République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Larbi Tbessi de Tebessa
Faculté des Sciences de l'ingénieur
Département d'Informatique

Mémoire

Présenté en vue de l'obtention du diplôme de Magistère en Informatique

Techniques de Navigation pour un Robot Mobile Autonome

Présenté Par Med Nadjib Zennir

Le jury est constitué de

Président :	Dr Salim Chikhi (Université de Constantine)
Rapporteur :	Pr Mohamed Benmohammed (Université de Constantine)
Examinateurs :	Dr Alaoua Chaoui (Université de Constantine)
	Dr Salah Merniz (Université de Constantine)

Table des matières

Table des figures

Liste des tableaux

Introduction

Dans sa quête permanente de perfectionnement, l'homme n'a de cesse de faire évoluer ses outils. Depuis ses premiers couteaux en silex, il montre sa redoutable qualité d'ingénieur. Le but ultime est évidemment de concevoir un outil qui pourrait le seconder ou le remplacer pour toutes les tâches ingrates et dangereuses voir même inaccessibles.

De nos jours, le mot "robot" désigne un dispositif mécanique qui peut accomplir de manière automatique des actions dangereuses ou pénibles pour l'homme. Étymologiquement, ce terme est d'origine slave désignant l'"esclave" ou le "travailleur dévoué", utilisé pour la première fois par l'écrivain tchécoslovaque Karel Capek dans sa pièce de théâtre R.U.R en 1920.

Cependant, c'est le visionnaire et auteur de romans de sciences-fictions Isaac Asimov qui, en 1950, dans son célèbre livre "I, Robot", fît connaitre le terme "robot" ainsi que le nom de la futur discipline "la robotique". C'est aussi dans cet ouvrage qu'Asimov fait, pour la première fois, référence aux fameuses "Trois lois de la robotique". Ces lois sont sensées réglementer les interactions hommes-robots dans une hypothétiques société future où les robots seront un fait dans le quotidien humain.

En attendant que le robot devienne l'égal de l'homme, beaucoup de problèmes restent à surmonter. Un de ces problèmes est la navigation autonome de ces robots dans des environnements dynamiques et incertains.

En effet, les capteurs généralement utilisés dans la robotique sont victimes de bruits parasitant les données. De ce fait, le robot se retrouve avec une cartographie le plus souvent infidèle à la réalité. La portée de ces mêmes capteurs pose aussi problème. Le robot ne possède qu'une petite partie de l'information concernant cet environnement. La problématique posée dans ce mémoire concerne la manière de concevoir des algorithmes ou des techniques capables de décider des actions du robot dans un environnement où il ne possède que des informations **partielles** et **bruitées**.

L'objectif de ce mémoire est de, premièrement, définir ce qu'est *un robot mobile autonome* à travers ses buts, ses constituants et de son environnement. Deuxièmement, définir le concept de **localisation** dans le domaine de la robotique et d'exposer différentes méthodes existantes (point de vue probabiliste, les filtres de Kalman, les

filtres particulaires et une approche par cartes de Kohonen). Troisièmement, définir le concept de **navigation**, dans un contexte d'environnement bruité et incertain, à travers l'exposé de différentes techniques existant dans la littérature (algorithmes génétiques, champs de potentiels, logiques floues, roadmaps et réseaux de neurones).

Notre objectif principal est de proposer deux méthodes de navigation de robot. L'une vise à utiliser les algorithmes génétiques mais sur des capteurs de portée locale contrairement aux modèles de la littérature qui travaillent sur des cartographies globales. La seconde méthode vise à utiliser les champs de potentiel munis d'équations de champs inédites plus proches de la réalité physique de ces phénomènes, nous visons aussi à résoudre le problème des minimas locaux propre à ces méthodes en utilisant la technique des entités virtuelles.

Le mémoire se divise en deux grandes parties. La première partie est un état de l'art du domaine de la robotique mobile autonome. Elle se divise en trois chapitres. Le premier est un état de l'art de la robotique "matérielle" puisque nous exposerons les différents capteurs et actionneurs utilisés dans la littérature. Le second chapitre traite de l'état de l'art de la localisation dans la robotique. Le troisième chapitre est un état de l'art des méthodes de navigation des robots mobile autonome dans des environnement incertains.

La seconde partie est la partie *"pratique"*. Elle est constituée de deux chapitres chacun exposant une proposition. Le premier chapitre présente notre proposition d'un algorithme de navigation utilisant les algorithmes génétiques pour un robot possédant une vue partielle de l'environnement. Ce chapitre contient aussi la phase de test de cet algorithme. Le second chapitre de cette partie présente notre seconde proposition d'algorithme de navigation basé sur les champs de potentiels, nous présenterons aussi la technique utilisée pour sortir le robot de minimas locaux, principal défaut des champs de potentiels, ainsi que des tests validant notre approche.

Le mémoire se termine par une conclusion générale qui discute les résultats obtenus, pose les critiques et les améliorations possibles et les perspectives envisageables.

Première partie

État de l'Art des techniques de localisation et de planification pour les robots mobiles autonomes

Chapitre 1

Quelques notions en robotique mobile autonome

Ce premier chapitre se propose, en premier lieu, de définir le robot mobile autonome dans ses différents aspects (capteurs, actionneurs, locomotion et environnement). Ensuite, à introduire la signification d'un environnement dynamique et incertain dans lequel évoluent généralement les RMA. Sera abordé aussi les différentes techniques permettant la cartographie de l'environnement et la localisation de ce robot dans cet environnement. Finalement, la dernière section sera réservée à un survol des différentes techniques et paradigmes utilisés dans la planification de chemins et l'évitement d'obstacle.

Actuellement, la majorité des robots crées en série sont des robots destinés à l'industrie. Ils sont conçus pour opérer de manière périodique et identique dans des environnements parfaitement contrôlés et statiques. L'incertitude étant nulle, ces robots sont généralement dépourvus de tout processus d'adaptation et d'apprentissage. Leurs opérations sont parfaitement connus dés la phase de conception et faîtes de manière optimal.

Dernièrement, avec l'apparition des centrales nucléaires, de l'exploration des planètes (telle que Mars), et la nécessité grandissante qu'a l'homme de s'affranchir de quelques travaux pénibles (travail en chantier, minier etc...), les objectifs des robots de nouvelle génération se trouvent agrandis. L'environnement devient inconnu, incertain et dynamique. Dans une telle optique, le robot doit être muni de systèmes qui lui permettent, de bien se localiser, de correctement cartographier l'environnement et d'interagir de manière efficace avec celui-ci.

1.1 Aspect physique du robot mobile autonome

L'architecture physique d'un robot mobile autonome est généralement constituée de trois couches :

- Les **organes de perception** (capteurs).
- Les **organes de locomotion** (actionneurs).
- Les **unités de traitements** (ordinateurs et systèmes embarqués).

Certains robots spécifiques sont munis d'équipements spéciaux. Par exemple, les bras robotisés utilisés en industrie sont munis d'un bras à plusieurs degrés de liberté et dépourvus d'organes de locomotion (car par principe ils sont statiques). Une autre catégorie de robots est totalement dépourvue d'organes de perception, l'environnement est statique et entièrement connu (cartographie constante stockée dans la mémoire du robot).

1.1.1 Types de capteurs utilisés dans les RMA

Les capteurs utilisés en robotique sont en nombre très important et en technicité très hétérogène. Cet état de fait constitue un obstacle pour couvrir efficacement ce domaine.

Il est généralement admis [Bra06] que pour classifier les capteurs on recourt à trois critères. Le premier critère (proprioceptif/extéroceptif)[Lar03] spécifie si le capteur capte une grandeur physique interne au robot (proprioceptif) ou une grandeur physique propre à l'environnement (extéroceptif). Le second critère, spécifie l'activité du capteur. Un capteur passif capte directement la grandeur physique au travers d'une cellule sensible (thermomètre, baromètre, anémomètre ...). Un capteur passif, capte l'information au travers de la réflexion d'un rayonnement produit par le capteur lui même, par exemple : une onde acoustique dans le cas d'un radar, un rayon laser dans le cas d'un capteur laser ou une émission de lumière dans le cas d'un capteur infrarouge. Le troisième critère est un critère de portée. Ce critère distingue les capteurs ayant une portée locale (caméra, Sonar) des capteurs ayant une portée globale (GPS, certains type de Radar). Le tableau 1.1 résume ces différents critères pour une liste non exhaustive de capteurs[Bra06].

1.1.1.1 Le Sonar

Le sonar, extéroceptif, actif, a été et est toujours le capteur de prédilection de la robotique. Cela est du essentiellement à son faible coût, son faible poids et ses dimensions réduites et sa basse consommation électrique comparés aux autres capteurs du marché[Abo03]. Ce capteur est utilisé pour mesurer la distance des obstacles potentiels par rapport à la source d'émission.

Ce type de capteurs possède un cône de détection d'environs 15^o. Typiquement, 24 capteurs répartis sur la circonférence du robot sont installés pour couvrir entièrement l'espace voisinant le robot. Le fonctionnement du Sonar suit le principe suivant : une émission périodique d'un signal acoustique à haute fréquence (ultrasons entre 50 et 250 kHz) et la mesure du temps que met l'onde acoustique pour revenir

	Local	**global**
Proprioceptif	**Passif :** - Capteur de batterie - Thermomètre de processeur - Accéléromètre - Gyroscope - Inclinomètre - Compas **Actif :**	**Passif :** **Actif :**
Extérioceptif	**Passif :** - Caméra à bord **Actif :** - Capteur SONAR - Capteur Infrarouge - Capteur Laser	**Passif :** - Caméra sur le toit - GPS par satellite **Actif :** - Capteur SONAR GPS

TAB. 1.1 – Classification des capteurs

au capteur. Le temps mesuré est proportionnel au double de la distance détectée (temps de l'aller et du retour). S'il n'y a aucun écho, cela se traduit par l'inexistence d'obstacle dans cette direction. Les mesures sont effectuées à un rythme de 20 fois par seconde ce qui donne ce son si caractéristique à ce type de capteurs[Bra06].

La précision de cartographie de ce type de capteur n'est pas très bonne. Cela est dû à de multiples inconvénients à amputer au Sonar :

Premièrement, une réflexion de l'onde acoustique avec un certain angle par rapport à un mur, donne l'illusion au robot que ce mur est soit plus éloigné que la réalité ou totalement inexistant (pas de retour de l'onde).Deuxièmement, le cas d'objets complexes générant de multiples échos rend l'interprétation de la réflexion problématique.Troisièmement, dans le cas d'un système multi-robots muni de Sonar, des interférences surviennent donnant aux robots l'illusion d'obstacles qui ne sont que les émissions d'autres robots. Jorg et Berg [Jor98] résolvent ce problème en intégrant des signatures (codes) pseudo-aléatoires aux émissions des robots ce qui permet un filtrage efficace. Quatrièmement, les effets des conditions atmosphériques (température, pression de l'air, humidité). Finalement, la perte exponentielle de la force du signal par rapport à la distance de la cible selon l'équation (1.1) :

$$S_{trans} = \frac{e^{-\alpha d}}{d^2} \tag{1.1}$$

Les auteurs {Aboshosha et Zell} [Abo03] proposent une solution pour palier à la relative imprécision du Sonar en le couplant à un capteur Laser(voir section suivante). Le couple Sonar-Laser se trouvent complémentaire l'un et l'autre chacun palliant (en partie) les inconvénients de l'autre. Ce système de détection hybride est donc plus précis.

1.1.1.2 Le capteur infrarouge

Le capteur infrarouge, extéroceptif, actif, n'utilise pas le même principe de détection que les Sonar. Le temps d'un aller-retour d'un photon est beaucoup trop court pour être mesuré par un équipement standard. Il utilise une LED infrarouge qui génère des pulsations de lumières à 40kHz. Le système est muni d'un vecteur de réception. La lumière infrarouge se réfléchira sur l'obstacle avec un angle qui varie selon la distance de cet obstacle. C'est l'angle de réflexion détecté par le vecteur de réception qui informera sur la distance de l'obstacle.

Ce système présente quelques défauts. Premièrement, la relation entre l'angle de réflexion et la distance n'est pas linéaire, nécessitant un calibrage du résultat à travers un post-traitement ou une table de correspondance. Le second problème est de taille. Le détecteur s'avère complètement inefficace pour des distances proches (inférieure à 6 cm), il retournera une distance plus grande que prévue.

1.1.1.3 Le capteur LASER

Les capteurs LASER 2D, extéroceptifs, actifs, sont beaucoup moins utilisés dans la robotique mobile autonome. Les raisons à cet état de cause est dû au prix relativement cher de l'équipement, un poids élevé (environ 5 kg) et la consommation électrique importante.

Le capteur LASER possède de nombreux avantages tels que la vitesse, la haute précision, la haute résolution angulaire et l'interprétation aisée des données fournies. Il calcul la distance des obstacles grâce à l'intersection du rayon laser avec les corps. La réflexion du rayon permet d'évaluer la distance parcourue par le rayon laser. La cartographie est effectuée par la rotation du rayon laser par rapport à un axe. On obtient des plans de coupe 2D de l'environnement en haute précision.

Le capteur Laser est désavantagé lorsque l'obstacle est d'une couleur trop sombre. En effet, le rayon est absorbé par le matériau et n'est donc plus réfléchi, le capteur retourne une distance infinie. Pareillement, si l'obstacle est translucide (porte en verre par exemple), le rayon Laser traverse le corps sans être réfléchi, le capteur ne détecte pas l'obstacle. Finalement, la coupe 2D de l'environnement est problématique. Si par exemple, l'obstacle est une table, le capteur détectera les pieds de cette table mais pas la table en elle même, le capteur retournera donc un obstacle beaucoup plus réduit qu'il ne l'est réellement. On peut émettre la même réserve concernant des escaliers qui ne sont pas détecter par cause de l'approche planaire (horizontale) de la détection.

1.1.1.4 Le compas

Le compas, extéroceptif, passif, est utilisé dans la localisation du robot par rapport à son environnement. Il est possible pour un robot mobile de se localiser par rapport à son environnement en traitant les données d'orientation et de rotation fournies par chaque roue et en connaissance de la position initiale (odomètrie). L'inconvénient de cette méthode est le cumul d'erreurs dû à l'imperfection des capteurs de direction et de rotation des roues qui par un effet papillon sur une durée suffisante finit par retourner une position théorique très éloignée de la position réelle.

On peut envisager deux solutions pour ce problème, le GPS ou le compas électronique. Le GPS est un type de compas dit *"satellitaire"* se basant sur une constellation de 24 satellites orbitant à 20220km [1]. Il permet une localisation par triangulation. Bien que très précis se relève complexe a utilisé est nécessite que l'environnement soit en extérieur. Un compas électronique à pour avantage de pouvoir fonctionner dans environnements aussi bien fermés qu'ouverts. Il détermine le champ magnétique à partir des propriétés électriques de certains matériaux soumis à un champ magnétique ; Les quatre principales technologies utilisées dans les compas électroniques sont le fluxgate, l'effet Hall, la magnétorésistivité et la magnétoinduction[2].

Le compas les plus simple d'utilisation est le compas analogique. Il distingue huit directions différentes qu'il retourne avec huit niveaux électriques différents. Le compas numérique est plus complexe mais aussi beaucoup plus précis, il atteint des résolutions de l'ordre du degré[Bra06].

1.1.1.5 Le gyroscope, l'accéléromètre et l'inclinomètre

Le triplet accéléromètre, gyroscope, inclinomètre est très utilisé dans la robotique de haut niveau qui requière une connaissance du positionnement rigoureuse et une information fiable pour l'orientation, la vitesse et l'accélération. On rencontre se type de contrainte dans les robots bipède, les avions autonomes, les robots tractés et les robots marcheurs (type insecte).

Généralement, ces capteurs ne captent une grandeur que dans une seule dimension. C'est pour cela qu'il n'est pas rare de trouver un jeu de capteurs positionnés sur différents axes pour capter les grandeurs dans plus d'une dimension.

Le principe de tous les accéléromètres est basé sur la loi fondamentale de la dynamique $F = m.a$ (F : force (N), m : masse (kg), a : accélération (m/s2) aussi notée γ. Plus précisément, il consiste en l'égalité entre la force d'inertie de la masse sismique du capteur et une force de rappel appliquée à cette masse. On distingue deux grandes familles d'accéléromètres : les accéléromètres non asservis et les accéléromètres à asservissement. Certains cristaux (quartz, sel de Seignette)

[1] http ://fr.wikipedia.org/wiki/Global_Positioning_System
[2] http ://fr.wikipedia.org/wiki/Compas_(navigation)

et certaines céramiques ont la propriété de se charger électriquement lorsqu'elles sont soumises à une déformation. Inversement, elles se déforment si on les charge électriquement, le phénomène est réversible. Le cristal se charge sur deux faces en regard avec des charges opposées lorsqu'on le soumet à une force exercée entre ces deux faces. Une métallisation des faces permet de recueillir une tension électrique qui pourra être utilisée dans un circuit [3].

Le fonctionnement du gyroscope est basé sur la précession existant dans tout corps en rotation. Il ne délivre pas une information d'orientation absolue mais plutôt le taux de changements dans une orientation donnée. Ceci est considérée comme étant un défaut car l'application n'est pas à l'abris d'une accumulation d'erreurs avec le temps (le même problème qu'avec l'odometrie).

Les inclinomètres mesurent l'orientation absolue d'un angle sur un intervalle de données qui dépend du modèle du capteur. Puisque l'inclinomètre détecte une valeur absolue et non pas un dérivatif, il est considéré plus apte à des mesures de direction qu'un gyroscope. Cependant, les inclinomètres souffrent de retards dans la délivrance des données, ce qui les rend peu aptes à des applications où la vitesse des réflexes est primordiales (d'où l'utilisation des gyroscopes).

1.1.1.6 La caméra numérique

La caméra numérique est le capteur le plus complexe utilisé en robotique. Ils n'ont été utilisé que récemment à cause de la puissance de calcul et de mémoire nécessaire au traitement du flux d'information.

Les rayons lumineux (constitués de photons), issus de la scène filmée (image) passent au travers d'un objectif optique puis vont frapper un capteur sensible CCD ou CMOS. Les photons reçus créent une charge au niveau des pixels du capteur et constituent ainsi une mémoire d'image. Chaque pixel est donc chargé de façon encore analogique à ce niveau. Un système électronique permet de "vider" régulièrement les charges analogiques de tous les pixels du capteur et les transforme en valeurs numérisées constituant ainsi l'image numérique. Ce circuit spécialisé traite image par image à intervalles réguliers, 24, 25, 30 fois par seconde suivant le réglage de la camera.

Une caméra numérique délivre un flux vidéo de résolution stable. Pour une utilisation en robotique mobile, une résolution de 60x80 s'avère pratique car étant suffisante pour la reconnaissance des obstacles et petite pour un traitement temps réel avec des algorithmes adéquats (algorithme de Sobel par exemple). A cette résolution, il est possibles de traiter un flux vidéo de 30 frames/s.

[3] http ://fr.wikipedia.org/wiki/Accéléromètre

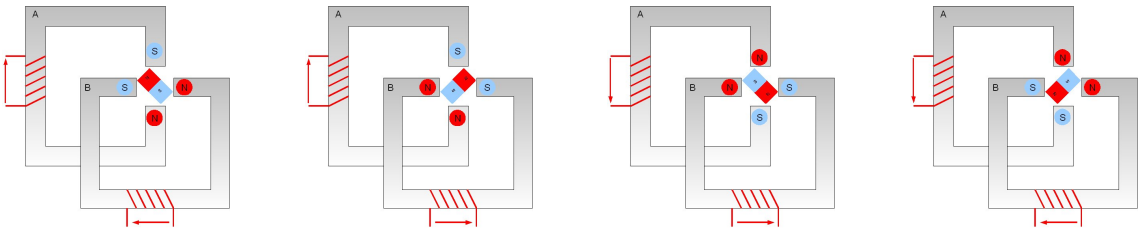


FIG. 1.1 – Les 4 phases d'un moteur pas à pas

1.1.2 Types d'actionneurs utilisés dans les RMA

Les actionneurs qu'il est possible de trouver sur un robot sont très variés. Ils dépendent du but du robot lui même et des buts qu'ils lui sont assignés.

1.1.2.1 Le moteur à courant continu

Les moteurs à courant continu sont les plus communément utilisés pour la locomotion dans la robotique mobile. Les moteurs à courant continu sont propres, silencieux et peuvent produire suffisamment de puissance pour une variétés de tâches. Ils sont beaucoup plus simples à contrôler que des moteurs pneumatiques qui sont principalement utilisés pour des tâches demandant beaucoup de puissance. De plus, les moteurs pneumatiques doivent être constamment reliés à des compresseurs et des pompes ce qui nuit gravement à la mobilité de tels systèmes.

1.1.2.2 Le moteur à pas

Il existe deux types de moteurs qui sont significativement différents des moteurs à courant continu. Les moteurs à pas qui seront discutés dans cette section et les servo-moteurs qui seront discutés dans la prochaine section.

Les moteurs à pas diffèrent des moteurs à courant continu par le fait qu'ils possèdent deux bobines qui peuvent être indépendamment contrôlées. Cela à pour effet de pouvoir contrôler ce type de moteur par des pulsations pour procéder à exactement un pas en arrière ou en avant en comparaison à un mouvement continu et fluide d'un moteur continu. Typiquement, il faut 200 pas pour effectuer une révolution. Cela induit que chaque pas est une rotation de 1.8^o. Certains types de moteurs à pas permettent même des demis-pas, cela augmente grandement la précision de tels systèmes. Il faut aussi remarquer que les moteurs à pas possèdent un nombre de pas maximal par seconde.

En comparaison, les moteurs à pas demandent moins d'efforts pour contrôler la vélocité et la position par rapport à un moteur à courant continu. Cependant, ils sont rarement utilisés pour conduire des robots mobiles. La raison à cela est leur absence de rétroaction et leurs limitation de vitesse en plus du faible rapport poids/performance par rapport au moteurs à courant continu.

1.1.2.3 Le servomoteur

Les servomoteurs sont assez comparables aux moteur à courant continu. Ils sont considérés comme des moteurs à courant continu de hautes qualité. Ce moteur est capable de faire face à de grandes variations de position, vitesse et rotation.

Les servomoteurs en-capsule une électronique pour contrôler les largeurs d'impulsion (PW ou LI en français). Contrairement aux moteurs à courant continu, les largeurs d'impulsion ne sont pas utilisés pour contrôler la vélocité mais plutôt la position désirée du disque de rotation du servomoteur. Le disque de rotation des servomoteurs ne font pas de rotation libre comme ceux des moteurs à courant continu, ils tournent dans un sens comme dans l'autre sur une plage d'angle définie de plus ou moins 120^o par rapport au milieu.

Le signal en LI utilisé dans un servomoteur est de fréquence constante de 50Hz, c-à-d qu'il y a une impulsion toutes les 20ms. La largeur des impulsions du signal déterminent l'angle du disque du moteur. Par exemple, une largeur de 0.7ms fera tourner le disque tout à fait à gauche (-120^o), une largeur de 1.7ms fera tourner le disque du moteur tout à fait à droite (120^o). Les valeurs exactes dépendent évidemment du modèle du moteur.

Comme pour les moteurs à pas, les servomoteurs sont de bons choix pour la robotique mobile (tourner l'axe des roues, manipuler des bras ...). Mais tout comme les moteurs à pas, le manque de rétroaction est un inconvénient majeur. On ne sait jamais si le moteurs est arrivé à la position désirée ou si au contraire le mouvement à été gêné par un quelconque élément de l'environnement.

1.1.3 Méthodes de locomotion utilisées en RMA

Bien que la robotique ne soit pas encore arrivée au niveau des attentes de la sciences-fiction, les avancées techniques sont importantes notamment dans l'allure de la locomotion qui se veut la plus naturelle possible. Cette partie introduit la taxonomie des méthodes de locomotion les plus représentatives ou en cours de développement.

1.1.3.1 Les robots à roues

Les robots à roues sont de loin les plus représentatifs des robots de recherche. Cela est dû à leur "simplicité" de réalisation. En effet, il n'y a pas à prendre charge les problèmes d'équilibre, de marche ou de stabilité. De plus, la mécanique pour contrôler l'ensemble est assez simple à mettre en oeuvre et de coût réduit.

1.1.3.2 Les robots sous-marins

Plus de 1000 robot sous-marins non habités sont en opération dans le monde. La majorité de ces robots sont des sous-marins commandés à distance commerciaux.

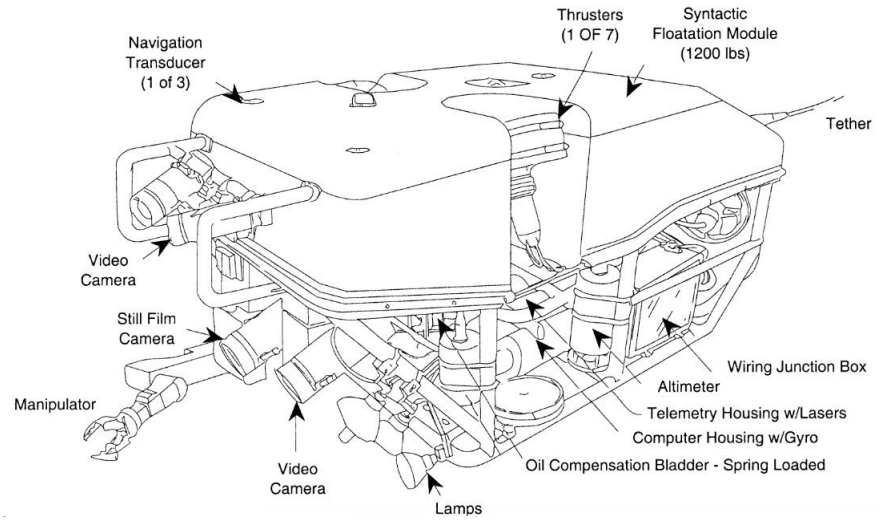


FIG. 1.2 – Le robot sous-marin Jason.

Leurs missions, inspections sous-marines, surveillance, construction et réparation. Ils oeuvrent à des profondeurs moyenne de 1000 m. Ils représentent un investissement lucratif surtout en ce qui concerne des opérations sur des plateformes pétrolières, des pipelines et les câbles de télécommunication.

Les robots sous-marins peuvent construire des cartographies du fond marin de grande précision grâce à des capteurs acoustique et des appareils de prises de vues haute définition. C'était précisément la mission de Jason illustré sur la figure1.2 et Argo II qui cartographièrent $2Km^2$ de fond marin par 4100 m de profondeur. Chose autrefois infaisable.

