REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA

Faculté de **T**echnologie

Département du **G**énie **E**lectrique

Mémoire de fin d'études de Master en

Filière : Automatique.

Spécialité : Automatique des Systèmes.

Planification et Optimisation de Trajectoire d'un Robot Mobile par l'Algorithme A star

---

Présenté par :

BOUAM Newres

BOUCHENAFA Adel

Proposé et Dirigé par :

Pr. HENINI Noureddine

Dr. MESSEI Adnane

Promotion 2022-2023

Remerciements

*Je remercie avant tout **Dieu** tout puissant de m'avoir donné la force et la volonté pour accomplir ce modeste travail.*

*Ce travail a été réalisé au sein du département de Robotique de la division d'électronique et d'instrumentation nucléaire du centre de recherche nucléaire de Birine. Je tiens à exprimer mes remerciements à Monsieur **ABDELLANI IDIR**, Directeur Général du CRNB, d'avoir accepter de m'accueillir dans son établissement et de mettre à ma disposition tous les moyens nécessaires pour l'accomplissement de mon travail.*

*J'exprime mes profondes gratitudes à **mes parents** pour leurs encouragements, leur soutien et pour les sacrifices qu'ils ont endurés.*

*Je tiens également à exprimer mes vifs remerciements à mon encadreur; le Professeur Mr. **HENINI Noureddine** pour avoir d'abord proposé ce thème et le suivi continuel tout le long de la réalisation de ce travail et qui n'a pas cessé de me donner ses conseils.*

*Que Monsieur le Co–Promoteur **le chercheur Senior du Centre de Recherche Nucléaire de Birine, Mr. MESSEI Adnane**, trouve ici ma gratitude pour son aide et ses conseils précieux qui m'ont permis de mener à terminé mon projet.*

Mes remerciements sont destinés aussi à Messieurs le président et les membres du jury de m'avoir honoré par leur présence pour juger et évaluer mon travail.

Enfin, je tiens à exprimer mes reconnaissances à tous mes amis et collègues pour le soutien moral.

Newres BOUAM

Remerciements

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je voudrais tout d'abord adresser toute ma reconnaissance à Monsieur HENINI NOUREDDIN Professeur à l'université de Médéa, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je désire aussi remercier le président et les membres de jury pour leur disponibilité et pour le temps qu'ils ont consacrés pour la lecture et l'examination de ce travail.

Je tiens à remercier Madame et Monsieur BOUAM pour le stage pratique, ainsi pour les outils qui m'ont fourni pour la réussite de ce projet.

J'adresse mes sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté de me rencontrer et de répondre à mes questions durant mes recherches.

J'aimerais aussi gratifier les efforts de mon binôme en l'occurrence Mlle BOUAM NAWRES, qui a eu l'amabilité de travailler ensemble dans ce sujet et atteindre les objectifs fixés.

Je voudrais exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche.

Enfin, je tiens à remercier mes chers parents, qui ont toujours été là pour moi, je remercie ma sœur Feriel et mes deux frères Rafik et Mounir pour leur soutien constant et encouragements.

Adel BOUCHENAFA

إهـــداء

اهدي هذا العمل المتواضع إلى:

أبي صاحب الهمة العالية

و إلى أمي الصبورة العزيزة الغالية

و إلى أصدقائي أصحاب النفوس الزكية

و إلى جميع أساتذتي منذ نشأت في الابتدائية

إلى كل هؤلاء و هؤلاء أهدي عملي

وارجوا من الله رب البرية

أن يتقبلها مني بإخلاص ونية

نـــورس

إهـــداء

اهدي هذا العمل المتواضع إلى:

أبي صاحب الهمة العالية

و إلى أمي الصبورة العزيزة الغالية

و إلى إخوتي أصحاب النفوس الزكية

و إلى جميع أساتذتي منذ نشأت في الابتدائية

إلى كل هؤلاء و هؤلاء أهدي عملي

وارجوا من الله رب البرية

أن يتقبلها مني بإخلاص ونية

عـــادل

ملخص :

الهدف من هذا العمل يتمثل في تطوير برنامج على Matlab والذي يسمح بتحديد المسار الأمثل في بيئة ثنائية الأبعاد في أقل وقت ممكن. تعتمد الطريقة على خوارزمية A (*) المستخدمة على نطاق واسع في الممارسة العملية وفقًا لمزاياها والنتائج التي تم الحصول عليها مقارنة بالخوارزميات الأخرى. تتوافق نتائج المحاكاة التي تم الحصول عليها مع تلك الموجودة في المراجع العلمية وتظهر قدرة هذه الخوارزمية على حل المشكلة المحددة.

الكلمات المفتاحية: خوارزمية (A *)، تحديد المسار الأمثل، تخطيط المسار.

Résumé :

L'objectif de ce travail consiste en l'élaboration de programme informatique sur le logiciel Matlab qui permet la détermination de la trajectoire optimale d'une navigation dans un environnement bidimensionnel dans un temps minimal. La méthode est basée sur l'algorithme A (*) star qui est très utilisée en pratique selon ses avantages et des résultats obtenus par rapport aux autres algorithmes. Les résultats de simulation obtenue sont en bon accord avec ceux de la littérature et montrent la capacité de cet algorithme de résoudre le problème donné.

Mots clés : Algorithme A star (A*), Navigation, Planification de trajectoire.

Abstract:

The objective of this work consists in the development of a computer program on Matlab software which allows the determination of the navigation optimal trajectory in a two-dimensional environment at minimal time. The method is based on the A (*) star algorithm which is widely used in practice according to its advantages and the results obtained compared to other algorithms. The simulation results obtained are in good agreement with those of the literature and show the ability of this algorithm to solve the given problem.

Keywords: A star (A*) algorithm, Navigation, Pathfindig.

Table des Matières

Table des matières…… i
Liste des figures…… iv
Liste des tableaux…… iv
Introduction Générale…… 01

Chapitre I : **Robots Mobiles**…… 03
I.1- Introduction…… 03
I.2- Définition d'un robot mobile…… 03
I.3- Historique…… 05
I.4- Avantages et inconvénients des robots mobiles…… 05
I.5- Classification des robots mobiles…… 06
I.6- Domaines d'application de la robotique …… 06
I.6.1. Utilisation civile…… 06
I.6.2- Santé…… 07
I.6.3- Industrie…… 07
I.6.4- Domaine militaire…… 08
I.6.5- Usage domestique…… 08
I.7- Types de robots mobiles…… 09
I.7.1- Robot mobile utilisant la chenille…… 09
I.7.2- Robot mobile à pattes…… 09
I.7.3- Robot mobile à roues…… 10
I.7.3.1- Robot uni-cycle…… 10
I.7.3.2- Robot tricycle…… 10
I.7.3.3- Robot voiture…… 11
I.7.3.4- Robot omnidirectionnel…… 11
I.7.3.5- Comparaison des différents types…… 12
I.8- Architecture des robots mobiles…… 12
I.8.1- Composants matériels d'un robot mobile…… 12
I.8.2- Energie utilisée dans les robots…… 12
I.8.3- Organes de sécurité…… 13
I.9- Moyens de perception et de locomotion en robotique mobile…… 13
I.9.1- Capteurs…… 13
I.9.1.1- Capteurs intéroceptifs…… 14
I.9.1.2- Capteurs extéroceptifs…… 14
I.9.2- Actionneurs…… 16
I.10- Conclusion…… 17

Chapitre II : **Principaux Algorithmes Utilisés dans la Navigation d'un Chemin**… 18
II.1- Introduction..…… 18
II.2- Solution du fainéant…… 18
II.3- Algorithme de Dijkstra…… 18
II.4- Depth-First Search (DFS)…… 19
II.5- Première exploration itérative en profondeur (IDDFS)……..…….... 19
II.6- Recherche du meilleur premier choix (B-F-S)…… 20
II.7- B * (B Star)…… 20
II.8- Recherche du meilleur premier (BFS Breadth First Search)…… 20
II.9- L'algorithme A*…… 23
II.9.1- Utilisation de la méthode heuristique A*…… 24
II.9.2- Vitesse ou exactitude...…… 25
II.9.3- Echelle…… 25
II.9.4- Heuristique exacte…… 26
II.9.5- Heuristique exacte pré-calculée……..…… 26
II.9.6- Heuristique pour des cartes de réseaux…… 26
II.10- Recherche du chemin…… 32
II.11- Conclusion…… 33

Chapitre III : **Planification de la trajectoire par A star (A*)**…… 34
III.1- Introduction…… 34
III.2- Déroulement de l'algorithme…… 34
III.2.1- Développement du nœud A…… 35
III.2.2- Développement du noeud D…… 35
III.2.3- Développement du noeud F…… 35
III.2.4- Arrivée au noeud I…… 35
III.3- Description de l'algorithme…… 37
III.3.1- Les listes A*…… 37
III.3.2- Détermination des noeuds voisins…… 38
III.3.3- Notion de parent d'un nœud…… 39
III.3.4- Récapitulation des étapes…… 39
III.4- Propriétés de A*…… 40
III.5- Organigramme de simulation…… 40
III.5.1- Organigramme de simulation…… 41
III.5.2- Présentation des différentes fonctions…… 42
III.5.2.1- Fonction "fill open list" (ou "remplir la liste ouverte") : [23-24]…… 42
III.5.2.2- Fonction distance A* (ou "heuristique")[23-24]…… 43
III.5.2.3- Fonction "expand FNC" [23-24]…… 44
III.5.2.4- Fonction "Min FNC" (ou "minimum FNC") [23-24]…… 45
III.5.2.5- Fonction "nod-index" (ou "index de nœud") [23-24]…… 45

III.6- Résultats de simulation……..… 46
III.6.1- Expérience 1 : Environnement sans obstacles……..…. 46
III.6.2- Expérience 2 : Environnement contient des murs comme obstacles…… 47
III.6.3- Expérience 3 : Robot mobile dans un labyrinthe…… 47
III.7- Conclusion…… 48

Conclusion Générale……..... 49
Références Bibliographiques……..…. 51

Liste des Figures

Figure	Titre	Page
(I-1)	*1950-Dr.Grey walter et sa tortue électronique [2].*	04
(I-2)	*Premiers robots mobiles [4].*	04
(I-3)	*Développent de la robotique mobile [5].*	05
(I-4)	*Robot d'exploration planétaire [6].*	05
(I-5)	*Robot absorbant la neige de la rue avec sa «bouche».*	06
(I-6)	*Robot assistant les chirurgiens.*	07
(I-7)	*Adaptation d'un robot à son environnement et à son utilisateur.*	07
(I-8)	*Robot de transport les paquetages des soldats de l'armée américaine.*	08
(I-9)	*Robot autonome aspirateur sur tous types de sol.*	08
(I-10)	*Robot à chenille.*	09
(I-11)	*Robot à pattes.*	09
(I-12)	*Robot mobile de type uni cycle.*	10
(I-13)	*Robot mobile de type tricycle.*	10
(I-14)	*Robot mobile de type voiture.*	11
(I-15)	*Robot omnidirectionnel.*	11
(I-16)	*Capteurs à ultrasons.*	15
(I-17)	*Capteurs à infrarouge.*	15
(I-18)	*Capteurs tactiles.*	15
(I-19)	*Boussoles.*	16
(I-20)	*Les balises.*	16
(I-21)	*GPS.*	16
(II-1)	*Algorithme de Dijkstra.*	19
(II-2)	*Recherche premier mieux (BFS).*	21
(II-3)	*Dijkstra avec des obstacles concaves.*	21
(II-4)	*BFS avec des obstacles concaves.*	22
(II-5)	*Recherche graphique dans un secteur sans obstacles due plus court chemin.*	23
(II-6)	*Algorithme A* avec un obstacle concave.*	24
(II-7)	*Distance du Manhattan.*	27
(II-8)	*Distance diagonale.*	27
(II-9)	*Distance Euclidienne.*	28
(II-10)	*Liens de casse dans les valeurs de f.*	29
(II-11)	*Graduation cassant lien supplémentaire à heuristique.*	30
(II-12)	*Graduation cassant lien supplémentaire heuristique travaille gentiment avec des obstacles.*	31
(II-13)	*Produit mutuel cassant lien supplémentaire à .heuristique, produit de jolis chemins.*	31
(II-14)	*Produit mutuel cassant un lien supplémentaire à heuristique avec obstacles (artefact).*	31
(III-1)	*Fonctionnement de l'algorithme A*.*	35
(III-2)	*Consistance de l'exploration A*.*	36
(III-3)	*Illustration de l'optimalité de A*.*	37
(III-4)	*Illustration d'une itération de l'algorithme A*.*	40
(III-5)	*Organigramme du fonctionnement du programme d'Algorithme A*.*	41
(III-6)	*Fonctions de A*.*	42
(III-7)	*Trajectoire planifiée par A* dans un environnement sans obstacles.*	47
(III-8)	*Trajectoire planifiée par A* dans un environnement contient des murs comme obstacles.*	47
(III-9)	*Trajectoire planifiée par A* dans un labyrinthe.*	47
(III-10)	*Trajectoire planifiée par A* à travers obstacles quelconques.*	48

Liste des tableaux

Tableau	Titre	Page
(I-1)	*Avantages et les inconvénients des différents types de robots à roues.*	12

Introduction Générale

Introduction Générale

Le robot mobile est un système qui intègre la conception de l'environnement, la prise de décision dynamique, le contrôle du comportement, la planification des tâches et l'exécution. Cela accomplit des fonctions telles que le mouvement, la navigation automatique, le contrôle multicapteur et l'interaction du réseau.

Les robots mobiles peuvent être largement utilisés dans différents domaines, à savoir : les gares, les aéroports, l'industrie, etec… et occupent rapidement des rôles importants dans notre vie quotidienne. En effet, ils ont permis de remplacer l'homme dans certains travaux difficiles, répétitifs ou pénibles et de lui apporter une plus grande précision.

Les robots doivent planifier un chemin approprié pour se déplacer lorsqu'ils naviguent dans un environnement complexe ou incertain. La planification de trajectoire détermine une trajectoire sans collision dans un environnement donné.

Au cours des derniers temps, différentes méthodes de planification de trajectoire [1–5] ont été proposées et démontrées sur des robots ou des véhicules guidés automatisés (VGA) avec diverses applications. Les algorithmes classiques de planification de chemin consiste en l'algorithme génétique (AG) [6, 7], l'algorithme d'optimisation des colonies de fourmis (ACO) [8–10], l'algorithme d'exploration rapide des arbres aléatoires (RRT) [11, 12] et l'algorithme A-Star (A *) [13–15].

L'algorithme A* est une technique qui est utilisée depuis longtemps dans la communauté de recherche. Son efficacité, sa simplicité et sa modularité sont souvent avantageux forts par rapport à d'autres algorithmes. En raison de son omniprésence et de son utilisation répandue, A* est devenu une option courante pour les chercheurs qui tentent de résoudre des problèmes.

L'objectif de ce travail consiste en l'élaboration d'un programme sur Matlab qui permet la détermination de la trajectoire optimale d'une navigation dans un environnement bidimensionnel à un temps minimal. La méthode est basée sur l'algorithme A (*) star qui est très utilisée en pratique selon ses avantages et des résultats obtenus par rapport aux autres algorithmes. Les résultats de simulation obtenue sont en bon accord avec ceux de la littérature et montrent la capacité de cet algorithme de résoudre le problème donné.

Ainsi, ce mémoire est structuré en trois chapitres, comme suit :

Le premier chapitre traite les robots mobiles. Le but est de dégager le type de l'architecture du robot à adopter.

Le deuxième chapitre traite le principe de navigation d'un chemin, et les divers algorithmes existent dans la littérature pour gérer, rechercher des plus courts chemins et des distances dans des environnements bidimensionnels exploitent la topologie particulière de navigation. Nous avons également étudié les différentes étapes ainsi que les avantages et les inconvénients de chaque algorithme.

Dans le dernier chapitre, la technique de l'algorithme A* a été détaillée et adapté à l'établissement d'un programme de navigation d'un un environnement de 2D sans et avec obstacles en utilisant le logiciel Matlab R2014a (8.3.0.532) où les différents résultats ont été présentés.

Le mémoire se terminera par une conclusion générale qui résumera les résultats obtenus et les perspectives futures de ce travail

Chapitre I :

Robots Mobiles

Robots Mobiles

I.1- Introduction

Les robots mobiles ont une large importance dans la vie quotidienne, l'industrie, la recherche scientifique, …etc. Dans ce chapitre, Nous expliquons l'état général dans lequel notre projet prend place. Nous présentons donc les concepts de base des robots mobiles, leurs applications, avantages et types. L'accent sera mis ensuite sur leur architecture et leurs moyens de perception.

