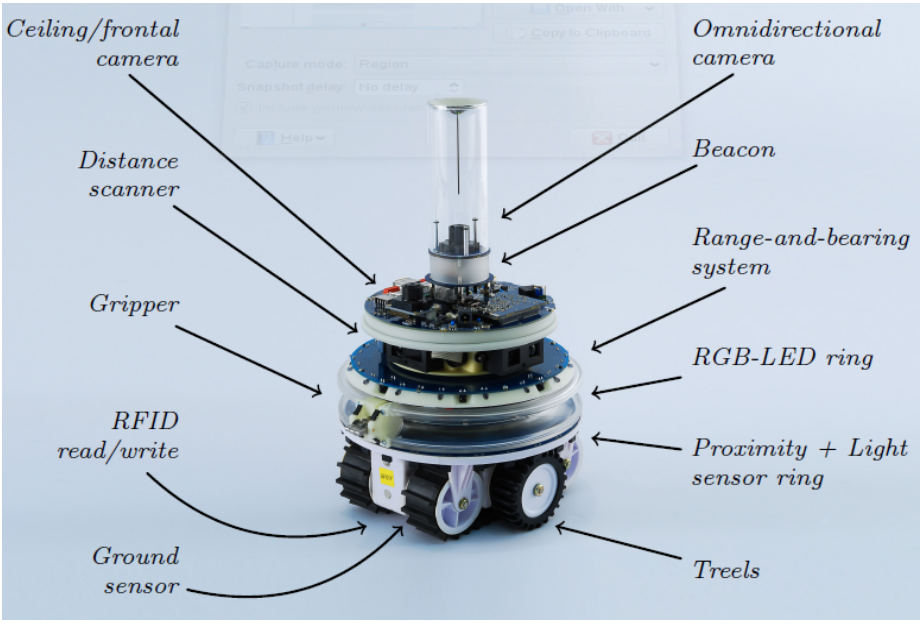

FIGURE 3.1 – Les différents senseurs et actuateurs d'un footbotCITATION

Le comportement en essaim des robots devra être simulé dans ARGoS. Ce dernier est un simulateur de robots open source qui a initialement été développé au sein du projet Swarmanoid. Pour l'exécuter, un fichier XML est nécessaire. Il permet la configuration du programme pour une expérience spécifique. Pour dicter la conduite

que doivent suivre les robots, un script est nécessaire. Ce script peut être écrit dans deux langages que gère ARGoS : C++ et Lua.

Vu que le script peut être écrit en Lua ou en C++, un choix entre ces deux langues doit être établi. Pour le bien du projet, il faut savoir optimiser le programme sans tomber dans un perfectionnisme inutile. On doit donc établir les avantages et inconvénients que possèdent ces deux langages tout en conservant en tête ce que requiert réellement le projet.

Chapitre 4

Choix du langage informatique

Lors de la présentation du simulateur ARGoS, deux choix de langage de programmation étaient possibles pour programmer les robots : C++ et Lua.

Après s'être familiarisé avec ces langages, la décision s'est portée sur Lua pour diverses raisons.

D'une part, les principaux avantages de C++, notamment, sa rapidité d'exécution et ses grandes librairies, ne semblaient pas primordiaux dans le cadre de ce projet. À cela s'ajoute les difficultés que ce langage aurait introduites. Plus spécifiquement, de meilleures connaissance aurait été nécessaire pour programmer, sans compter le temps perdu dans les débuguage du code. D'autre part, Lua est proche du langage Python. Ce dernier a été étudié en BA1 à l'école polytechnique de Bruxelles. Comme Lua est un langage de scripting, il était plus adapté aux besoins du groupe grâce à sa facilité de concevoir des prototypes de programmes rapidement. Sa simplicité de compréhension a aussi favorisé une plus grande clarté du code des autres membres, ce qui a permis de dédier plus de temps à l'avancement du projet.

Il faut aussi souligner le fait que la nouvelle version d'ARGoS permet de compiler un fichier Lua en un seul clic grâce au compilateur intégré à ARGoS. Pour compiler un fichier C++, il faut passer par le terminal et appeler le compilateur pour ensuite exécuter le fichier compilé dans ARGoS. Cette étape supplémentaire peut être néfaste à la productivité sachant que l'on requiert souvent de tester le code qui a été écrit. Cette nouvelle version possède aussi, par défaut, un contrôleur Lua qui associe les différents capteurs à des fonctions de base exploitable dans un code Lua. Sur les quatre fonctions prédéfinies de LUA combiné a ARGoS (init, step, reset et destroy), les trois premières sont primordiales pour qu'ARGoS fonctionne. En effet ces fonctions s'exécutent une à une sur un seul robot puis l'autre et ainsi de suite. En C++, il aurait fallu réaliser cette étape en plus et même changer l'implémentation des fonctions de base.