Les obstacles techniques au travail sous-marins sont différents de ceux rencontrès sur terre, dans les airs ou même dans l'espace. Premièrement, le problème de l'atténuation rapide de toute forme de signaux : acoustique ou électromagnétique. La conséquence directe de ce problème et la difficulté de communication d'une part et l'obligation pour le robot d'être proche de la cible (pour une cartographie par exemple) avec tous les dangers que cela représente. Deuxièmement, la haute pression ambiante des environnement sous-marins posent un réel problème pour les véhicules sous-marins. De nos jours, seuls quelques machines peuvent descendre à plus de 1000 m de profondeur. Très rares sont ceux qui arrive à 6500 m. Un seul peut opérer à 11000m. [Whi00]

1.1.3.3 Les robots aériens ou drones

Engins volants de taille réduite, moins chers et plus simples à mettre en oeuvre qu'un aéronef (la présence d'un pilote étant l'un des éléments les plus dimensionnant pour la conception d'un tel système), ils sont également plus discrets et leur perte n'est pas aussi lourde de conséquence que celle d'un appareil complet et de

FIG. 1.3 – Le Drone Predator.

son pilote. Ils représentent une alternative intéressante grâce à leur faible coût de fabrication et d'utilisation.

Au fur et à mesure que les technologies, informatiques notamment, progressent, les drones peuvent être utilisés comme plate-forme de désignation de cible ou comme armes. Ils sont aussi utilisés comme précurseurs d'opérations, souvent à des fins de recueil de renseignements. Ses missions sont alors de la surveillance et de la reconnaissance.

Leurs applications civiles incluent les contrôles sur le trafic, les opérations de recherches aériennes et de sauvetage, la récolte de données pour la prédiction météorologique, le relais d'informations...

Leurs tailles sont variées : de quelques centimètres à plusieurs mètres. Leurs formes également, tout comme leurs types de propulsion : certains sont équipés de réacteurs, d'autres d'hélices, quand d'autres utilisent des rotors, à l'instar des hélicoptères. Des drones terrestres commencent à faire également leur apparition.

Enfin, certains servent tout simplement de démonstrateurs technologiques, validant à moindre coût, grâce à la réduction d'échelle et donc la quantité de matériaux nécessaires, certaines formules aérodynamiques ou certains équipements, sans pour autant risquer la vie d'un pilote d'essais.

Ce type de tests permet d'atteindre les limites d'un appareil, en dépassant celles que peut supporter un pilote (+9g/-3g maximum), afin d'en confirmer la solidité.[4]

1.1.3.4 Les robots bipèdes

Les robots bipèdes sont un type de robots se mouvant sur leurs membres arrières et utilisant la marche comme méthode de locomotion. Seuls les humains, quelques primates et exceptionnellement les chiens peuvent utiliser cette méthode de locomotion. Cependant, la marche n'est pas récente. Des preuves formelles attestent de l'existence de cette méthode depuis le temps des dinosaures et leurs représentant le plus célèbre le Tyrannosaure Rex.

La marche est la méthode de locomotion la plus économe en énergie[5]. Elle est aussi la moins stressante pour le corps pour une vitesse moyenne de 5 km/h. De nombreuses études et recherches ont pour seul but de copier la marche humaine. Asimo, le plus connu des robot marcheur consomme 10 fois plus d'énergie que l'homme.

1.1.3.5 Les robots quadrupèdes

A mi-chemin entre la marche bipède et la marche insectoide, la marche quadrupède se fait sur quatre membres. Ce type de marche est caratéristique de la majorité des mammifères et des dinosaures herbivores.

Ce type de marche est plus complexe que la marche des insectes vu que pendant le pas, le polygone de sustentation est un triangle. Elle demande donc une coordination en temps réel. Elle est cependant moins complexe que la marche bipède bien que plus gourmande en énergie.

Ce type de marche a été implémentée par les étudiant du MIT dans le robot appelé BigDog. L'informatique de bord gère l'orientation, la direction, ainsi que la distribution d'énergie entre les différents servos lorsque les conditions changent. Le contact avec le sol et la pression de la charge sur le "dos" sont constamment pris en compte, tandis qu'un gyroscope et un système de vision stéréoscopique déterminent l'orientation et la voie à suivre pour parvenir au but assigné, ou déterminé par le logiciel de bord. Bien entendu, tous les paramètres internes propres au fonctionnement du moteur principal sont pris en charge, ce qui fait de BigDog un engin totalement autonome[6].

1.1.3.6 Les robots insectes

Les robots insectes sont un type particulier de robots possédant quatre pattes ou plus (jusqu'à 32). Les robots insectes sont une extension des robots quadrupèdes

[4]http ://fr.wikipedia.org/wiki/Drone

[5]http ://www.larecherche.fr/content/impression/article ?id=10544

[6]http ://www.futura-sciences.com/fr/news/t/robotique/d/bigdog-la-perfection-a-quatre-pattes_15006/

FIG. 1.4 – Bigdog, la meilleur réalisation de robot quadrupède.

vues précédemment. Tous ce qui sera relaté dans cet section peut être appliqué aux robots quadrupèdes.

Ils sont comparativement aux robots bipèdes moins complexes à contrôler, ce qui explique, d'un point de vue évolutif, que la locomotion des insectes soit chronologiquement antérieur à celle des bipèdes. La multitude de pattes que possède un insecte lui confère une stabilité accrue et une simplicité à préserver l'équilibre. En effet et généralement, à chaque phase de la marche, l'insecte possède au moins trois pattes en appuie. Ces pattes en appuie génèrent un polygone (virtuel) de sustentation qui peut être triangulaire, carré ou plus. Si la projection du centre de gravité de l'insecte se trouve au sein de ce polygone, l'insecte est en équilibre [Zen04].

La marche des insectes et des bipèdes (comme celle des quadrupèdes) sont tous deux bien adaptés aux terrains accidentés ou présentant de fortes variations (même si la marche bipède est plus efficace pour les terrains à fortes variations tels que les escaliers). Cependant, la différence évolutive qui à conduit à l'apparition de la marche bipède est la consommation d'énergie. La marche bipède est plus économe en énergie.

Matt Moses à élégamment décrit dans son papier [Mos00] une méthode de locomotion minimaliste se basant sur le concept de Nv Neuron et de topologie de connexion pour obtenir, ce qu'il nomme, les différents motifs de locomotion.

1.1.4 Exemple de locomotion avec un réseau de neurones

1.1.4.1 Nv Neuron :

Le Nv Neuron est un élément de circuit de retard contenant une résistance, une capacité et un trigger de Schmidt dont le schéma est illustré dans la figure 1.5. Dans ce qui suit, le point A de la figure sera dénoté "Entrée du réseau", le point B "Point de stimuli" et le point C "la sortie". Quand la sortie du neurone est "haute", le neurone est inactif. Si la sortie du neurone est "basse", le neurone s'active.

Généralement, les neurones sont structurés sous forme de boucles. La sortie réseau du premier neurone est connecté à l'entrée réseau du neurone suivant et ainsi de suite. La sortie réseau du dernier neurone est connectée à l'entrée du premier neurone. Ces réseaux de par leur topologies et les nombreuses configurations de stimulations ouvrent la voie à de nombreux motifs de locomotion. La formule 1.2 décrit le temps de réponse d'in neurone :

$$t_1 - t_2 = RC\ ln\frac{V_h}{V_{thl}} \tag{1.2}$$

V_h est le voltage maximum de sortie du Trigger de Schmidt, V_{thl} est le voltage seuil de transition descendante. Généralement, R prend des valeurs dans les megaOhms, C dans les microfarads et le durée de l'impulsion de l'ordre de quelques secondes.

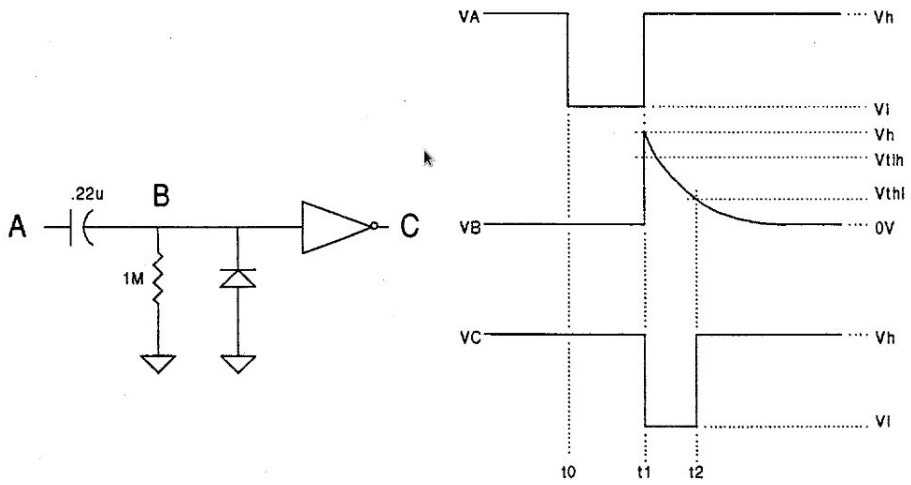


FIG. 1.5 – Le shéma d'un Nv-Neuron et le diagramme de sa réponse à une impulsion.

1.1.4.2 Génération de motifs de locomotion avec un réseau de neurones

Les neurones sont généralement structurés en boucles ou en chaines. Quand un premier neurone est actif, il génère une impulsion qui active le deuxième neurone.

Quand le second neurone est actif, le premier se désactive après l'arrivé à terme de son délai de $t_2 - t_1$ secondes. Le seconde transmet une impulsion au troisième pour débuter une nouvelle phase équivalente à la précédente.

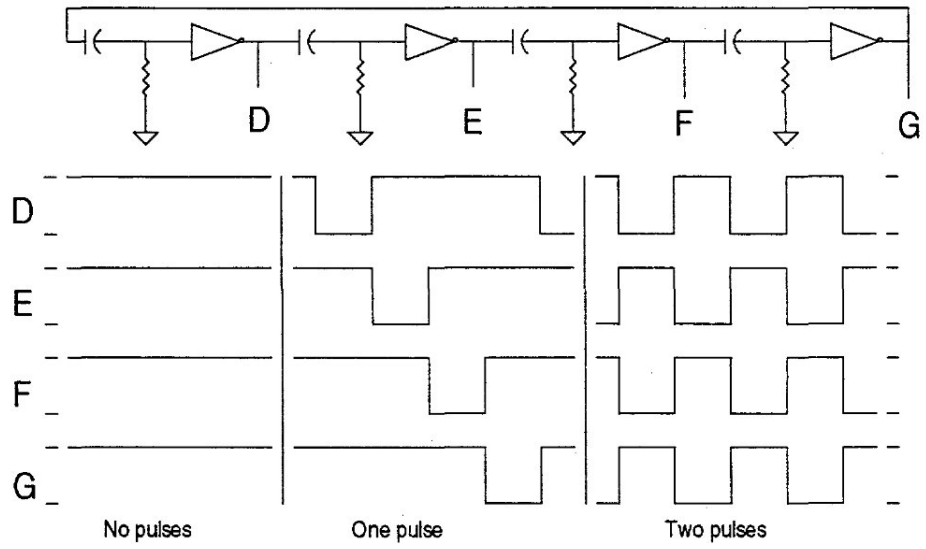


FIG. 1.6 – Diagramme temporel d'une boucle de neurones

Si les neurones sont en chaine, le dernier neurone transmet l'impulsion vers l'extérieur de la chaine et se désactive en mettant fin au cycle unique de celle-ci. Si les neurones sont structurés en boucle, le dernier neurone transmet l'impulsion vers le premier pour débuter un nouveau cycle. Les boucles et les chaines sont illustrées respectivement dans la figure 1.6 et dans la figure 1.7.

Les sorties des neurones peuvent être amplifiées et utilisées pour commander des moteurs (voir section 1.1.2). Les boucles de neurones, de par les combinaisons possibles de stimulation, peuvent générer une grande variété de motifs de signaux. en contrôlant la topologie du système, c-à-d quel neurone contrôle quel moteur, la configuration initiale de stimulation, il est possible d'inculquer une grande variété de modes de locomotion à un robot tel que la marche, la nage, le trot, le saut etc ...

La figure 1.7 illustre la manière de contrôler le pas d'une patte grâce à une chaine de Nv-Neurons. L'impulsion retardée traverse la chaine neurone après neurone par phase d'activation et de désactivation séquentielle telle que représentée par le diagramme. Les deux moteurs A et B sont connectés au réseau de tel manière à obtenir l'oscillation voulue à chaque temps. A+ est connecté à D, A- à F, B+ à E et B- à G. Le mouvement se fait en quatre temps.

Au premier temps, l'impulsion se trouve en sortie du premier neurone à D. Cela génère une différence de potentiel avec F aux pôles du moteur A qui initie une rotation.

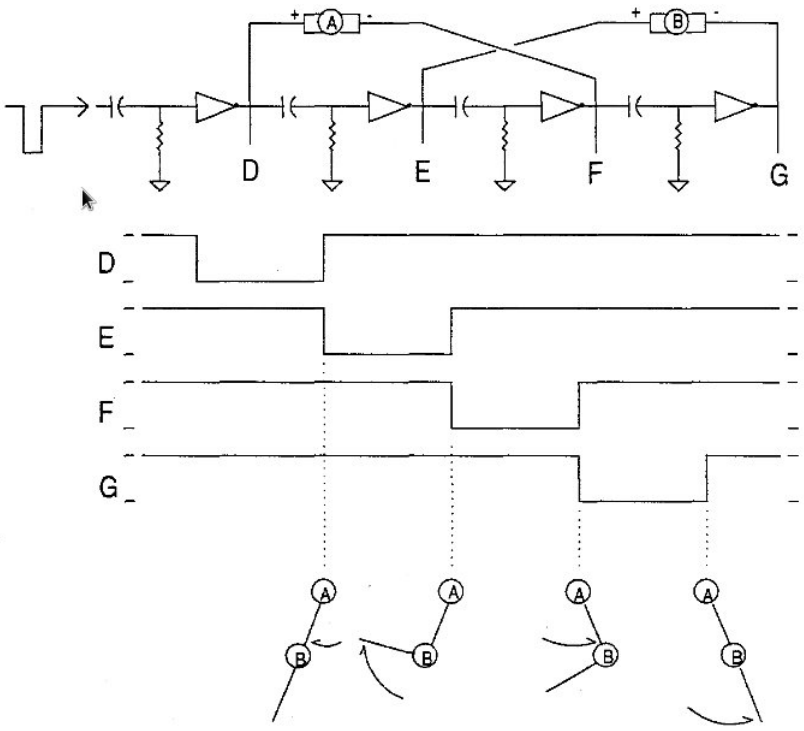


FIG. 1.7 – Une chaine de neurones contrôlant le pas d'une patte.

Au deuxième temps, l'impulsion se trouve en sortie du deuxième neurone à E. Ceci génère une différence de potentiel avec G aux pôles du moteur B qui initie une rotation. La seconde section de la patte bouge en se repliant.

Au troisième temps, l'impulsion se trouve en sortie du troisième neurone à F. On se retrouve avec un potentiel inverse à celui du premier temps faisant tourner le moteur dans le sens contraire à celui du premier temps.

Au dernier temps, l'impulsion se trouve à la sortie du dernier neurone c-à-d au niveau de G. On se retrouve avec un potentiel inverse au second temps. Le moteur B initie une rotation contraire. La patte se retrouve à son état initial et est prête à recommencer un nouveau cycle.

Si les bornes des moteurs se trouvent inversées, le robot effectuera une marche vers l'arrière. De même, une variation dans V_h et V_{thl} induit un changement dans la valeur de $t_2 - t_1$ en accord avec l'équation 1.2, cela modifie la vitesse de progression du robot.

La figure 1.8 illustre un robot équipé d'un réseau de quatre Nv-Neurons structurés en boucle. Les cercles représentent ces neurones, les cercle coloriés en noir représentent un neurone actif. Les ellipses représentent les pattes du robot qui intègrent un réseau de neurones en chaine semblable à celui de la figure 1.7 et

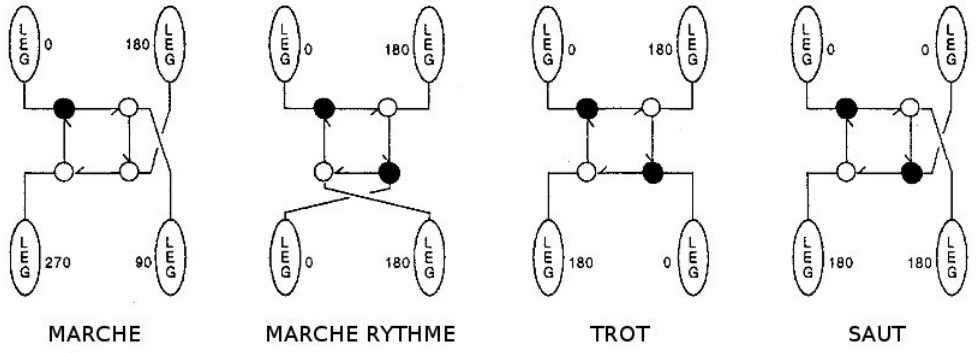


FIG. 1.8 – Le shéma d'un Nv-Neuron et le diagramme de sa réponse à une impulsion.

qui effectue un cycle (pas) à la réception d'une impulsion. Les nombres apparaissant aux cotés des ellipses représente la phase du pas. Puisque, comme vu précédemment, le pas se fait en quatre temps, chacun de ces temps se voit attribuer une valeur de phase respectivement 0, 90, 180 et 270.

La figure 1.8 illustre clairement comment un changement de topologie, de stimulation et de phases de pattes peuvent radicalement modifier le mode de locomotion. Il est envisageable d'intégrer un multiplexeur au système capable de modifier le mode de locomotion du robot en plein action.

1.2 Environnement dynamique et incertain

1.2.1 L'aspect dynamique

Un environnement est considéré dynamique si les objets qui le composent peuvent changer de position (une sphère qui roule), changer de forme (un piéton qui promène son chien), ou apparaitre et disparaitre (porte coulissante par exemple) au cours du temps.

1.2.2 L'aspect incertain

Dans la suite de ce mémoire, nous allons considéré comme [Lar03] que l'information sur l'environnement est incertaine, si elle est bruitée (mauvaises conditions de mesures), incomplète (obstruction d'un capteur ou portée limitée, absence d'informations sur l'évolution d'un objet ou un phénomène) ou imprécise (les glissement des roues sont observables mais rarement mesurables avec précision).

Chapitre 2

Approches de localisation

"Où suis-je ?" est peut-être la phrase que se posent inconsciemment et chaque seconde des milliards d'êtres humains. A la base des informations que reçoivent nos sens, le cerveau essaye constamment de répondre à cette question dont dépend l'optimalité de la prise de décision (tel que la sauvegarde de l'équilibre ou la recherche d'un itinéraire). Un seul dysfonctionnement sera sévèrement ressentie par le sujet que cela soit pas un stress intense ou le fameux "mal de mer".

L'auto-localisation d'un robot par rapport à son environnement revête la même importance que pour son équivalent humain. C'est une phase critique dont dépend grandement la prise de décision au niveau de la navigation et de la planification de chemins qui seront relatés au prochain chapitre ou au niveau de la sauvegarde de l'équilibre pour des systèmes bipèdes ou quadrupèdes. Il n'est pas rare de voir de tels robots s'affaler à terre à cause d'un gyroscope bloqué ou se perdre à cause d'une défaillance de capteurs.

L'imperfection des mesures que permettent les différents capteurs sont un obstacle pour une bonne localisation. L'essentiel des approches qui seront relatées dans ce chapitre, reflète la détermination à extraire l'information la plus fiable à partir de systèmes très imparfaits. A la lumière de ce qui a été dit, il va sans dire que la majorité des approches empruntent à l'intelligence artificielle ses paradigmes les plus connus .

2.1 Point de vue probabiliste

Les meilleurs méthodes de localisation sont probabilistes.(Thrun 2000 ; Dissanayake, Newman et al. 2001 ; Montemerlo, Thrun et al. 2003). Il y a une variété de manières différentes d'implémenter une approche probabiliste. Il y a aussi différents types d'algorithmes qui sont en cours de recherche et on ne peut pour l'instant juger lequel est le meilleur.

Ces algorithmes varient tous dans leurs hypothèses concernant l'environnement, le type de capteur associé, les capacités de calculs demandées, scalabilité, flexibilité

ainsi que beaucoup d'autres paramètres. Certains de ces algorithmes sont purement probabilistes et cela jusqu'aux fondements mathématiques, d'autres le sont moins et s'apparente plus à des heuristiques ou à des méthodes hybrides lorsqu'ils sont combinés à des méthodes plus exotiques. Dans cette section nous allons nous penchés sur les algorithmes des plus connus du domaine :

- les filtres de Kalman et,
- les filtres particulaires.

Le problème de localisation peut être considéré comme étant un *problème d'estimation Bayésienne*. Le but est d'estimer la localisation du robot à partir de mesures bruitées. Sous un point de vue probabiliste, nous pouvons considérer que le robot possède des *croyances* à propos de sa localisation. A chaque instant, il ne considère pas une seule possibilité de localisation mais bien l'espace de localisation en entier. A partir des informations disponibles, le robot peut croire qu'il a une certaine localisation avec un certain degré de croyance. Le problème de localisation reviens ici à estimer la densité de probabilité de tous l'espace de localisation.

Il est a noté que ce système de localisation se suffit à lui même et n'a pas besoin de planificateur de chemins vu que le modèle d'action (section 2.1.2.2) est essentiel pour le processus de localisation (en fournissant les mesures relatives) et décide de la prochaine action à faire.

2.1.1 Croyances

Le robot possède une croyance sur sa position. C'est une densité de probabilité pour toute localisation $x \in \Xi$, où Ξ est l'ensemble des localisations. La croyance est notée Bel :

$$Bel(x_k) = P(x_k|d_{0...k}) \tag{2.1}$$

Cette équation exprime la probabilité que le robot soit à la localisation x_k en considérant les informations $d_{0...k}$ à ce moment là. Ces informations peuvent aussi inclure des données à priori telles que la carte de l'environnement. Cette densité de probabilité possède un maximum qui exprime la localisation du robot la plus probable.

Pendant la navigation, le robot accède à des informations aussi bien relatives qu'absolues. Le robot incorpore ces mesures à ses croyances pour en générer de nouvelles à propos de sa position. Généralement, deux types de croyances sont à distinguer. Les croyances *à priori* et les croyances *à posteriori*. La croyance à priori $Bel^-(x_k)$ est celle qui est obtenue après l'incorporation de toutes les informations jusqu'à l'étape x_k, ce qui inclue les données relative, mais exclue la donnée absolue à l'étape k. La croyance à posteriori $Bel^+(x_k)$ est celle qui est obtenue après l'inclusion de la dernière mesure absolue.

2.1.2 Perceptions et actions probabilistes

Le point de vue probabiliste de la localisation oblige à exprimer en termes probabilistes les perceptions du robot et les actions à entreprendre. Cette section introduira deux modèles. Le modèle perceptif, qui fournit les mesures absolues, et le modèle d'action, qui fournit les mesures relatives.

2.1.2.1 Perception

Il est possible de décrire les perceptions du robot en termes probabilistes. Soit S l'espace de toutes les mesures possibles par les capteurs du robot, soit s_k la mesure observée au moment k. Il est possible de décrire la probabilité qu'un capteur observe s_k à partir de la localisation x_k au moment k par la densité de probabilité

$$P(s_k|x_k) \tag{2.2}$$

Cette équation est appelée *perception* ou *modèle perceptif*. Ce modèle est invariant avec le temps d'où la facultativité de l'indice k. Cette densité de probabilité est difficilement calculable. La raison est dû aux cas où l'espace de mesure S est de taille trop importante. Prenons le cas d'une caméra : prendre en compte toutes les possibilités d'images à toutes les localisations est tout bonnement irréalisable dans des délais raisonnables.

2.1.2.2 Action

Les actions permettent au robot de se mouvoir dans l'environnement. Soit A l'ensembles des actions possibles par le robot, soit a_k l'action effectuée par le robot au moment k. Il est possible d'exprimer le changement de position par une transition de probabilité

$$P(x_k|x_{k-1}, a_{k-1}) \tag{2.3}$$

Cette densité de probabilité donne pour un instant $k-1$ à la position x_{k-1}, lorsque le robot effectue l'action a_{k-1}, la probabilité qu'il se retrouve à la position x_k. Autrement dit, la densité de transition décrit de quelle manière les actions changent la position du robot. Cette densité est nommée *modèle d'action*. Elle est invariante dans le temps (indice k superflu).

Le modèle d'action fournit le système en mesures relatives. En pratique, le modèle d'action est défini grâce à des mesures odométriques effectuées au préalable.

2.1.3 Localisation probabiliste

La localisation d'un robot dans un point de vue probabiliste suit un schéma. Premièrement, le robot possède des croyances initiales sur sa position. Le modèle d'action effectue une action est renvoie vers le système une mesure relative de la position en cours (équation 2.3). Le système calcule à la base de cette action une croyance à priori. Le système met à jour la croyance à priori tant que le modèle

d'action le fournit en mesures relatives. Lorsque les capteurs envoient une mesure vers le modèle perceptif, celui-ci fournit au système une mesure absolue (équation 2.2) qui lui permet de calculer une croyance à posteriori qui servira à mettre à jour les croyances futures. Ce cycle est répété jusqu'à atteindre un but ou satisfaire un critère d'arrêt.

2.1.3.1 Croyances initiales

Au début, le robot possède une croyance sur sa position initiale. Cette croyance est notée $Bel^{-}(x_0)$. Si le robot *sait* où in se trouve au départ, $Bel^{-}(x_0)$ est une distribution nulle avec un pic au niveau de la position initiale. Si le robot *ne sait pas* où il se trouve, $Bel^{-}(x_0)$ est une distribution uniforme. Finalement, si le robot *croit* être à une position donnée, $Bel^{-}(x_0)$ est une distribution classique avec un maximum au niveau de la position la plus probable.