I.2- Définition d'un robot mobile

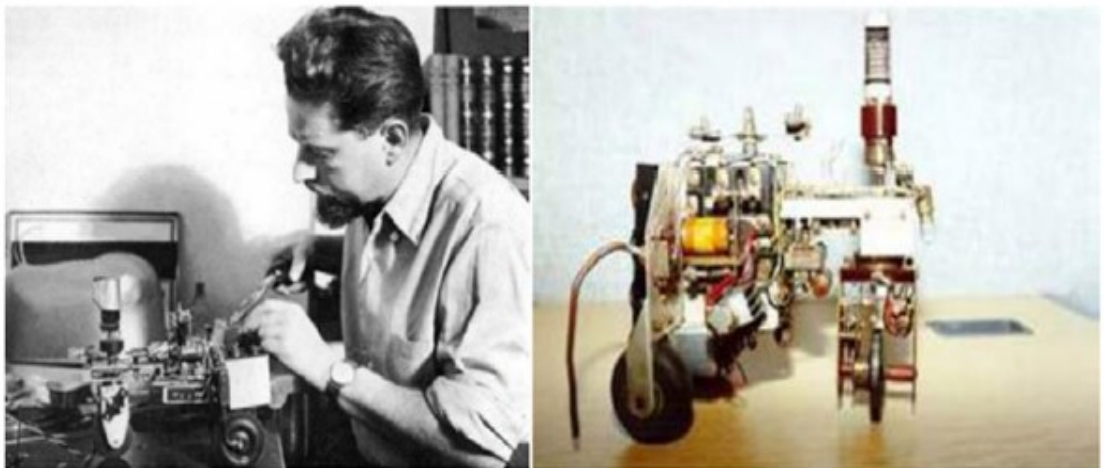
Un robot mobile est un système électronique et informatique qui interagit physiquement avec son environnement pour atteindre un objectif spécifique. Il est capable de s'adapter à des variations dans ses conditions de fonctionnement et possède des fonctions de perception, de décision et d'action. Par conséquent, le robot peut effectuer diverses tâches de différentes manières et accomplir sa propre tâche correctement même dans des situations imprévues.

Les robots mobiles sont des machines capables de se déplacer dans leur environnement. Cependant, leur mode de déplacement varie en fonction du domaine d'application. Par exemple, la mobilité sur roues est la structure mécanique la plus couramment utilisée. Cette technique permette un déplacement dans toutes les directions avec une accélération et une vitesse importantes en fonction de l'agencement et des dimensions des roues [1].

I.3- Historique

Le mot robot vient du tchèque "robota" qui signifie "travail forcé" ou "servitude". Dans la pièce, les robots sont présentés comme des serviteurs dociles et efficaces pour réaliser les tâches pénibles mais qui se rebellent contre leurs créateurs. [1].

1948 : Grey Walter a inventé la première tortue robotique mobile autonome qui se dirigeait vers les sources de lumière qu'elle percevait. Cependant, ce robot n'était pas programmable. La mise en place de robots programmables n'a été possible que dans les années 50 avec la création des transistors et des circuits intégrés [2].



a) Dr. *Grey walter.* **b)** *Tortue électronique.*

Figure (I-1) : *1950-Dr.Grey walter et sa tortue électronique [2].*

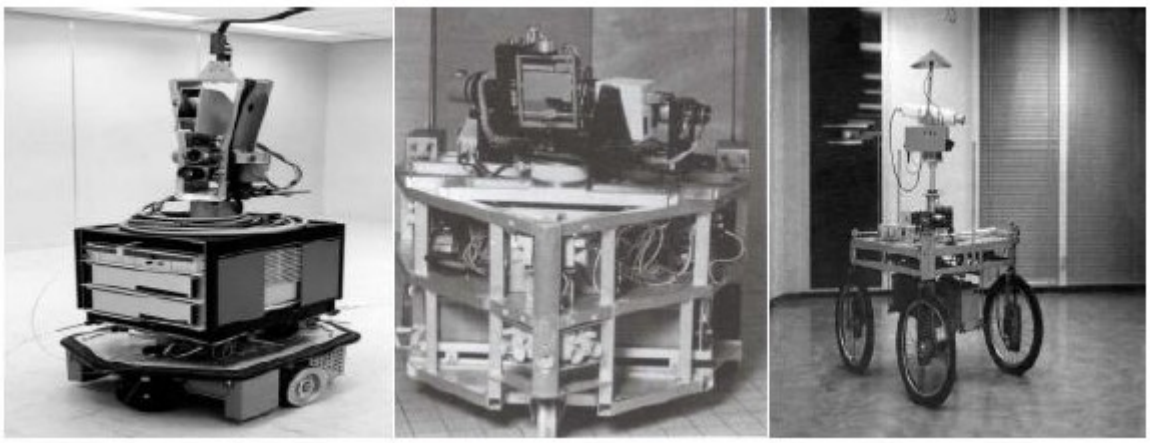
Dans les années 60, les recherches en électronique ont conduit à réaliser des robots plus complexes grâce à l'apparition du transistor. Ces robots étaient capables de réaliser des tâches similaires. Par exemple, le robot "Beast" de l'université John Hopkins était capable de se déplacer au centre des couloirs en utilisant des capteurs ultrasons, de chercher des prises électriques (noires sur des murs blancs) en utilisant des photodiodes et de s'y recharger [3].

1960-1980 :

Fin des années 60 : Mise en place de «shakey» premier robot mobiles intégrant perception, planification et exécution ;

1970 : Sandford cart ;

1977 : Premier robot mobile français HILARE au LAAS (CNRS Toulouse).



a) *Shakey.* **b)** *Hilare.* **c)** *Stanford cart.*

Figure (I-2) : *Premiers robots mobiles [4].*

1990-2000 : Développent de la robotique mobile ;

1995 : Mise en place de la RoboCup ;

1997 : Premier robot mobile extra planétaire sur mars ;

1999 : Lancement des robots chien Aibo, par SONY.



a) *Robocup.* **b)** *Aibo.* **c)** *Sojourner.*

Figure (I-3) : *Développent de la robotique mobile [5].*

Depuis 2000 :

2003 : Projet «Mars Exploration Rover»

2009 : Projet «Mars Science Laboratory» Le projet Rover a été suivi par l'envoi de Curiosity fin [2].



a) *Robot Spirit.* **b)** *Curiosity.*

Figure (I-4) : *Robot d'exploration planétaire [6].*

I.4- Avantages et inconvénients des robots mobiles

Divers avantages se résument :

*Accroissement de la capacité de production,

* Remplacement de l'homme dans l'exécution des taches pénibles ou dangereuses,

*Manutentions.

* Une meilleure qualité, Augmenter la productivité, Plus de sécurité ...

Inconvénients particuliers liés à la conception de tels robots sont [2] :

∗ La conception mécanique liée à la mobilité,

* La détermination de la position et de la latitude (orientation),

* La détermination du chemin optimal pour atteindre le lieu de la tâche.

* Investissement initial élevé, des experts en robotiques peuvent être difficiles à trouver, Coûts permanents.

I.5- Classification des robots mobiles

Une proposition de classification définissant le degré d'autonomie des robots mobiles :

- Voiture téléguidé par un opérateur qui lui oblige chaque opération élémentaire à réaliser.
- Voiture téléguidé au sens d'une opération à effectuer. La voiture vérifie automatiquement ses tâches.
- Voiture semi-autonome réalisant sans l'aide de l'opérateur des fonctions programmées.
- Voiture autonome qui réalise des tâches semi-programmées. Ce type de véhicule pose des difficultés d'un niveau de complexité haut de représentation des connaissances, de capacité décisionnelle et de génération de plans qui sont résolus à bord dans la mesure du possible.

I.6- Domaines d'application de la robotique

La robotique actuelle est utilisée dans de différents domaines :

I.6.1- Utilisation civile

De plus en plus de tâches sont confiées aux robots. Ils servent à remplacer les personnes qui sont chargées des opérations civiles (Nettoyer la ville, aider la population, s'occuper des lieux publics...). Le robot présenté sur la figure ci-dessous est équipé de deux caméras et d'un GPS. Il se déplace dans la rue de manière autonome et ramasse la neige avec sa "bouche".



Figure (I-5) : *Robot absorbant la neige de la rue avec sa «bouche».*

I.6.2- Santé

Les robots sont de plus en plus utilisés dans le domaine médical. Bien qu'ils ne soient pas complètement autonomes, ils assistent les médecins ou les chirurgiens et permettent même des opérations médicales à distance. Cette pratique de chirurgie assistée est émergente et bien que peu répandue, elle est en passe de devenir la chirurgie du futur. La figure (I-6) montre un robot constitué de quatre bras mobiles avec des mini-pinces à leur extrémité. Un chirurgien est assis devant une machine équipée de pédales et de joysticks.



Figure (I-6) : *Robot assistant les chirurgiens.*

I.6.3- Industrie

Le but principal des robots est de remplacer l'homme dans des activités difficiles et fatigantes pour l'employeur (figure I-7). Les robots ont commencé à être utilisés dans les chaînes d'assemblage industrielles. On y trouve des robots soudeurs, manipulateurs et peintres…



Figure (I-7) : *Adaptation d'un robot à son environnement et à son utilisateur.*

I.6.4- Domaine militaire

Les robots sont de plus en plus utilisés dans le domaine militaire, figure (I-8). Aujourd'hui, il est plus facile de créer des robots discrets et dotés de nombreux capteurs, ce qui est idéal pour des missions d'espionnage ou d'éclairement.



Figure (I-8) : *Robot de transport les paquetages des soldats de l'armée américaine.*

I.6.5- Usage domestique

Ces dernières années, de nombreux robots ont été conçus pour les particuliers afin d'effectuer des fonctions à la place de leur propriétaire. Ils sont capables de faire le ménage, tondre la pelouse, nettoyer la piscine… Ce qui conduit certains clients à se assurer ces domestiques contemporains.



Figure (I-9) : *Robot autonome aspirateur sur tous types de sol.*

I.7- Types de robots mobiles

I.7.1- Robot mobile utilisant la chenille

Lorsque le terrain est accidenté, les roues perdent leur efficacité de locomotion. Ce type de système de locomotion restreint la liberté de mouvement du robot mobile qui l'utilise. En revanche, les chenilles offrent plusieurs avantages, tels qu'une meilleure adhérence au sol et la capacité de surmonter des obstacles plus importants.



Figure (I-10) : *Robot à chenille.*

I.7.2- Robot mobile à pattes

Lorsque le terrain devient encore plus imprévisible, avec de grandes variations de hauteur telles qu'un escalier ou un terrain très accidenté, les deux types de systèmes précédemment mentionnés ne sont plus efficaces, et l'on se tourne vers les robots mobiles à pattes. Ces robots possèdent des points d'appui discrets sur le terrain, ce qui les rend adaptés à ce problème de mouvement. Cependant, concevoir et contrôler un robot à pattes est extrêmement complexe. De plus, leur vitesse de déplacement est généralement très réduite. La commande est très difficile et dépend de la multiplicité des actionneurs utilisés. Un exemple de robot mobile à pattes est l'Aibo de Sony [3].

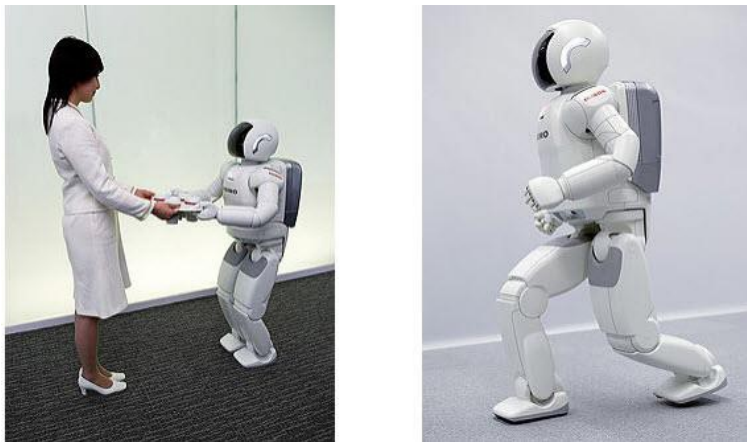


Figure (I-11) : *Robot à pattes.*

I.7.3- Robot mobile à roues

Les robots à roues sont classés en différentes catégories en fonction de la position et du nombre de roues utilisées. Dans ce contexte, nous présenterons les quatre principales classes de robots à roues, [4] :

I.7.3.1- Robot uni-cycle

Un robot de type uni-cycle est propulsé par deux roues distinctes, et peut également comporter des roues libres pour assurer sa stabilité. Son point central de rotation se trouve sur l'axe qui relie les deux roues motrices.

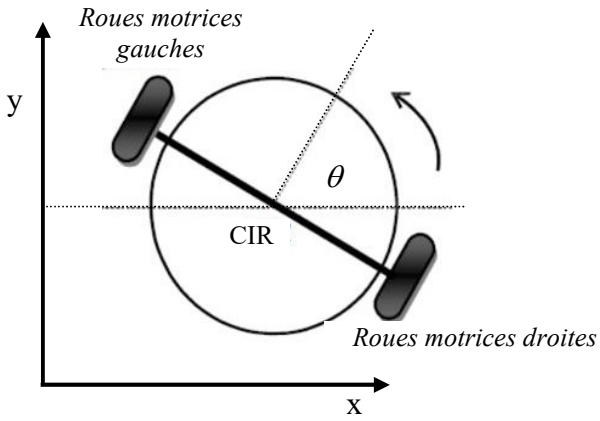


Figure (I-12) : *Robot mobile de type uni-cycle.*

I.7.3.2- Robot tricycle

Un robot de type tricycle se compose de deux roues fixes positionnées sur un même axe, ainsi que d'une roue centrale orientable placée sur l'axe longitudinal. Le mouvement du robot est déterminé par la vitesse des deux roues fixes et l'orientation de la roue orientable. Son point de rotation se trouve à l'intersection de l'axe des roues fixes et de l'axe de la roue orientable.

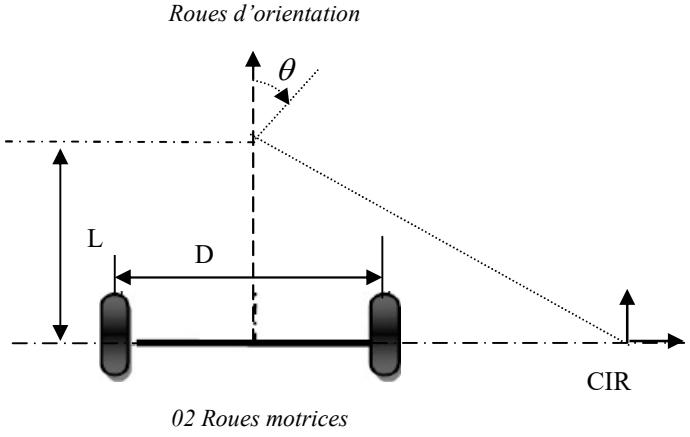


Figure (I-13) : *Robot mobile de type tricycle.*

I.7.3.3- Robot voiture

Un robot de type voiture présente des similarités avec le tricycle, car il se compose de deux roues fixes sur un même axe, ainsi que de deux roues centrales orientables également sur un même axe.

Cependant, le robot de type voiture est plus stable car il dispose d'un point d'appui supplémentaire. Toutes les autres caractéristiques du robot voiture sont identiques à celles du robot tricycle. En fait, le robot tricycle peut être ramené au robot voiture en remplaçant les deux roues avant par une seule roue placée au centre de l'axe, tout en maintenant le point de rotation inchangé.

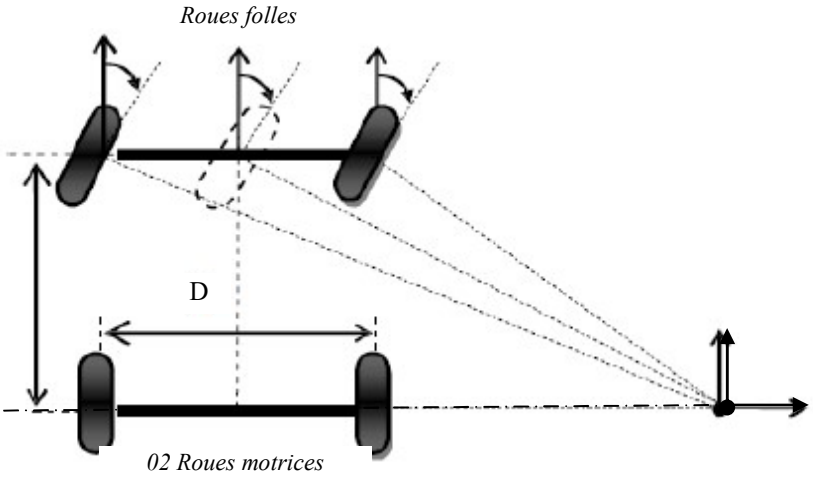


Figure (I-14) : *Robot mobile de type voiture.*

I.7.3.4- Robot omnidirectionnel

Un robot omnidirectionnel est capable de se déplacer librement dans toutes les directions. Il est généralement composé de trois roues décentrées orientables placées selon un agencement en triangle équilatéral.