En vue des raisons énoncées ci-dessus, Lua a été choisi comme langage de programmation.

Chapitre 5

Trajectoire

5.1 Déplacement élémentaire

Comme il a été présenté dans l'introduction sur l'intelligence artificielle, et vu en pratique dans la structure de base d'un comportement Lua interprétable par ARGoS, un footbot exécute à chaque step une séquence d'opérations. On peut distinguer dans cette séquence trois types d'opérations (cf chapitre 2) : l'écoute des capteurs, la prise de décision et les interactions sur l'environnement par l'intermédiaire des effecteurs, que l'on nommera ici simplement sous le nom d'actions. Ces actions sont donc limitées et déterminées par les effecteurs dont dispose le robot et qui sont décrits au chapitre 3. Dans ce chapitre-ci, l'un des effecteurs primordiaux d'un footbot sera examiné : son déplacement.

Tout d'abord il est intéressant de se pencher sur le lien entre les paramètres physiques du robot et la manière dont celui-ci peut effectuer l'action simple «se rendre d'un point A à un point B» dans le cas où le robot agit seul dans un environnement sans obstacles. Ensuite, nous verrons comment le robot peut s'accommoder des obstacles (fixes, prévisibles) et autres robots (mobiles, imprévisibles) à partir d'une ou plusieurs de ces actions simples.

L'effecteur dont un footbot dispose afin de se déplacer est une paire de roues dont les vitesses peuvent être fixées de manière indépendante. A chaque instant, les seuls deux mouvements auxquels peut accéder le robot sont donc une translation parallèle aux roues et une rotation autour d'un point au milieu de l'axe des roues. La mécanique [6] nous indique que la composition de ces deux mouvements est suffisante pour permettre au robot de se déplacer librement dans un plan mais surtout, elle nous donne la relation entre les vitesses des deux roues et la vitesse générale ainsi que la vitesse de rotation du robot :

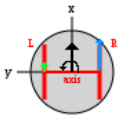

FIGURE 5.1 – Signification physique de v_r, v_l, v_g *et* ω_g CITATION

$$\begin{cases} v_g = \frac{v_r + v_l}{2} \\ \omega_g = \frac{v_r - v_l}{l_{axe}} \end{cases} \tag{5.1}$$

A partir de cette loi des vitesses, il est aisé de construire un algorithme permettant à un robot de converger vers sont but selon une trajectoire souple et à vitesse constante.

Algorithm 5.1 Convergence with no obstacle avoidance

Require: $SPEED :=$ Fixed speed of the footbot > 0
goal in arena
Ensure: footbot converges towards the goal at speed $SPEED$

```
1 while goal not reached do
2   update footbot position and orientation
3   θ ← angle between the direction of the goal from the footbot and footbot orientation
4   right velocity ← convergence(θ, SPEED)
5   left velocity ← 2 · SPEED − right velocity %so that overall speed stays equal to SPEED
6   robot.wheels.set_velocity(left velocity, right velocity)
7 end while
```

Où convergence(θ, SPEED) fixe la convergence du robot vers son goal. Elle doit satisfaire :

$$\begin{cases} convergence(0, SPEED) = SPEED \\ convergence(\overbrace{0 < \theta \leq \pi}^{\text{goal à gauche du robot}}, SPEED) > SPEED \\ convergence(\overbrace{0 > \theta \geq -\pi}^{\text{goal à droite du robot}}, SPEED) < SPEED \end{cases} \tag{5.2}$$

Pour une fonction convergence($\theta, SPEED$) donnée satisfaisant à cette condition (par exemple dépendance linéaire en θ), le footbot peut donc se rendre d'un point A à un point B, tant qu'il ne rencontre pas d'obstacles sur son trajet. Notons que cet algorithme s'intègre particulièrement bien dans la fonction loop demandée par ARGoS. Dans notre projet nous avons choisi

$$convergence(\theta, SPEED) = (1 + \kappa\frac{\theta}{\pi})SPEED$$

Où $0 < 2\kappa \leq 2$ est un paramètre qui fixe l'intensité de la convergence.