2.1.3.2 Mise à jour des croyances

A la base de la croyance initiale, le robot commence à interroger ses capteurs et à faire diverses actions dans l'environnement. Les résultats des mesures doivent être incorporer pour mettre à jour les croyances du robot. Au moment k, à la réception de l'action a_{k-1} et avant la réception de la mesure absolue s_k, la densité de probabilité de la croyance à priori est :

$$Bel^{-}(x_k) = P(x_k|s_1, a_1, s_2, a_2, ..., s_{k-1}, a_{k-1}) \tag{2.4}$$

A chaque fois qu'une mesure absolue est disponible, le système met à jour la croyance à priori qui devient une croyance à postériori $Bel^{+}(x_k)$ dont la densité de probabilité est :

$$Bel^{+}(x_k) = P(x_k|s_1, a_1, s_2, a_2, ..., s_{k-1}, a_{k-1}, s_k) \tag{2.5}$$

Toute la problématique réside dans la manière de calculer ces densités de probabilités.

2.1.3.3 Incorporation des actions

Assumons que le robot a accompli une action. La mesure de la position relative doit être incluse dans ces croyances. Dans l'équation 2.4, nous définissons la croyance dans laquelle la dernière information d'action est incorporée, la croyance à priori $Bel^{-}(x_k)$. Grâce au théorème de probabilité totale et l'utilisation du corolaire de Markov, nous pouvons réécrire la définition originale par une formule calculable sur ordinateur.

Probabilité Totale Le théorème de probabilité totale énonce que la probabilité d'un évènement est égale à la somme des probabilité des évènement partiels et dépendants le composant. En accord avec ce théorème, nous pouvons réécrire la définition d'une croyance à priori 2.4 en :

$$Bel^{-}(x_k) = \int_{\Xi} P(x_k|x_{k-1}, z_1, a_1, ..., z_{k-1}, a_{k-1}) \times P(x_{k-1}|z_1, a_1, ..., z_{k-1}, a_{k-1}) dx_{k-1} \tag{2.6}$$

Cette equation exprime la croyance à priori d'être dans l'état x_k comme étant la sommation des probabilités de transition de l'état x_{k-1} vers l'état x_k sachant toutes les actions et les mesures passées, $P(x_k|x_{k-1}, z_1, a_1, ..., z_{k-1}, a_{k-1})$ multipliée par la probabilité d'être dans l'état x_{k-1} sachant toutes les mesures et actions passées, $P(x_{k-1}|z_1, a_1, ..., z_{k-1}, a_{k-1})$.

Le second terme de l'intégrale dans 2.6 est la probabilité d'être à la position x_{k-1} sachant toute les information jusqu'à $k-1$. En particulier une action accomplie à $k-1$. Cependant, la position physique du robot à $k-1$ ne dépend pas de l'action accompli à cette étape. Donc, nous n'avons pas à prendre a_{k-1} en considération lorsque nous exprimons cette probabilité. En utilisant la définition de la croyance à posteriori depuis 2.5, nous réécrivons 2.6 en :

$$Bel^{-}(x_k) = \int_{\Xi} P(x_k|x_{k-1}, z_1, a_1, ..., z_{k-1}, a_{k-1}) \times Bel^{+}(x_{k-1}) dx_{k-1} \tag{2.7}$$

Pour simplifier l'expression du premier terme dans l'intégrale de 3.9, nous avons recourt au corolaire de Markov qui énonce qu'à une connaissance donnée dans une certaine étape, le passé est indépendant du futur et vice-versa. Avec la connaissance de la location précédente, il n'est pas important de savoir les sensation mesurées juste après la dernière action d'où :

$$P(x_k|x_{k-1}, z_1, ..., z_{k-1}, a_{k-1}) = P(x_k|x_{k-1}, a_{k-1}) \tag{2.8}$$

Nous obtenons une équation qui peut être utilisée pour efficacement incorporer les actions du robot dans ces croyances.

$$Bel^{-}(x_k) = \int_{\Xi} P(x_k|x_{k-1}, a_{k-1}) \times P(x_{k-1}|z_1, a_1, ..., z_{k-1}, a_{k-1}) dx_{k-1} \tag{2.9}$$

2.1.3.4 Incorporation des sensations

Assumons que le robot possède la croyance à priori $Bel^{-}(x_k)$, la croyance dans la position après avoir effectuer le dernier mouvement. Le robot prélève des mesures de l'environnement est extrait le vecteur z_k. Nous voulons incorporer ces mesures dans les croyances à priori pour former la croyance à postériori définie dans l'équation 2.5. Avec la règle de Bayes, et le corolaire de Markov, nous pouvons réécrire la croyance à postériori dans une forme réellement programmable.

La règle de Bayes Elle explique comment le robot doit changer ces croyances quand de nouvelle mesures se présentent. En utilisant la règle de Bayes et la définition de croyance à priori 2.4, nous pouvons réécrire 2.5

$$\begin{aligned} Bel^{+}(x_k) &= \frac{P(z_k|x_k,z_1,a_1,...,z_{k-1},a_{k-1})P(x_k|z_1,a_1,...,z_{k-1},ak-1)}{P(z_k|z_1,a_1,...,z_{k-1},ak-1)} \\ &= \frac{P(z_k|x_k,z_1,a_1,...,z_{k-1},a_{k-1})Bel^{-}(x_k)}{P(z_k|z_1,a_1,...,z_{k-1},ak-1)} \end{aligned} \tag{2.10}$$

Comme pour l'incorporation des actions, il est possible de simplifier cette équation en vue d'une implémentation toujours au moyen du corolaire de Markov qui énonce (par le même principe précédant) :

$$P(z_k|x_k, z_1, a_1, ..., z_{k-1}, a_{k-1}) = P(z_k|x_k) \tag{2.11}$$

En remplaçant ces expressions dans 2.10, nous pouvons obtenir la version simplifier suivante qui sera implémentée grâce à laquelle le robot sera en mesure d'incorporer les sensations dans ces croyances :

$$Bel^{+}(x_k) = \frac{P(z_k|x_k)Bel^{-}(x_k)}{P(z_k|z_1, a_1, ..., z_{k-1}, ak-1)} \tag{2.12}$$

2.1.4 Les filtres de Kalman

La méthode basée sur les filtres de Kalman traque la positions des marques ou des éléments de l'environnement lorsque le robot est en mouvement. C'est l'une des rares méthodes qui maintient l'aspect d'incertitude dans une cartographie, mais sont limités par l'allure gaussienne de l'incertitude (relatif aux formules décrivant cette incertitude). Cette limitation a été partiellement levée par l'introduction des filtres de Kalman étendus. La complexité de l'algorithme est proportionnelle à $O(n^2)$, n étant le nombre de marques ou d'éléments de l'environnement.

Pour implémenter la méthode de filtre de Kalman dite "vanilla", les hypothèses habituelles sont que le robot démarre en un point inconnu dans un environnement inconnu. Pour modéliser le mouvement du robot à travers l'environnement, un modèle cinématique est indispensable. Généralement, ce modèle incorpore du bruit. Beaucoup de méthodes utilisent des modèle à temps linéaire discret pour modéliser le mouvement. Cependant, les mouvements des robot sont le plus souvent non-linéaires, en segmentant le chemin en parties assez petites, nous pouvons considérer ces minuscules parties comme étant des segments de droite (linéaires). La mise à jour de l'état du robot, $x_v(k+1)$ peut être calculée comme suit :

$$x_v(k+1) = F_v(k)x_v(k) + u_v(k+1) + v_v(k+1) \tag{2.13}$$

où le v en indice dénote la variable spécifique au robot véhicule, k et l'indice de pas, $F_v(k)$ est la matrice de transition des états, $u_v(k)$ est un vecteur de contrôle des entrées, $x_v(k)$ est l'état du robot et $v_v(k)$ est un vecteur décrivant les erreurs dans le mouvement. Le bruit dans le mouvement est généralement décrit comme ayant

une moyenne nulle.

Les robot requièrent des capteurs qui peuvent mesurer la position relative des marques (ou des éléments) par rapport au robot. Les observations, comme le mouvement, sont caractérisés par l'incertitude. La sortie des capteurs pour la i^{eme} marque, $z_i(k)$, est donnée par :

$$\begin{aligned} z_i(k) &= H_i x_a(k) + w_i(k) \\ &= H_{pi} p_i - H_v x_v(k) + w_i(k) \end{aligned} \tag{2.14}$$

où $x_a(k)$ est un vecteur contenant les états du véhicule et des marques, $w_i(k)$ est un vecteur pour mémoriser les erreurs d'observation, généralement avec une moyenne nulle comme avec le modèle du mouvement, H_i est le modèle d'observation qui cartographie les sorties de capteurs $z_i(k)$ au vecteur d'état du robot et p_i est la position absolue de la i^{eme} marque.

Les techniques du filtre de Kalman "vanilla" requière un modèle de processus linéaire muni de bruit gaussien. Le filtre de Kalman étendu a été développé pour étendre ses capacités vers des processus non-linéaires. Les filtres de Kalman linéaires utilisent les dérivations partielles du processus et des fonctions de mesures pour continuellement linéariser aux alentours des estimations courantes. Cette technique permet au filtre de Kalman originel d'être étendu vers des systèmes simples non-linéaires.

Les méthodes standards des filtres de Kalman ne peuvent traiter la détection de deux marques indistinguables à des positions différentes de l'environnement. La détection des deux marques induit une distribution multimodale représentant la localisation de cette marque. Ceci est incompatible avec l'hypothèse d'un bruit unimodal gaussien du filtre "vanilla". Les implémentation les plus répandues ignorent simplement les élément de l'environnement qui ne seraient pas uniques, avec le désavantage certains de perdre de précieuse informations à propos de l'environnement.

2.1.5 Les filtres particulaires

Les filtres particulaires représentent la croyance de position d'un robot en utilisant un ensemble d'exemples ou de particules distribuées en accord avec la distribution future des positions du robot. Dans le domaine de la localisation robotique ces méthodes sont souvent définies comme étant des méthodes de Monte Carlo de localisation. Plus que d'approximer les localisations futures, ces techniques utilisent une distribution de particules pondérées qui approximent la distribution désirée. Ceci élimine toutes les restrictions à propos de la forme ou des caractéristiques des distributions futures.

Les filtres particulaires utilisent récursivement des filtres de Bayes pour estimer la distribution future des positions du robot. Le filtre estime la probabilité de densité à travers l'espace d'état. La croyance de la position est donnée par l'équation 2.1.

Comme vu dans le premier chapitre, les capteurs peuvent nous retourner deux types de grandeurs, proprioceptifs et extrioceptifs. Donc, des données internes au robot et des données externes celles de l'environnement.

En faisant cette distinction, et en exploitant quelques corolaires et en répétant des intégrations, nous pouvons arriver à la mise à jour récursives du filtre de Bayes. Le corolaire de Markov, qui énonce que les états futurs sont indépendants des états passés tant que l'état courant est connu, est très utile. En l'utilisant plusieurs fois, l'équation de mise à jour pour les positions possibles du robot, $Bel(x_t)$, obtenue est :

$$Bel(x_t) = \eta p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{2.15}$$

où, η est une constante de normalisation, o_t est un ensemble d'observation (perceptions externes) au temps t et a_{t-1} est l'ensemble des perceptions internes au temps $t-1$. Cette équation est la base de nombreux algorithmes de la localisation à base de Monte Carlo.

Si les mises à jour sont seulement basées sur des perceptions internes, la croyance en la position va graduellement s'étaler dû à l'accumulation de l'incertitude dans le mouvement du robot. La figure 2.1 montre une progression d'un exemple d'approximations de la position d'un robot lorsqu'il suit une chemin rectangulaire. Le robot démarre correctement, mais deviens de moins en moins certain de sa localisation après chaque mouvement. La forme de la distribution démontre une caractéristique importante du modèle, les incertitudes dans les mouvement angulaires sont plus critiques que les incertitudes dans les mouvements de translation.

Le composant $p(o_t|x_t)$ de l'équation 2.15 est la clé pour éviter le déclin infini dans l'approximations de la position du robot car il intègre des probabilité sur des observations externes à même de "corriger" les erreurs de progression.

Une fois que le robot possède une représentation partielle de son environnement, les observations répétées en utilisant les capteurs du robot font que les particule se localise des positions plausibles, les particules indésirables disparaissent. Les robots peuvent se localiser rapidement dans de grand environnement complexes avec un temps de traitement raisonnable. Un autre avantage de la méthode et de pouvoir adapter l'algorithme avec la puissance des machines, plus la machine est puissante, plus les résultats sont précis.

La principale critique que l'on pourrait émettre et le fait que les filtres particulaires "ne sont que" des algorithmes de localisation, ils ne peuvent pas effectuer de cartographie.

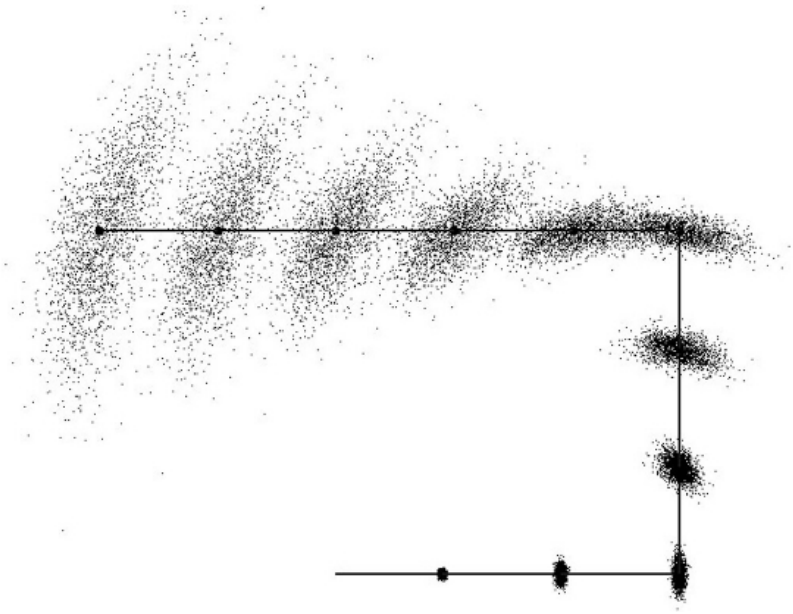


FIG. 2.1 – Phénomène d'étalement de la croyance de la position en raison de la grande incertitude.

2.2 Approches connectionnistes

L'approche connectionniste pour la localisation d'un robot dans l'environnement est surtout représentée par les cartes auto-organisatrices et la plus célèbres d'entre elles, celle de Kohonen. Ce type de localisation est surtout utilisé dans les espaces fermés (pièces). Il s'apparente le plus souvent aux méthodes de reconnaissance de caractères.

2.2.1 Localisation par carte auto-organisée de Kohonen

Le point commun entre la reconnaissance de caractères et la localisation dans l'espace fermé sont les trois contraintes d'invariances dans l'espace, d'invariance de translation et d'invariance de rotation. Ce constat fait, l'image issue de la perception de la pièce (le plus souvent grâce aux capteurs ultrasons) est considérée comme étant un caractère. Tout comme la reconnaissance de caractères, nous supposons que le plan détaillé des pièces est connu et stocké en mémoire. Dés lors, le principe est simple : La carte de Kohonen renvoi le plan réel le plus ressemblant à l'image bruitée des capteurs.

Janet et al [JAJ96] se sont intéressés à cette question. Leurs objectif est de permettre à un robot une localisation dite globales (donc en un environnement restreint et fermé) en utilisant les cartes de Kohonen. Ils considèrent un robot muni de capteurs ultrason.

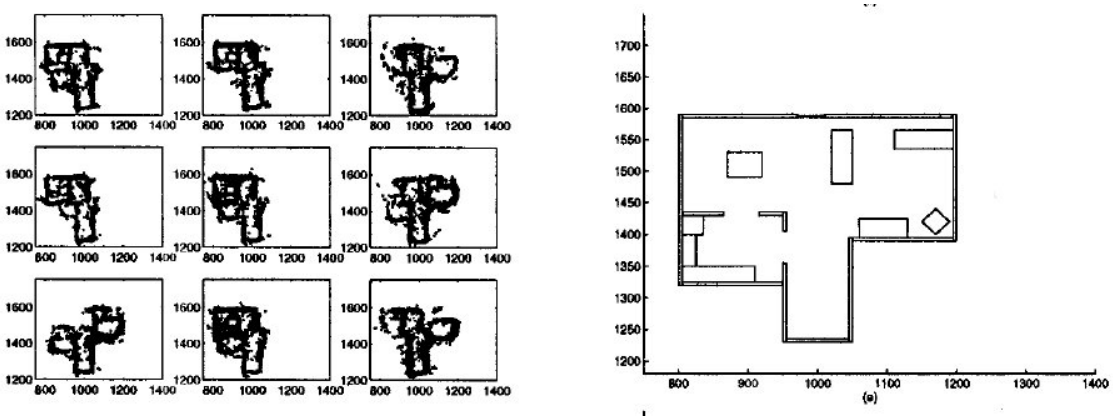


FIG. 2.2 – A droite, la plan réel de la chambre. A gauche, le plan simulé

Le robot étant simulé, il était donc nécessaire de simuler le bruit dû aux capteurs. Pour cela, ils induisent aux valeurs réelles des distances une erreur obéissant à une loi gaussienne 6σ. La figure 2.2 illustre le plan d'une chambre et ses équivalents simulés après perception par des capteurs ultrasons. De plus, puisque le déplacement se fait sur un principe odométrique, les auteurs ont introduits un bruit gaussien 6σ sur les croyances du robot que cela soit dans la rotation (l'erreur dans $[-\pi, \pi]$) ou dans la translation (l'erreur est dans $[-1, 1]$ pour les coordonnées).

2.2.1.1 La stratégie

Les auteurs utilisent plusieurs cartes de Kohonen de 21 par 21 éléments. Donc la carte peut représenter 441 vecteurs descriptifs (un vecteur par neurone). Malheureusement, les auteurs ont été très discrets sur la structure du neurone et ce qu'il représente réellement, nous ne pouvons qu'émettre des hypothèses sur son contenu pour la suite.

Chaque carte de Kohonen est utilisée pour mémoriser une chambre. Il y a donc 10 cartes de Kohonen qui subiront le même processus d'apprentissage. La base d'apprentissage de chaque carte est constituée des images bruitées des chambre équivalentes.

A coté du processus standard d'apprentissage de la carte, les auteurs comptabilisent les fréquences d'activation des neurones. Cette comptabilité est utilisée plus tard dans un processus qu'ils nomment *pruning* et qui consiste à mieux utiliser les neurones les moins sollicités (optimisation).

Pour tester la carte, des données lui seront fournis par le robot virtuel comme si celui-ci se déplace à travers la pièce. Trois étapes de préparation de données sont comptabilisées.

2.2.1.2 Entrainement de la carte

L'entrainement de la carte se fait au moyen de l'algorithme standard. Dans un premier temps, Un neurone gagnant est choisi qui représente le neurone le plus proche au vecteur d'entrée :

$$||X - W_w|| = min_{i,j=1,2,..,21}||X - W_{ij}|| \tag{2.16}$$

Pour bien contrôler l'apprentissage de la carte, les auteurs ont tablé sur un facteur d'apprentissage décroissant avec le temps, c'est un processus de raffinage.

$$\eta_\zeta = (\frac{n_0 - n_f}{\zeta_{max}})\zeta - \eta_0 \tag{2.17}$$

La mise à jour des neurones se fait selon l'équation standard suivante :

$$W_w^{k+1} = W_w^k + \eta_\zeta[X - W_w^k] \tag{2.18}$$

Pour obtenir un apprentissage plus efficace, l'impact sur le voisinage est une fonction exponentielle inverse facteur de la distance du voisin par rapport au neurone vainqueur.

$$\eta_N = (\eta_\zeta)(0.7^B) \tag{2.19}$$

L'équation de mise à jour des poids choisies par les auteurs est donc du style :

$$W_{N_w}^{k+1} = W_N^k + \eta_N[X - W_{N_w}^k] \tag{2.20}$$

2.2.1.3 Rappel

Cette phase consiste à trouver la carte de Kohonen dont les données sont les plus proches des données d'entrées produites par les capteurs. Les données sont présentées à chaque carte, pour chacune d'entre elles, un calcul de distance est effectuer. Il s'agit de la sommation des distances entre chaque neurone et sous-ensemble de données le représentant. La quantité $\sum_{rk}$ représente la distance de la carte k par rapport aux données présentées.Pour décider de la carte la plus représentative des données, nous considérons le $\sum_{rk}$ minimum :

$$\Sigma_{rw} = min_{k=1..21}\Sigma_{rk} \tag{2.21}$$

Le robot considère qu'il est dans la chambre représentée par la chambre w qui a retourné la distance $\sum_{rw}$.

2.2.1.4 Préparation des données

Cette étape a été intégrée au processus de localisation par les auteurs dans l'espoir d'améliorer les capacité de reconnaissance de leurs système. Ils ont recours à trois processus distincts :

Gross shifting Cette étape consiste à garantir une invariance de la translation. C'est une sorte de normalisation où les données de la chambre sont ramenée à l'origine en effectuant une soustraction du vecteur de déplacement.

Fine shifting Cette étape consiste à essayer de matcher le contenu de la carte avec les données du robot. En effet, puisque la carte contient les informations bruitées des chambres et que le robot bouge pendant l'acquisition des données, il n'est pas certain que les données qu'elle contient actuellement soit dans une plage de données identique à celle des données du robot. Donc, la carte elle même subie une adaptation par translation par rapport à l'origine.

Pruning Tout les neurones n'entre pas forcément en compte (ou alors faiblement) pour la reconnaissance d'une chambre. Cette inactivité peut même pénaliser le processus de rappel en bruitant $\sum_{rk}$. Il s'agit d'ignorer les neurones dont les fréquences d'activation sont inférieur à un certain seuil.

2.2.1.5 Critiques

La méthode de localisation par carte de Kohonen présentée par [JAJ96] est bien représentative de ce qui se fait dans le domaine. La principale critique qu'on pourrait lui reprocher est sa rigidité. En effet, la méthode ne fonctionne qu'en milieu clos, pour des espaces restreints et de plus pour un nombre limité d'endroits. Par exemple, si le système gère 10 cartes de Kohonen, il ne serait pas possible de stocker une onzième chambre.

Le second défaut est que le système est incapable de détecter un obstacle en mouvement. Il ne peut pas se représenté un chat qui traverserait la chambre, d'où les grands risques de collisions que cela pourrait occasionner.

Nous concluons que cette méthode ne présente pas les critères nécessaires pour une navigation autonome d'un robot mobile dans un environnement dynamique et incertain.

2.3 Approches floues

La localisation dans les approches floues est indissociable de la navigation elle même. Plutôt que de parler de localisation et d'omettre de parler de la prise de décision quant au mouvement, nous avons décider de regrouper le processus perception-localisation-déplacement dans une même section à savoir la section 3.4 du troisième chapitre.

Chapitre 3

Approches de planification de chemins

La planification de chemins a pour but de *définir à partir d'une représentation de l'environnement, un chemin de risque de collision nul et satisfaisant certains critères d'optimisation.*

La navigation et la planification de chemins dans un environnement parfaitement connu est un sujet bien maitrisé. Ceci n'est pas le cas pour un environnement inconnu. L'obtention d'une carte parfaite, totalement conforme à la réalité du terrain, est pour le moment une chose utopique. Ceci est du au bruit qui est généré lors de la phase de perception (intrinsèque au capteurs) ainsi qu'à la nature heuristique ou imprécise de certains algorithmes de cartographie (algorithmes génétiques, réseaux de neurones, logique floue ...). Dans ce contexte, planifier un chemin sûr et sans collision est une tâche difficile.

La première stratégie naturelle qui se présente à l'esprit pour planifier un chemin est de sélectionner un chemin à partir des connaissances présentes, évoluer un certain temps sur ce chemin tout en collectant de nouvelles informations, re-planifier un nouveau chemin à la lumière des nouvelles connaissances et ainsi de suite. C'est une stratégie extrêmement répandue dans la littérature [KHS03][SS96][Fri07] et basée sur le local Path-Planning, c-à-d qu'à chaque instant, le robot déduit son chemin sur une représentation locale de l'environnement. Cette méthode est utilisé lorsque la portée des capteurs n'est pas suffisante à une cartographie globale de l'environnement.

D'autres stratégies sont basées sur le global Path-planning. Le robot possède une cartographie complète de l'environnement. Dans ce cas, la planification de chemin s'apparente à la recherche de chemin dans un graphe (Traveler Sale Problem). Le problème étant NP-complet, différentes stratégies sont envisageables. L'un des plus connues sont les Algorithmes Génétiques.

3.1 Approches évolutives

Dans la littérature, beaucoup d'auteurs se sont penchés à la manière d'utiliser le paradigme évolutionniste pour résoudre le problème de planification de chemins. Le problème de recherche de chemins est un problème NP-Complet. L'espace de recherche est l'ensemble de toutes les configurations de chemins possibles dans l'environnement. Généralement, en entrée, l'algorithme exige une cartographie globale de la topologie et des obstacles dans l'environnement. En sortie, l'algorithme génère un chemin supposé optimal (ou proche de l'optimalité).

Ces approches sont généralement peu utilisées dans la Robotique Mobile Autonome car il est difficile de générer une cartographie globale de l'environnement, il faut pour cela que le robot soit muni, par exemple, d'un système de positionnement pas satellite (GPS), d'auxiliaires officiants depuis une altitude plus élevée (caméra sur toit, drones ou avions de surveillance) ou alors de Radars.

3.1.1 Apercu rapide de l'approche évolutionniste

L'approche évolutive s'inspire de l'évolution naturelle pour explorer un grand espace de recherche de solutions. Haupt et al [RH04], décrivent l'analogie entre l'approche évolutive numérique et biologique comme montré dans la figure 3.1.Les algorithmes génétiques utilisent un codage des paramètres, et non les paramètres eux mêmes. Ils travaillent sur une population de points, au lieu d'un point unique et ils n'utilisent que les valeurs de la fonction étudiée, pas sa dérivée, ou une autre connaissance auxiliaire.

Résoudre un problème en AG revient à satisfaire une contrainte (recherche d'un maximum ou d'un minimum local ou global d'une fonction) dans un problème d'optimisation. Sa force réside dans la multiplicité des chemins de recherche qui est d'autant plus grande que la population s'accroit. Chaque individu représente à lui seul un chemin de recherche. Comme dit précédemment, l'évolution de cette population est calquée sur le vivant. L'algorithme génétique comporte 5 grandes phases.