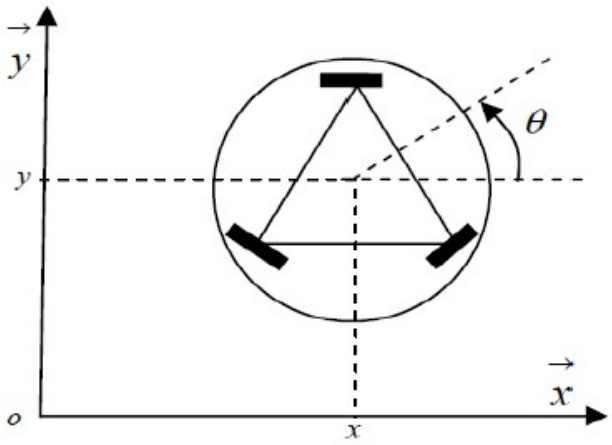


Figure (I-15) : *Robot omnidirectionnel.*

I.7.3.5- Comparaison des différents types

Le tableau ci-dessous présente un récapitulatif des avantages et des inconvénients des différents types de robots à roues.

Type du robot à roues	Avantages	Inconvénients
Uni-cycle	* Stable, * Rotation sur lui-même, * Complexité mécanique faible,	
Tricycle	*Complexité mécanique modérée	* Peu stable * Pas de rotation sur soi-même
Voiture	* Présente une stabilité notable, * La complexité mécanique est de niveau modéré	* Pas de rotation sur lui-même
Omnidirectionnel	* Présente une stabilité notable, * Capable de rotation sur place,	* Complexité mécanique importante

Tableau (I-1) : *Avantages et Inconvénients des différents types de robots à roues.*

I.8- Architecture des robots mobiles

L'architecture des robots mobiles comprend quatre composants essentiels :

- La structure mécanique et la propulsion.
- Le système de positionnement.
- Les dispositifs de sécurité.
- Le système d'analyse des informations et la gestion des fonctions.

I.8.1- Composants matériels d'un robot mobile

Un robot mobile est essentiellement composé de composants matériels et logiciels. Parmi les composants matériels, on retrouve une plateforme mobile qui sert de base à tous les autres éléments tels que les capteurs, les actionneurs et une source d'énergie telle que des batteries.

D'autres parties, tels que des bras manipulateurs, peuvent être additionnés au robot pour des applications spécifiques.

I.8.2- Energie utilisée dans les robots

Les robots se déplacent grâce à des moteurs électriques, thermiques ou hydrauliques. L'utilisation courante de l'énergie électrique présente l'avantage d'une commande facile,

mais elle peut poser des défis en termes de transport et de génération. Différentes méthodes sont employées pour résoudre ces problématiques :

- L'énergie nécessaire peut être fournie par des batteries qui peuvent être rechargées de manière automatique ou manuelle, ou bien échangées avec d'autres lorsque leur charge est épuisée.
- Une autre méthode consiste à utiliser un groupe électrogène embarqué, mais cela entraîne l'inconvénient d'un poids élevé. Dans ce cas, l'énergie de base est généralement de nature thermique.
- Il est également possible d'utiliser un cordon ombilical pour alimenter le robot en énergie, mais cela réduit son autonomie.

L'énergie thermique est principalement utilisée par des véhicules à forte puissance, que ce soit pour la traction ou pour activer un compresseur hydraulique.

I.8.3- Organes de sécurité

En fonction de la tâche qui lui est assignée, un robot peut être amené à travailler à proximité du personnel. Il est donc impératif qu'il soit équipé de dispositifs assurant sa sécurité. Des capteurs sont positionnés tout autour du robot afin de détecter les obstacles sur une large zone. Deux types de capteurs sont utilisés : les capteurs de proximité pour détecter les obstacles avant une collision (ultrasons, hyperfréquences, infrarouges, etc.), et les capteurs de contact qui détectent une collision ou un choc avec l'environnement (contact électrique sur les pare-chocs, résistance variable, fibres optiques, etc.). Ces dispositifs agissent en redondance par rapport aux capteurs précédemment mentionnés.

I.9- Moyens de perception et de locomotion en robotique mobile

La perception joue un rôle crucial dans le domaine de la robotique. C'est à partir de ce concept que la structure d'un robot capable d'exécuter des tâches complexes ou d'évoluer dans un environnement inconnu ou mal connu est construite. L'élément fondamental du système de perception est le capteur, dont le rôle est de traduire les caractéristiques de l'environnement en données exploitables.

Il existe de nombreux moyens utilisés pour la perception de l'environnement. Nous pouvons citer :

- Les systèmes de vision globale.
- Les télémètres laser et ultrasonores.
- Les capteurs optiques et infrarouges.

- Les capteurs tactiles.

I.9.1- Capteurs

La commande des robots mobiles repose sur deux types d'informations essentielles : les informations proprioceptives et les informations extéroceptives. Dans un environnement encombré d'obstacles fixes ou mobiles (comme d'autres robots), le système de perception revêt une grande importance pour assurer la sécurité du robot. Dans ce chapitre, nous nous concentrerons spécifiquement sur les capteurs pertinents pour la tâche de navigation [5].

I.9.1.1- Capteurs intéroceptifs

Les capteurs proprioceptifs fournissent des données sur l'état interne du robot, telles que la vitesse, la position et l'orientation. Ces informations permettent au robot de se renseigner sur son déplacement dans l'espace, c'est-à-dire sa localisation. Bien que ces capteurs puissent être utilisés directement, ils ont tendance à subir une dérive au fil du temps, ce qui limite leur efficacité en tant que capteurs uniques. Parmi les exemples de capteurs proprioceptifs, on peut citer l'odomètre, le radar Doppler et les systèmes inertiels.… [6].

I.9.1.2- Capteurs extéroceptifs

Les capteurs extéroceptifs ont pour but d'acquérir des informations sur l'environnement proche du robot. Ils fournissent des mesures caractéristiques de la position du robot dans son environnement en détectant les objets qui l'entourent. Ces informations peuvent prendre différentes formes. Parmi les exemples de capteurs extéroceptifs, on trouve les télémètres à ultrasons, infrarouges, lasers, les caméras, etc. Dans le contexte de la navigation autonome d'un robot mobile et en fonction de la mission à accomplir, on peut également utiliser les capteurs suivants, [7] :

a) Les capteurs à ultrasons : Les capteurs mentionnés ici se distinguent des autres types de capteurs car ils émettent une onde qui se propage généralement dans l'air. Cela entraîne une augmentation significative du temps d'acquisition des données, car la vitesse du son est bien plus lente que celle de la lumière. Cependant, la mesure par temps de vol est facile à réaliser avec une électronique simple. Il est important de noter que l'onde émise subit une atténuation notable dans cet environnement, ce qui limite les distances mesurées à des valeurs relativement courtes (quelques dizaines de mètres), mais cela reste suffisant pour une utilisation en intérieur.



Figure (I-16) : *Capteurs à ultrasons.*

b) Les capteurs à infrarouge : Ces capteurs sont utilisés de manière similaire aux lasers. Les mesures de distance peuvent être effectuées par triangulation, mesure d'intensité et mesure de temps de vol. De plus, il peut arriver que ces capteurs aient des angles d'ouverture importants lors de l'émission.



Figure (I-17) : *Capteurs à infrarouge.*

c) Les capteurs tactiles : Ces capteurs sont généralement utilisés pour des arrêts d'urgence lorsque le système de perception principal n'a pas détecté un obstacle. Ils sont conçus pour réagir rapidement et prendre des mesures appropriées pour éviter une collision ou minimiser les dommages potentiels.



Figure (I-18) : *Capteurs tactiles.*

d) Les boussoles : Ces capteurs utilisent la mesure du champ magnétique terrestre pour déterminer la direction du nord. Ils peuvent utiliser différentes technologies et présentent l'avantage de fournir une référence directionnelle stable au fil du temps.



Figure (I-19) : *Boussoles.*

e) Les balises : Dans certaines applications, il est possible d'utiliser des balises dont la position est connue pour faciliter la localisation du robot. Ces balises peuvent être facilement détectées par le robot, qui est équipé d'une antenne directionnelle lui permettant de déterminer la direction des différentes balises. En utilisant ces informations directionnelles, le robot peut déduire sa propre position.



Figure (I-20) : *Balises.*

f) Le GPS (Global Positioning System) : Le système mentionné est un système de balises universel où les balises sont positionnées sur des satellites en orbite terrestre. Ce système permet d'obtenir une mesure de position dans un référentiel global couvrant toute la Terre, avec une précision pouvant varier de quelques dizaines de mètres à quelques centimètres en fonction des équipements utilisés.



Figure (I-21) : *GPS.*

I.9.2- Actionneurs

Pour se déplacer et interagir avec son environnement, un robot est équipé d'actionneurs et peut être équipé d'un ou plusieurs moteurs pour faire tourner ses roues et effectuer des déplacements. Généralement, les roues du robot sont contrôlées par deux commandes motrices : la vitesse de déplacement et le taux de rotation. Habituellement, ces commandes

sont exprimées en mètres par seconde (m/s) pour la vitesse de déplacement et en degrés par seconde (deg/s) pour le taux de rotation. [8].

Le type de locomotion impose deux types de contraintes [9]:

- Les contraintes cinématiques, qui concernent la géométrie des mouvements possibles du robot.
- Les contraintes dynamiques, qui sont liées aux effets du mouvement (accélérations limitées, vitesses limitées, présence de forces d'inertie ou de frottement).

I.10- Conclusion

Dans ce chapitre, nous avons décrit une brève présentation sur les robots mobiles autonomes. Ces derniers peuvent être classés en plusieurs types selon le critère choisi : type de locomotion, degré d'autonomie, …etc. A cause de sa stabilité et sa faible complexité mécanique, le choix donc sera porté sur le type robot à roue uni-cycle. Afin de réaliser sa tâche, un robot mobile doit avoir un comportement intelligent et donc complexe. Les principaux algorithmes utilisés dans la navigation d'un chemin seront l'objectif du chapitre suivant.

Chapitre II :

Principaux Algorithmes Utilisés dans la Navigation d'un Chemin

Principaux Algorithmes Utilisés dans la Navigation d'un Chemin

II.1- Introduction

Les algorithmes de recherche les plus innovants en intelligence artificielle sont conçus pour des graphes arbitraires plutôt que pour des jeux basés sur un réseau. La difficulté principale réside dans le fait qu'il existe des aspects du comportement humain qui étaient auparavant difficiles à transposer pour une prise en charge informatique dans le but d'animer des personnages virtuels. Dans la suite, nous allons décrire plusieurs contributions et tentatives visant à résoudre le problème de recherche de chemins.

II.2- Solution du fainéant

Une approche initiale consiste à tracer une ligne droite entre les points de départ et d'arrivée, et à demander à l'acteur ou à l'agent de suivre cette ligne. Lorsqu'il rencontre un obstacle, il le contourne soit par la droite, soit par la gauche.

Cette méthode fonctionne assez bien pour les obstacles convexes, mais elle présente de nombreuses limites lorsqu'il s'agit d'obstacles concaves. Il est impensable de l'utiliser seule dans n'importe quel jeu respecté. Cependant, cet algorithme, combiné à un ou plusieurs autres, peut s'avérer très utile.

II.3- Algorithme de Dijkstra

L'algorithme de Dijkstra, proposé par Dijkstra en 1959, explore toutes les cases possibles en prenant en compte leurs coûts respectifs. Il fonctionne en parcourant les nœuds d'un graphe, en commençant par le point de départ de l'objet. L'objectif de l'algorithme de Dijkstra est de trouver le chemin le plus court entre deux nœuds d'un réseau.

Le principe de fonctionnement est assez similaire :

- L'acteur démarre du point de démarrage et calcule les poids des points voisins.
- Il traite le point ayant le poids le plus bas.
- Ensuite, il calcule les poids des points voisins et met éventuellement à jour leurs poids s'ils sont plus bas.
- Il passe ensuite au point ayant le poids le plus bas parmi tous les points.
- Ce processus se répète jusqu'à ce que tous les points aient été examinés.

Pour implémenter cet algorithme, on utilise une structure de file. Lorsqu'un point est ajouté à la file, il est placé en fonction de son poids calculé. Le prochain point à examiner est alors celui ayant le poids le plus bas, qui se trouve en tête de la file.

Cette fois-ci, le poids de chaque cellule est pris en compte, mais comme dans l'algorithme précédent, les directions sont considérées comme équivalentes.

Avantage

L'algorithme de Dijkstra assure la découverte du chemin le plus court entre le point de démarrage et le point d'arrivée, à condition qu'aucun des bords ne présente un coût négatif. Dans le courbe suivante, le carré rose représente le point de démarrage, le carré bleu représente le point d'arrivée, et les zones de couleur turquoise (éliminés) montrent les secteurs parcourus par l'algorithme de Dijkstra.

Les secteurs de couleur turquoise les plus clairs indiquent les zones les plus éloignées du point de départ et constituent ainsi la "frontière" de l'exploration, Figure (II-1).

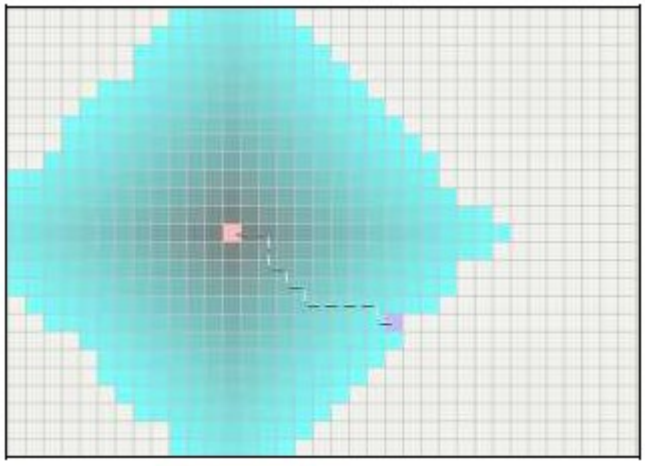


Figure (II-1) : *Algorithme de Dijkstra.*

II.4- Depth-First Search (DFS)

Cet algorithme est également connu sous le nom de "parcours en profondeur" en français. Il explore chaque chemin jusqu'à sa fin, en les parcourant un par un au fur et à mesure. Cependant, il n'est pas très efficace lorsque le nombre de chemins est élevé et que ces chemins sont longs. Il s'arrête dès qu'il trouve un chemin. (Cormen, 2001).

II.5- Première exploration itérative en profondeur (IDDFS)

Il s'agit d'une variante du parcours en profondeur appelée "parcours en profondeur itératif", (Demyen, 2007).

À la différence du parcours en profondeur classique, dans cette méthode, le nombre de cases explorées dans chaque chemin augmente à chaque itération. L'algorithme explore d'abord le premier chemin jusqu'à une certaine profondeur seuil, puis passe au deuxième chemin et ainsi de suite, en augmentant la profondeur seuil à chaque itération. Cela permet de

combiner les avantages du parcours en largeur et du parcours en profondeur. L'algorithme s'arrête dès qu'il trouve un chemin valide.

II.6- Recherche du meilleur premier choix (B-F-S)

Best First Search (Dechter et Pearl, 1985) A* et B* sont des algorithmes de recherche de chemin de type "Best First Search" (meilleure recherche d'abord). Ces algorithmes sont utilisés pour calculer le chemin le plus prometteur en utilisant une heuristique qui est une estimation de la distance jusqu'à la destination. Ils évaluent les nœuds à explorer en fonction d'une fonction de coût, qui est la somme du coût réel pour atteindre ce nœud depuis le départ et de l'estimation du coût restant pour atteindre la destination. L'algorithme A* utilise une heuristique admissible, c'est-à-dire une estimation qui ne surestime pas le coût réel, tandis que l'algorithme B* utilise une heuristique optimiste qui peut surestimer le coût réel. Ces algorithmes sont efficaces pour trouver rapidement un chemin optimal dans des espaces de recherche étendus.

II.7- B * (B Star)

B Star est un algorithme de type Best First Search (Berliner, 1979). Cet algorithme utilise une structure de données arborescente pour explorer tous les chemins possibles. Il se concentre ensuite sur la branche ayant le coût le plus faible. Si cette branche ne mène pas à la destination, l'algorithme choisit une autre branche ayant un coût relativement faible pour poursuivre l'exploration.