La manière la plus directe de permettre au robot d'éviter des obstacles quels qu'ils soient est d'exécuter une routine d'évitement à la place d'une routine de convergence lorsqu'un obstacle est détecté.

Algorithm 5.2 Convergence with obstacle avoidance

Require: $SPEED :=$ Fixed speed of the footbot > 0
goal in arena
Ensure: footbot converges towards the goal at speed $SPEED$ while avoiding obstacles

```
1  while goal not reached do
2    update footbot position and orientation
3    read proximity sensors %or whatever other sensor in use
4    if no obstacles then
5      θ ← angle between the direction of the goal from the footbot and footbot orientation
6      right velocity ← convergence(θ, SPEED)
7    else
8      right velocity ← avoidance(proximity sensor reading, SPEED)
9    end if
10   left velocity ← 2 · SPEED − right velocity %so that overall speed stays equal to SPEED
11   robot.wheels.set_velocity(left velocity, right velocity)
12 end while
```

Où avoidance(proximity sensor reading) fixe la routine d'évitement du robot. Son implémentation est très libre et peut fortement varier en fonction du capteur utilisé pour détecter les obstacles. On peut par exemple utiliser le senseur proximity du footbot, qui associe à 24 directions autour du robot une valeur entre zéro et un : une valeur zéro indique qu'aucun obstacle n'est perçu dans la direction donnée tandis qu'une valeur supérieure indique qu'un objet a été détecté. Cette valeur augmente au fur et à mesure que le robot se rapproche de l'obstacle [7]. Dans notre projet nous avons choisi

$$avoidance(direction, proximity) = \begin{cases} \frac{-\alpha+(1-proximity)^{\beta}\cdot direction}{11} SPEED & \text{si } direction \leq 12 \\ \frac{(22+\alpha)-(1-proximity)^{\beta}\cdot(25-direction)}{11} SPEED & \text{si } direction \geq 12 \end{cases}$$

Où $1 \leq \alpha \leq 12$ est un paramètre qui fixe l'influence de la direction de l'obstacle le plus proche et $0 \leq \beta$ fixe l'influence de la proximité de cette obstacle.

Grâce à une routine d'évitement efficace prenant en compte la direction de l'obstacle perçu et sa proximité [8], ainsi qu'en utilisant l'algorithme présenté plus haut, il est déjà possible de produire une solution fonctionnelle à la partie déplacement du cahier des charges, même si on peut améliorer cette solution en fonction de la connaissance de son environnement dont dispose le robot.

Ainsi, dans le cas omniscient ou si le robot est capable de construire une carte de son environnement reprenant la position des différents obstacles il peut-être judicieux d'utiliser un algorithme de recherche du plus court chemin. La manière la plus directe de faire est de donner au footbot une liste de goal successifs qui le mèneront au goal final. Ceci permet de réutiliser facilement les algorithmes déjà présentés tout en étant parfaitement compatible avec les valeurs de retour typiques d'un algorithme de recherche du plus court chemin. En effet, la plupart des recherches du plus court

chemin utilisent une représentation en graphe d'un environnement. La valeur de retour d'une telle recherche est donc une liste des nœuds qu'il faut parcourir dans le graphe afin d'arriver au but final, ce qui est précisément ce que cet algorithme fait.

Algorithm 5.3 Convergence with path finding

Require: $SPEED :=$ Fixed speed of the footbot > 0
intermediate goals list := list of points which lead to the goal while avoiding the obstacles
goal in arena
Ensure: footbot converges towards the goal at speed $SPEED$ while avoiding obstacles

```
1  for intermediate goal in intermediate goals list do
2    while goal not reached do
3      update footbot position and orientation
4      read proximity sensors %or whatever other sensor in use
5      if no obstacles then
6        θ ← angle between the direction of the goal from the footbot and footbot
         orientation
7        right velocity ← convergence(θ, SPEED)
8      else
9        right velocity ← avoidance(proximity sensor reading, SPEED)
10     end if
11     left velocity ← 2 · SPEED − right velocity %so that overall speed stays
       equal to SPEED
12     robot.wheels.set_velocity(left velocity, right velocity)
13   end while
14 end for
```

L'implémentation de l'algorithme de recherche du plus court chemin qui fournit intermediate goals list est un problème à part entière. Avant de l'examiner plus en détail, il faut noter que malgré l'utilisation d'une recherche du plus court chemin qui devrait a priori permettre d'éviter les obstacles, le test d'obstacles est toujours présent, ainsi que la possibilité d'évitement. Il est évident que ceci est fait pour permettre d'éviter des objets inattendus tels que d'autres robots, par exemple. Cependant, on peut dès lors se demander s'il ne faudrait pas aussi chercher de nouveau un plus court chemin après un évitement imprévu ou si le robot a dévié d'une distance significative de sa trajectoire prévue.