Initialement, une population d'individus possédant des gènes aléatoires est créée (naissances). On évalue leur performances (fitness) relatives. Pour varier les chemins de recherches, un croisement s'avère nécessaire. Puisque dans la nature seuls les individus les plus robustes s'accouplent (en adéquation avec les théories Darwinienne), une phase de sélection est nécessaire. Les individus dont la fonction de fitness est la plus proche de l'extremum voulu (minimum ou maximum) sont choisis. Un croisement s'opère entre ces individus générant de nouveaux individus (enfants). Toujours selon les lois Darwinienne,seuls les plus forts restent, donc les individus dont le fitness se trouve loin de l'extremum considéré sont détruits. Nous répétons le processus : sélection, croisement, destruction un nombre suffisant de fois (chaque étape est appelée génération). Pour éviter à l'algorithme d'être piégé dans un extremum local,

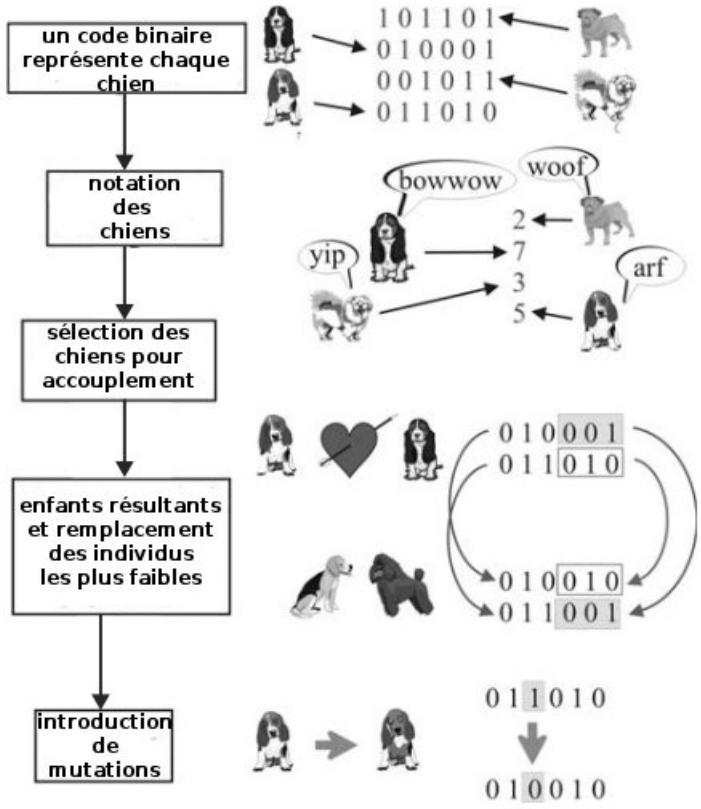


FIG. 3.1 – Analogie entre génétique numérique et génétique biologique.

un opérateur de mutation est introduit, il change arbitrairement une ou plusieurs valeurs dans les gènes d'un individu prit au hasard. La cardinalité de la population reste constante le long du processus (nombre de naissances = nombre de morts).

3.1.1.1 La codification

Cette phase est représentée par une fonction qui transforme une entité depuis l'espace du problème vers l'espace utilisé par l'algorithme génétique. Dans ce qui concerne ce chapitre, c'est généralement un chemin qui sera transformé depuis son tracé vers des valeurs numériques utilisées par l'algorithme génétique. Chaque auteur propose généralement une manière innovante de coder le chemin. Le codage peut être binaire, multi-caractères (généralement le plus utilisé pour la planification de chemins) ou sous forme d'arbres [Mit98]. Remarque, le choix d'une codification est un choix critique pour le bon déroulement de l'algorithme.

3.1.1.2 La calcul de fitness

La première ligne de ce chapitre est une définition de la planification de chemins : " La planification de chemins a pour but de *définir à partir d'une représentation de l'environnement, un chemin de risque de collision nul et satisfaisant certains critères*

d'optimisation". Il existe pour chaque problème des milliers de chemins possibles, cependant, il est possible de créer une métrique pour les différencier et de quantifier leur qualité.

Noter des individus (chemins) est le fait d'attribuer **automatiquement** un fitness (une note) à la base du code que renferme cet individu au travers d'une formule mathématique (fonction de coût). Pour le cas des chemins, la formule prendra en compte des paramètres aussi divers que, la distance, le nombre de collisions, le nombre de virage, l'énergie gaspillée etc ... Remarque, le choix d'une bonne fonction de fitness est un choix critique pour le bon déroulement de l'algorithme.

3.1.1.3 La sélection

La sélection est un opérateur pour les algorithmes génétiques. Il permet de déterminer les couples d'individus les plus aptes à s'accoupler (transmission de gènes vers la prochaine génération) selon leur fitness. Cette phase se fait dans l'espoir qu'au moins un enfant des enfants issus des croisements soit de fitness supérieur à celui de ses parents. La sélection est généralement de nature probabiliste mais favorise les individus aux meilleurs fitness.

Parmi les politiques de sélection, la sélection par roulette, où chaque individu représente une portion de la roulette d'autant plus grande que son fitness est meilleur, en faisant tourner la roulette, les individus occupant les plus grandes portions sont favorisés. Une autre politique est la sélection par rang qui choisit toujours les individus possédant les meilleurs scores d'adaptation, le hasard n'entre donc pas dans ce mode de sélection. Certains algorithmes utilisent la sélection par tournoi. Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit pour ces paires l'individu qui a le meilleur score d'adaptation[Mit98].

D'autres méthodes de sélection existent dans la littérature spécialisée tel que la sélection de Boltzmann, l'élitisme ou la sélection Gamma. Elles ont des utilisations particulières pour des types très spécifiques de problèmes.

3.1.1.4 Le croisement

L'opérateur de croisement est sûrement l'opérateur le plus connu dans ce paradigme de l'IA. Le croisement à point unique en est la forme la plus simple : une position de croisement unique est choisi aléatoirement(un seul bit), cette partie est échangée entre les deux parents pour générer deux enfants. L'idée est élégante mais très réductrice. Cette technique a pour défaut de n'inter-changer qu'un seul bit, lorsque les bits à inter-changer sont les mêmes (ce qui arrive dans 50% des cas) le croisement s'avère sans efficacité.

Pour palier au problème de position unique, il a été envisager d'inter-changer des portions entières de code. Soit deux individus A=10110011 et B=01001101. La première étape du croisement consiste a choisir aléatoirement (et en nombre

aléatoire) les indices des positions à inter-changer. Supposons dans l'exemple le choix des indices aléatoires : 2,5,8. La deuxième étape consiste à inter-changer les bits des parents A et B aux positions indexées précédemment, c-a-d à inter-changer le deuxième, le cinquième et le dernier bit de chaque parents. Cela génère deux enfants, E1 et E2 tels que E1 = 11111011 et E2 = 0000101.

Ce qui précède n'est pas une liste exhaustive des méthodes de croisements possibles. On ne peut pas juger de la qualité d'une méthode par rapport à une autre, il semblerait que tout problème possède un degré d'affinité avec une des méthodes de croisements possible [Mit98].

3.1.1.5 La mutation

A l'instar de tous les algorithmes d'optimisation et de recherche d'optimum, les AG se heurtent au problème crucial des optimums locaux. Généralement, les AG sont utilisés pour la recherche d'optimum de fonctions fitness complexes, multivariées, donc impossible à résoudre analytiquement (ou demande un temps de calcul astronomique). Le second opérateur des AG, la mutation, sert justement à sortir des optimums locaux et à explorer un autre chemin de recherche.

La mutation d'un individu peut être réalisée de différentes manières. La fréquence de mutation dans une population est un paramètre important pour l'AG. Elle est habituellement basse (une mutation toutes les 10 générations par exemple). L'individu mutant représente une nouvelle voie d'exploration qui pourrait tirer la population vers un nouvel optimum de meilleur qualité que le précédant en transmettant ses gènes à travers l'opérateur de croisement génétique vu précédemment. Lorsque, statistiquement, il est décidé d'opérer à une mutation dans une génération donnée, la première étape consiste à choisir un individu de manière aléatoire parmi la population. Pour cet individu, un gène est choisi aléatoirement pour modification. Comme déclarer en introduction, l'opérateur se fera sur un codage binaire, la mutation consiste alors en l'inversion d'un bit aléatoire dans le gène. Cette simple modification, génère un individu nouveau, son fitness est recalculé [Mit98].

3.1.2 Utilisation de l'approche évolutive dans quelques travaux

Pour illustrer l'utilisation des approche évolutive pour la planification des chemins, une sélection de trois travaux ont été comparés les unes par rapport aux autres dont les différents choix (par rapport à la codification, et les paramètres des opérateurs de l'algorithme génétique)effectués par les auteurs pour obtenir les chemins optimaux. En fin de section, Une critique sera émise pour chacun des travaux.

Remarque que le travail de Sughihara et Smith [SS96] considère un environnement et un mouvement en 3D. Cependant, les auteurs effectuent des projection de l'environnement sur les plans xy et xz. Ils appliquent à ces deux plans une planification 2D par algorithme génétiques dans le sens classique du terme. Le chemin

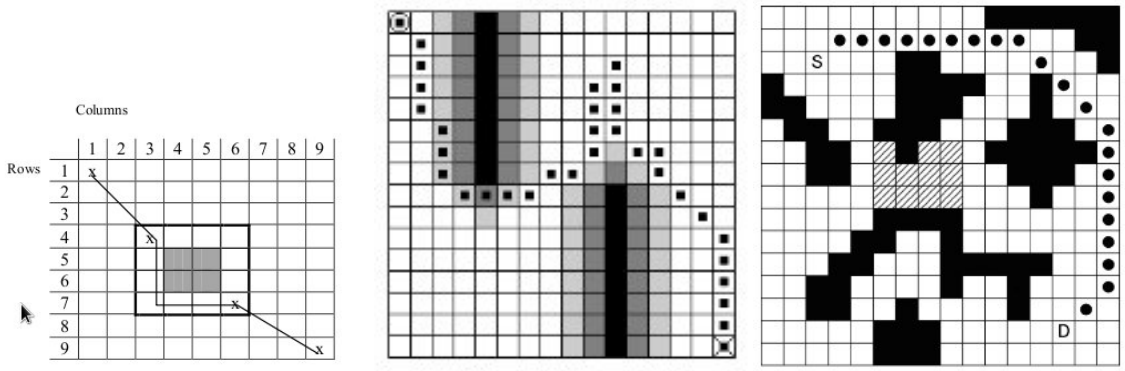


FIG. 3.2 – La représentation de l'environnement par [KHS03][SS96][Fri07].

en 3D est reconstitué en reconstituant le chemin optimal dans l'espace à partir des meilleurs chemins obtenus séparément dans les deux sous-plans xy et xy. C'est la planification des chemins dans ces deux sous plans dont il est objet par la suite.

3.1.2.1 La représentation de l'environnement

Dans les cas les plus simples de l'utilisation de l'approche évolutive dans la littérature, l'environnement est décri comme étant fermé, rectangulaire, représenté sous forme d'une grille. Les obstacles sont signalés par des cases noires, le reste par des cases blanches [Fri07][SS96][KHS03], certains auteurs comme [SS96] expriment la difficulté du terrain par une nuance grise d'autant plus sombre que la difficulté est grande (jusqu'à devenir noire "un obstacle"), la Figure 3.2 illustre ces représentations.

3.1.2.2 La codification des chemins

La codification des individus est obtenues le plus souvent en analysant le mouvement du robot dans l'environnement. Les auteurs Kamran et al[KHS03], identifient deux méthodes de mouvements pour leur robot. Le "Row-Wised Movement"(RWM) et le "Column-Wised Movement"(CWM). Le RWM (resp CWM) part du principe que pour se déplacer dans l'environnement, le robot traverse les lignes (resp les colonnes) de la grille par un seul point et ne retourne jamais en arrière (resp à gauche). Un drapeau ou gène(booléen) dans le chromosome de l'individu permet de préciser si le chemin est en RWM ou en CWM. L'approche des auteurs permet aussi de mixer les deux méthodes de déplacement. Des gènes dans le chromosome permettent de spécifier à quel position le mouvement change (Path Switch), c-à-d quand est ce qu'il passe de RWM vers CWM ou vice-versa. Les auteurs précisent qu'il n'est possible de changer que deux fois de méthodes pendant le trajet.

Sughihara et Smith [SS96], codifient en binaire les chemins possibles. Ils intègrent aussi la notion de Row-Wised Movement qu'ils nomment x-monotone et Column-Wised Movement qu'ils nomment y-monotone. Ils assument qu'un trajet ne peut

être que x ou y-monotone. Le premier bit du génome de chaque individu précise si justement le chemin est x ou y-monotone. Le reste du chromosome est une suite de (n-1) couples (direction,distance) pour un chemin de n cases. Le couple est de taille 3+log(n+1) bits. Les deux premiers bits désignent la direction de la prochaine case (00,01,10,11) pour (vertical, diagonale supérieure, horizontal, diagonale inférieure) pour y-monotone et (horizontal, diagonale gauche, vertical, diagonale droite) pour x-monotone. Les bits suivants n'ont de signification que lorsque la direction est 00 et représente la distance en entier signé.

Comme pour Sughihara et Smith, Fries [Fri07] codifie les chemins en binaire. Les chemins sont strictement x-monotones (CWM). De ce fait, le chromosome n'est constitué que de couples (direction,distance). Puisque le chemin est strictement x-monotone, le choix des nouvelles cases se résument à seulement trois (codées 00,01 et 10). Le reste des bits du couple codent la distance parcourue dans la direction choisie. Pour une grille de nxn, le maximum de distance qui peut être parcouru est $n-1$. Pour coder $n-1$ en binaire, il faut $Log_2(n-1)$ bits, la longueur totale d'un couple est de $2 + Log_2(n-1)$.

3.1.2.3 L'estimation des chemins

Les auteurs Kamran et al[KHS03], effectuent une évaluation de la population des individus à chaque itération. Leur fonctions se base sur trois paramètres, le nombre de collision, la distance du chemin et le nombre de détours. Puisque les chemins peuvent être changer de x-monotone à y-monotone et vice versa, ils peuvent être muni de détours(virages) les auteurs ont pris le nombre de détour comme étant un malus puisque le robot perdra beaucoup plus de temps sur un chemin riche en virages. Dans l'équation, le paramètre f_{length} dénote d'une quantité entre 0 et 1 qui représente la distance du chemin. Le chemin le plus court dans la population correspond $f_{length} = 1$, le chemin le plus long $f_{length} = 0$, les chemins restants sont échelonnés entre 0 et 1 selon leur propre taille. Le paramètre $f_{collision}$ dénote aussi d'une quantité entre 0 et 1 où le chemin qui possède le maximum de collisions à un paramètre $f_{collision} = 0$ et celui qui possède le minimum de collisions $f_{collision} = 1$, les chemins restants sont échelonnés entre 0 et 1 selon leur nombre de collisions. Finalement, le paramètre $f_{numbreofTurns}$ dénote d'une quantité entre 0 et 1 où le chemin qui possède le minimum de virages reçoit $f_{numbreofTurns} = 0$, celui qui possède le maximum de virages $f_{numbreofTurns} = 1$ les chemins restants sont échelonnés entre 0 et 1 selon leur nombre de virages. Nous obtenons des individus dont le fitnesse est entre 0 et 1 selon l'équation :

$$f_{path} = f_{collision}.[L.f_{Length} + T.f_{numbreofTurns}]/\frac{100}{L+T} \tag{3.1}$$

Sughihara et Smith [SS96], décrivent une formule de coût plus simple. Ils considèrent le chemin comme étant pondéré. Comme évoqué plus haut, les auteurs considèrent des niveaux d'obstacles différents représentés par leurs niveaux de gris, plus une case

est sombre plus l'obstacle est difficile est plus le poids de cette case est proche de 1. Pour chaque individu qui représente un chemin de taille n et dont l'obstacle le plus difficile est de poids w_{max}, le fitness est de valeur :

$$f_{path} = (1 + w_{max})n^2 \tag{3.2}$$

Dans son papier,Fries [Fri07] ne mentionne pas directement l'équation utilisée pour calculer le fitness des individus. Cependant, il précise que l'équation est fonction de la longueur du chemin et de la difficulté de terrain au centre de toutes les cases traversées par le trajet. L'auteur relate la difficulté d'estimer la difficulté du terrain au sein des cases. Il utilise pour cela une approche floue basée sur des variables linguistiques qui génère des nombres floues triangulaires. Le résultat de la fuzzification est une valeur entre 0 et 1 qui détermine la difficulté du terrain à cette case. La difficulté totale du terrain est surement obtenue par une moyenne pondérée des difficultés aux seins des cases.

3.1.2.4 La sélection de chemins

Les auteurs Kamran et al[KHS03], utilisent une sélection par rang (voir section 3.1.1.3), donc il n'y pas de hasard et c'est toujours les meilleurs individus qui sont choisis pour le croisement. Sughihara et Smith [SS96] ne cite pas directement la méthode de sélection utilisée, cependant, il mentionne dans leur survol des algorithmes génétiques que : *"Quelques individus sont choisis autant que parents en se basant sur leur valeur de fitness"*, ce qui laisse penser qu'il y a une part de hasard (sinon ils auraient dit que les individus au fitness maximum sont choisis) d'où une sélection par la roulette, toute autre méthode exotique aurait été mentionnée par les auteurs. Fries [Fri07] aborde la question de la sélection dans une seule phrase dans son article où il déclare que : *"Les meilleurs parents sont choisis pour croisement et création d'enfants"*, ce qui laisse penser que la méthode de sélection est une sélection par rang.

3.1.2.5 Croisement de chemins

Généralement, le croisement entre chemins se fait directement en échangeant du code entre les parents pour générer les enfants. L'information échangée doit essentiellement porter sur la direction et la distance plutôt que d'autres informations. Par exemple, Kamran et al [KHS03], échangent essentiellement les informations codant les distances et directions et précisent que c'est un non sens que d'échanger des informations tel que les "switch points".

En échangeant le code entre deux parents avec un croisement en un seul point, Kamran et al génèrent deux enfants. Sugihara et Smith utilisent une méthode analogue (point de croisement unique) pour croiser les chemins. Cependant, une question se pose sur la validité d'un croisement entre deux chemins de monotonie différentes.

En effet, qu'elle apport générationnel peut apporter un croisement entre deux parents dont l'un est x-monotone et le second y-monotone ?

Fries effectue aussi un croisement en un point unique. Il précise que le taux de croisement γ qui contrôle le pourcentage de parents concernés par le processus est de 0.8. Ce taux a été obtenu par l'expérience.

3.1.2.6 Mutation

Kamran et al effectuent la mutation des individus en parcourant un à un tous les gènes des chromosomes, une variable aléatoire décidera s'il faut ou non altérer le gène en question. La mutation peut altérer aussi bien une information de direction, de distance, les drapeaux et même (contrairement au croisement) altérer les "switch points". Les auteurs appliquent aussi le principe d'élitisme qui consiste à sauvegarder le patrimoine du meilleur individu en le copiant tel qu'il est (clonage) dans un enfant pour le préserver de la mutation.

Sugihara et Smith effectuent des mutation avec un taux non déterminé. Dés que le processus de mutation a été décidé pour un individu, un bit de son patrimoine génétique est aléatoirement choisi pour être inversé.

Fries, situe la probabilité de mutation de chaque bit à $\nu = 0.02$, une grandeur qu'il a obtenu empiriquement. C'est à dire qu'à chaque bit, il décide avec une probabilité de 0.02 s'il doit être inverser ou non. C'est une méthode assez simple et efficace vu la simplicité du code génétique des individus.

3.2 Approches par champs de potentiel

Les potentiels de champs [HJ08] est peut être l'approche la plus utilisée dans la planification d'un chemin. Cela est dû au fait de sa simplicité d'implémentation, de paramétrage (par rapport à un algorithme génétique par exemple) et de sa vitesse d'exécution. Cette méthode utilise une analogie du mouvement d'une particule chargée au sein d'un champs électrostatique. Cette section va introduire des généralités physiques en rapport aux champs électrostatique et aux potentiels de champs. Elle traitera aussi des différentes méthodes présentes dans la littérature pour palier aux inconvénients de la méthode tels que les minimas locaux.

3.2.1 Généralités

En physique, on désigne par champ électrique un champ créé par des particules électriquement chargées. Un tel champ permet de déterminer en tout point de l'espace la force électrique exercée à distance par ces charges. Dans le cas de charges fixes dans le référentiel d'étude le champ électrique est appelé champ électrostatique. Lorsque les charges sont en mouvement dans ce référentiel il faut y ajouter un champ

électrique induit dû aux déplacements des charges pour obtenir le champ électrique complet.

Toute particule chargée immobile crée un champs électrostatique $\vec{E}$. Son expression est directement issue de la loi de coulomb. Si l'on considère une charge q comme source du champ, celui-ci est orienté des sources vers le point considéré et a pour valeur :

$$E = \frac{1}{4\pi\epsilon}\frac{q}{r^2} \tag{3.3}$$

Le champs électrostatique est aussi directement lié au potentiel électrique V par la formule 3.4. On dit que le champs dérive d'un potentiel par l'intermédiaire d'un gradient.

$$\vec{E} = -\vec{\nabla}V = -\frac{\partial V}{\partial x}\vec{i} - \frac{\partial V}{\partial y}\vec{j} - \frac{\partial V}{\partial z}\vec{k} \tag{3.4}$$

Dans le vide, deux particule chargée électriquement exercent l'une sur l'autre une force. La loi de Coulomb décrit cette force et s'énonce comme suit : L'amplitude de la force électrostatique entre 2 charges est proportionnelle au produit des deux charges et est inversement proportionnelle au carré de la distance entre les deux charges. Elle aussi issue du produit direct entre la charge de la première particule par la valeur de champs électrostatique généré par la seconde charge :

$$F_{12} = q_1\ E_2 = \frac{q_1 q_2}{4\pi\epsilon_0 r_{12}^2} \tag{3.5}$$

Il est parfois nécessaire dans quelques applications de disposer de la direction de cette force. L'équation précédente possède une version vectorielle. Elle permet de précisément déterminer l'orientation de l'action d'une force au sein du champs électrostatique.

$$\vec{F}_{12} = \frac{q_1 q_2}{4\pi\epsilon_0||\vec{r}_{12}||^2}\frac{\vec{r}_{12}}{||\vec{r}_{12}||} \tag{3.6}$$

$$\vec{F}_{12} = \frac{q_1 q_2}{4\pi\epsilon_0||\vec{r}_{12}||^3}\vec{r}_{12} \tag{3.7}$$

$$\vec{E_T} = \vec{E_1} + \vec{E_2} + \vec{E_3} + \ldots + \vec{E_n} = \vec{E_T}(M) = \sum_{i=1}^{n}\frac{q_i}{4\pi\varepsilon_0}\frac{\vec{P_iM}}{|\vec{P_iM}|^3} \tag{3.8}$$

$$\vec{F_T} = q_1\vec{E_T} = q_1\sum_{i=1}^{n}\frac{q_i}{4\pi\varepsilon_0}\frac{\vec{P_iM}}{|\vec{P_iM}|^3} \tag{3.9}$$

Les équations précédentes décrivent donc les forces exercés mutuellement par une ou plusieurs charges. Il est important de noter que la force sera attractive si les charges sont inverses, répulsive dans le cas contraire.

La force est aussi déductible (plus facilement) depuis la formule du champs de potentiel V. En effet, la force est (comme le champs) un dérivatif du potentiel, donc pour un point quelconque de l'espace vide, une charge exerce une force telle que décrite dans la formule suivante :

$$\vec{F} = -\overrightarrow{\text{grad}}\,(V) = -\overrightarrow{\text{grad}}\left(\frac{q}{4\pi\varepsilon_0\,|\vec{r}|}\right) = -\vec{\nabla}\left(\frac{q}{4\pi\varepsilon_0\,|\vec{r}|}\right) = \frac{q\vec{r}}{4\pi\varepsilon_0\,|\vec{r}|^3} \tag{3.10}$$

3.2.2 Application à la planification de chemin

Le principe de l'application des champs de force pour la planification de chemins est d'utiliser le phénomène d'attraction et de répulsion que subit une charge électrique à l'intérieur du champs. L'introduction de ce chapitre défini la planification de chemin comme étant la recherche d'un trajet (optimal) entre un point de départ et un point d'arrivé (but) tout en évitant d'entrer en collision avec les obstacles.

Cette méthode utilise l'analogie, le robot, le but et les obstacles sont tous représentés par des charges électriques. Le robot étant une charge, il doit être attiré par le but et répulsé par les obstacle. Donc, le robot possède une charge de signe contraire au but (attraction) et de même signe que les obstacle (répulsion) comme illustré dans la figure 3.4. La force résultante F_T est la somme des forces exercés par les obstacles et la force du but.

$$\vec{F_T} = \vec{F_o} + \vec{F_b} \tag{3.11}$$

A chaque moment, le robot subi une force exercée par les charges qui l'entourent selon l'équation 3.9. Il est possible à partir de cette force de calculer le mouvement du robot avec la variation du temps. Les équations suivantes définissent l'accélération subie par le robot sous l'influence de la force totale :

$$\vec{F_T} = m_{robot}\vec{\gamma} \tag{3.12}$$

$$\vec{\gamma} = \frac{1}{m_{robot}}\vec{F_T} \tag{3.13}$$

A partir de l'accélération, il est possible de calculer la vitesse du robot, par récursion, la position aussi est calculable à condition de posséder précisément les conditions initiales de vitesse et de position. Les équations suivantes expriment les relations existant entre les trois notions de la dynamique, l'accélération $\vec{\gamma}$, la vitesse $\vec{v}$ et la position $\vec{p}$:

$$\vec{\gamma} = \frac{\partial\vec{v}}{\partial t} \tag{3.14}$$

$$\vec{v} = \frac{\partial\vec{p}}{\partial t} \tag{3.15}$$

Dans la pratique, pour calculer les positions du robot sous l'influence de la force, il

faut avoir recours à la méthode de Runge-Kutta (ou la méthode d'Euler). Avec un pas suffisamment petit et de bonnes valeurs initiales, la simulation est généralement de bonne qualité.

$$\gamma_t = F_t / m_{robot} \tag{3.16}$$

$$v_t = v_{t-1} + \gamma_{t-1} \tag{3.17}$$

$$p_t = p_{t-1} + v_{t-1} \tag{3.18}$$

3.2.3 Représentations du champs de potentiel dans la littérature

Bien que les lois de l'électrostatique et des potentiels de champs soient parfaitement définis par la physique, leurs applications directes peuvent faire surgir quelques problèmes. Les charges électrostatique sont généralement de poids insignifiant. Ils n'est pas rare que l'application des lois de Coulomb fassent mouvoir ces particules à des vitesses et des accélérations gigantesques (de l'ordre de milliers de kilomètres par seconde ou seconde au carré). De tels accélérations sont tout à fait impensables pour un robot mobile autonome. De ce fait, les auteurs d'articles dans ce domaine, ont pris beaucoup de libertés à assigner au champs des formules canoniques imaginaires. Chaque auteur propose sa propre définition du champs à même de faire mouvoir le robot à des vitesse et des accélération envisageables dans la pratique quitte à être en contradiction avec la réalité.