II.8- Recherche du meilleur premier (BFS Breadth First Search)

Il s'agit de l'algorithme de parcours en largeur avec une heuristique. Il explore tous les chemins à partir d'une seule case à chaque itération, mais avec une évaluation heuristique spécifique. Au lieu de sélectionner le nœud le plus proche du point de départ, il choisit le nœud le plus proche de l'objectif. L'algorithme continue jusqu'à ce qu'il trouve un chemin entre le point de départ et l'arrivée

Inconvénient

BFS ne garantit pas de trouver le plus court chemin.

Avantage

Cet algorithme converge beaucoup plus rapidement que celui proposé par Dijkstra.

Ceci est dû au fait qu'il emploie la fonction heuristique pour guider sa voie vers le but très vite.

Effectivement, dans le cas où le but se trouve au sud de la position de départ, l'algorithme BFS (parcours en largeur avec heuristique) aura tendance à privilégier les chemins orientés vers le sud. Cela peut être observé dans le diagramme où les nœuds jaunes représentent les chemins avec une valeur heuristique élevée (coût élevé pour atteindre le but), tandis que les nœuds noirs représentent les chemins avec une valeur heuristique basse (coût bas pour atteindre le but). Cette particularité permet à l'algorithme BFS de trouver des chemins rapidement, en comparaison avec l'algorithme de Dijkstra.

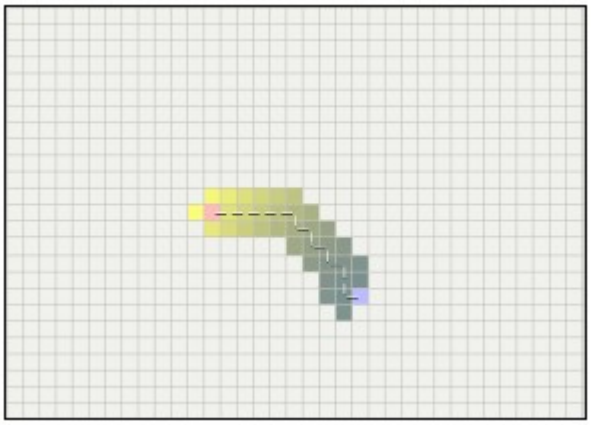


Figure (II-2) : *Recherche du premier mieux (BFS)*

Effectivement, dans des cas simples où la carte ne présente aucun obstacle et où le chemin le plus court est une ligne droite, les deux algorithmes peuvent donner des résultats similaires. Cependant, lorsque des obstacles concaves sont présents, l'algorithme de Dijkstra peut rencontrer des difficultés pour trouver le plus court chemin. En revanche, il garantit toujours de trouver le chemin le plus court, même dans des situations complexes. Cela signifie que même si l'algorithme de Dijkstra peut nécessiter plus de temps de calcul et d'efforts dans de tels scénarios, il fournit une solution optimale, figure (II-3).

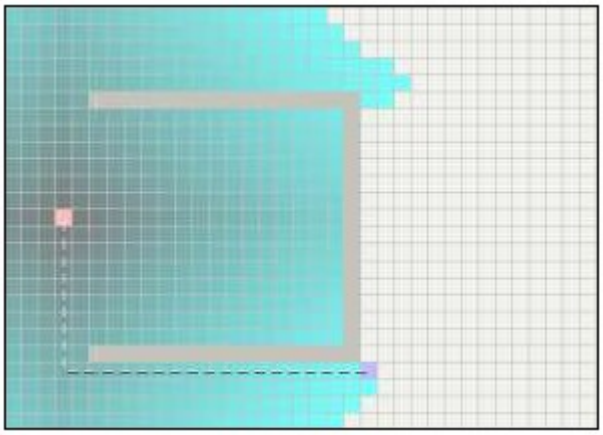


Figure (II-3) : *Dijkstra avec des obstacles concaves.*

Effectivement, l'algorithme BFS nécessite moins d'efforts en termes de calculs et de complexité, mais il ne garantit pas de trouver le meilleur chemin dans tous les cas. Dans des situations plus complexes, où des obstacles concaves sont présents, l'algorithme BFS peut donner un chemin qui n'est pas visiblement le plus court. Cela est dû à la nature de l'algorithme qui explore les chemins de manière plus uniforme, sans prendre en compte l'heuristique ou l'estimation de la distance jusqu'à l'arrivée.

Ainsi, l'algorithme BFS peut être plus rapide dans certains scénarios, mais il peut donner des résultats suboptimaux en termes de longueur du chemin. Par conséquent, dans des situations où la recherche du chemin le plus court est primordiale, l'algorithme de Dijkstra est préférable, malgré sa complexité accrue, figure (II-4).

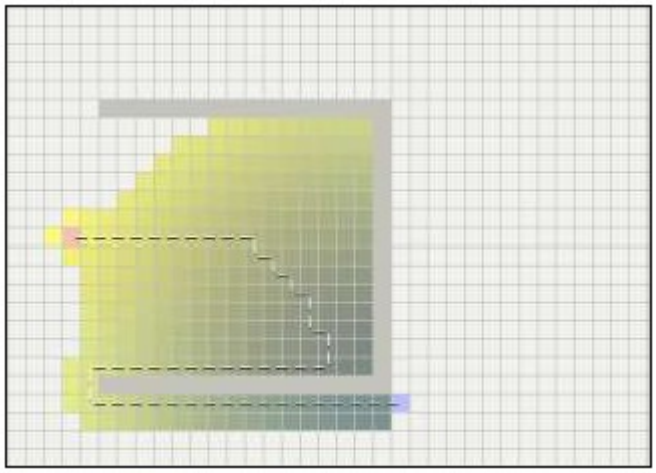


Figure (II-4) : *BFS avec des obstacles concaves.*

Effectivement, l'algorithme BFS peut être passionné dans sa recherche du chemin vers le but, ce qui peut conduire à des situations où il continue d'avancer sur un chemin malgré un coût élevé. Cela est dû à la nature de l'algorithme qui explore tous les chemins de manière égale sans prendre en compte le coût accumulé jusqu'à présent.

L'algorithme BFS se base uniquement sur l'exploration en largeur, en considérant les nœuds les plus proches du point de départ en premier, puis en s'étendant progressivement aux nœuds adjacents. Cette approche peut être rapide pour trouver un chemin vers le but, mais elle ne garantit pas d'obtenir le chemin le plus court en termes de coût total.

En revanche, l'algorithme de Dijkstra, qui prend en compte le coût accumulé jusqu'à chaque nœud, peut éviter cette situation où l'algorithme BFS continue d'avancer sur un chemin avec un coût élevé. Il évalue constamment et met à jour le chemin le plus court en fonction du coût accumulé, garantissant ainsi la recherche du chemin le plus court possible.

Il est donc important de prendre en considération ces différences entre les algorithmes BFS et Dijkstra lors du choix de l'algorithme de recherche de chemin en fonction des exigences spécifiques de la situation donnée.

II.9- L'algorithme A*(Hart, Nilsson et Raphael, 1968)

Effectivement, l'algorithme A* a été développé en 1968 pour combiner des approches heuristiques telles que l'algorithme BFS et des approches formelles comme l'algorithme de Dijkstra. L'objectif était de bénéficier des avantages des deux approches, en utilisant l'heuristique pour guider la recherche tout en garantissant la découverte du chemin le plus court.

Contrairement à l'algorithme BFS qui donne généralement une solution approximative sans garantie de trouver la meilleure réponse, l'algorithme A* est construit sur le sommet de l'heuristique. L'heuristique fournit une estimation du coût restant jusqu'au but, ce qui permet à l'algorithme A* de guider efficacement la recherche en priorisant les nœuds qui semblent les plus prometteurs.

Grâce à cette combinaison d'heuristique et d'évaluation précise du coût accumulé, l'algorithme A* peut garantir la découverte du chemin le plus court, sous réserve que certaines conditions soient respectées. Cependant, il est important de noter que la qualité de l'heuristique utilisée peut avoir un impact significatif sur les performances de l'algorithme A* et la qualité des résultats obtenus, [16-18].

En résumé, l'algorithme A* est conçu pour combiner les approches heuristiques et formelles, en utilisant l'heuristique pour guider la recherche et garantir le chemin le plus court. Il a été développé comme une amélioration de l'algorithme BFS, offrant une solution plus précise et fiable dans de nombreux problèmes de recherche de chemin, comme indique la figure (II-5).

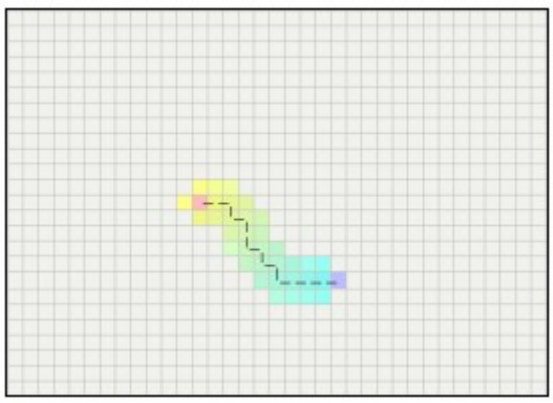


Figure (II-5) : *Recherche graphique dans un secteur sans obstacles due plus court chemin.*

Dans le cas d'un obstacle concave, l'algorithme A* est capable de trouver un chemin aussi bon que celui trouvé par l'algorithme de Dijkstra, voire meilleur dans certains cas. Cela est dû à l'utilisation de l'heuristique dans l'algorithme A*Figure (II-6):

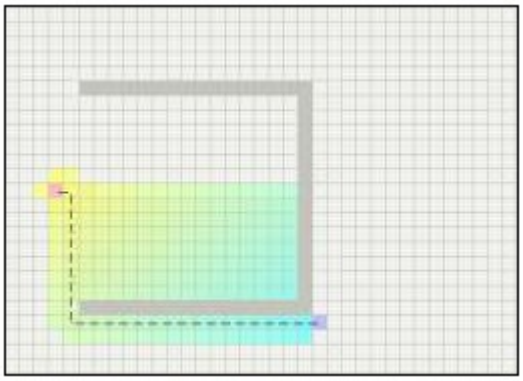


Figure (II-6) : *Algorithme A* avec un obstacle concave.*

L'algorithme A* qui est un algorithme de recherche de chemin. Il combine des morceaux d'information de l'algorithme de Dijkstra (favorisant le sommet qui est proche du point de départ) et de l'algorithme BFS (favorisant le sommet qui est proche du but). Cela permet à l'algorithme A* d'être plus efficace que les deux autres algorithmes.

Dans la terminologie standard utilisée dans un algorithme A*:

G (n) : représente le coût du chemin du point de départ à n'importe quel sommet n.

H (n) : représente le coût heuristique estimé du sommet n au but.

Dans les diagrammes suscités, *h* (en jaune) représente les sommets qui sont loin du but, et *g* (bleu) représente les sommets loin du point de départ.

L'algorithme A* équilibre les deux tout en se déplaçant du point de départ au but. A chaque pas de la boucle principale, il examine le sommet n qui possède le coût le plus bas.

$$F(n) = G(n) + H(n) \qquad \text{(II-1)}$$

II.9.1- Utilisation de la méthode heuristique A*

L'heuristique peut être utilisée pour contrôler le comportement de l'algorithme A* [19].

* A une extrême, si (h(n) = 0), alors seul *g(n)* joue un rôle, l'algorithme A* combiné à l'algorithme de Dijkstra, affirme l'existence du chemin le plus court.

* Si *h(n)* est toujours inférieur (ou égal) au coût de déplacement de *n* vers le but, on garantit donc que l'algorithme A* permet de trouver, lentement, le chemin le plus court.

* Si *h(n)* est exactement égal au coût de déplacement de n vers le but, alors l'algorithme A* suivra uniquement et très vite le meilleur chemin, et n'étendra jamais son domaine de recherche. Dans ce cas l'algorithme A* présente un comportement parfait.

* Si *h(n)* est parfois supérieur au coût de déplacement de n vers le but, dans ce cas on ne garantit pas que l'algorithme A* puisse trouver le plus court chemin, mais il peut converger plus rapidement.

* À l'autre extrême, si *h(n)* est très supérieur à *g(n)*, dans ce cas seul *h(n)* joue un rôle dans la convergence de l'algorithme A* par rapport à l'algorithme BFS.

La situation dans laquelle nous nous trouvons est intéressante car elle nous offre la possibilité de prendre des décisions et d'exploiter l'algorithme A* de différentes manières. Dans ce contexte, nous pouvons viser l'obtention rapide des chemins les plus courts. Si nous réduisons notre niveau d'exigence, nous continuerons à obtenir des chemins optimaux, mais peut-être à une vitesse réduite. En revanche, si nous élevons notre niveau d'exigence, nous abandonnerons la recherche des chemins les plus courts, mais l'algorithme A* convergera plus rapidement.

Dans le contexte d'un jeu, cette caractéristique de l'algorithme A* peut s'avérer extrêmement utile. Par exemple, il peut y avoir des situations où trouver le chemin optimal n'est pas crucial, et où un chemin satisfaisant est suffisant. Pour ajuster l'équilibre entre la fonction *g(n)* et la fonction *h(n)*, il est possible de modifier chacune de ces deux quantités. Cela permet de jouer sur les priorités et de trouver un compromis adapté aux besoins spécifiques de la situation.

II.9.2- Vitesse ou exactitude

L'adaptabilité de l'algorithme A* repose sur l'utilisation d'heuristiques et de fonctions de coût, qui peuvent être extrêmement utiles dans la conception de jeux. L'équilibre entre vitesse et précision peut être exploité pour créer des jeux plus rapides. Dans la plupart des jeux, il n'est pas nécessaire d'obtenir absolument le meilleur chemin entre deux points, il suffit d'avoir un chemin qui permet de rejoindre ces points en étant suffisamment proche.

II.9.3- Echelle

L'algorithme A* calcule $f(n) = g(n) + h(n)$. Pour sommer les deux valeurs, ils doivent être à la même échelle.

II.9.4- Heuristique exacte

Lorsque l'heuristique est égale à la distance réelle le long du chemin optimal, on peut observer que l'algorithme A* explore très peu de nœuds, comme illustré dans le diagramme présenté dans la section suivante. À l'intérieur de l'algorithme A*, la fonction *f(n)* = *g(n)* + *h(n)* est calculée pour chaque nœud. Lorsque *h(n)* correspond exactement à *g(n)*, la valeur de *f(n)* reste constante le long du chemin. Les nœuds qui ne sont pas sur le chemin optimal auront une valeur f plus élevée, [19].

II.9.5- Heuristique exacte pré-calculée

Une approche pour construire une heuristique exacte consiste à calculer à l'avance la longueur du chemin le plus court entre chaque paire de points. Cependant, cette méthode n'est pas réalisable pour la plupart des cartes de jeu. Heureusement, il existe des méthodes qui permettent d'obtenir une heuristique qui se rapproche de cette exactitude.

Il est possible d'ajouter à l'heuristique h une fonction qui calcule le coût pour se déplacer depuis n'importe quel emplacement jusqu'au prochain nœud du chemin. L'heuristique finale devient alors :

$$H(n) = h'(n, w_1) + \text{distance}(w_1, w_2), h'(w_1, but) \qquad \text{(II-2)}$$

Une autre option consiste à choisir une heuristique plus précise mais plus coûteuse, en évaluant toutes les paires w_1 et w_2 qui sont proches du nœud actuel et de l'objectif, respectivement

Heuristique linéaire exacte.

Dans des circonstances particulières, il est possible d'adopter une heuristique exacte sans effectuer de pré-calcul. Lorsque nous nous trouvons dans une situation sans obstacles ni terrain difficile, le chemin le plus court entre le point de départ et la destination doit être une ligne droite. En fin de compte, cette heuristique correspond à une heuristique exacte.

II.9.6- Heuristique pour des cartes de réseaux

Sur un réseau, il y a des fonctions heuristiques bien connues pouvant être employées, [4]

*** Distance de Manhattan**

La norme utilisée dans notre heuristique est la distance de Manhattan. Le coût de déplacement d'un espace à un espace voisin est représenté par *D*. Par conséquent, l'heuristique dans notre jeu est définie comme suit :

$$h(n) = D * \left(abs(x_n - x_{but}) + abs(y_n - y_{but}) \right) \qquad \text{(II-3)}$$

L'échelle qui doit être utilisée est celle qui correspond à la fonction du coût.