5.2 Recherche du plus court chemin

5.2.1 Algorithme A*CITATION

En informatique, A * est un algorithme informatique qui consiste à mettre en place un processus de traçage d'un chemin traversable efficace entre des points. Les points sont considérés comme des noeuds.

A* utilise une «best-first» recherche et trouve le chemin possédant le moindre coût à partir d'un nœud initial donné à un nœud but.

Pour cela, il utilise une fonction de coût pour déterminer l'ordre dans lequel les visites de recherche des nœuds de l'arbre vont s'effectuer. Cette fonction de coût est la somme de deux autres fonctions. La fonction coût de la trajectoire passée qui est la distance connue à partir du nœud de départ au dernier parcouru lors de la recherche. Et la fonction coût du chemin futur qui est une estimation heuristique de la distance entre le dernier nœud parcouru et le nouveau nœud à atteindre .

Cet algorithme possède quand même des limites. Il est effectivement efficace dans le cas où l'on considère les robots comme omniscient, connaissant l'environnement et, donc, connaissant la position de la source et des obstacles se trouvant dans l'environnement. Dans le cas de non-omniscience, l'arène est inconnue et on ne possède aucunes données à propos de celles-ci. Et c'est ici que se trouve le plus grand défaut de l'algorithme A*.

A* détermine un chemin complet de nœuds pour arriver d'un noeud départ à un noeud but. Lorsqu'il ne possède pas de données complètes à propos de l'arène, il est bloqué lors de son exécution et ne peut donc pas déterminer le chemin que doit suivre le robot. Il faudra adapter l'algorithme A* afin de remédier à ce problème.

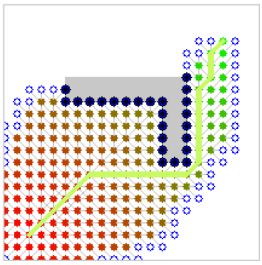

FIGURE 5.2 – Représentation d'une éxécution de A*

5.2.2 Algoritme de Dijkstra CITATION

L'algorithme de Dijkstra est un algorithme servant à résoudre le problème du plus court chemin. Le principe de l'algorithme est le suivant :

Il s'agit de mettre en place progressivement un sous-graphe dans lequel sont classés les différents sommets. Un ordre croissant est établit entre les sommets et il est fixé en fonction de la distance minimale qui éloignent ces sommets à celui de départ. Cette distance correspond à la somme des nœuds parcourus.

Au début, les distances de chaque sommet par rapport au sommet de départ sont considérées comme infinie et on attribue à celui-ci une distance de 0.

Ensuite, au cours de chaque itération, les distances des sommets reliés par un nœud au dernier du sous-graphe sont mis à jour. Cette mis à jour consiste à ajouter la valeur du nœud à la distance séparant le sommet de départ à ce dernier sommet. Après cette mise à jour, l'ensemble des sommets, ne faisant pas partis du sous-graphe, sont examinés et celui qui possède la distance minimal y est ajouté.

Enfin, on répète l'exécution jusqu'à l'épuisement des sommets ou jusqu'à la sélection du sommet d'arrivée. Voici 3 figures qui représentent un exemple du principe utilisé. Le but est de trouver le plus court chemin entre le point A et le point J. Comme dit précédemment, après chaque mise à jour, le sommet possédant la distance minimale est rajouté au sous graphe. La figure 7 représente bien cela.

En effet, le sommet E est rajouté au sous graphe et non le sommet I car la distance à parcourir entre A et E est plus petite que entre A et I.

Il faut souligner que cet algorithme possède le même inconvénient qu'A*.

Chapitre 6

Communication et prise de décision

6.1 Gestion de l'autonomie

Chapitre 7

Résultats obtenus

Chapitre 8

Fonctionnement du groupe

8.1 Général

8.2 Organisation

8.3 Communication

Chapitre 9

Conclusion