L'équation 3.10 décrit la force qu'exerce une charge sur tout point vide (où il n'y a pas d'autre particule chargée) de l'espace. Par extension, l'équation 3.7 décrit la force exercée par une particule chargée sur une autre particule possédant aussi une charge non nulle. Ces deux formules peuvent être considérées comme les équations de force canoniques régissant les champs électrostatiques.

Quasiment tous les auteurs, au moment de décrire les formules des forces dans les champs électrostatiques, expriment deux équations de potentiels différents. Une équation pour les potentiels attractifs, une autre pour les potentiels répulsifs qui sont, le plus souvent, radicalement différentes. Pourtant, comme il a été démontré plus haut, il n'y a en valeur absolue aucune différence entre un potentiel attractif ou répulsif pour une même distance si ce n'est par le signe.

Le tableau 3.1 récapitule les formules utilisés par trois auteurs pour représenter les disparités par rapport aux formules réelles. Dans le tableau, F_a représente la force attractive, F_r la force répulsive. Dans la formulation de Chirikjian et Wang, d représente la distance par rapport au but, D_{ro} la distance par rapport au obstacle et a une constante. Dans la formulation de Koren et Borenstein, F_{ca} et F_{cr} sont deux constantes régulant respectivement la force attractive et la force répulsive. W la taille du robot, C_{ij} la "certitude" de la cellule (i,j). Cette formulation est appliquée à un environnement calqué sur une grille. La forme des obstacles remplie les cellules de la grille (cellules actives) plus ou moins complètement d'où le paramètre

Auteurs	Formules
Gregory S. Chirikjian,Yunfeng Wang [GSC00]	$F_a = \frac{1}{d}$ $F_r = \frac{1}{(aD_{ro})^n}$
Y. Koren, J. Borenstein [YK91]	$\vec{F_a} = F_{ca}\left(\frac{x_i - x_0}{d_t}\vec{i} - \frac{y_i - y_0}{d_t}\vec{j}\right)$ $\vec{F_r} = \frac{F_{cr}W^n C_{ij}}{d^n(i-j)}\left(\frac{x_i - x_0}{d(i,j)}\vec{i} - \frac{y_j - y_0}{d(i,j)}\vec{j}\right)$
Min Gyu Park, Jae hyun Jeon [MGP01]	$F_a = -K_d\lvert x - x_0\rvert$ $F_r = \begin{cases} K_r\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2} & \text{si } \rho < \rho_0 \\ 0 & sinon \end{cases}$

TAB. 3.1 – Formules de représentation des forces par différents auteurs

C_{ij}. Dans la formulation de Park et Jeon, K_d et K_r sont deux constantes régulant respectivement la force attractive et la force répulsive, x_0 la position du but, ρ la distance par rapport à l'obstacle, ρ_0 le seuil d'influence d'un potentiel.

Park et Jeon [MGP01] n'hésitent pas à inverser la formulation naturelle de la force. En effet, en physique, la force générée par une charge électrique est toujours inversement proportionnelle au carré de la distance. Cependant, dans leurs formulations, les auteurs relient avec une proportionnalité linéaire la force et la distance. En résumé, plus la grande est la distance, plus grande est la force. Même si Chirikjian et Wang [GSC00] relient avec une proportionnalité inverse la force à la distance, elle ne l'est pas avec un degré suffisant pour calquer la réalité. L'obstacle des formules réelles est que $1/d^2$ est problématique. En effet, lorsque la charge est trop loin du but, la force attractive est insignifiante, sinon écrasée par les forces répulsive. Inversement, lorsque la charge se trouve trop prés du but, elle est démesurée. Elle est soumise à de très grandes accélérations qui peuvent, dans le pire des cas, la faire entrer en collision avec des obstacles qui ne créent pas assez de force pour la freiner. Si les auteurs décrivent deux équations différentes pour la force attractive et répulsive, c'est dans le but d'avantager la force attractive. Dans tous les cas, le degré du polynôme décrivant la force répulsive et inférieur au degré du polynôme décrivant la force attractive.

3.2.4 Inconvénients de la méthode

Bien que facilement implémentable, cette méthode souffre d'incovénients contreignant. Le premier est de loin le plus connu est le problème de minimas locaux communs à tant de paradigmes. Le second est le problème dans les couloirs étroits

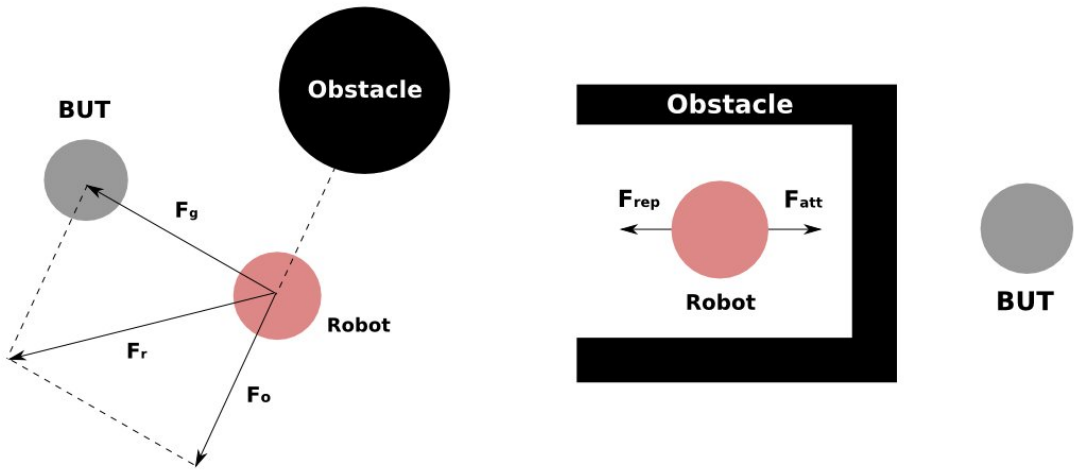


FIG. 3.3 – A droite, la résultante des forces en action sur le robot. A gauche, le problème du minima local

ou le robot entre en oscillation, le dernier et celui du passage dans les petits espaces.

3.2.4.1 Le piège du minima local

Il y a un risque non nul qu'à un certain moment, les forces exercées sur le robot s'annulent. Cette configuration se retrouve, comme illustré dans la figure 3.4, en présence d'obstacles en forme de U. La force de répulsion générée par cet obstacle s'annule avec la force d'attraction du but. Ceci forme un minima local. Dans le minima local, le robot s'immobilise en l'absence de force motrice. Pour éviter ce désagrément, deux approches sont envisageables, soit d'éviter de tomber dans un minima local (Local Minima Avoidance LMA), ou alors de s'échapper de ce minima (Local Minima Escape LME).

L'approche LMA tente d'obtenir une cartographie de champs, où il n'y a pas de minima local autre que celui du but, en modifiant les différents potentiels. Plusieurs méthodes sont envisageables telles que les fonctions de navigations et les fonctions harmoniques. Cependant, ces méthodes sont très gourmandes en temps de calculs et ne sont pas envisageables pour le temps-réel[MM08].

L'approche LME (Local minima Escape) ne tente pas d'éviter les minimas locaux. C'est une approche qui essaye d'extraire le robot du minimum une fois qu'il se trouve piégé. Il y a dans la littérature beaucoup d'approche utilisée pour arriver à ce but. La plus célèbre et celle des entités virtuelles. Le principe se base sur la génération d'obstacles[MGP03] ou de buts[ZXy03] virtuels. Lorsque le robot se retrouve dans un minimum local, cela sous entend que la sommation de toutes les forces exercées sur le robot est nulle. La méthode des entités virtuels vise à considérer un but (ou un obstacle) qui ne représente rien dans le réel perçu mais qui malgré tout possède

une charge. Placée à un endroit stratégique, cette entité virtuelle perturbe la force exercée sur le robot. La sommation n'étant plus nulle, le robot se met en mouvement et sort du minima local. Cette méthode est simple à réaliser, le véritable défi et de choisir un emplacement et une charge optimal à cette entité virtuelle à même de sortir le robot du minima sans collision et garantissant que le robot ne se retrouvera plus dans ce même minima.

3.2.4.2 Passage dans les petits espaces

Les petits espaces formés par deux obstacles peuvent poser problème à l'algorithme. Cela vient du fait que les obstacles empêchent le robot de passer, il y a répulsion. Dans le meilleur des cas, si le robot ne se retrouve pas dans l'axe, il contournera les obstacles alors qu'il aurait été préférable qu'il passe entre eux, dans le pire des cas, il se retrouve dans un minimum local.

Ce genre de configuration se retrouve surtout dans des environnements avec des obstacles fortement disséminés.

3.2.4.3 Oscillation dans les couloirs étroits

La méthode des champs de potentiel est aussi désavantagée lorsque le robot évolue dans un couloir étroit présentant quelques irrégularités. La proximité des parois du couloir fait que le robot oscille lors de son trajet provoquant parfois même des collisions. En effet, la force répulsive d'une paroi rejette le robot vers la paroi opposée qui effectue de même récursivement tel qu'illustré dans la figure 3.4.

Koran et Borenstein [YK91] ont calculé la limite minimal de la largeur d'un couloir pour éviter à un robot d'entrer en oscillation. Cette limite est fortement dépendante des paramètres du champs de potentiel, de la vitesse du robot et de la taille de ce dernier.

$$L^{n+1} > 2nW^nV(T+\tau)f \tag{3.19}$$

3.3 Approches connectionnistes

Il est naturel de penser que les entités biologiques sont les systèmes autonomes navigants les plus aboutis qui existent. En faisant abstraction de tous les mécanismes qui permettent le mouvement (muscles, os, articulations), le siège de la navigation de ces systèmes est sans conteste le système cérébrale. Il est donc logique que les spécialistes du domaine ce soient tournés vers les réseaux de neurones pour essayer de modéliser l'incroyable complexité du système de navigation dans le cerveau.

Historiquement parlant, McCulloch et Pitts [WSM43] ont été les premiers, en 1943, à avancer l'idée d'un neurone formalisé mathématiquement. Ce premier neurone était binaire, muni d'un délai d'activation et d'un seuil fixe. Surtout, les poids d'un réseau étaient constants dans le temps interdisant un quelconque apprentissage.

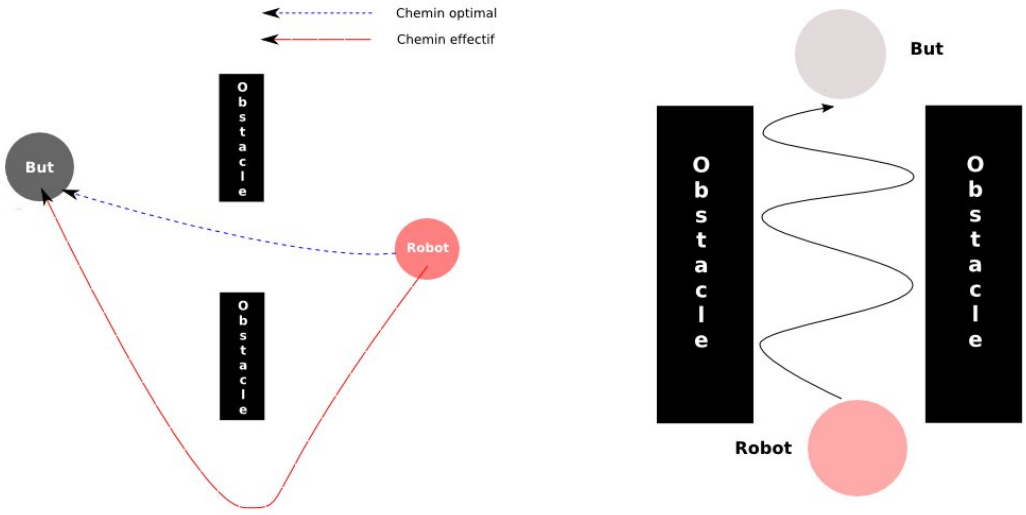


FIG. 3.4 – A droite, le problème face au passage par de petits espaces. A gauche, le problème de l'oscillation dans les couloirs étroits

Le premier réseau qui admettait l'idée d'un changement de poids avec le temps (apprentissage) fût formaliser par Hebb en 1949. Cette règle suggère que lorsque deux neurones sont excités conjointement, cela renforce le lien qui les unit[1].

L'interconnection de plusieurs neurones produit un *réseau de neurones artificiel* plus communément appelé *réseau de neurone*. C'est un modèle mathématique (ou informatique) qui tente de reproduire la structure et/ou les aspects fonctionnels des réseaux de neurones biologiques. Il traite l'information selon une approche connectionniste. Dans la majorité des cas, un réseau de neurones est un système adaptatif qui peut changer sa structure en réponse à des informations externes ou internes pendant la phase d'apprentissage. Ce sont aussi des outils de modélisation de données statistiques non-linéaires. Ils peuvent être utilisés pour modéliser la relation complexe liant les entrées aux sorties ou à trouver des motifs dans des données.

3.3.1 Apprentissage

Il existe trois paradigmes majeurs d'apprentissage. Chacun correspond à une tâche d'apprentissage abstraite particulière. Il y a l'apprentissage supervisé, l'apprentissage non-supervisé et l'apprentissage par renforcement. Généralement, l'algorithme d'apprentissage de n'importe quel réseau appartient à l'une de ces catégories.

3.3.1.1 Apprentissage supervisé

L'apprentissage supervisé est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'ap-

[1] http ://fr.wikipedia.org/wiki/Règle_de_Hebb

prentissage contenant des exemples de cas déjà traités.

Plus précisément, la base de données d'apprentissage est un ensemble de couples entrée-sortie $(x_n, y_n)_{1 \leq n \leq N}$ avec $x_n \in X$ et $y_n \in Y$, que l'on considère être tirées selon une loi sur $X \times Y$ inconnue, par exemple x_n suit une loi uniforme et $y_n = f(x_n) + w_n$ où w_n est un bruit centré.

Le but de la méthode d'apprentissage supervisé est alors d'utiliser cette base d'apprentissage afin de déterminer une représentation compacte de f notée g et appelée fonction de prédiction, qui à une nouvelle entrée x associe une sortie $g(x)$. Le but d'un algorithme d'apprentissage supervisé est donc de généraliser pour des entrées inconnues ce qu'il a pu "apprendre" grâce aux données déjà traitées par des experts, ceci de façon "raisonnable".[2].

La rétropropagation est une règle de l'apprentissage supervisé qui consiste à rétropropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés. Pour les réseaux de neurones, on utilise habituellement la rétropropagation du gradient de l'erreur, qui consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs : les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.

3.3.1.2 Apprentissage non-supervisé

Dans un apprentissage non-supervisé des donnés x sont présentées au réseau ainsi qu'une fonction de coût qui doit êter minimisée et peut être n'importe quel fonction de x et des sorties du réseau. La fonction de coût est définie par les contraintes de l'application. Cette apprentissage est généralement utilisé pour les problèmes d'estimation,, compression, filtrage, séparation de données aveugle et clustering.

3.3.1.3 Apprentissage par renforcement

Dans l'apprentissage par renforcement, l'entrée x est généralement inconnue mais généré par un agent en interaction avec l'environnement. A chaque étape t, l'agent décide d'une action y_t est l'environnement génère une observation x_t et un coût instantané c_t, en accord avec certaines dynamiques (généralement inconnues). Le but est de découvrir une politique de sélection d'actions qui minimise quelques mesures du coût à long terme. Les dynamiques de l'environnement et le coût à long terme pour chaque politique sont généralement inconnus mais peuvent être estimés. Les réseaux de neurones sont généralement utilisés dans l'apprentissage par renforcement comme partie d'un système plus grand. Ce type d'apprentissage est généralement utilisé pour les problèmes de contrôle, les jeux et autre tâche à décisions séquentielles.

[2]http ://fr.wikipedia.org/wiki/Apprentissage_supervisé

3.3.2 Types de réseaux de neurones

Il existe plusieurs types de réseaux de neurones dans la littérature, nous présentons dans ce qui suit les modèles les plus populaires.

3.3.2.1 Réseau FeedForward(Multicouches)

Le réseau de neurones feedforward a été le premier et le plus simple des réseaux de neurones artificiels. Dans ce réseau, l'information circule dans une seule direction, de l'arriére, depuis les neurones d'entrée, à travers des neurones cachés, vers les neurones de sortie. Ils sont aussi appelés *preceptrons multi-couches*. Ils n'ont ni cycles ni boucles. Généralement, l'algorithme d'apprentissage utilisé pour l'entrainement est la rétropropagation du gradient.

3.3.2.2 Réseaux récurrents

Les réseaux de neurones récurrents sont des systèmes dynamiques constitués d'unités (neurones) inter-connectés interagissant non-linéairement, et où il existe au moins un cycle dans la structure. Les unités sont reliées par des arcs pondérés. La sortie d'un neurone est un combinaison non linéaire de ses entrées. On peut étudier leurs comportements avec la théorie des bifurcations, mais la complexité de cette étude augmente très rapidement avec le nombre de neurones.

Réseau récurrents simples Un réseau simple récurrent est une variation du perceptron multi-couches, quelques fois appelé "réseau Elman" au nom de son inventeur Jeff Elman. Un réseau à trois couches est utilisé, avec l'ajout d'un ensemble d'unités de contexte dans la couche d'entrée. Il y a des connections depuis la couche cachée vers ces unités de contexte avec des lien pondérés fixés à un. A chaque pas, l'entrès est propagée vers l'avant comme dans un PML normal et la règle d'apprentissage (généralement la rétropropagation) est appliquée. Les liens à pondération fixe des unités de contexte contient toujours une copie des valeurs précédentes de la couche cachée. Ces réseaux peuvent maintenir un certain état, leur permettant de bien traiter des problèmes de prédictions au delà des capacités d'un PML.

Réseau de Hopfield Le réseau de neurones d'Hopfield [Hop82] est un modèle de réseau de neurones récurrents à temps discret dont la matrice des connexions est symétrique et nulle sur la diagonale et où la dynamique est asynchrone (un seul neurone est mis à jour à chaque unité de temps). Il a été découvert par le physicien John Hopfield en 1982. Sa découverte a permis de relancer l'intérêt dans les réseaux de neurones qui s'était essoufflé durant les années 1970 suite à un article de Marvin Minsky et Seymour Papert.

Un réseau de Hopfield est une mémoire adressable par son contenu : une forme mémorisée est retrouvée par une stabilisation du réseau, s'il a été stimulé par une partie adéquate de cette forme.

Réseau echo state Les réseau echo state [HJ04] est un réseau de neurones récurrent avec une connexion vers une couche cachée très creuse (habituellement moins d'1% de connectivité). La connectivité et les poids de la couche cachée sont assignés aléatoirement au départ et demeurent fixes. Les poids des neurones de sortie sont modifiés dans la phase d'apprentissage de manière à reproduire des motifs temporels spécifiques.

L'interet principal des ces réseaux est son comportement non-linéaire, les seuls paramètres sont les poids de la couche de sortie. La fonction de cout (erreur) est une fonction quadratique qui respecte le vecteur de paramètres et qui peut au besoin être approché à un système linéaire.

3.3.2.3 Réseau à base de fonctions radiales (RBF)

Un réseau à base de fonctions radiales est un réseau de neurones artificiel qui utilise les fonction à base radiales comme fonctions d'activation. C'est une combinaison linéaire de fonctions à base radiale. Il est généralement utilisé pour l'approximation de fonctions, prédiction de séries temporelles et le contrôle.

3.3.2.4 Cartes de Kohonen

Les cartes auto-organisées (SOM), inventées par Teuvo Kohonen, se basent sur une forme d'apprentissage non-supervisé. Un ensemble de neurones artificiels apprenant à lier des points dans un espace d'entrée à des coordonnées dans un espace de sortie. L'espace d'entrer à généralement une dimension et une topologie différentes de l'espace de sortie, la carte essaye de préserver cela.

3.3.2.5 Réseau de neurones stochastique

Un réseau de neurones stochastique diffère d'un réseau de neurones typique car il introduit le concept de variation aléatoire. D'un point de vue probabiliste, ces variations aléatoires peuvent être considérées comme un échantillonnage statistique à la manière d'un échantillonnage de Monte Carlo.

Machines de Boltzmann La machine Boltzmann peut être approché à un réseau de Hopfield bruité. Inventée par Geoff Hinton et Terry Sejnowski en 1985, la machine Boltzmann est importante car étant le premier réseau de neurones à démontrer un apprentissage latent des variables des unités cachées. L'apprentissage d'une machine Boltzmann était au départ long à simuler, cela a pu être améliorer grâce à l'algorithme divergeant contrasté de Geoff Hinton.

3.3.3 Propriétés théoriques

Les propriétés théoriques des réseaux de neurones sont l'approximation de fonction, la classification et la capacité.

3.3.3.1 Approximation de fonctions

Pour faire de l'approximation de fonction, Georges Cybenko [Cyb89] a démontrer, en 1989, dans son article *"Approximation by Superpositions of Simoidal Fonction"*, qu'un réseau multicouche, avec une seule couche cachée de neurones sigmoides et une couche de sortie avec des neurones linéaires permet d'approximer n'importe quelle fonction d'intérêt avec une précision arbitraire à condition de disposer de suffisamment de neurones sur la couche cachée. Intuitivement, un peu a la façon des séries de Fourier qui utilisent des sinus et cosinus, cette preuve passe par la démonstration que l'on peut approximer n'importe quelle fonction d'intérêt par une combinaison linéaire de sigmoides.

3.3.3.2 Classification

Pour faire de la classification, on utilisera des réseaux soit à deux, soit a trois couches de neurones sigmoides. On peut montrer qu'une seule couche cachée suffit a engendrer des frontières de décision convexes, ouvertes ou fermées, de complexité arbitraire, alors que deux couches cachées permettent de créer des frontières de décision concaves ou convexes, ouvertes ou fermées, de complexité arbitraire. La figure 3.3 montre en deux dimensions différents types de frontières de décision. Intuitivement, on peut voir que la première couche cachée d'un tel réseau sert a découper l'espace d'entrée a l'aide de frontières de décision linéaires, comme on l'a vu pour le perceptron simple, la deuxième couche sert a assembler des frontières de décision non-linéaires convexes en sélectionnant ou en retranchant des régions engendrées par la couche précédente et, de même, la couche de sortie permet d'assembler des frontières de décision concaves en sélectionnant ou en retranchant des régions convexes engendrées par la couche précédente.

3.3.3.3 Capacité

Le modèle d'un réseau de neurones artificiel a une propriété appelée capacité. Elle correspond grossièrement à la capacité d'un réseau à modeler une certaine fonction. Ce concept est aussi lié à la charge d'information qui peut être enregistrée dans le réseau et à la notion de complexité.

3.3.4 Approches neuronales pour la planification de chemins

En octobre 2004, à l'Université de Floride, Thomas DeMarse déclare avoir fait voler un avion de combat F22 (en simulateur) intégralement commandé par 25000 vrais neurones de rat. Au début de l'expérience, les neurones n'étaient pas reliés entre eux. Après les premiers signaux transmis par électrodes aux cellules nerveuses, celles-ci ont commencé à s'organiser entre elles, en se connectant les unes aux autres pour analyser les données et y répondre. Ces réponses modifient les commandes de l'avion, qui envoient en retour des nouvelles informations au réseau de neurones, etc.

Une boucle sans fin.

"Initialement, quand nous avons raccordé ce cerveau au simulateur de vol, il ne savait pas comment contrôler l'avion" explique DeMarse. *"L'avion volait à la dérive. A mesure que les informations arrivaient, cela a lentement modifié le réseau qui a progressivement appris à faire voler l'avion."* Selon les dires du jeune scientifique, ce "réseau de neurones" est désormais capable de piloter l'avion par tous les temps, du ciel bleu au beau milieu d'une tempête forte comme un cyclone.[3]

Cette expérience prouve, qu'à terme, les réseaux neuronaux sont les plus aptes à planifier un chemin pour n'importe quel système de navigation car ces tissus de cerveau de rat ont été capable de gérer des systèmes aussi hétérogène qu'un rat et un avion de chasse F22.

Nous allons dans ce qui suit présenter trois travaux de la littérature. Le premier est un cas classique d'utilisation d'un perceptron multicouches écrit par Janglova [Jan04]. Le Deuxième est un travail de Ritthipravat et al, basé sur un réseau de Hopfield modifié pour l'évitement d'obstacles [PRN02]. Le dernier travail est celui de Z.Hendzel [Z.H05] qui se base sur les carte de Kohonen pour planifier des chemins sans collision.

Janglova utilise une simulation de robot à deux roues pour effectuer son implémentation. Ce robot est virtuellement muni de vingt-neuf capteurs ultrasons qui recouvrent l'environnement devant le robot sur 240^o. Elle utilise un PMC (perceptron multi-couche) comme réseau de traitement couplé à un réseau PCA (Principal Composant Analysis) en entrée.

Ritthipravat et al Comme introduit dans le paragraphe précédant, les auteurs se basent sur un réseau de Hopfield pour planifier le chemin. Contrairement au réseau Hopfield traditionel, le réseau introduit par les auteurs est asymétrique. Leur principe se base sur la construction d'un réseau copiant l'environnement, c-à-d que chaque neurone représente une zone de l'environnement et les liens entre ces neurones ont un poids en relation directe à la distance par rapport au but.

Hendzel propose de diviser le comportement du robot en deux comportements distincts. Le premier comportement est d'atteindre le but, le second est d'éviter les obstacles. Chacun de ces comportement se voit assigner une carte de Kohonen séparée. L'idée est d'ensuite traiter les deux cartes avec une stratégie coopérative pour sélectionner le prochain pas à effectuer.