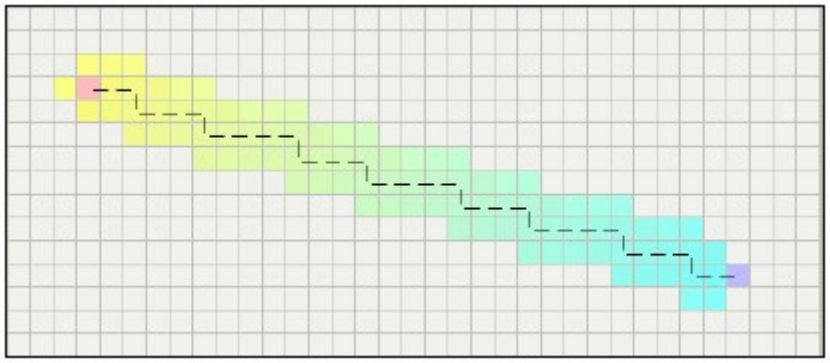


Figure (II-7) : *Distance du Manhattan.*

*** Distance diagonale**

Si nous autorisons les déplacements en diagonale sur notre carte, nous aurons besoin d'une heuristique différente. La distance du Manhattan pour (4 est, 4 nord) sera *8*D*. Cependant, nous pourrons simplement nous déplacer (4 nord-est) au lieu de cela, donc l'heuristique doit être *4*D*. Cette fonction prend en compte les déplacements en diagonale, en veillant à ce que les mouvements directs et les mouvements en diagonale aient un coût de *D* :

$$h(n) = D * Max\left(abs(x_n - x_{but}) + abs(y_n - y_{but})\right) \qquad \text{(II-4)}$$

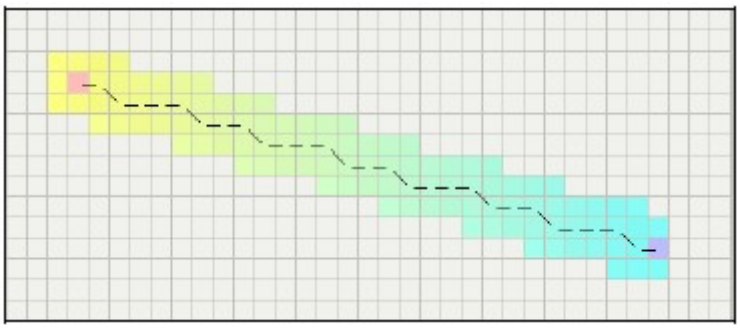


Figure (II-8) : *Distance diagonale*

Si le coût en diagonale du mouvement n'est pas *D*, mais une quantité semblable à $D_2 = \sqrt{2} * D$, ladite heuristique ne sera pas juste pour nous. Nous voudrions, au lieu de cela, quelque chose de plus sophistiqué :

$$h_{Diagonal}(n) = Minute\left(Abs(x_n - x_{but}) + Abs(y_n - y_{but})\right) \qquad \text{(II-5)}$$

$$h_{Straight}(n) = \left(Abs(x_n - x_{but}) + Abs(y_n - y_{but})\right) \qquad \text{(II-6)}$$

$$h(n) = D_2 * h_{Diagonal}(n) + D * h_{Straight}(n) - 2 * h_{Diagonal}(n) \qquad \text{(II-7)}$$

Ici, nous calculons:

h_diagonale (n) = le numéro de pas que nous pouvons prendre le long d'une diagonale.

h_straight (n) = la distance du Manhattan et combiner ensuite deux en considérant tous les pas diagonaux pour coûter à D_2 et ensuite tout le maintien directement marche (notez que c'est le numéro de droit intervient la distance du Manhattan, moins deux directement pas pour chaque pas diagonal que nous avons pris au lieu de cela) coûtent *D*.

*** Distance Euclidienne**

Si les unités peuvent se déplacer suivant un angle (au lieu des directions du réseau), on peut donc employer une distance de ligne droite: [20] :

$$h(n) = D * \sqrt{(x_n - x_{but})^2 + (y_n - y_{but})^2} \qquad \text{(II-8)}$$

Cependant, si c'est le cas, on peut avoir donc des difficultés pour l'utilisation directe de l'algorithme A* parce que la fonction du coût g ne correspondra plus à la fonction heuristique *h*.

De plus, pour cette méthode, la distance Euclidienne est plus courte que la distance de Manhattan ou la distance diagonale. On obtient toujours le chemin les plus courts, cependant l'algorithme A* prendra plus de temps à l'obtenir.

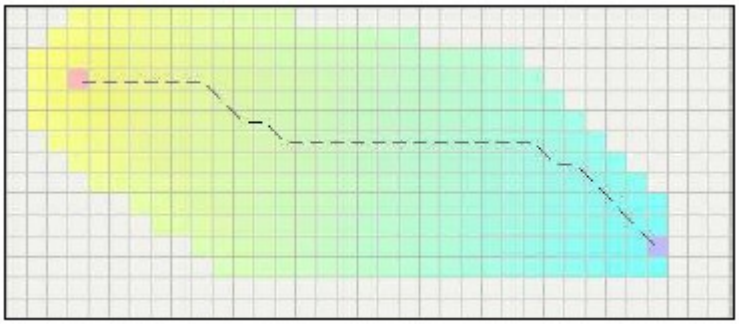


Figure (II-9) : *Distance Euclidienne*

*** Liens de casse**

Les connexions dans l'heuristique peuvent parfois entraîner une exécution inefficace. Lorsque plusieurs chemins ont la même valeur de *f*, tous ces chemins sont explorés alors qu'il suffirait d'en explorer un seul :

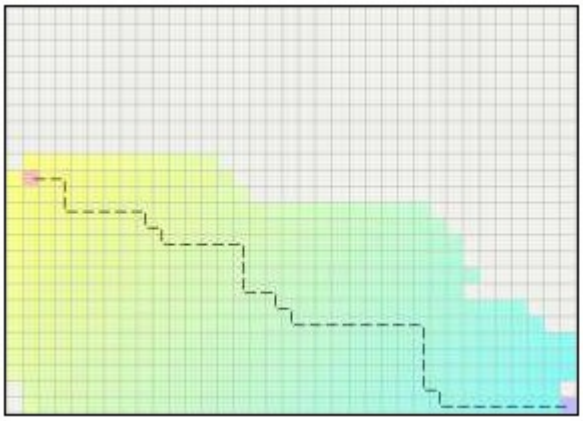


Figure (II-10) : *Liens de casse dans les valeurs de f.*

Pour résoudre ce problème, une solution consiste à introduire un petit facteur de correction au niveau de l'heuristique. Ce facteur de correction doit être déterminé en fonction du sommet et doit permettre d'obtenir des valeurs différentes pour *f*. Cela revient en fait à explorer les différentes possibilités de valeurs de *f*.

Une façon de rompre les liens est d'ajuster légèrement l'échelle de la fonction h. Si l'échelle est faible, cela fait en sorte que f augmente progressivement à mesure que l'on se rapproche de l'objectif. Cependant, cela signifie malheureusement que l'algorithme A* aura tendance à explorer les sommets proches du point de départ plutôt que ceux proches de l'objectif. Par conséquent, au lieu d'augmenter légèrement l'échelle de la fonction h (par exemple, de 0,1 %), l'algorithme A* préférera étendre les sommets proches de l'objectif.

Heuristique * = (1.0 + p).

Le facteur p doit être sélectionné de manière à : p < (coût minimum pour un pas) / (la longueur maximale de chemin attendue).

En supposant que nous ne nous attendons pas à ce que les chemins aient une longueur supérieure à 1000 pas, nous pouvons choisir p = 1/1000. Cette correction permet de rompre les liens et donne lieu à une nouvelle variante de l'algorithme A* qui explore une partie plus restreinte de l'espace de recherche par rapport aux variantes précédentes, comme illustré dans la figure (II.11).

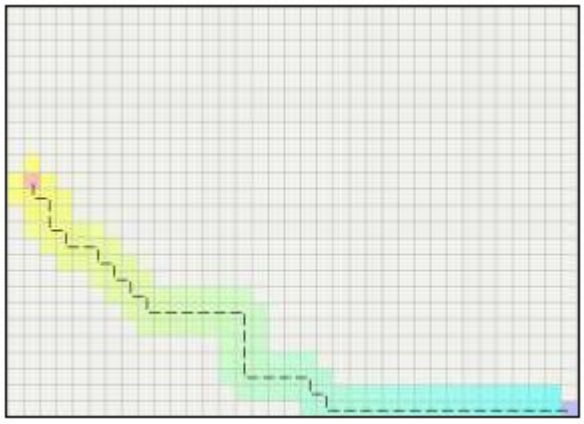


Figure (II-11) : *Graduation cassant lien supplémentaire à heuristique.*

Lorsqu'il y a des obstacles, l'algorithme doit toujours entreprendre une exploration pour trouver un chemin contournant ces obstacles. Cependant, il est important de noter qu'une fois l'obstacle contourné, l'algorithme A* explore un champ de possibilités plus limité, comme le montre la Figure (II-12).

Une façon différente de casser des liens est de préférer les chemins qui sont le long de la ligne droite du point de démarrage vers le but:

$$Dx_1 = x_{current} - x_{goal} \quad \text{(II-9)}$$

$$Dy_1 = y_{current} - y_{goal} \quad \text{(II-10)}$$

$$Dx_2 = x_{start} - x_{goal} \quad \text{(II-11)}$$

$$Dy_2 = y_{start} - y_{goal} \quad \text{(II-12)}$$

$$Croix = abs\ (dx_1 * dy_2 - dx_2 * dy_1) \quad \text{(II-13)}$$

$$Heuristique\ +\ =\ cross * 0.001 \quad \text{(II-14)}$$

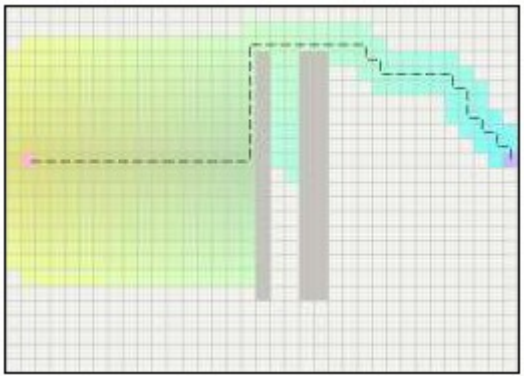


Figure (II-12) : *Graduation cassant lien supplémentaire heuristique travaille gentiment avec des obstacles.*

Le code calcule le produit vectoriel croisé entre le vecteur du point de départ à la destination et le vecteur du point actuel à la destination. Lorsque ces vecteurs ne sont pas alignés, le produit vectoriel croisé sera plus grand. En conséquence, ce code donnera une légère préférence à un chemin qui suit la ligne droite reliant le point de départ et la destination. Ainsi, lorsque aucun obstacle n'est présent, l'algorithme A* explore non seulement moins d'espace de recherche, mais génère également des chemins plus réalistes, comme illustré dans la Figure (II-13).

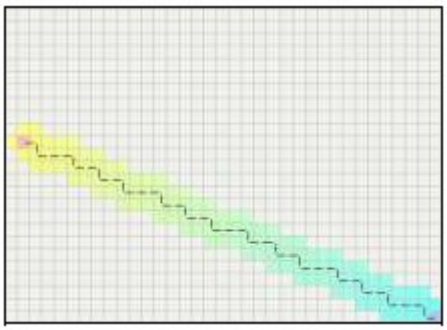


Figure (II-13) : *Produit mutuel cassant lien supplémentaire à .heuristique, produit de jolis chemins.*

Cependant, étant donné que cet interrupteur de lien favorise les chemins suivant la ligne droite entre le point de départ et la destination, des artefacts discrets peuvent apparaître lorsque l'acteur rencontre des obstacles, comme illustré dans la Figure (II-14) (notez que le chemin reste toujours optimal, bien que cela puisse sembler étrange).

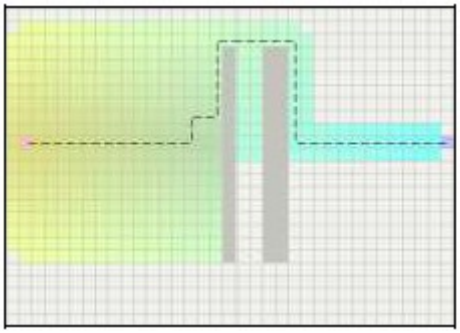


Figure (II-14) : *Produit mutuel cassant un lien supplémentaire à heuristique avec obstacles (artefact).*

Pour explorer l'amélioration de cet interrupteur de lien dans un mode interactif, vous pouvez utiliser l'applette de James Macgill A*. En sélectionnant le mode "Claire", la carte est nettoyée et deux points opposés aux coins de la carte sont choisis. Lorsque la méthode classique de l'algorithme A* est utilisée, les effets des liens deviennent évidents. En revanche, en utilisant la méthode "Fudge" (ou d'évitement), on observe l'impact supplémentaire du produit vectoriel sur l'heuristique.

Il existe également une autre approche pour rompre les liens, qui consiste à construire soigneusement la file d'attente de priorités de l'algorithme A* de manière à ce que les nouvelles insertions avec une valeur de f spécifique soient toujours classées comme étant meilleures (inférieures) que les anciennes insertions avec la même valeur de f. Cette méthode permet de rompre les liens et d'améliorer la recherche de chemins dans l'algorithme A*

*** Recherche d'un secteur**

Si notre objectif est de rechercher une tâche ou un endroit près de n'importe quel point d'arrivée plutôt qu'un espace spécifique, nous pouvons construire une heuristique alternative, notée $h'(x)$, qui est le minimum des valeurs de $h_1(x)$, $h_2(x)$, $h_3(x)$, etc., où h_1, h_2, h_3 représentent les heuristiques pour chaque tâche ou endroit voisin.

Cependant, une approche plus rapide consiste à laisser l'algorithme A* rechercher le centre du secteur d'arrivée. Une fois qu'un espace voisin de la zone de destination est atteint, nous pouvons arrêter l'algorithme et construire un chemin à partir de là.

II.10- Recherche de chemin

Le déplacement des humanoïdes à l'intérieur d'un environnement virtuel est souvent lié à l'objectif d'atteindre une position spécifique. Cela correspond au problème traité par la recherche de chemins, qui consiste à trouver un chemin depuis une position donnée vers une position souhaitée tout en évitant les obstacles statiques de l'environnement. Pour générer un chemin réaliste, il est préférable de rechercher un chemin qui minimise certains critères tels que la distance ou le coût énergétique. Bien que cette recherche d'un chemin optimal ne soit pas nécessairement alignée avec la navigation humaine [WM03], elle permet de générer des chemins plausibles en évitant les détours excessifs qui seraient moins réalistes. Deux grandes approches peuvent être distinguées :

- Les approches basées sur des graphes ;
- Les approches basées sur des champs de potentiel.

II.11- Conclusion

Dans ce chapitre, nous avons introduit le principe de la navigation par chemin et exploré les différents algorithmes disponibles pour la gestion de celle-ci, en particulier la recherche des plus courts chemins et des distances dans des graphes plats en exploitant la topologie spécifique de l'entrée graphique. Nous avons examiné les différentes étapes ainsi que les avantages et les inconvénients de chaque algorithme. L'algorithme A* tire parti des avantages de l'algorithme BFS et de l'algorithme de Dijkstra. Il ressemble à l'algorithme de Dijkstra car il peut être utilisé pour trouver le plus court chemin, et il ressemble également à l'algorithme BFS car il peut utiliser une heuristique pour guider sa recherche. Nous avons également présenté l'heuristique associée à l'algorithme A* et examiné les différentes variantes de cet algorithme. Selon les avantages de l'algorithme A*, son application à la Planification de la trajectoire d'une navigation dans un environnement sans et avec obstacles sera étudiée et simulée dans le chapitre suivant.

Chapitre III :

Planification de la Trajectoire par A Star (A*)

Planification de la Trajectoire par A star (A*)

III.1- Introduction

Pour trouver un chemin plus court et moins coûteux d'un point à un autre, l'algorithme A* utilise une approche simple et efficace pour ce genre du problème. A chaque itération, l'algorithme cherche à se rapprocher de la destination en privilégiant les possibilités qui sont directement plus proches de celle-ci, laissant de côté les autres.

Les possibilités qui ne permettent pas de se rapprocher de la destination sont mises de côté mais conservées dans une liste pour une exploration ultérieure, au cas où la solution actuellement explorée se révèle inefficace. En effet, il est impossible de savoir à l'avance si un chemin sera viable ou s'il sera effectivement le plus court. Si le chemin conduit à une impasse, cette solution devient inutilisable.

L'algorithme se dirige donc d'abord vers les chemins les plus directs. Si ces chemins ne mènent nulle part ou s'avèrent inefficaces par la suite, l'algorithme examine les solutions mises de côté. C'est grâce à cette capacité de revenir en arrière pour explorer les solutions mises de côté que l'algorithme garantit de toujours trouver une solution, si elle existe bien sûr.