Hendzel simule un robot à roue d'une cinématique assez complexe illustrée sur la figure 3.5. Le robot est supposé-ment muni de huit capteurs ultrasons. Les simu-

[3] http ://www.futurinc.lautre.net/breve.php3 ?id_breve=299

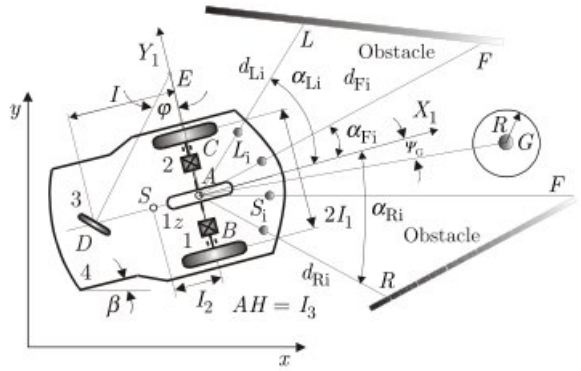


FIG. 3.5 – Le modèle cinématique du robot de Hendzel.

lations ont été faites dans un environnement fermé (10x10 $[m^2]$) et le but se trouve à une distance maximale de 6 mètres.

3.3.4.1 Les entrées et sorties considérées

Cette section décrit, pour chaque auteur, les entrées considérées au réseau ainsi que les sorties attendues.

Janglova injecte en entrée de son réseau les distances retournées par les vingt-neuf capteurs du robot qu'elle note V_i. Elle injecte aussi des informations relatives à la position du but qu'elle note S_i. Le système étant classique, les entrées sont elles aussi classique tout comme les sorties qui sont notées O_i qui représentent la direction (azimut) du prochain pas à entreprendre.

Ritthipravat et al n'ont pas explicitement exprimé les entrées utilisées pour leur réseau. Cependant, il est assez simple de faire des hypothèses sur la question. Premièrement, le robot détecte la direction du but, à travers des sonars ultrasons ou autre capteurs à portée locale. Les auteurs considèrent le neurone central de leurs topologie (voir section suivante) comme étant la position du robot. La valeur des poids est distribuée selon la distance de chaque neurone par rapport au but. La sortie est représentée par la position du neurone activé qui désigne la direction à entreprendre par rapport au neurone central.

Hendzel a muni son robot de huit capteurs ultrasons retournant chacun une distance d_i par rapport au but ou aux obstacles. Ces capteurs étant fixés, Hendzel fournit au robot les valeurs angulaires dont est chargé chaque capteurs. En entrée, donc, le robot reçoit un couple (angle,distance) depuis chaque capteur $u_i = [\psi_i, d_i]$ qu'il présente à chaque carte. La carte responsable de l'atteignabilité du but ne s'intéresse qu'aux coordonnées du but $u_G = [\psi_G, d_G]$ normalisés, la carte responsable de l'évitement d'obstacle ne considère que les coordonnées des obstacles qui

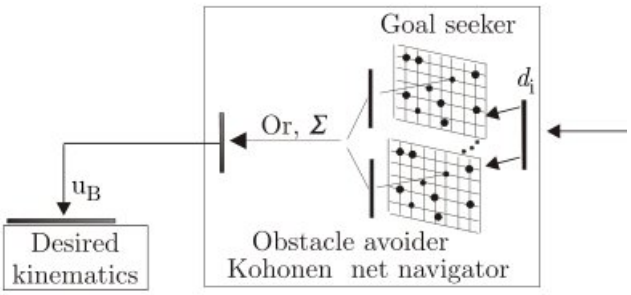


FIG. 3.6 – Le Navigateur de Hendzel, deux cartes de Kohonen pour deux comportements distincts.

subissent une transformation de manière à garantir que le robot se trouve au milieu de l'espace libre. Cette transformation prend en compte la distance minimal de gauche et de droite et aussi la distance de front normalisée avec la portée maximale des capteurs. En sortie, chaque carte propose des coefficient de multiplication générant une transformation spatiale. A la manière des champs de potentiels, l'auteur propose de faire une sommation entre les deux vecteurs issues de chaque carte (après multiplication), le vecteur résultant sera la sortie finale du système vers le controleur.

3.3.4.2 Topologies des réseaux

La topologie du réseau est une propriété importante et décisive pour la qualité du modèle de planification que proposent les auteurs. Nous allons présenter dans cette section les contraintes imposées par l'environnement à la définition de ces topologies.

Janglova propose une topologie du réseau composé d'un réseau PCA qui traite les vingt-neuf entrées. Ce réseau PCA transorme les 29 entrées en 18 sorties qui seront les entrées du PMC. La couche d'entrée du PMC est donc constituée de 18 neurones, 9 V_i et 9 S_i. La couche cachée est elle constitué de 18 neurones et 9 neurones pour la couche de sortie (figure 3.7).

Ritthipravat et al présentent une topologie sous forme de grille semblable a ce qui se fait sur une carte de Kohonen. Cette grille est une représentation réduite de l'environnement immédiat entourant le robot. Cette grille est carrée de 3x3 neurones. Le neurone central désigne la position courante du robot.

Les poids sont distribués selon une métrique précise qui est la distance par rapport au but. Nous pouvons voire sur la partie gauche de la figure 3.8 que le lien reliant le centre au but est le plus puissant (1^{st}). Puis, les neurones directement adjacents au neurone but sont liés au centre par les seconds poids les plus puissant (2^{nd}) et ainsi de suite jusqu'au neurone qui se trouve complètement à l'opposé du

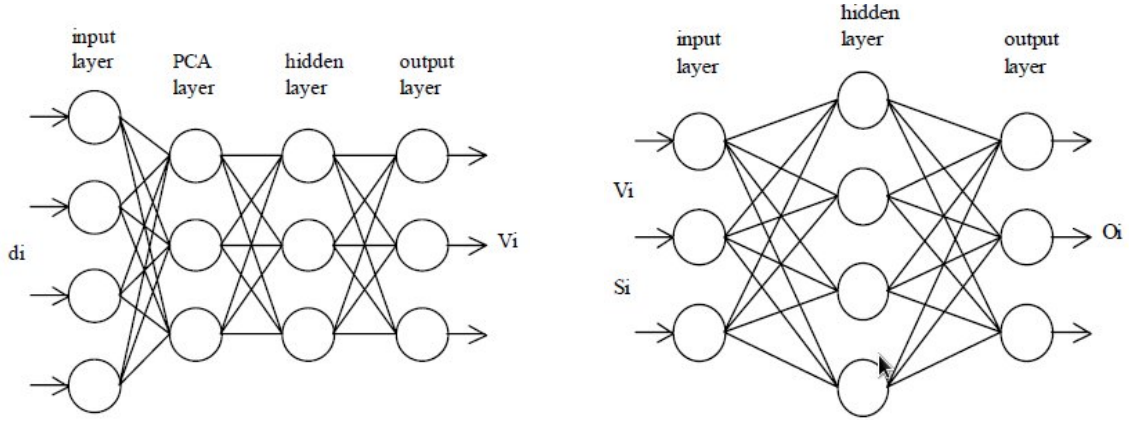


FIG. 3.7 – A gauche, Le réseau PCA qui fournit les entrées du réseau PMC à droite dans les travaux de Janglova

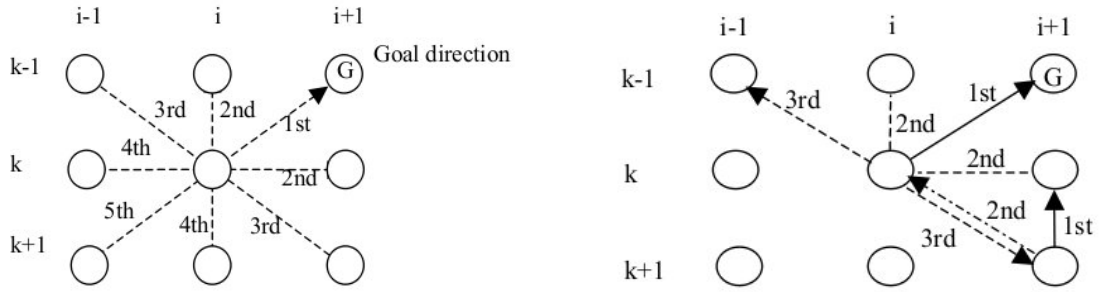


FIG. 3.8 – A gauche, La topologie du réseau de Ritthipravat et al, à droite, description de l'asymétrie du réseau.

but qui reçoit la pondération la plus faible.

L'asymétrie du réseau est bien définie par la partie droite de la figure 3.8. Le neurone complètement à droite vers le bas (k+1,i+1) est relié au centre par un grand poids (2^{nd}) car s'approchant du but, par contre le centre est reliè à ce neurone par un poids moyen (3^{rd}) car s'éloignant du but.

Les auteurs voulant activé les neurones dont les poids des liens en entrée sont les plus puissant ont utilisé une sommation classique avec rajout d'un biais. Cette sommation est du genre :

$$u_i = \sum_{i=1}^{m} w_{ij} v_j + \theta_i \tag{3.20}$$

où u_i est l'entrée du neurone i, w_{ij} le poids du lien entre le neurone j et le neurone i, v_j la sortie du neurone j et θ_i un biais. Ce biais a été introduit par les auteurs pour garantir un chemin sans collision et une atteignabilité du but. L'expression du biais est telle que :

$$\theta_j = \begin{cases} +\infty & si\ un\ but \\ -\infty & si\ un\ obstacle \\ 0 & sinon \end{cases} \tag{3.21}$$

Ceci a pour effet de désactiver les neurones représentant les obstacles (pas de déplacement dans ce sens) et d'activer à coup sur le neurone de but. Les auteurs ont opté pour une fonction hyperbolique dans le domaine positif, nulle sinon :

$$\Phi(u_i) = \begin{cases} \frac{1-e^{u_i}}{1+e^{u_i}} & u_i >= 0 \\ 0 & sinon \end{cases} \tag{3.22}$$

Hendzel propose deux cartes de Kohonen de 5x5 neurones. Pour la carte responsable de l'atteignabilité du but, chaque neurone contient un couple de positionnement par rapport au but u_G. C-à-d que chaque neurone contient un vecteur (angle,distance) qui pointe vers le neurone représentant le but $u_i = [\psi_i, 0]$. Pour la seconde carte, chaque neurone contient un vecteur de perception pondéré. Le lien entre deux neurones est un lien de voisinage d'autant plus fort que la distance entre ces neurones est petite.

3.3.4.3 Algorithme d'entrainement

L'apprentissage et l'adaptation sont deux propriétés recherchées dans les réseaux de neurones. Pour atteindre ces buts ultimes, un bon algorithme d'apprentissage est indispensable. Nous présenterons dans cette section les algorithmes utilisés par les auteurs dans leurs méthodes.

Janglova propose deux algorithmes d'apprentissage. Le premier est propre au réseau PCA qui subit un apprentissage qui combine le mode supervisé et non-supervisé. PCA une procédure linéaire non-supervisé qui recherche un ensemble de motif non-corrélés à partir de l'entrée. Un réseau feedforward est utilisé pour permettre une classification non-linéaire des ces composants. PCA est une méthode de réduction de données qui condense les informations d'entrée en quelques composantes principales.

Le PMC quant à lui subit un entrainement typique à ce type de réseaux, c-à-d qu'il est muni d'un algorithme de rétropropagation statique de l'erreur.

Ritthipravat et al ne décrivent pas de schéma d'entrainement car il en n'existe pas. Comme vu précédemment, les pondérations des liens ne varient pas en accord à un apprentissage mais en accord à une métrique (distance par rapport au but). Cependant, le fonctionnement du réseau reste classique c-à-d une activation asynchrone et aléatoire des neurones du réseau et choix du lien reliant les neurones les plus actifs (règle Hebbienne). Les auteurs désactivent les neurones représentant les obstacles mais aussi les neurones qui représentent l'ancien emplacement pour, comme ils le déclarent, "éviter les boucles et retours arrières". Nous émettons une réserve quant à ce choix car dans quelques cas (labyrinthes) les retours arrières sont indispensables.

Hendzel utilise l'algorithme d'entrainement classique pour les cartes de Kohonen, c-à-d que pour chaque entrée x, il commence par calculer la distance de cette entrée par rapport à tout les neurones représentés par leurs vecteur de poids w :

$$d(x, w_s) = ||x - w_s||^2 \tag{3.23}$$

Ensuite, pour élire le neurone vainqueur, le neurone présentant une distance minimale par rapport à l'entrée est choisi :

$$w = arg\ min_s d(x, w_s) \tag{3.24}$$

Le vecteur de poids w du neurone élu subit une variation qui tend à l'approcher du vecteur d'entrée selon la règle :

$$w_w^{n+1} = w_w^n + c(t)h(s, w)(x - w_w^n) \tag{3.25}$$

où $c(t)$ est un pas d'apprentissage décroissant avec le temps, $h(s, w)$ est un coefficient de voisinage qui sera surtout utile lors du changement des vecteur de poids des neurones environnants.

après aprentissage, la carte présentera au final une grille de prototype de perception. Le neurone élu de la première carte donnera un prototype de perception du but, par conséquent, un prototype de son orientation après la multiplication vectorielle effectuée. La seconde carte donnera un prototype de perception vers le milieu de l'espace libre, par conséquent un prototype de l'orientation de ce milieu après la multiplication matricielle. La somme de ces prototypes d'orientation donnera le vecteur de déplacement final.

3.3.4.4 Critiques

Janglova propose une méthode simple à base de PMC. Sa contribution se situe au niveau du PCA à l'entrée qu'elle utilise pour condenser les données et ainsi augmenter le taux de réussite de classification du PMC en aval. Il est à remarquer que cette étape pouvait très bien être effectuée par une analyse en composante principale classique qui aurait été un gain de temps considérable vu que le PCA de l'auteur possède un algorithme d'apprentissage composite (supervisé-non-supervisé) surement très gourmand en temps de calcul. Mais il est clair que cette méthode peut être encourageante dans le cas ou on envisage une méthode de navigation entièrement neuronale avec les dispositifs physiques adéquats à sa conception.

Ritthipravat et al proposent une méthode simple d'implémentation mais trop réductrice. Le robot ne peut se mouvoir que dans 8 directions et la précision de l'environnement simulé est trop basse. Les auteurs n'utilisent pas les deux propriétés qui font la force d'un réseau de Hopfield c-à-d l'adaptation et l'apprentissage. De plus, en éliminant les retours arrières, le robot ne peut explorer son environnement ni résoudre un chemin dans un labyrinthe.

Hendzel a proposé une approche très originale de la planification de chemins, ces expérimentations ont donné des résultats très probants. La seule ombre au tableau pourrait être la probabilité d'un minima local vu que le vecteur de direction final est une sommation des vecteurs issus des deux cartes. L'une générant un vecteur attracteur, l'autre un vecteur répulseur. il semblerait qu'un phase d'apprentissage en plus sorte le robot de ces minimas, cela est surement du au coefficient d'apprentissage $c(t)$ qui varie avec le temps et déstabilise la sommation.

3.4 Approches floues

La logique floue (fuzzy logic, en anglais) est une technique utilisée en intelligence artificielle. Elle a été formalisée par Lotfi Zadeh en 1965 et utilisée dans des domaines aussi variés que l'automatisme (freins ABS), la robotique (reconnaissance de formes), la gestion de la circulation routière (feux rouges), le contrôle aérien, l'environnement (météorologie, climatologie, sismologie), la médecine (aide au diagnostic), l'assurance (sélection et prévention des risques) et bien d'autres. En fait, le simple fait de noter, déjà sous Jules Ferry, un élève dans différentes disciplines et de lui calculer un rang par application de coefficients à ses notes était déjà faire de la logique floue sans le savoir

Elle s'appuie sur la théorie mathématique des ensembles flous. Cette théorie, introduite par Zadeh, est une extension de la théorie des ensembles classiques pour la prise en compte d'ensembles définis de façon imprécise. C'est une théorie formelle et mathématique dans le sens où Zadeh, en partant du concept de fonction d'appartenance pour modéliser la définition d'un sous-ensemble d'un univers donné, a

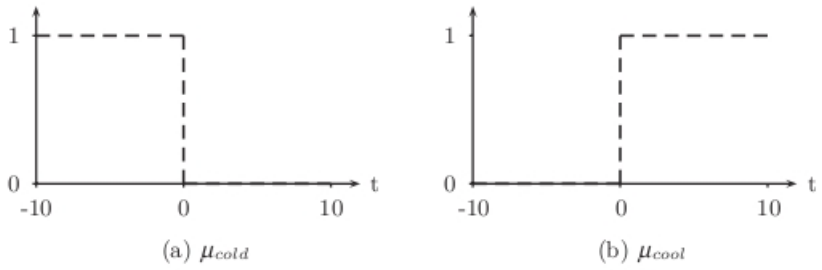


FIG. 3.9 – Ensembles classiques.

élaboré un modèle complet de propriétés et de définitions formelles. Il a aussi montré que cette théorie des sous-ensembles flous se réduit effectivement à la théorie des sous-ensembles classiques dans le cas où les fonctions d'appartenance considérées prennent des valeurs binaires (0,1).

Elle présente aussi l'intérêt d'être plus facile et meilleur marché à implémenter qu'une logique probabiliste, bien que cette dernière seule soit stricto sensu cohérente (voir Théorème de Cox-Jaynes). Par exemple la courbe $Ev(p)$ peut être remplacée par trois segments de droite sans perte excessive de précision pour beaucoup d'applications considérées ci-dessus.

Pour la présentation théorique de la logique floue, nous nous sommes basé sur le livre de Nadia Nedjah : *Evolvable machines : theory and practice*[NN05].

3.4.1 Ensembles flous

Conventionnelement, un ensemble S est dit classique, si et seulement si pour un élément $x \in U$, U est appelé univers de discours, nous avons soit $x \in S$ ou $x \notin S$. Pour les ensembles flous, ceci n'est pas vrai. Chaque élément $x \in U$ appartient à un ensemble flou mais avec un degré d'appartenance. Un ensemble S devient flou, s'il est couplé à une fonction $\mu_S : U \rightarrow [0, 1]$ appelée fonction d'appartenance qui, pour chaque élément $x \in U$, définit son degré d'appartenance à S. Un couple $(x, \mu_S(x))$ est appelé *singleton* et chaque ensemble flou peut être considéré comme l'ensemble classique définit par l'union de tout les singletons $(x, \mu_S(x))$, pour tout $x \in U$.

$$\mu_S(x) = \begin{cases} 1 & \text{si } x \in S \\ 0 & \text{si } x \notin S \end{cases} \tag{3.26}$$

La littérature foisonne d'exemples comparant les ensembles logiques et flous. Peut-être, le meilleur exemple pour illustrer la nécessité du flou est quand nous voulons définir le concept de froid et glacial. La version classique qui définit ces concept pour être tels que définis dans les équations 3.26.

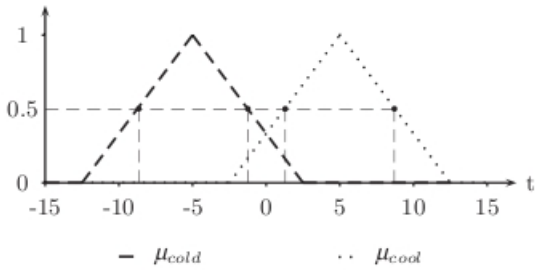


FIG. 3.10 – Représentation de deux ensembles flous.

$$\mu_{glacial}(t) = \begin{cases} 1 & \text{si } -10^o < t < 0^o \\ 0 & sinon \end{cases}$$
$$\mu_{froid}(t) = \begin{cases} 1 & \text{si } 0^o < t < 10^o \\ 0 & sinon \end{cases} \tag{3.27}$$

La représentation graphique de ces ensembles est relatée par la figure 3.9, le concept d'eau froide et glaciale son mutuellement exclusif. Certainement, il est juste de dire qu'une eau à -10^oC est plus froide qu'une eau à 1^oC. Par opposition, dans ces ensembles classiques, une eau à 10^oC est autant froide qu'une eau à 0^oC. D'un autre coté, les ensembles classiques ne peuvent pas définir une transition être ces deux états. Nous savons que dans le monde réel, la transition entre l'état glacial et l'état froid se fait en douceur est de manière fine. Ces considération ne sont pas prises en compte dans la définition classique.

Il est commun d'utiliser des graphiques pour décrire la fonction d'appartenance à un ensemble flou. Pour simplifier, des graphiques triangulaires, trapézoidales, sigmoidales ou Gaussiens sont préférés. La représentation triangulaire floue des concepts froid et glacial est définie dans l'équation 3.27 est représentée sur le graphique de la figure 3.10.

$$\mu_{glacial}(t) = \begin{cases} \frac{2}{5}t + \frac{5}{3} & \text{si } -12.5^o < t < -5^o \\ \frac{-2}{5}t + \frac{1}{3} & \text{si } -5^o < t < 2.5^o \\ 0 & sinon \end{cases}$$
$$\mu_{froid}(t) = \begin{cases} \frac{2}{5}t + \frac{1}{3} & \text{si } 2.5^o < t < 5^o \\ \frac{-2}{5}t + \frac{5}{3} & \text{si } 5^o < t < 12.5^o \\ 0 & sinon \end{cases} \tag{3.28}$$

En considérant la figure 3.10, nous pouvons constater qu'une eau à 0^o est à un tiers glaciale et à un tiers froide. Pour permettre la transition entre deux (ou plusieurs) états en logique floue, une superposition de 10% voir même 50% des ensembles flous de ces états est généralement utilisée.

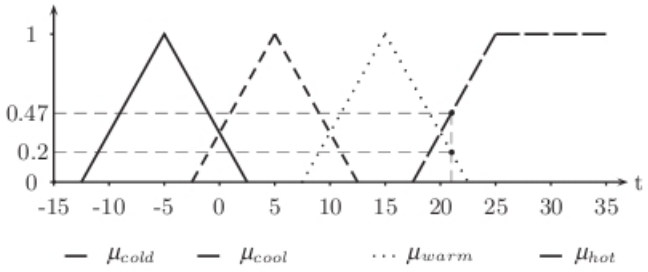


FIG. 3.11 – Représentation de plusieurs ensembles flous.

3.4.2 Termes et variables linguistiques

Les deux aspects essentiels dans la logique flou sont le concept et les caractéristiques de ce concept. Par exemple, pour les concept température, age et taille, nous pouvons définir respectivement ces caractéristiques : froid et glacial, jeune et agé, petit, moyen et grand. Les concepts quantifiés par la logique flou sont appelés "variables linguistiques". Leurs caractéristiques sont appelés "termes linguistiques".

Toute variable linguistique V telle que $V \in R \subset U$, R étant le référentiel de variables et sous-ensemble de l'univers de discours U, est quantifiée grâce à un ensemble de termes linguistiques $T_V^1, T_V^2, T_V^3, ..., T_V^n$. Chaque terme linguistique T_V^i est un ensemble flou possédant sa propre fonction d'appartenance $\mu_{T_V^i}$.

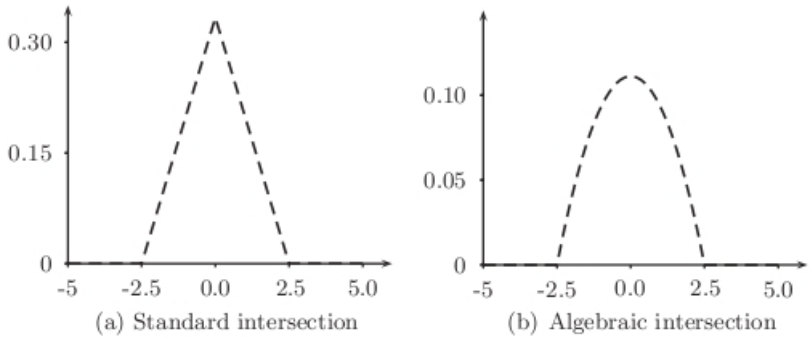
La figure 3.11 contient le graphe des différentes fonctions d'appartenance possibles des termes linguistiques "glacial", "froid", "tiède" et "chaud" du concept "température". Le référentiel $R = [-15, 35]$.

3.4.3 Opérateurs

Dans le but de pouvoir utiliser efficacement la logique floue, les opérateurs d'ensembles flous s'avèrent incontournables. Ces opérateurs sont des extensions des opérateurs habituels de la logique classique. Dans ce qui suit, sera détaillé l'utilité de chacun de ces opérateurs. Pour l'exemple, nous définissons deux ensembles flous A et B possédant des fonctions d'appartenance μ_A et μ_B.

3.4.3.1 Intersection d'ensembles flous

Intuitivement, l'intersection de deux ensembles flous noté $\prod$ produit un ensemble flou. Dans la littérature spécialisée, nous rencontrons plusieurs méthodes d'intersection. Les plus connues sont : "l'intersection de Zadeh ou standard 3.29", "produit d'intersection ou algébrique 3.30", "intersection de Lukasiewicz 3.31" et "intersection robuste 3.32". La figure 3.12 contient le graphe montrant la différence entre une intersection standard et une intersection algébrique.



(a) Standard intersection (b) Algebraic intersection

FIG. 3.12 – Intersection standard et algébrique.

$$(\mu_A(x) \prod \mu_B(x)) = min(\mu_A(x), \mu_B(x)) \quad (3.29)$$

$$(\mu_A(x) \prod \mu_B(x)) = \mu_A(x).\mu_B(x) \quad (3.30)$$

$$(\mu_A(x) \prod \mu_B(x)) = max(\mu_A(x) + \mu_B(x) - 1, 0) \quad (3.31)$$

$$(\mu_A(x) \prod \mu_B(x)) = \begin{cases} \mu_A(x) & \text{si } \mu_B(x) = 1 \\ \mu_B(x) & \text{si } \mu_A(x) = 1 \\ 0 & \text{sinon} \end{cases} \quad (3.32)$$

3.4.3.2 Union d'ensembles flous

Comme pour le cas de l'intersection, de nombreuses méthodes d'union sont citées dans la littérature spécialisée. Notée $\cup$, les plus connues sont : "L'union de Zadeh ou standard (3.33)", "produit d'union ou algébrique (3.34)", "union de Lukasiewicz (3.35)" et "union robuste (3.36)".

$$(\mu_A(x) \cup \mu_B(x)) = max(\mu_A(x), \mu_B(x)) \quad (3.33)$$

$$(\mu_A(x) \cup \mu_B(x)) = \mu_A(x) + \mu_B(x) - \mu_A(x).\mu_B(x) \quad (3.34)$$

$$(\mu_A(x) \cup \mu_B(x)) = min(1, \mu_A(x) + \mu_B(x)) \quad (3.35)$$

$$(\mu_A(x) \cup \mu_B(x)) = \begin{cases} \mu_A(x) & \text{si } \mu_B(x) = 0 \\ \mu_B(x) & \text{si } \mu_A(x) = 0 \\ 1 & \text{sinon} \end{cases} \quad (3.36)$$

3.4.3.3 Complémentaire d'ensembles flous

Le complément d'un ensemble flou noté η n'a pas de définition claire. Comme pour l'union et l'intersection, plusieurs méthodes sont citées dans la littérature tels que : "le complement standard (3.37)", "le complément de Sugeno (3.38)" et "le complement de Yager (3.39)".

$$\eta(\mu_A(x)) = 1 - \mu_A(x) \tag{3.37}$$

$$\eta_\sigma(\mu_A(x)) = \frac{1 - \mu_A(x)}{1 + \sigma\mu_A(x)} \tag{3.38}$$

$$\eta_v(\mu_A(x)) = (1 - \mu_A(x)^v)^{\frac{1}{v}} \tag{3.39}$$

3.4.4 Commande floue

La Commande floue est l'application la plus utilisée de la logique floue. Elle consiste à remplacer les algorithmes de réglage conventionnels par des règles linguistiques du type : "SI la voiture qui précède est proche ET que je roule vite ALORS il faut freiner rapidement". Ainsi on obtient un algorithme linguistique qui se prête mieux que les méthodes traditionnelles à la commande d'un processus.

3.4.4.1 Fuzzification

Le premier module traite les entrées du système (valeurs réglantes). On définit tout d'abord un univers de discours, un partitionnement de cet univers en classes pour chaque entrée, et des fonctions d'appartenance pour chacune de ces entrées (par exemple pression grande, petite, faible et changement d'écart mesure consigne de débit de matériau sortant d'une trémie très élevé, élevé, moyen, négatif, très négatif). La première étape, appelée fuzzification, consiste à attribuer à la valeur réelle de chaque entrée, au temps t, sa fonction d'appartenance à chacune des classes préalablement définies, donc à transformer l'entrée réelle en un sous ensemble floue. Soit V une entrée, le fuzzifié de V est tel que définit dans l'équation (3.40)

$$fuzzy(V) = [x_V^1, x_V^2, x_V^3, ..., x_V^n] \tag{3.40}$$

x_V^k est le degré d'appartenancede la valeur V à l'ensemble flou S_k.

3.4.4.2 Déductions floues

Le deuxième module consiste en l'application de règles de type "si l'écart de température est grand, diminuer le débit du fuel". Ces règles vont, comme dans l'exemple introductif, permettre de passer d'un degrés d'appartenance d'une grandeur réglante au degrés d'appartenance d'une commande. Ce module est constitué d'une base de règles et d'un moteur d'inférence qui permet le calcul. A partir de la base de règles (fournie par l'expert) et du sous ensemble flou X_0 correspondant à la fuzzification du vecteur de mesure $x_0 = [x_0^1, x_0^2, ..., x_0^n]^T U$, le mécanisme d'inférence calcule le sous-ensemble flou $\mu(x_0)$ relatif à la commande du système. En général, plusieurs valeurs de variables floues, convenablement défini par des fonctions d'appartenance, sont liées entre elles par des règles, afin de tirer des conclusions. On parle alors de déductions floues.

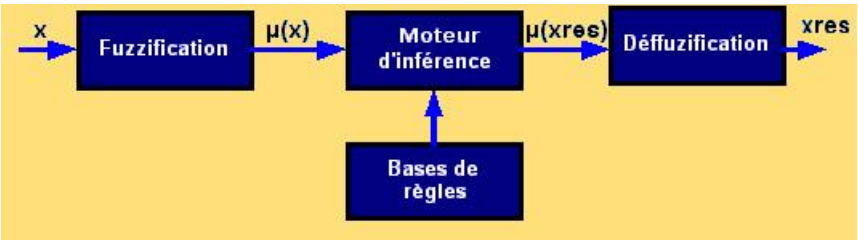


FIG. 3.13 – Shéma d'un système de commande.

3.4.4.3 Défuzzification

Le moteur d'inférence fournit une fonction d'appartenance résultante $\mu(x_R)$ pour la variable de sortie x_R. Il s'agit donc d'une information floue. Etant donné que l'organe de commande nécessite un signal de commande u_{cm} précis à son entrée, il faut prévoir une transformation de cette information floue en une information déterminée. Cette information est appelée défuzzification.

La défuzzification ou concrétisation, consiste donc à combiner ces coefficients avec les sous-ensembles de sortie, pour les convertir en un ou plusieurs signaux de commande. C'est l'opération inverse de la pondération. Il existe plusieurs méthodes pour calculer la valeur représentative d'un ensemble de sortie dont voici une liste non exhaustive :

AI (integration adaptative), BADD (basic defuzzification distributions), CDD (constraint decision defuzzification), COA (centre de surface), COG (centre de gravité), ECOA (centre de surface étendu), EQM (méthode de qualité étendue), FCD (défuzzi-fication à clustering flou), FM (moyenne floue), FOM (premier maximum), GLSD (generalized level set defuzzification), ICOG (centre de gravité indexé), IV (valeur d'influence), LOM (dernier maximum), MeOM (moyenne du maximum), MOM (médiane du maximum), QM (méthode de qualité), RCOM (choix aléatoire du maximum), SLIDE (défuzzification semi-linéaire), WFM (Moyenne floue pondérée)

Exemple : Défuzzification par centre de gravité La méthode de défuzzification la plus utilisée est celle de la détermination du centre de gravité de la fonction d'appartenance résultante $\mu(x_R)$. Dans ce contexte il suffit de calculer l'abscisse x_R^*. L'abscisse du centre de gravité peut être déterminée à l'aide de la relation générale suivante :

$$CdG = \frac{\int_{-1}^{1} x_R \mu(x_R) dx_R}{\int_{-1}^{1} \mu(x_R) dx_R} \tag{3.41}$$

L'intégrale du dénominateur donne la surface, tandis que l'intégrale du numérateur correspond au moment de la surface.

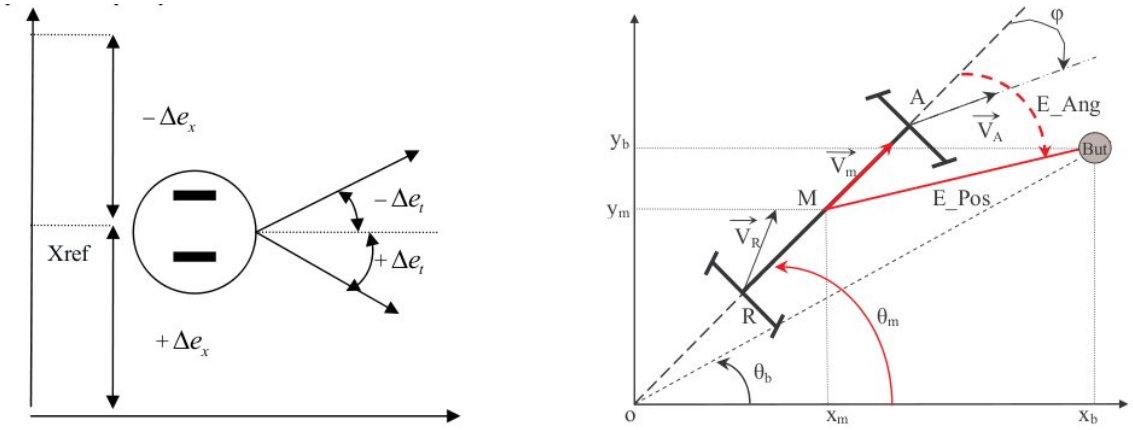


FIG. 3.14 – A droite, Le modèle cinématique de Ouedah et al. A gauche, celui de Peri et al.

3.4.5 Applications de la logique floue dans la planification de chemins

En réalité, c'est le coté commande floue de la théorie de la logique floue qui est utilisée dans la robotique mobile autonome. Nous allons présenter deux travaux se basant sur la commande floue. Les deux travaux sont des réalisations de robots à roues.

Le premier travail est celui de *Ouadah et al* intitulé *Implémentation d'un contrôleur floue pour la commande d'un robot de type voiture* [Oua06]. Le second travail est celui de *Peri et Simon* intitulé *Fuzzy Logic Control for an Autonomous Robot*[VMP06].

3.4.5.1 Les entrées et sorties considérées

Dans cette partie nous nous intéressons aux données que vont traiter les contrôleurs flous des auteurs. La figure 3.14 illustre les modèles cinématiques des deux groupes de recherche.

Ouadah et al A droite de la figure est illustré le modèle cinématique de la réalisation des auteurs appelée "Robucar". Leur implémentation se résume en un régulateur flou. La position et l'orientation actuelle sont calculés par odométrie. En entrée, le régulateur reçoit la distance robot-but (désignée E_{Pos} sur le schéma) ou l'erreur de position, et l'erreur angulaire entre sa position et celui du but (notée E_{Ang}). En sortie, le régulateur transmet la vitesse de translation noté V_m et l'angle de braquage noté ψ nécessaires pour approcher le but.

Peri et Simon La figure 3.14 illustre, à gauche, le robot dans un espace cartésien. il est clair que les deux entrées sont les mêmes que celles de Ouadah et al, à savoir, une erreur sur la distance notée Δe_x et une erreur sur l'angle d'orientation que les auteurs ont noté Δe_t. Les sorties du contrôleur sont, elles, différentes. Ce sont

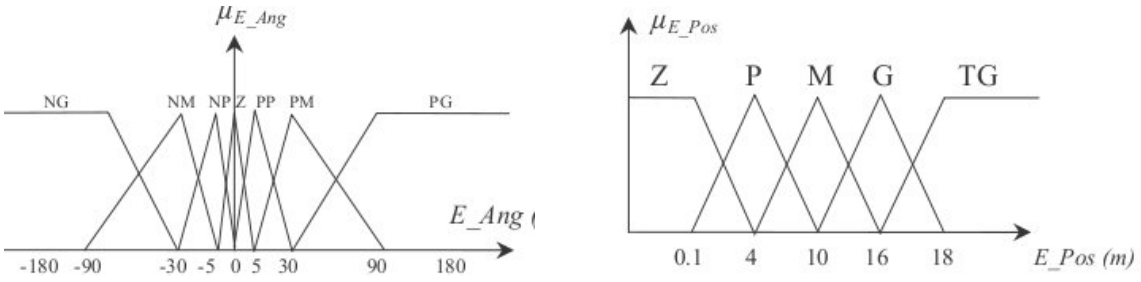


FIG. 3.15 – A droite, le diagramme de la fonction d'appartenance de la variable E_{Pos}. A gauche, le diagramme de la fonction d'appartenance de la variable E_{Ang}

des pulsations modulées en largeur (PML voir servomoteur en section 1.1.2.3) qui contrôlent indépendamment deux servomoteurs, un pour la roue gauche et un autre pour la roue droite. Les auteurs ont noté ces pulsations Δw_r et Δw_l pour, respectivement, la roue gauche et la roue droite.

3.4.5.2 La fuzzification

Dans cette section nous allons présenter l'univers de discours et les fonctions d'appartenance utilisés par les auteurs pour fuzzifier leurs variables d'entrées vues dans la section précédente.

Ouadah et al ont un univers de discours assez riche pour les variables d'entrée constitué de pas moins de douze termes linguistiques qui sont respectivement : Négatif Grand, Négatif Moyen, Négatif Petit, Zéro, Positif Petit, Positif Moyen et Positif Grand pour l'erreur angulaire. Zéro, Petit, Moyen, Grand, Très Grand pour l'erreur de position. La figure 3.15 illustre les deux diagrammes des deux fonctions d'appartenance de E_{Pos} et E_{Ang}.

Peri et Simon utilisent un univers de discours assez simple qui contient en tout trois termes linguistiques : Négatif, Zéro et Positif pour les variables d'entrées. Dans leur papier il n'y a pas de diagramme de fonction d'appartenance mais des définition mathématiques de ces ensembles. Ces définitions sont assez floues et sont du type :

$$f_{ij}(z_j) = \begin{cases} 1 + (z_j - c_{ij})/b_{ij}^- \\ 1 - (z_j - c_{ij})/b_{ij}^+ \\ 0 \end{cases} \tag{3.42}$$

Leurs fonction étant composée, elle devrait normalement posséder un domaine de définition pour chaque partie. Chose qu'il n'y a pas. On ne sait donc pas à quel intervalle la fonction est nulle. Mais nous pouvons deviner que l'allure de la fonction est triangulaire. Les auteurs n'expliquent pas très bien le sens de c_{ij}, b_{ij}^- et b_{ij}^+.

Erreur de Positin (*E_Pos*)			Erreur Angulaire (*E_Ang*)						
			NG	NM	NP	Z	PP	PM	PG
	Z	φ	*PM*	*PP*	*Z*	*Z*	*Z*	*NP*	*NM*
		V_m	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*
	P	φ	*PG*	*PG*	*PM*	*Z*	*NM*	*NG*	*NG*
		V_m	*F*	*F*	*F*	*F*	*F*	*F*	*F*
	M	φ	*PM*	*PM*	*PP*	*Z*	*NM*	*NG*	*NG*
		V_m	*F*	*F*	*M*	*M*	*M*	*F*	*F*
	G	φ	*PM*	*PP*	*PP*	*Z*	*NP*	*NP*	*NM*
		V_m	*F*	*M*	*G*	*G*	*G*	*M*	*F*
	TG	φ	*PM*	*PM*	*PP*	*Z*	*NP*	*NM*	*NM*
		V_m	*F*	*M*	*G*	*TG*	*G*	*M*	*F*

FIG. 3.16 – Table d'inférence Ouadah et al.

3.4.5.3 Le système d'inférence flou

Cette concerne l'élaboration des règles qui détermines le comportement attendu du robot. Généralement, les règles se présentent sous la forme d'une (ou plusieurs) tables qui pour chaque degrés d'appartenance de chaque entrée à chaque ensemble flou, la table retourne le fuzzifié de la réponse attendue.

Ouadah et al ont une table telle qu'illustrée dans la figure 3.16. Sa grande taille est principalement due à la richesse de l'univers de discours de leurs régulateur. Le mode d'utilisation de cette table est illustré par cet exemple (cases en ligne doublée) : Si E_{Pos} est petite (P) et E_{Ang} est nulle (Z), alors le but est tout droit mais à une petite distance. Dans ce cas, le robot doit avancer avec une vitesse V_m faible (F) sans changer de direction (ψ est Z).

Les auteurs constatent que le choix optimal des fonctions d'appartenance et des règles représente une tâche difficile. Elle requièrt du temps, de l'expérience et de l'habilité de la part de l'expérimentateur.

Peri et Simon proposent deux tables (les tables 3.2 et 3.3), la première pour la vélocité angulaire de la roue droite Δw_r, la seconde pour la vélocité de la roue gauche Δw_l. Il est possible à partir des termes linguistiques définissant les variables d'entrée d'assigner un terme linguistique à Δw_l et Δw_r.

$\Delta e_x/\Delta e_t$	N(negative)	Z(zero)	P(positive)
N(negative)	S(slow)	S(slow)	S(slow)
Z(zero)	S(slow)	F(fast)	M(medium)
P(positive)	S(slow)	M(medium)	F(fast)

TAB. 3.2 – Table de règles pour Δw_r.

$\Delta e_x/\Delta e_t$	N(negative)	Z(zero)	P(positive)
N(negative)	F(fast)	M(medium)	S(slow)
Z(zero)	M(medium)	F(fast)	S(slow)
P(positive)	S(slow)	S(slow)	S(slow)

TAB. 3.3 – Table de règles pour Δw_l.

3.4.5.4 La défuzzification

La dernière étape est la transformation des valeurs de commande du domaine flou vers le domaine réel.

Ouadah et al ont opté pour un calcul du centre de gravité. Le passage vers une forme calculable transforme l'intégrale en une sommation :

$$z^* = \frac{\sum_{i=1}^{m} z_i \mu(z_i)}{\sum_{i=1}^{m} \mu(z_i)} \tag{3.43}$$

Dans cette formulation, m est le nombre de règles, z la variable de commande.

Peri et Simon ont de même opté pour le calcul du centre de gravité. Cependant, leurs méthode est plus complexe que celle de Ouadah et al et se base sur trois équations.

3.5 Les méthodes carte routières (RoadMap)

3.5.1 Roadmap

Une carte routière R ou Roadmap est l'union de courbes uni-dimensionnelles de manière à ce que pour tout q_{depart} et q_{but} dans FS (l'espace libre non obstrué), il existe un chemin entre q_{depart} et q_{but} si et seulement si :

- il existe un chemin depuis $q_{depart} \in FS$ vers $q'_{depart} \in FS$ (accessibilité),
- il existe un chemin depuis $q_{but} \in FS$ vers $q'_{but} \in FS$ (atteignabilité),
- il existe un chemin depuis q_{depart} vers q'_{but} dans R (connectivité).

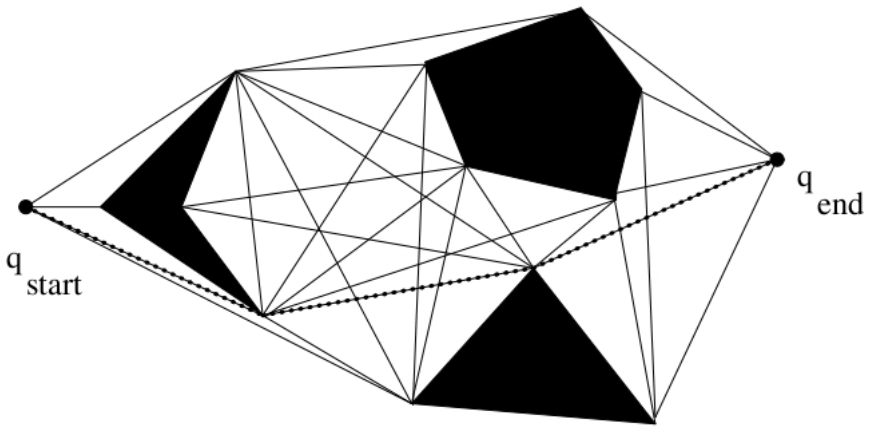


FIG. 3.17 – Un graphe de visibilité.

En utilisant une Roadmap, un planificateur peut construire un chemin entre deux points quelconques en commençant par trouver un chemin sans collision (accessibilité), traversé la roadmap vers le but (connectivité) et alors construire un chemin sans collision depuis un point quelconque de la roadmap vers le but (atteignabilité).

3.5.2 Graphe de visibilité

Les noeuds d'un graphe de visibilité incluent la position de départ, le position du but et toutes les arêtes des obstacles (ou la configuration de l'espace d'obstacles). Les arcs du graphes de visibilité sont des segments de droites rectilignes qui connectent deux noeuds sans pénétrer l'espace d'obstacle. L'arête d'un polygone d'obstacle peut aussi servir d'arc pour le graphe de visibilité vu qu'il n'entre pas en collision avec l'espace d'obstacles. Il est possible de rechercher dans le graphe pour le chemin le plus court dans R^2 (fig 3.17). Il n'est pas certain de trouver le chemin le plus court pour des dimensions supérieurs à 2.

3.5.3 Rétraction

Une rétraction est un mapping continu de l'espace libre FS en un sous-ensemble uni-dimensionnel de FS. L'image de la rétraction est la roadmap. La rétraction la plus populaire dans la planification de chemins est appelée diagramme de Voronoi.

Cette structure est un graphe dont les arcs sont l'ensemble de points qui ont une distance minimale par rapport aux frontière de l'espace libre en deux points différents et dont les sommets sont l'ensemble de points qui ont une distance minimale par rapport aux frontière de l'espace libre en trois points différents (où les arcs se rencontrent).

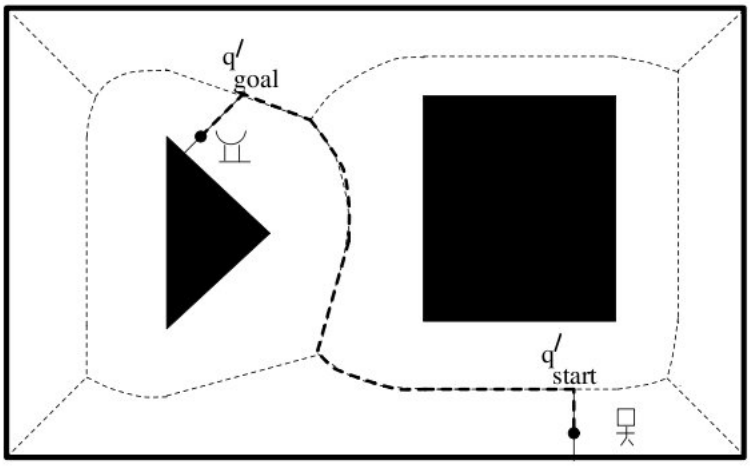


FIG. 3.18 – Un graphe de visibilité.

L'accessibilité (resp. l'atteignabilité) est garantie entre le point de départ et d'arrivée en faisant une projection directe à partir d'un de ces points sur l'arc le plus proche (voir figure 3.18). Initialement, les diagrammes de Voronoi sont plutôt utilisés dans le domaine de la géomètrie. Le diagramme de Voronoi originel est l'ensemble de points ayant une distance minimal par rapport à un ou plusieurs sites.

L'avantage des diagrammes de Voronoi généralisés est qu'ils peuvent produire un chemin qui évite au maximum les obstacles. L'un de ces inconvénients est qu'il ne peut fonctionner correctement dans des dimensions supérieurs à 2.

3.5.4 Méthode de la Silhouette

Il existe une approche roadmap pouvant travailler sur des espaces de dimensions supérieur à 2. Un hyperplan, appelé membrane, traverse l'espace de travail ou l'espace de paramètres de configurations, contenant les obstacles. Généralement, la membrane est perpendiculaire à un axe de coordonnées. Chaque membrane, continent des points exprimant les extremum d'une fonction. Ces point tracent des courbes appelées silhouette de courbe. En général, ces courbes ne sont pas forcément connectés. L'union des silhouettes de courbes forment la roadmap.

3.5.5 Décomposition cellulaire

Dans la décomposition cellulaire, l'espace libre du robot est décomposé en régions appelées cellules. Il y a deux types de décompositions cellulaires : exacte et approximative. La décomposition cellulaire exacte décompose l'espace en cellules dont l'union produit l'espace libre lui même. Dans cette approche, les limites adjacentes entres cellules voisines ont généralement un sens physique comme un soudain changement de la proximité d'un obstacle.

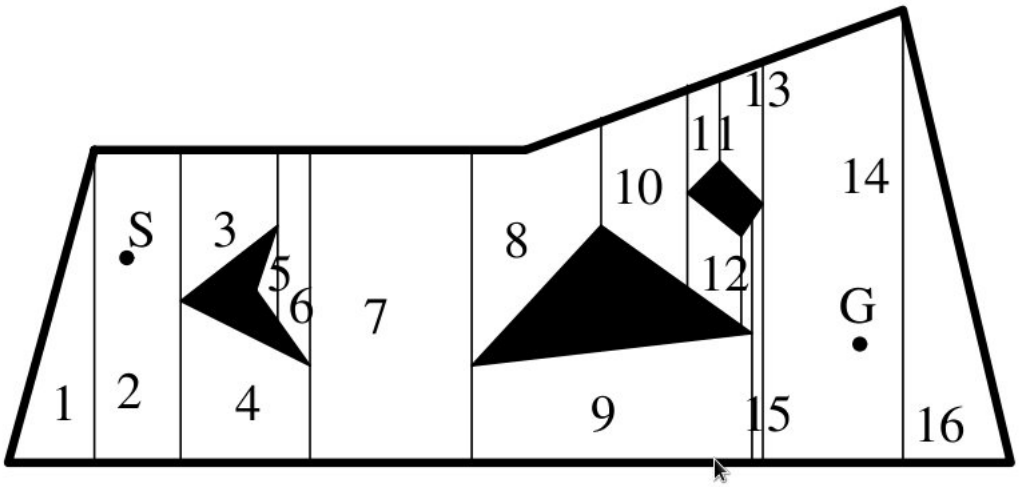


FIG. 3.19 – Une décomposition trapézoide.

Dans les décomposition approximative, l'espace libre est décomposé en cellule ayant des formes prédéfinies comme des pixels ou des images. Généralement, l'union de toutes ces cellules approximatives est un sous-ensemble de l'espace libre.

Le planificateur de chemin doit construire un graphe de connectivité ou chaque sommet représente une cellule et les arcs représentent des relations de voisinages directes entre ces cellules. La recherche d'un chemin dans le réel perçu s'apparente alors à la recherche de chemins dans un graphe.

3.5.6 Décomposition trapézoide

Un exemple d'une décomposition cellulaire exacte peut être trouver dans la figure 3.19. Chaque cellule est soit un triangle ou un trapèze lors du dessin de lignes verticales à travers chaque sommet des polygones. L'extrémité de chaque segment est la limite de l'espace libre. Deux cellules sont adjacentes si elles partagent une frontière en commun. Le graphe de connectivité peut alors être construit pour rechercher un chemin entre le départ et le but. Un graphe de connectivité de la décomposition cellulaire de la figure 3.19 est visible sur la figure 3.20.

3.5.7 Décomposition généralisée de Voronoi

La figure 3.21 contient une décomposition selon la méthode de Voronoi ou chaque cellule est l'ensemble de points le plus proche d'un objet en particulier. Le graphe de connectivité 3.22 de cet environnement est le dual du diagramme généralisé de Voronoi de la figure 3.18.