Ainsi, on peut fournir à l'algorithme A* un terrain avec autant d'obstacles complexes qu'on le souhaite. S'il existe une solution, A* la trouvera, sans se laisser décourager par la présence d'obstacles ou la complexité du problème.

III.2- Déroulement de l'algorithme

Lorsqu'il s'agit d'explorer un graphe, notamment dans le domaine de la planification de trajectoires, il est assez simple d'utiliser la distance par rapport au nœud d'arrivée comme heuristique. L'heuristique $h(n)$ est une estimation du coût minimal pour atteindre le nœud d'arrivée à partir du nœud courant n. En connaissant le coût $g(n)$ entre le nœud de départ et le nœud courant n, nous obtenons donc une estimation du coût total de la solution, noté $f(n)$. La stratégie d'exploration de l'algorithme A* consiste alors à choisir le prochain nœud ayant le coût total estimé le plus bas, f(n).

$$f(n) = g(n) + h(n) \qquad \text{(III-1)}$$

où

$$h(n) = distance(n, nfinal) \qquad \text{(III-2)}$$

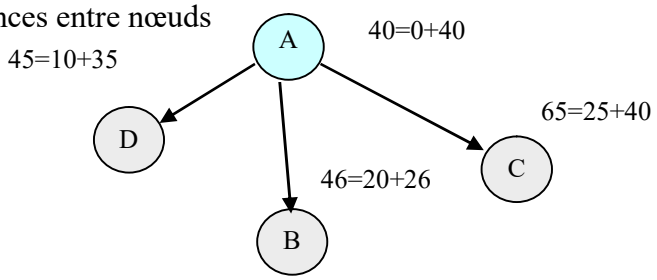
et

$$g(n) = \sum_{départ}^{n} cout(noeud_i, noeud_j) \qquad \text{(III-3)}$$

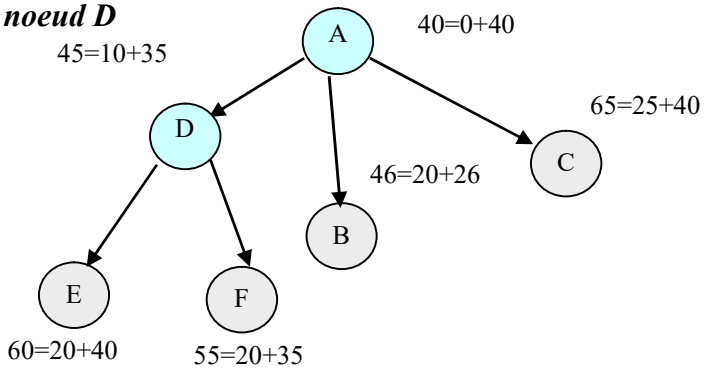
Le schéma graphique ci-dessus illustre le déroulement de l'algorithme A* lors de l'exploration des nœuds de A à I, où A est le nœud de départ et I est le nœud d'arrivée, [21].

III.2.1- Développement du nœud A

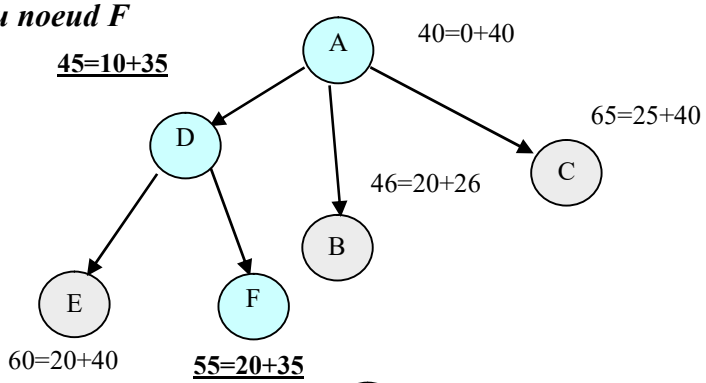
Donnants plusieurs distances entre nœuds



III.2.2- Développement du noeud D



III.2.3- Développement du noeud F



III.2.4- Arrivée au noeud I

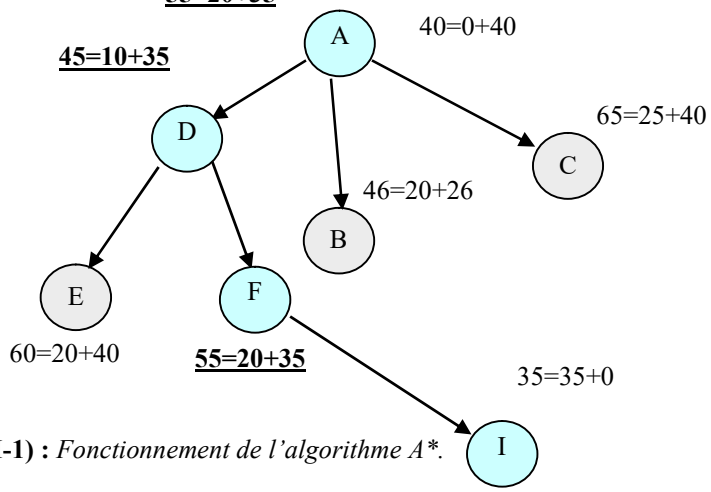


Figure (III-1) : *Fonctionnement de l'algorithme A*.*

Chaque fois qu'un nœud est développé, c'est-à-dire lorsqu'on examine les nœuds suivants disponibles à partir de ce nœud, notre arbre de recherche s'étend. Cependant, lors de l'exploration en graphe, il est important de prendre en compte la possibilité de répétition

d'états. Ainsi, lorsque nous atteignons un état déjà visité, nous conservons uniquement le chemin menant à ce nœud avec le coût le plus bas.

Comme mentionné précédemment, l'algorithme A* est optimal dans certaines conditions, que nous décrirons dans ce paragraphe. Dans le cas de l'exploration en graphe, la condition d'optimalité de A* est que l'heuristique doit être consistante (Russel, 1995). Une heuristique *h(n)* est consistante si, pour tout nœud n_1 et pour tout voisin n_2 de n_1, le coût estimé $h(n_1)$ du chemin entre n_1 et le nœud d'arrivée n'est pas supérieur à la somme du coût de l'étape entre n_1 et n_2 et du coût estimé $h(n_2)$ du chemin entre n_2 et le nœud d'arrivée.

$$h(n_1) \leq coût(n_1, n_2) + h(n_2) \qquad \text{(III-4)}$$

Cette condition peut être satisfaite en utilisant, par exemple, la distance cartésienne entre le nœud n_1 et le nœud d'arrivée comme heuristique :

$$h(n) = dist(n, n_{final}) = \sqrt{(x - x_{final})^2 + (y - y_{final})^2} \qquad \text{(III-5)}$$

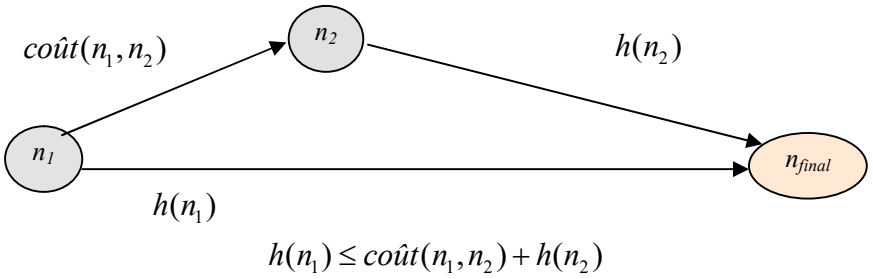


Figure (III-2) : *Consistance de l'exploration A*.*

Dans ce cas, l'inégalité triangulaire impose que :

$$dist(n_1, n_{final}) \leq dist(n_1, n_2) + dist(n_2, n_{final}) \qquad \text{(III-7)}$$

et vu que :

$$coût(n_1, n_2) \geq dist(n_1, n_2) \qquad \text{(III-8)}$$

Notre heuristique satisfait la condition de consistance comme illustré dans la Figure (III-2). Il serait facile de le vérifier expérimentalement, car la condition de consistance implique que la séquence de nœuds développés soit dans l'ordre croissant des valeurs de *f(n)*, comme on peut le voir dans la Figure (III-1).

Cela implique également que lorsque nous atteignons le nœud final pour la première fois, nous avons déjà atteint notre objectif avec le chemin optimal. Comme le montre la Figure (III-3), supposons que le nœud I ait déjà été parmi les voisins du nœud A. À ce stade, l'algorithme d'exploration ne se serait pas arrêté, car nous avons encore des nœuds avec des estimations meilleures [22].

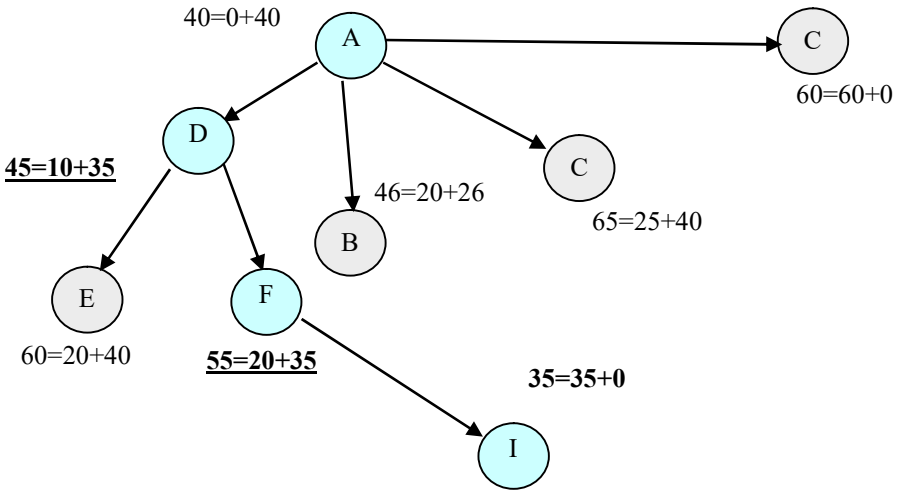


Figure (III-3) *: Illustration de l'optimalité de A*.*

III.3- Description de l'algorithme

*III.3.1- Les listes A**

A* utilise deux listes, ces listes contiennent des points. Pour être plus général, on peut même dire que ces listes comportent des noeuds d'une courbe, lequel représentant notre environnement.

La première liste, connue sous le nom de "liste ouverte", contient tous les nœuds qui ont été examinés. Lorsque l'algorithme se penche sur un nœud du graphe, il est ajouté à la liste ouverte s'il n'y est pas déjà présent.

La deuxième liste, appelée "liste fermée", contient tous les nœuds qui ont été considérés à un moment donné comme faisant partie du chemin solution. Avant d'être ajouté à la liste fermée, un nœud doit d'abord passer par la liste ouverte. Il doit être examiné avant d'être considéré comme une option valide.

Nous avons une idée générale du chemin solution, en sachant que le nœud voisin du nœud actuel est susceptible de nous rapprocher de la solution. Nous examinons donc chaque nœud voisin du nœud courant pour déterminer celui qui a le plus de chances de faire partie du chemin solution.

La recherche du chemin débute par le premier point, en examinant tous ses voisins, en calculant leur qualité, et en sélectionnant le meilleur pour continuer. Chaque nœud examiné est ajouté à la liste ouverte, et le meilleur nœud de cette liste est déplacé vers la liste fermée. Ce nœud devient la référence pour la recherche suivante.

Remarque :

Pour déterminer le meilleur point pour poursuivre le chemin, il ne suffit pas de chercher le meilleur parmi les voisins du point actuel, mais parmi tous les nœuds de la liste ouverte. C'est ainsi que nous pouvons abandonner un chemin qui semblait prometteur au départ mais qui ne l'est pas en réalité.

A chaque itération, nous examinons tous les nœuds qui ont été étudiés (et qui n'ont pas encore été sélectionnés) pour trouver celui qui a la meilleure qualité. Il est tout à fait possible que le meilleur nœud ne soit pas un voisin direct du nœud courant.

Cela signifie que le nœud courant nous éloigne de la solution et qu'il doit être corrigé. L'algorithme s'arrête lorsque la destination est atteinte ou lorsque toutes les solutions mises de côté ont été examinées et qu'aucune d'entre elles ne s'est avérée satisfaisante, ce qui correspond au cas où il n'y a pas de solution.

III.3.2- Détermination des noeuds voisins

Si nous considérons notre domaine de recherche comme une carte, les points adjacents à prendre en compte sont ceux qui se trouvent en haut, en bas, à gauche, à droite et en diagonale, à condition qu'ils ne contiennent pas d'obstacles infranchissables tels que des murs, des montagnes ou des rivières.

Avant d'évaluer la qualité de chaque nœud adjacent, il est important de ne considérer que ceux qui sont réellement utilisables. De plus, il faut exclure les nœuds qui sont déjà présents dans la liste ouverte ou dans la liste fermée. Pour être plus précis, un point ne doit pas être pris s'il se trouve déjà dans la liste ouverte, à moins qu'il ne présente une meilleure qualité, auquel cas nous mettrons à jour la liste ouverte.

Voici ce qu'il faut vérifier pour chaque nœud voisin potentiel :

- Est-ce un obstacle ? Si oui, nous ignorons ce nœud.
- Est-il dans la liste fermée ? Si oui, cela signifie que ce nœud a déjà été étudié ou est en cours d'étude, donc nous n'effectuons aucune action.
- Est-il dans la liste ouverte ? Si oui, nous calculons la qualité de ce nœud et, si elle est meilleure que celle de son homologue (ayant la même position dans la grille) dans la liste ouverte, nous mettons à jour les valeurs de $f(n)$ et le parent du nœud correspondant dans la liste ouverte.
- S'il n'est pas dans la liste ouverte, nous l'ajoutons à cette liste et calculons sa qualité $f(n)$.

III.3.3- Notion de parent d'un nœud

Chaque nœud possède un parent qui représente le chemin optimal jusqu'à ce nœud. Le parent joue un rôle crucial à la fin de l'algorithme pour retrouver le chemin solution. Il est également essentiel lors de la mise à jour d'un nœud dans la liste ouverte. Lorsque nous mettons à jour la qualité d'un nœud dans la liste ouverte, nous devons également mettre à jour son parent afin de spécifier clairement quel parent permet d'obtenir cette qualité améliorée.

Une fois que la destination a été atteinte, il est nécessaire de retracer le chemin en suivant les parents des nœuds présents dans la liste fermée. Nous remontons ainsi la chaîne des parents jusqu'à atteindre le point de départ.

III.3.4- Récapitulation des étapes

- Nous commençons par le nœud de départ, qui devient le nœud courant.
- Nous examinons tous les nœuds voisins du nœud courant.
- Si un nœud voisin est un obstacle, nous l'ignorons.
- Si un nœud voisin est déjà dans la liste fermée, nous l'ignorons.
- Si un nœud voisin est déjà dans la liste ouverte, nous mettons à jour la liste ouverte si le nœud correspondant a une qualité inférieure (et nous n'oublions pas de mettre à jour son parent).
- Sinon, nous ajoutons le nœud voisin dans la liste ouverte avec le nœud courant comme parent.
- Nous recherchons le meilleur nœud dans la liste ouverte. Si la liste ouverte est vide, cela signifie qu'il n'y a pas de solution, et nous terminons l'algorithme.
- Nous déplaçons ce nœud dans la liste fermée et le supprimons de la liste ouverte.
- Nous répétons le processus avec ce nœud comme nœud courant jusqu'à ce que le nœud courant soit le nœud de destination.

Voici une représentation d'une itération de l'algorithme. Notre objectif est de passer du point orange au point bleu. Les nœuds voisins du nœud orange sont représentés par les cases marquées en vert, et ils sont ajoutés à la liste ouverte. Pour chaque nœud voisin, nous calculons les coûts G et H pour atteindre le nœud orange et pour atteindre la destination. Dans cet exemple, j'ai utilisé la distance à vol d'oiseau comme heuristique. La case qui présente le coût total le plus bas est la case verte en dessous, donc elle est déplacée dans la liste fermée. L'algorithme va maintenant itérer à partir de cette case.

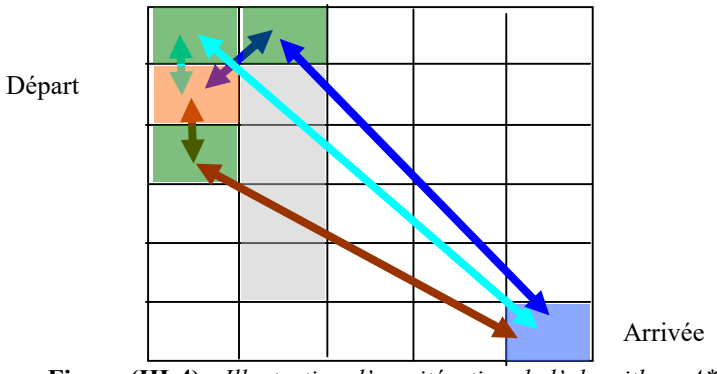


Figure (III-4) *: Illustration d'une itération de l'algorithme A*.*

III.4- Propriétés de A*

– ***Complétude :*** À moins qu'il y est une infinité de noeuds.

– ***Complexité de temps :*** Exponentielle, Selon la longueur de la solution.

– ***Complexité en espace :*** Exponentielle (1- Selon la longueur de la solution. 2- Elle garde tous les noeuds en mémoire).

– ***Optimale :*** il demande énormément d'espace mémoire de travail que de temps de calcul, [23].

III.5- Organigramme de simulation

III.5.1- Organigramme de simulation

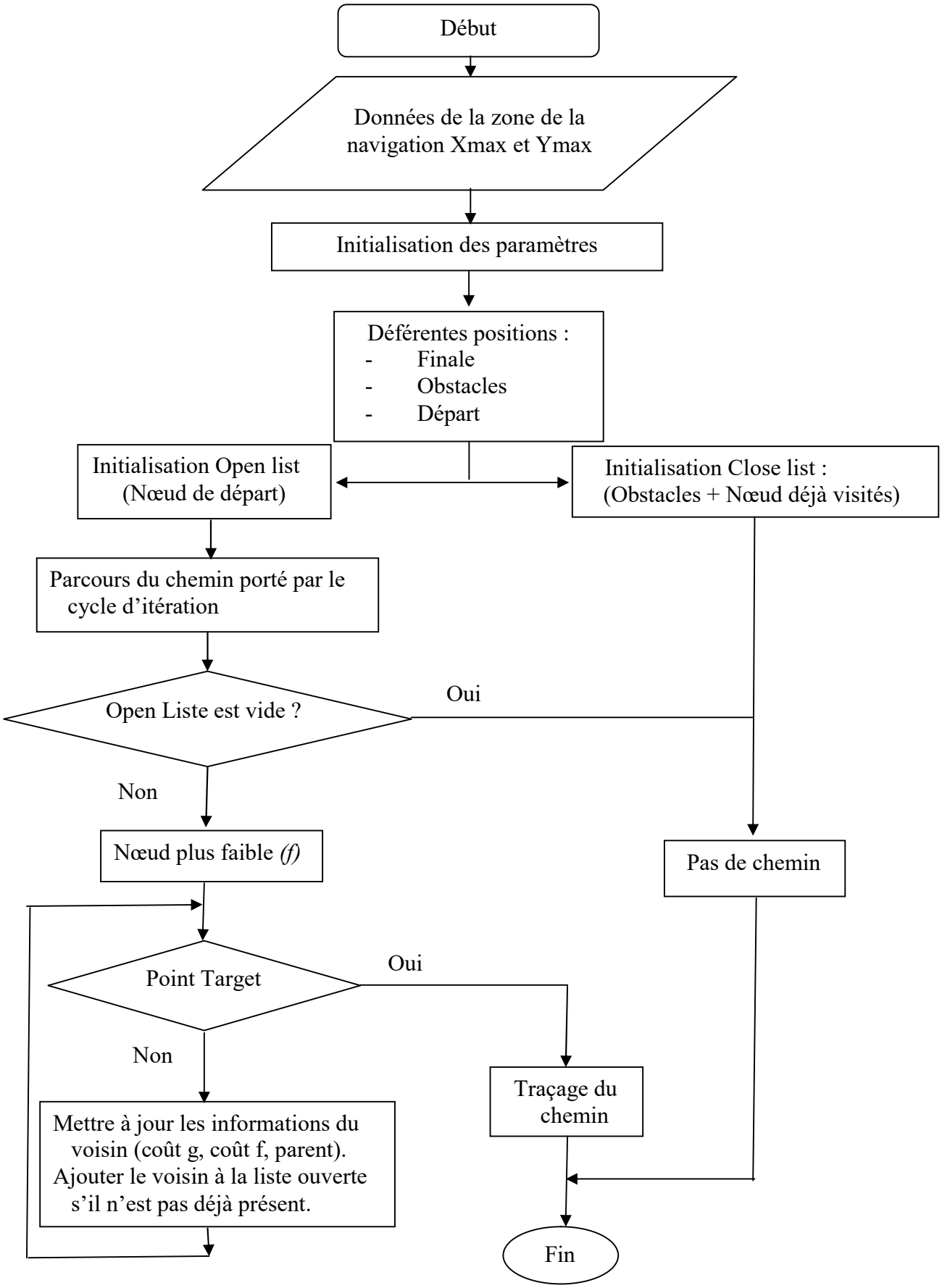


Figure (III-5) *: Organigramme du fonctionnement du programme d'Algorithme A*.*

III.5.2- Présentation des différentes fonctions

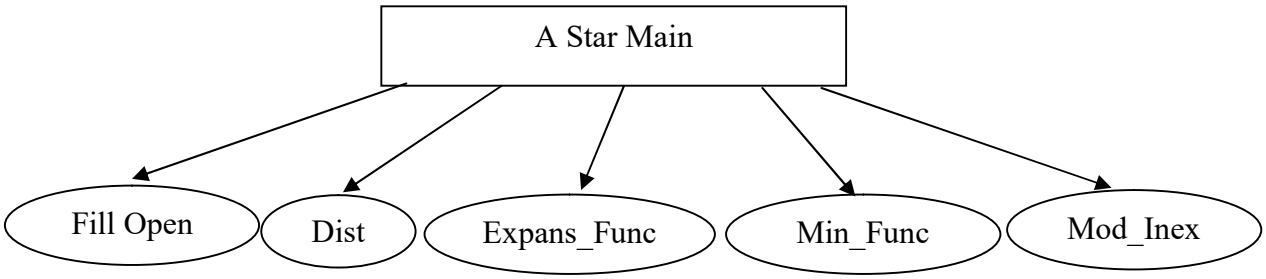


Figure (III-6) : *Fonctions de A*.*

III.5.2.1- La fonction "fill open list" (ou "remplir la liste ouverte") : [23-24]

La fonction "fill open list" est une fonction clé de l'algorithme A* (A-star) qui est utilisée pour mettre à jour la liste ouverte des nœuds qui ont été générés mais qui n'ont pas encore été visités.

L'algorithme A* utilise une stratégie de recherche de chemin qui consiste à explorer les nœuds voisins du nœud courant en fonction de leur coût total estimé, qui est calculé en combinant la distance traversée depuis le nœud de départ avec la distance restante jusqu'à la destination (heuristique). Les nœuds sont stockés dans une liste ouverte, qui est triée par ordre croissant de leur coût total estimé.

La fonction "fill open list" est utilisée pour ajouter de nouveaux nœuds dans la liste ouverte et pour mettre à jour les nœuds existants lorsqu'un meilleur chemin est trouvé. Cette fonction utilise une technique appelée "open counter A*" pour gérer la liste ouverte.

L'open counter A* consiste à ajouter un compteur d'itérations à chaque nœud lorsqu'il est généré. Ce compteur est ensuite utilisé pour résoudre les conflits de priorité qui peuvent se produire lors de la mise à jour de la liste ouverte. Lorsqu'un nœud est ajouté à la liste ouverte, son compteur d'itérations est incrémenté. Si deux nœuds ont le même coût total estimé, le nœud avec le compteur d'itérations le plus petit est choisi en premier.

Le rôle de la fonction "fill open list" avec open counter A* est de trier les nœuds dans la liste ouverte en fonction de leur coût total estimé et de leur compteur d'itérations. La fonction prend en entrée la liste ouverte, le nœud courant et les nœuds voisins du nœud courant, et renvoie la liste ouverte mise à jour.

Pour chaque nœud voisin, la fonction calcule son coût total estimé et son compteur d'itérations. Si le nœud voisin est déjà présent dans la liste ouverte, la fonction compare le

coût total estimé et le compteur d'itérations du nœud existant avec ceux du nœud voisin. Si le nœud voisin a un coût total estimé inférieur et/ou un compteur d'itérations plus petit, la fonction met à jour le nœud existant avec les nouveaux coûts et le nouveau compteur d'itérations.

Si le nœud adjacent n'est pas présent dans la liste ouverte, la fonction additionne le nœud à la liste avec les coûts et le compteur d'itérations calculés.

En somme, la fonction "fill open list" avec open counter A* est une méthode efficace pour gérer la liste ouverte dans l'algorithme A*. En utilisant cette technique, l'algorithme est en mesure de résoudre les conflits de priorité de manière efficace et de trouver rapidement le parcours le plus court entre le nœud de départ et le nœud cible.

III.5.2.2- La fonction distance A* (ou "heuristique") [23-24]

La fonction distance A* est une fonction qui est utilisée dans l'algorithme de recherche de chemin A* (A-star) pour estimer la distance restante entre un nœud donné et le nœud cible dans le graphe.

Le rôle de la fonction distance A* dans l'algorithme A* est de fournir une approximation heuristique de la distance entre le nœud courant et le nœud cible. Cette estimation heuristique est ensuite utilisée pour évaluer la priorité de chaque nœud dans la liste ouverte, afin de décider lequel explorer en premier.

Plus précisément, la fonction distance A* effectue les étapes suivantes :

1- Récupère les coordonnées du nœud courant.

2- Récupère les coordonnées du nœud cible.

3- Calcule la distance euclidienne entre les deux nœuds à l'aide de la formule mathématique suivante :

$$Dist = \sqrt{\left(x_{cible} - x_{cournt}\right)^2 + \left(y_{cible} - y_{cournt}\right)^2} \qquad \text{(III-9)}$$

4- Retourne la distance heuristique calculée.

La fonction distance A* doit être admissible, c'est-à-dire qu'elle doit toujours fournir une approximation de la distance restante inférieure ou égale à la distance réelle restante. Si la fonction distance A* est admissible, l'algorithme A* garantit de trouver le chemin le plus court entre le nœud de départ et le nœud cible.

En somme, le rôle de la fonction distance A* est crucial dans l'algorithme A*, car elle permet de guider la recherche vers les nœuds les plus prometteurs, c'est-à-dire ceux qui sont susceptibles de mener rapidement à la destination finale tout en minimisant le coût total.

III.5.2.3- La fonction "expand FNC" [23-24]

La fonction "expand FNC" est une étape importante de l'algorithme A* (A-star) utilisé pour déterminer le trajet le plus court entre deux points sur une carte ou un graphe. Cette fonction a pour rôle d'étendre le nœud courant en générant tous les nœuds voisins et en mettant à jour les coûts associés.

Plus précisément, la fonction "expand FNC" effectue les étapes suivantes :

1. Récupère le nœud courant à partir de la liste ouverte.

2. Vérifie si le nœud courant est le nœud cible. Si oui, l'algorithme est terminé et le chemin optimal a été trouvé.

3. Marque le nœud courant comme visité et le supprime de la liste ouverte.

4. Utilise la fonction "fill open list" pour ajouter tous les nœuds voisins du nœud courant à la liste ouverte, en calculant le coût de déplacement et la distance heuristique pour chaque nœud.

5. Pour chaque nœud voisin, met à jour les coûts associés en utilisant la formule suivante :

$$g(n) = g(n_{parent}) + coût(n_{parent}, n) \qquad \text{(III-10)}$$

$$f(n) = g(n) + h(n) \qquad \text{(III-11)}$$

où :

* $g(n)$ est le coût pour arriver le nœud n à partir du nœud de départ.
* $g(n_{parent})$ est le coût pour arriver le nœud parent à partir du nœud de départ.
* $coût(n_{parent},n)$ est le coût de déplacement pour aller du nœud parent au nœud n.
* $h(n)$ est la distance heuristique entre le nœud n et le nœud cible.
* $f(n)$ est le coût total pour atteindre le nœud n en passant par le nœud parent.

6- Ajoute chaque nœud voisin à la liste ouverte avec son coût total mis à jour.

Cette fonction est appelée à chaque itération de l'algorithme A* pour explorer les nœuds les plus prometteurs à partir du nœud courant. Le rôle de la fonction "expand FNC" est donc de mettre à jour les coûts des nœuds visités et de générer les nœuds voisins pour continuer la recherche, tout en assurant que l'algorithme suit toujours la direction du chemin le plus court vers la destination.

III.5.2.4- La fonction "Min FNC" (ou "minimum FNC") [23-24]

La fonction "Min FNC" est une fonction utilisée dans l'algorithme A* (A-star) pour sélectionner le nœud avec la plus petite valeur de la fonction *f(n)* parmi tous les nœuds présents dans la liste ouverte.

La fonction *f(n)* est une fonction qui calcule le coût total d'un nœud donné, qui est la somme de deux éléments :

1. Le coût *g(n)* pour atteindre ce nœud depuis le nœud de départ, qui est accumulé en traversant chaque nœud parent.

2. La distance heuristique *h(n)* entre ce nœud et le nœud cible.

La fonction "Min FNC" est appelée à chaque itération de l'algorithme A* pour sélectionner le nœud avec le coût total minimum à partir de la liste ouverte.

Le rôle de la fonction "Min FNC" est de fournir un moyen efficace de trouver le nœud le plus prometteur à explorer en premier, en minimisant la somme de la distance parcourue jusqu'à ce nœud et de la distance restante jusqu'à la destination. Cela permet d'explorer les nœuds les plus proches de la destination en premier, tout en évitant d'explorer les nœuds qui ne sont pas pertinents pour la recherche.

La fonction "Min FNC" compare les valeurs de la fonction *f(n)* pour chaque nœud dans la liste ouverte et retourne le nœud avec la plus petite valeur. Si plusieurs nœuds ont la même valeur de *f(n)*, la fonction peut utiliser d'autres critères pour sélectionner le nœud le plus approprié, tels que la distance parcourue pour atteindre ce nœud (c'est-à-dire le coût *g(n)*) ou la distance restante jusqu'à la destination (c'est-à-dire la fonction heuristique *h(n)*).

En somme, la fonction "Min FNC" est cruciale pour l'efficacité et la rapidité de l'algorithme A*, car elle permet de sélectionner rapidement les nœuds les plus prometteurs à explorer en premier et de garantir que l'algorithme suit toujours la direction du chemin le plus court vers la destination.

III.5.2.5- La fonction "nod-index" (ou "index de nœud") [23-24]

La fonction "nod-index" est une fonction utilisée pour rechercher un nœud donné dans une liste de nœuds.

Lors de l'exécution de l'algorithme A* (A-star), les nœuds sont stockés dans une liste ouverte, qui contient tous les nœuds qui ont été générés mais qui n'ont pas encore été visités.

Cette liste ouverte peut contenir des milliers de nœuds, chacun ayant un identifiant unique qui le distingue des autres.

La fonction "nod-index" est appelée pour trouver le nœud dans la liste ouverte qui correspond à un nœud donné. La fonction prend en entrée la liste ouverte et l'identifiant du nœud à rechercher, et renvoie l'indice du nœud dans la liste. Si le nœud n'est pas trouvé dans la liste, la fonction renvoie une valeur spéciale pour indiquer que le nœud n'existe pas dans la liste.

Le rôle de la fonction "nod-index" est de faciliter la mise à jour des nœuds dans la liste ouverte pendant l'exécution de l'algorithme A*. En effet, lorsque l'algorithme explore les nœuds voisins d'un nœud donné, il peut générer des nœuds qui ont déjà été visités précédemment. Dans ce cas, il est essentiel de vérifier si le nœud nouvellement considéré à un coût total plus faible que l'ancien nœud et, le cas échéant, de mettre à jour le nœud existant avec les nouveaux coûts.

En utilisant la fonction "nod-index", il est facile de rechercher un nœud dans la liste ouverte et de mettre à jour ses coûts en utilisant son indice dans la liste. Cela permet d'optimiser la recherche et la mise à jour des nœuds, ce qui contribue à accélérer la convergence de l'algorithme et à améliorer la qualité de la solution trouvée.

III.6- Résultats de simulation

Dans cette partie, nous allons appliquer l'algorithme de planification A* pour une navigation d'une trajectoire dans plusieurs environnements et nous allons présenter le chemin optimal parcouru pour assurer le lissage du chemin.

L'algorithme A*, la position du départ, la position du but, la position des obstacles comme des données du problème et le chemin planifié défini par un ensemble de points comme résultats. La simulation a été faite par le logiciel Matlab R2014a (8.3.0.532), [26].

III.6.1- Expérience 1 : Environnement sans obstacles.

La figure (III-7) la montre navigation du point de départ au point final, le chemin sous forme une droite, qui confirme l'itinéraire plus court c'est une droite liée les deux points départ et arrivée, c'st un parcourt plus optimisé.

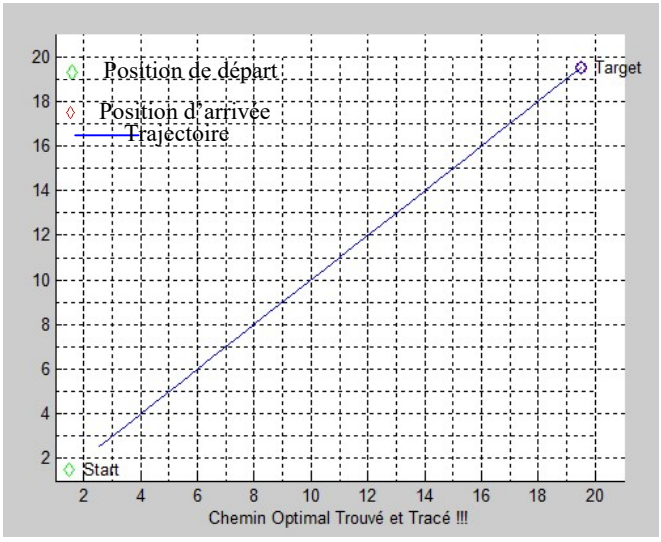
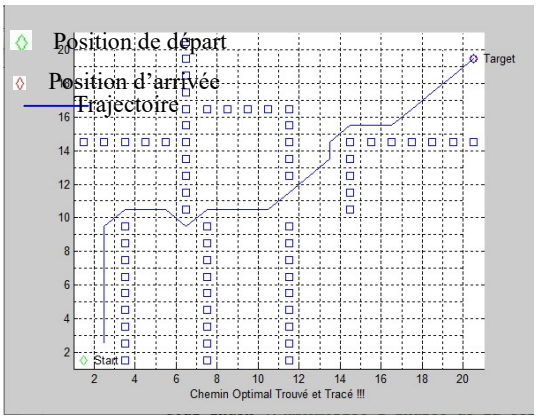
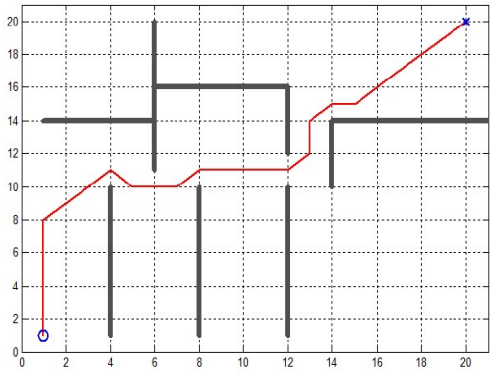


Figure (III-7) : *Trajectoire planifiée par A* dans un environnement sans obstacles.*

- Temps d'exécution : 0.025 s
- Longueur de chemin : 26.1725 um

III.6.2- Expérience 2 : Environnement contient des murs comme obstacles.



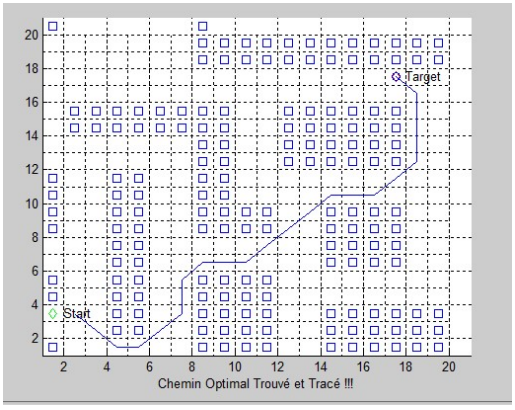


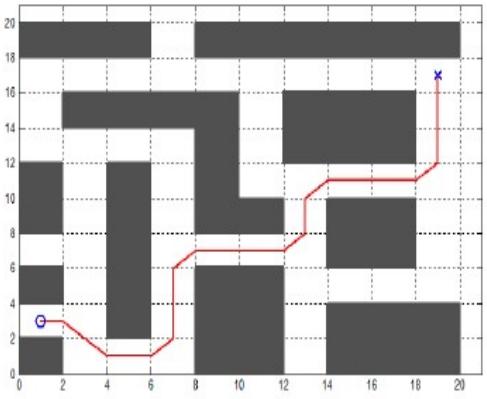
Notre simulation Réf [21]

Figure (III-8) : *Trajectoire planifiée par A* dans un environnement contient des murs comme obstacles.*

Dans cette expérience nous avons utilisé un environnement sous forme 2D comporte plusieurs obstacles proposés comme murs. Les résultats obtenus, figure (III-8) sont en accord avec ceux de la référence [21]. Qui confirme le bon fonctionnement de notre programme.

III.6.3- Expérience 3 : Robot mobile dans un labyrinthe.

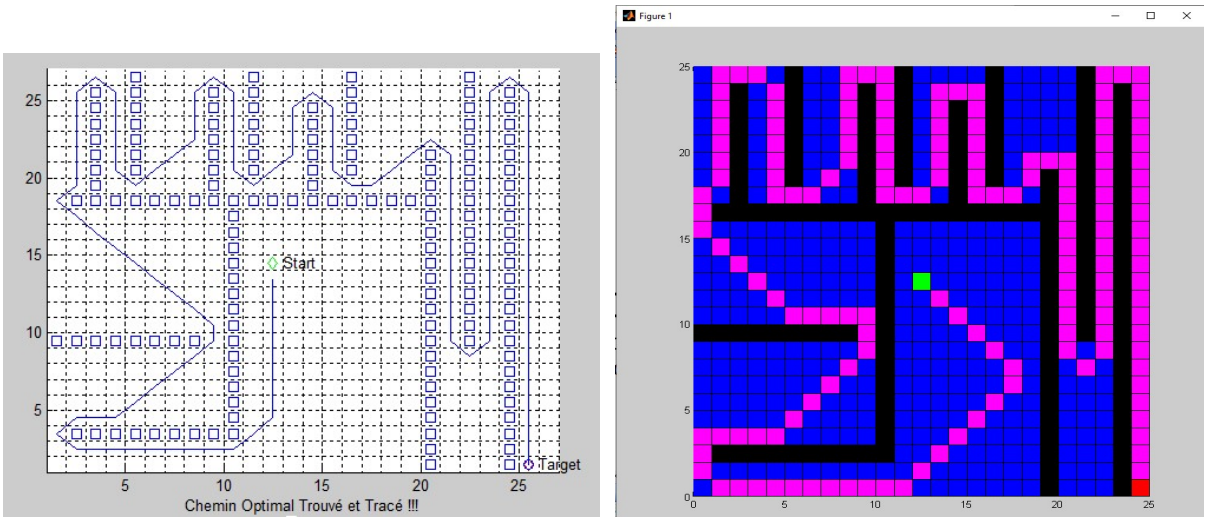




Notre simulation Réf.[21]

Figure (III-9) : *Trajectoire planifiée par A* dans un labyrinthe.*

Durant ce test nous avons utilisé un environnement sous forme 2D « un labyrinthe » comporte plusieurs obstacles proposés comme murs. Les résultats obtenus, figure (III-9) sont en accord avec ceux de la référence [21]. Qui confirme le bon fonctionnement de notre programme.



Notre simulation Réf.[25]

Figure (III-10) : *Trajectoire planifiée par A* à travers obstacles quelconques.*

Ce dernier test présente une navigation dans un environnement sous forme 2D quelconque comporte plusieurs murs pour vérifier la validité de notre programme et faire une comparaison ave un code de la littérature Réf.[25], La figure (III-10) illustre les résultats obtenus.

III.7- Conclusion

Après avoir effectué des simulations, nous pouvons tirer les conclusions suivantes concernant l'algorithme A* :

- L'algorithme A* permet aux robots mobiles de planifier leurs trajectoires dans n'importe quel environnement en deux dimensions, quel que soit le degré de complexité (nombre et formes des obstacles).
- Cet algorithme se distingue par son temps d'exécution minimal et la qualité optimale des chemins générés, ce qui en fait une solution efficace pour ce type d'application. Par conséquent, A* a été choisi comme technique de planification préférée.

Conclusion Générale

Conclusion Générale

On peut définir la robotique comme un domaine regroupant des techniques et des études visant à concevoir des systèmes mécaniques, informatiques ou hybrides capables de remplacer les fonctions motrices, sensorielles et intellectuelles de l'homme.

En particulier, la robotique mobile coopérative est un domaine très actif sur le plan recherche et de développent destiné à l'industrie. La difficulté de travailler sur ce domaine réside dans son aspect pluridisciplinaire d'une part et distribué d'autre part. En effet, les systèmes multi-robots nécessitent des techniques très avancées d'intelligence artificielle.

L'objective de ce travail d'une étude de Utilisation de l'algorithme A* pour la planification et l'optimisation de trajectoire d'un robot mobile consiste en la poursuite de trajectoires d'un robot mobile dans un plan bidimensionnel comporte d'obstacles.

Nous avons commencé notre étude par une description succincte sur la robotique mobile. En effet, différents types de robots mobiles à roues, leurs structures, les classes des robots mobiles à roues ont été présentés. Ensuite, nous avons donné un rappelle succinct sur la modélisation cinématique et dynamique des robots mobiles notamment pour celui de type unicycle.

Dans le deuxième chapitre, nous avons exposé le principe de la navigation d'un chemin optimal, et les principaux algorithmes disponible dans la littérature. Cet outil permet de gérer la découverte des chemins les plus courts et des distances dans des environnements plats en exploitant la topologie spécifique de l'entrée graphique.

Nous avons décrits aussi, les différentes étapes, puis les avantages et les inconvénients de chacun de ces techniques. L'algorithme A* exploite des avantages de l'algorithme BFS (Breadth-First Search) et de l'algorithme de Dijkstra. Il partage des similitudes avec l'algorithme de Dijkstra, car il peut être utilisé pour trouver le chemin le plus court. Il partage également des similitudes avec l'algorithme BFS, car il utilise une heuristique pour guider sa recherche. Nous

avons présenté l'heuristique spécifique à l'algorithme A* ainsi que les différentes formes de cet algorithme.

Ensuite, la technique de l'algorithme A* a été détaillée et adaptée à l'établissement d'un programme de navigation d'un un environnement bidimensionnel sans et avec obstacles en utilisant le logiciel Matlab R2014a.

La performance de l'algorithme A* se reflète principalement dans la vitesse de planification du chemin et la robustesse du chemin planifié. Bien que les chercheurs aient effectué de nombreuses recherches sur l'algorithme A *.

En ce qui concerne le robot réel, le principe reste le même, sauf en ce qui concerne la transmission de données via la carte

En se basant sur les résultats obtenus, on peut en tirer la conclusion que le but initial de ce travail est atteint, car les ces résultats sont acceptables et sont en bon accord avec ceux de la littérature. Toutefois, de nombreuses actions peuvent faire suite à ce travail dans le but de d'améliorer le comportement des robots à travers les points suivants : -

- Affectation le programme de simulation à un seul robot à l'aide carte Arduino dans un environnement comportant plusieurs obstacles mobiles,
- Affectation le programme de simulation aux plusieurs robots à l'aide de cartes Arduino dans un environnement comportant plusieurs obstacles (étude statique et étude dynamique).
- Il est recommandé d'explorer les autres algorithmes pour résoudre le problème de planification.
- Lors de la planification, il est important de prendre en compte les dimensions réelles du robot ainsi que ses contraintes mécaniques (telles que la non-holonomie, etc.).

Références Bibliographiques

Références Bibliographiques

[1]- Lemond, J, P., "*La Robotique Mobile*", Editions Hermès, Paris, 2001.

[2]- Sutton, R, S., Barto, A, G., "*Reinforcement Learning: An Introduction*", MIT Press, Cambridge, MA, 1998.

[3]- Bayle, B., "*Robotique Mobile*", école Nationale Supérieure de Physique de Strasbourg, Université Louis Pasteur, 2007

[4]- Belkhadria Khemisti, "*Commande d'un Robot Mobile par Réseaux de Neurones Artificiels*", Mémoire de magister en électronique, Université El Hadj Lakhdar, Batna, 2013.

[5]- Shuzhi S. G., Lewis F. L., "*Autonomous Mobile Robots, Sensing, Control, Decision, Making and Applications*", Taylor and Francis Group, 2006.

[6]- Filliat D., "*Robotique Mobile*", Cours à l'école Nationale Supérieur des Techniques Avancées ENSTA, Paris, 2004.

[7]- Laurin E., "*Système Intelligent d'Assistance à la Perception dans la Conduite de Véhicule*", Thèse de Doctorat, Université de Sherbrooke, 2000.

[8]- Beaudry E., "*Planification de tâches pour un robot mobile autonome*", Thèse Maître ès sciences (M.Sc.) Faculté des sciences, Université de Sherbrooke, Canada, 2006

[9]- Large F., "*Navigation autonome d'un robot mobile en environnement dynamique et incertain*", Thèse de doctorat, Institut National de recherche en informatique et en automatique, France, 2003.

[10]- A. Khemissat, "*Plateforme de prototypage rapide pour la robotique mobile : Application à un robot Auto-balancé*", Mémoire de Master, Université M'hamed Bougara-Boumerdes, 2016.

[11]- Z. Oussama et N.Med Islam "*Localisation et navigation d'un robot mobile par webcam*" Mémoire de Magister, Université Saad Dahlab de Blida 2014/2015.

[12]- Laëtitia Matignon, "*Introduction à la robotique*", support de cours Licence 1iere année GREYC-CNRS Université de Caen, France, 2011/2012.

[13]- WWW.usinenouvelle.COM

[14]- WWW.robot.gmc.ulaval.ca

[15]- WWW.david.filliat.divloper.com

[16]- S. Zertal, "*Evolution Artificielle d'une population de robots humanoïdes*" Mémoire de Magister en Informatique, Université Mohamed Khider Biskra, 2011.

[17]- Shakour, Marc, "*Conception Et Implémentation D'un Algorithme De Planification De Chemin Dans Un Jeu Vidéo Comportant Un Environnement Triangularisé*", Mémoire du Maîtrise En Informatique, Université Du Québec A Montréal, 2012

[18]- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.htm

[19]- http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

[20]- http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes,

[21]- H. Tazebinte & K. Lameche, "*Planification et optimisation de trajectoire d'un robot mobile par les algorithmes évolutionnaires*", Mémoire de master en Automatique, Ecole Nationale Polytechnique, Elharrach, Alger, 2012

[22] D. Garcia, "*Planification de trajectoire dans un atlas de cartes*", Mémoire de maîtrise des sciences appliquées (automatique et systèmes), Ecole Polytechnique de Montréal, CANADA, 2008.

[23]- Safia Kedad-Sidhoum, "*Plus courts chemins : Algorithme A**", cours d'algorithmique avancée, Laboratoire d'Informatique de Paris 6 (LIP6), 2005-2006.

[24]-Alexandru Rusu, "*Planification de chemin et navigation autonome pour un rover d'exploration planétaire*", Thèse de doctorat, Institut Supérieur de l'Aéronautique et de l'Espace (ISAE), 2014,

[25]- https://github.com/santolucito/Robot-Navigation/blob/master/RobotNavigation.java,

[26]- Logiciel Matlab R2014a (8.3.0.532).

ملخص :

الهدف من هذا العمل يتمثل في تطوير برنامج على Matlab والذي يسمح بتحديد المسار الأمثل في بيئة ثنائية الأبعاد في أقل وقت ممكن. تعتمد الطريقة على خوارزمية A (*) المستخدمة على نطاق واسع في الممارسة العملية وفقًا لمزاياها والنتائج التي تم الحصول عليها مقارنة بالخوارزميات الأخرى. تتوافق نتائج المحاكاة التي تم الحصول عليها مع تلك الموجودة في المراجع العلمية وتظهر قدرة هذه الخوارزمية على حل المشكلة المحددة.

الكلمات المفتاحية: خوارزمية (A *)، تحديد المسار الأمثل، تخطيط المسار.

Résumé :

L'objectif de ce travail consiste en l'élaboration de programme informatique sur le logiciel Matlab qui permet la détermination de la trajectoire optimale d'une navigation dans un environnement bidimensionnel dans un temps minimal. La méthode est basée sur l'algorithme A (*) star qui est très utilisée en pratique selon ses avantages et des résultats obtenus par rapport aux autres algorithmes. Les résultats de simulation obtenue sont en bon accord avec ceux de la littérature et montrent la capacité de cet algorithme de résoudre le problème donné.

Mots clés : Algorithme A star (A*), Navigation, Planification de trajectoire.

Abstract:

The objective of this work consists in the development of a computer program on Matlab software which allows the determination of the navigation optimal trajectory in a two-dimensional environment at minimal time. The method is based on the A (*) star algorithm which is widely used in practice according to its advantages and the results obtained compared to other algorithms. The simulation results obtained are in good agreement with those of the literature and show the ability of this algorithm to solve the given problem.

Keywords: A star (A*) algorithm, Navigation, Pathfindig.