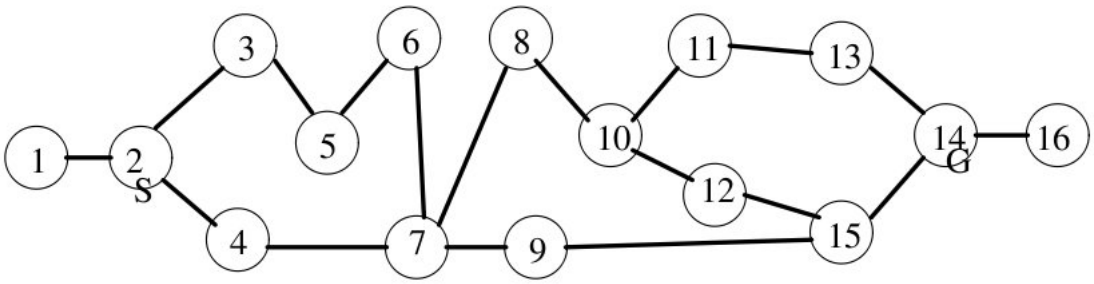


FIG. 3.20 – Le graphe de connectivité equivalent à la décomposition trapézoidale

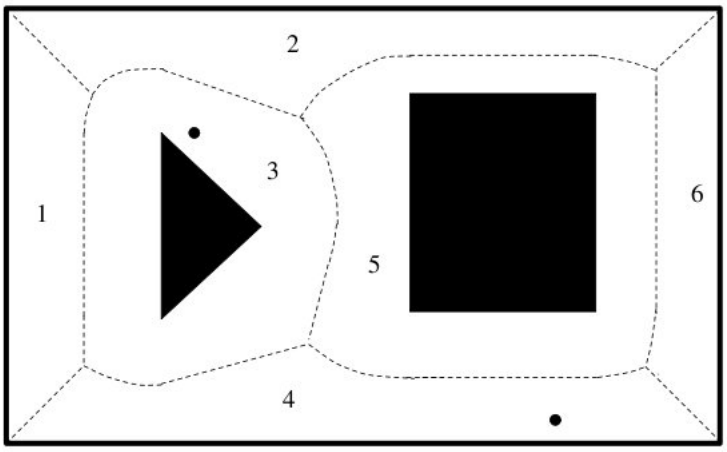


FIG. 3.21 – Une décomposition selon Voronoi

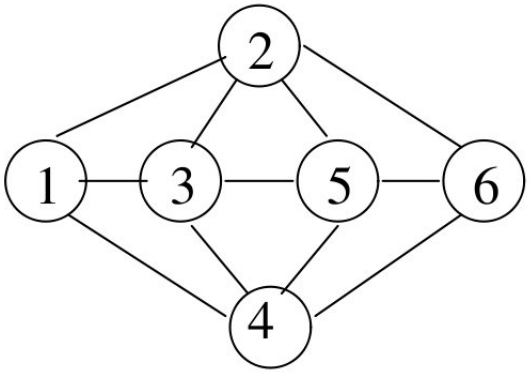


FIG. 3.22 – Le graphe de connectivité correspondant à la décomposition selon Voronoi

3.5.8 Critiques de méthodes roadmap

Les méthodes vues dans cette section sont très robustes à l'exécution, mais elles souffrent de certains défauts insurmontables pour une navigation autonome d'un robot.

Premièrement, ces méthodes nécessitent de connaitre l'ensemble de l'environnement en question, c-a-d de posséder une cartographie globale de ce dernier. Il a été démontré dans le premier chapitre que les capteurs nécessaires à une telle cartographie pour un environnement inconnu sont soit inexistants (pour un environnement fermé) soit très chers (pour un environnement ouvert avec cartographie satellite ou aérienne en temps réel).

Deuxièmement, ces méthodes ne prennent en aucun cas la dynamicité de l'environnement comme contrainte. L'environnement est statique. Nous pouvons imaginer que ces algorithmes soient appliqués plusieurs fois pas seconde a une cartographie globale dynamique, encore faut-il pouvoir obtenir cette cartographie.

Finalement, les cartographies effectuées en haute altitude ne permettent pas de distinguer certaines issues. Par exemple, ces méthodes considèreront un pont comme étant un obstacle et n'envisageront pas la possibilité de passer en dessous de lui, même chose pour les tables, les toits, les plateformes élevées etc.

Deuxième partie

Proposition de Champs de Potentiels et d'Algorithmes Génétiques pour la planification de chemins

Chapitre 4

Proposition d'algorithmes génétiques pour la planification locale de chemins

Dans le chapitre précédant, nous avons évoqué les différentes méthodes utilisées pour planifier un chemin de manière évolutionniste. La remarque principale que nous pouvons en tirer, est que, ces algorithmes ne peuvent fonctionner que dans le cadre d'une cartographie globale.

Une seconde remarque est que ces algorithmes sont appliqués sur des populations de chemins complexes. En effet, tous les auteurs partent de l'hypothèse que les chemins doivent obligatoirement relier la position initiale à la position finale. Cet particularité doit être constante le long des générations. Nous savons combien il est complexe pour un algorithme génétique de trouver un chemin optimal dans cet optique surtout dans le cadre d'un environnement dont les obstacles forment des motifs compliqués tels que les labyrinthes.

Précédemment, les auteurs, dans leurs approches, se heurtent à divers problèmes tels que les "switching point" et la difficulté de les intégrer dans une fonction de fitness.

Nous proposons dans ce chapitre une approche évolutionniste qui se base sur une cartographie locale. La portée des capteurs (que nous supposons être des Sonars) ne permet pas une approche évolutionniste classique telle que présentée dans le second chapitre. Donc, le processus évolutionniste se fait en étape, c-a-d qu'à chaque position, le robot évalue les chemins en sa disposition dans la population, effectue des croisements et mutation, élimine les chemins les moins intéressants, choisi le meilleur de ces chemins et bouge selon celui-ci. Une fois le chemin parcouru, le robot réitère le processus en créant une nouvelle population etc ...

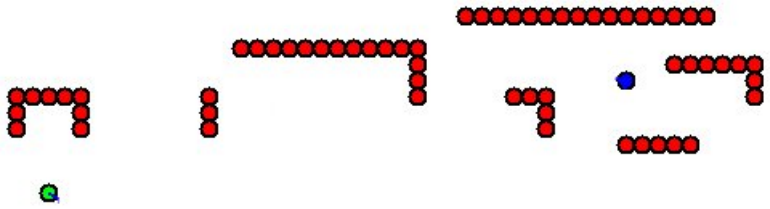


FIG. 4.1 – L'environnement de simulation.

4.1 La représentation de l'environnement

Puisque nous utilisons une cartographie locale, l'environnement que perçoit le robot dépend directement de la portée de ses capteurs. Pour la simulation, nous avons opté de considérer les obstacles comme des cercles de diamètre de 10 unités. La forme circulaire des obstacles permet un calcul de collision plus aisé, les petites dimensions de ces derniers nous permettent d'envisager des obstacles aux formes les plus extravagantes.

Nous considérons les dimensions du robot de même grandeur que celles des obstacles élémentaires et ceci dans un souci de simplification. Finalement, le but est aussi de même dimension pour une cohérence globale de la simulation.

4.2 Ères et générations

Contrairement aux algorithmes évolutionnistes classiques qui considèrent une ère unique subdivisée en une quantité de générations (plusieurs centaines), nous considérons notre algorithmes comme un ensemble d'ère (une dizaine) subdivisé en quelques dizaines de générations.

Cela résulte du fait que chaque ère nous produit un mini-chemin élémentaire d'un chemin plus grand. Ce chemin élémentaire est évidemment le meilleur individu issu du processus d'évolution entamé de cette ère. Une fois le chemin parcouru, une tout nouvelle population doit être générée pour remplacer l'ancienne inaugurant une nouvelle ère.

Le chemin obtenu n'est au final que la concaténation des chemins représentants les meilleurs individus résultant du processus évolutif de chaque ère.

4.3 Création de la population

La création de la population est l'étape initiale de toute ère dans l'algorithme. La population est un ensemble de chromosomes représentant chacun un chemin. Chaque

chromosome est un ensemble de gène représentant un mouvement élémentaire. Soit l'ensemble de toutes les population possibles Pop et C l'ensemble de tous les chromosomes :

$$P = \bigcup_{i=0}^{n} Ch_i : Ch_i \in C, P \in Pop \tag{4.1}$$

4.3.1 Le gène

Le gène est le constituant élémentaire du chromosome. Dans le réel perçu, il représente un mouvement élémentaire. Ce mouvement élémentaire est un couple $(\Delta x, \Delta y)$. Le pas est un couple de scalaires dont les éléments sont compris entre -9 et +9 qui représentent le nombre d'unité de distance à parcourir dans l'axe des ordonnées et des abscisses. Algébriquement parlant un couple $(\Delta x, \Delta y)$ appartient à l'ensemble des gène G ssi :

$$(\Delta x, \Delta y) \in G : \Delta x \in [-9, +9], \Delta y \in [-9, +9] \tag{4.2}$$

Puisque $card([-9, +9]) = 19$ donc, l'ensemble des gène est de cardinalité finie $card(G) = 19^2 = 361$.

4.3.2 Le chromosome

Le chromosome est un ensemble de gène. Donc, c'est un ensemble cohérent de déplacements élémentaires formant un chemin. Soit Ch un chromosome de l'ensemble de chromosome C, nous avons :

$$Ch = \bigcup_{i=0}^{n} g_i : g_i \in G, Ch \in C \tag{4.3}$$

Il faut remarquer que pour une nombre n de déplacements, il est possible de construire au maximum 361^n chromosomes syntaxiquement différents. Syntaxiquement, car il existe une multitude de chemins équivalents issus de retours arrière comme par exemple : "monter de 5 unités" et "descendre de 3 unités" qui équivalente à "monter de 2 unités".

4.4 La fonction de fitness

La fonction de fitness est la brique essentielle de l'algorithme. Elle a pour but d'évaluer (noter) les individus. La fonction de fitness est en rapport direct avec les contraintes de la navigation, c-a-d, l'absence de collisions, une taille de chemin réduite (autonomie des batteries) et la proximité du but.

4.4.1 Calcul de la taille du chemin

Les chemins ne contenant pas de boucles sont les chemins les plus courts. Or, il est essentiel de favoriser les chemins ne contenant pas de boucles. En effet, un chemin non-bouclé satisfait le critère d'économie d'énergie dans la navigation de robots mobiles autonomes.

Vu la manière que nous avons utilisé pour exprimer un chemin dans un chromosome, la taille d'un chemin se réduit à une sommation directe de touts les pas de direction contenus dans les gènes de celui-ci selon l'équation suivante :

$$T_{Ch} = \sum_{i=1}^{card(Ch)} \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2}\ ,\ avec\ (\Delta x_i, \Delta y_i) \in g_i \in Ch \tag{4.4}$$

Exemple : Soit le chemin défini par le chromosome $\{(4,5),(1,8),(7,2)\}$:
La taille du chemin représentée par ce chromosome réduit est : $\sqrt{(4^2+5^2)}+\sqrt{(1^2+8^2)}+\sqrt{(7^2+2^2)} = 21,8$

4.4.2 Calcul du nombre de collision

Notre environnement est constitué d'éléments circulaires de rayon de 5 unités. Notre trajet est lui aussi constitué de mouvements élémentaires (voir section 4.3.1). Géométriquement, le trajet est un ensemble connecté de segments de droite.

L'interet de cette considération est de pouvoir détecter les collisions de trajet avec les obstacles $o_i(x_i, y_i)$ qui sont à portée du capteur (cartographie locale), c-à-d $RO_i < S$ où R est la position du robot, O_i la position de l'obstacle et S le seuil de la portée du capteur (dans notre simulation considéré à 40 unités).

L'équation décrivant la distance d'un point par rapport à une droite est telle que :

$$d(A,(d)) = AA_h = \frac{|ax_A + by_A + c|}{\sqrt{a^2+b^2}} \tag{4.5}$$

Calculer la distance de tous les obstacles par rapport à tous les segments de droite constituant le trajet grace à l'équation précédente n'est pas une garantie suffisante quant à l'existence réelle d'une collision. Il ne faut pas oublier que l'équation considère une droite (AB) plutôt d'un segment de droite $[AB]$ ce qui donne lieu à une multitude de faux positifs tels que représentés dans la figure ...

Pour éliminer les cas de faux positifs, nous envisageons une triangulation, c-a-d si nous considérons les points A et B appartenant à un segment de droite du trajet et que O est la position de l'obstacle, il faudra calculer toutes les combinaisons de distance de points et de segments de droite,ie , calculer $d(O,(AB)), d(A,(OB)), d(B,(OA))$.

Le critère pour une collision est d'avoir cette assertion vraie :

$$\begin{array}{l} \left\{ \begin{array}{l} d(O,(AB)) < r_{obstacle}\ et \\ d(O,(AB)) < d(A,(OB))\ et \\ d(O,(AB)) < d(B,(OA)) \end{array} \right. \\ ou \\ \left\{ \begin{array}{l} d(O,B) < r_{obstacle}\ ou \\ d(O,A) < r_{obstacle} \end{array} \right. \end{array} \tag{4.6}$$

Si cette assertion est vraie pour certains segments pour un chemin, cela incrémente autant de fois qu'il y a de collision la variable N_c qui représente le nombre de collision pour ce chemin.

4.4.3 La fonction finale

La fonction de fitness est un paramètre critique de l'algorithme qu'il faut choisir avec beaucoup de parcimonie. Il doit répondre aux attentes de la navigation des robots mobiles autonomes quant à l'économie de l'énergie, en préférant les chemins courts, et l'absence de collision, en éliminant les chemins qui en contiennent. Le but ultime étant bien sûr d'atteindre la cible.

Des sous-sections précédentes, nous avons relatés la manière de calculer pour un chemin, sa longueur T_{ch} et son nombre de collisions N_c. Puisque le robot doit se diriger vers la cible, la distance du dernier point du trajet (position future du robot) par rapport à la cible d_{cible} est un paramètre incontournable dans la fonction de fitness.

Il n'existe aucune règle ou loi ou algorithme qui nous permette de choisir cette fonction. Donc, nous devons le faire de manière empirique grâce à l'expérience. Pour calculer le fitness d'un chromosome ch, nous avons opté pour une équation du genre :

$$F_{ch} = \frac{K}{A\ N_c + B\ T_{ch} + C\ d_{cible}} \tag{4.7}$$

où K,A,B et C sont des constantes à choisir.

4.5 La sélection de chemins

La selection de chemins est une opération élémentaire pour la programmation génétique. Elle permet de choisir les individus les plus aptes à transmettre leurs gênes pour les générations suivantes tout en respectant le souci d'améliorer la population.

Plus précisemment, la sélection est l'étape qui vient en amont de l'étape de croisement qui elle permet la création de nouveaux individus dits "enfants" moyennant

un brassage génétique des deux parents.

Notre application utilise une selection par rang. Cela veut dire que les parents sont choisis au hasard mais en favorisant les individus ayant les meilleurs fitness. Pour effectuer ce choix nous faisons une analogie à la roulette. Tous les individus sont représentés par une portion sur cette roulette dont la taille est proportionnelle aux fitness de ces derniers.

Algorithmiquement, nous effectuons une sommation des fitness de tous les individus qui représente le portour de la roulette :

$$T = \sum_{i=1}^{Tpop} fitness_i \tag{4.8}$$

Donc chaque fitness représente une partie de cette somme. Nous choisissons aléatoirement un nombre compris entre 0 et T soit a :

$$a \in [0, T] \tag{4.9}$$

Une troisième variable S est initialisée à 0 que nous incrémentons avec les valeurs de fitness des individus. Cela est l'analogie de la roulette qui tourne. Nous nous arrêtons lorsque S est égale à a. L'individu dont le fitness est le dernier à être incrémenté est choisi.

4.6 Le croisement de chemins

Une fois deux chemins sélectionnés par l'opérateur de sélection, ils seront considérés comme parents. L'opérateur de croisement permet de générer des enfant qui hériteront du mélange du code génétique entre le père et la mère.

Nous avons opté pour un croisement à point unique. En effet, nous considérons la plus petite taille entre les deux parents (qui sont considérer comme étant une liste de gènes) et *len* un opérateur qui retourne la taille d'une liste :

$$l = min(len(mere), len(pere)) \tag{4.10}$$

Nous choisissons un point aléatoire $p \in [1, l]$, si nous considérons les parents comme étant des listes et l'opérateur de concaténation +, les deux enfants issus du croisement sont :

$$\begin{aligned} enfant1 &= pere[1:p] + mere[p:len(mere)] \\ enfant2 &= mere[1:p] + pere[p:len(pere)] \end{aligned} \tag{4.11}$$

Les deux enfants sont ensuite ajoutés dans la population.

4.7 La mutation de chemins

La mutation de chemins permet à l'algorithme génétique de ne pas rester piégé dans des optimums locaux. En effet, lors de l'apparition d'individus au fitness fort (super-individus), ces derniers ont tendance a être choisis à excès et de pousser la population a hérité de leurs codes génétiques au détriment de la diversité possible.

Le processus de mutation se fait avec une fréquence relativement faible de l'ordre de 2 générations sur 100. C'est une modification directe sur un ou plusieurs gènes du chromosome.

Cette opération débute par le choix aléatoire d'un individu (les probabilités de choix sont uniformes). Un gène pris au hasard dans ce code est modifié. Soit g le gène choisi pour mutation :

$$g = (\Delta x, \Delta y) \Rightarrow g' = (\Delta x', \Delta y) \; ou \; g' = (\Delta x, \Delta y') \; ou \; g' = (\Delta x', \Delta y') \qquad (4.12)$$

Evidemment, g' est le gène qui remplacera désormais g dans le chromosome.

4.8 L'élimination des individus faibles

L'application répétée de l'opérateur de croisement conduit à une surpopulation. Il est essentiel de garder une taille constante au début de chaque génération. L'étape d'élimination permet de supprimer les individus les plus faibles pour ramener la taille de la population à la constante voulue.

Dans notre application, nous n'utilisons pas une méthode basée sur l'hasard. Nous utilisons une méthode inverse à l'élitisme. Nous choisissons les individus aux fitness les plus faibles et nous les éliminons.

4.9 Algorithme d'une génération

La toute première génération débute par la création d'une population initiale aléatoire. Cette population contient un nombre défini de chemins. Les chemins ont tous une longueur maximale égale à la portée du capteur du robot, cela pour éviter que le chemin ne se prolonge vers un obstacle qui n'est pas à portée du capteur. Tous ces chemins sont évalués et leurs valeurs de fitness sont stockées.

Dés lors et pour le reste des générations, un cycle s'engage. Il y'a d'abord la sélection d'un couple qui préfigure la phase de croisement. Le croisement génère deux enfants dont la valeur de fitness est stockée. Pour une moyenne de deux générations sur 100, une phase de mutation d'enclenche. Finalement, deux individus, les plus faibles, sont éliminés pour garder une cardinalité constante pour la population.

Génération :
Sélection des parents
Croisement des parents
Calcul de fitness des enfants
Mutation d'un individu (si il y a lieu de le faire)
élimination des individus les plus faibles.

Ainsi se termine une génération qui transmet à la suivante une population de taille stable est légèrement modifiée (voir améliorée).

4.10 Algorithme de l'ère et critère d'arrêt

La première ère inaugure la première génération qui a été évoquée plus haut. Celle ci prend pour point de repère la position initiale du robot qui sera utilisée lors des générations pour l'élaboration des chemins possibles. Une ère voit s'écouler plusieurs dizaines de générations. A la fin de la dernière génération, le meilleur chemin est choisi. Il sera considéré comme étant le chemin élémentaire qui augmentera le trajet du robot.

Lors des ères suivantes, la position finale à laquelle le robot se trouve après avoir suivi le meilleur chemin de l'ère précédente est choisi comme étant le nouveau repère de la nouvelle ère. La population est intégralement recréée. L'ère débute sur de nouvelles bases.

ère :
Considérer le dernier point du chemin précédant comme base
$i \leftarrow 0$
tq $i < N_{generation}$ faire
début
génération
$i \leftarrow i + 1$
fin

Le critère d'arrêt de l'algorithme est l'arrivée au but ou du moins à proximité de ce dernier. Cela se traduit géométriquement par une distance entre le point final et le but inférieure à un seuil :

$$distance(fin, but) < Seuil \tag{4.13}$$

Ce critère n'est malheureusement pas suffisant pour garantir l'arrêt de l'algorithme. En effet, comme il n'est pas possible de prouver que l'algorithme atteindra le but à chaque fois et qu'il ne se trouve pas piéger dans des boucles. Pour cela, une variable de sécurité est ajoutée qui est périodiquement incrémentée et qui arrête

l'algorithme si celui-ci tourne trop longtemps. L'algorithme générale est de la forme :

Algorithme :

Considérer le position initiale comme base
$securite \leftarrow 0$
tq $distance(fin, but) < Seuil$ et $securite < N_{eremax}$ faire
début
  ère
  $securite \leftarrow securite + 1$
fin

4.11 Test de l'approche évolutionniste

Dans un souci de valider l'efficacité de la méthode proposée dans le chapitre précédant, nous avons développé une application en Python sous l'environnement Linux. Nous avons crée notre propre librairie d'algorithme génétique et cela dans un souci de contrôle absolu de la solution proposée.

Ce chapitre va dans un premier temps introduire les spécificités de l'aspect logiciel de la simulation. Les grandes lignes y seront présentées. Dans un second temps, nous allons définir l'environnement matériel sur lequel s'est effectué la simulation, pour avoir une bonne idée de la masse de calcul nécessaire par la suite. Finalement, puisque la méthode est très paramétrique, nous allons varier certains variables de l'application et constater l'impact globale se produisant sur la durée de calcul et la qualité de la solution.

4.11.1 Application développée

Nous avons développé une application qui reprend les idées proposées dans le chapitre précédant. Cette application a été écrite en Python sous le système d'exploitation Linux Ubuntu 8.04 LTS. Le choix du langage est dû principalement aux énormes qualités et le succès indiscutable qui le qualifient.

Traditionnellement, ce genre de méthodes sont développés sous C ou C++ voir même Java. Mais il est indéniable que travailler sous ces langages demandent un temps et un effort intellectuel importants. Python base sa philosophie sur la simplicité d'écriture de code. Écrire un code sous Python apporte un gain de temps d'un facteur oscillant entre 2 et 10. Donc, l'étudiant, plutôt que de réflichir à la manière d'implémenter une solution va plutôt travailler à améliorer celle-ci sans contrainte de programmation.

Nous avons opté pour le système libre Linux Ubuntu 8.04 LTS pour s'essayer à l'univers de développement libre et tester les différentes alternatives qu'il propose

par rapport à d'autre solution payante.

Python étant Orienté Objet, nous avons trouvé une grande facilité à développé le *moteur* génétique. En effet, l'algorithme génétique et une simple classe au sein de laquelle se retrouve toutes les opérandes habituelles au concept.

L'algorithme du tableau 4.1 est un résumé de la classe qui, en réalité, contient des algorithmes de calculs de collisions, de longueurs de chemins et de distance de but pour évaluer les individus (voir le chapitre 3). Elle contient aussi les règles de réductions de chemins pour obtenir le chemin optimal d'une classe de chemins équivalents. Nous avons essayer, sans grands succès, d'implémenter une technique d'élimination de boucles, cependant, nous avons remarqué que nous pouvons obtenir des chemins sans boucles en paramétrant correctement la fonction d'évaluation et en choisissant un nombre suffisant de'individus dans la population.

Pour l'interface de l'application, nous avons opté pour Tk. Tk est une librairie fournie avec Python assez simpliste mais qui se prête parfaitement à nos besoins. Cette librairie nous offre les mécanismes nécessaires à l'affichage de l'environnement, à sa modification à la volée, le tout pour une consommation de processeur et de mémoire négligeables.

4.11.2 Configuration du matériel

La configuration matérielle est nécessaire si nous voulons nous faire une idée de la masse de calcul nécessaire à la génération d'un chemin. Il est de notoriété publique que les algorithmes génétiques sont très gourmands en temps de calcul. A la base de notre configuration chacun peut extrapoler quant au temps de calcul que cela prendrait sur des plateformes plus ou moins récente que la notre.

Le tableau 4.2 résume l'environnement sur lequel a été effectuer la simulation et à partir duquel nous avons obtenus nos résultats lors de l'expérimentation.

4.11.3 Expérimentations paramétriques

Dans cette section sera relaté la sensibilité de l'algorithme génétique aux différents paramètres. Par sensibilité de l'algorithme nous sous-entendant la durée d'exécution et la qualité des résultats produits. Par paramètres nous sous-entendant le nombre d'individus dans la population, le nombre de générations dans chaque ère et aussi le ratio de mutation. Il existe bien plus de paramètres possibles tels que la méthode de sélection (qui n'influe pas réellement sur le processus) ou le type de croisement (difficile à implémenter plus d'un type).

Pour ce qui suit, nous allons adopté des abréviations pour les différentes variables étudiées dans cette expérimentation. T_p exprimera la taille de la population, T_e

Algorithme :

```
class genetic() :
    def init(self) :
        pass

    def creer_population(self) :
        ...
        return population

    def evaluation(self,population) :
        ...
        return population

    def selection(self,population) :
        ...
        return individu

    def croisement(self,pere,mere) :
        ...
        return [enfant1,enfant2]

    def elimination(self,population) :
        return population

    def generation(self,population) :
        evaluation(population)
        mere = selection(population)
        pere = selection(population)
        population += croisement(pere,mere)
        elimination(population)

    def era(self,nb_generation) :
        population = creer_population()
        i = 1
        while (i¡nb_generation) :
            generation(population)
            i += 1
```

TAB. 4.1 – Squelette de la classe genetic écrit en Python

Ordinateur	HP Laptop 510
Processeur	Intel Inside Centrino Mobile Technology 2.0 GHz
RAM	512 Mo DDR
Cache	256 Ko
DD	60 Go
OS	Linux Ubuntu 8.04 LTS

TAB. 4.2 – Environnement de simulation

exprimera le temps d'exécution, N_g exprimera le nombre de générations dans une ère. Il y aura autant d'ère que nécessaire pour atteindre le but.

4.11.3.1 Variations de T_e par rapport à T_p

Dans cette section nous allons expérimenter la variation du temps d'exécution par rapport à la taille de la population. Ce test à pour but de se faire une idée de la complexité de l'algorithme en fonction du nombre d'individus.

Pour ce test, l'algorithme a été paramétré sur 25 générations par ère. Le temps est exprimé en secondes. Il faut remarquer que le temps dont il est question dans le tableau 4.3 désigne la durée globale nécessaire à l'algorithme pour que le robot atteigne son but depuis son emplacement initial.

T_p	$1^{er}\ Test$	$2^{eme}\ Test$	$3^{eme}\ Test$	$4^{eme}\ Test$	$5^{eme}\ Test$	$Moyenne$
50	44	62	75	66	59	**60**
100	79	52	52	42	59	**56**
150	66	59	62	91	71	**70**
200	105	93	93	99	85	**95**

TAB. 4.3 – Variations du temps d'exécution par rapport à la taille de la population

Le graphe 4.2 exprime la variation du temps d'exécution par rapport à la taille de population. La tendance générale est à l'accroissement ce qui s'explique logiquement par l'influence de la taille de la population sur l'algorithme de la génération notamment dans les phases de sélection et d'élimination qui prennent plus de temps avec la croissance de la population. Nous remarquons un phénomène intéressant lors le test à 100 individus qui s'avère moins consommateur en temps qu'un algorithme à 50 individus. L'explication que nous donnons est que bien que l'ère de 100 individus prenne plus de temps que celle de 50 individus, leur nombre dans le chemin global est plus réduit grâce à la meilleure qualité de ses individus. Au dessus de 100 individus le graphe prend une allure classique.

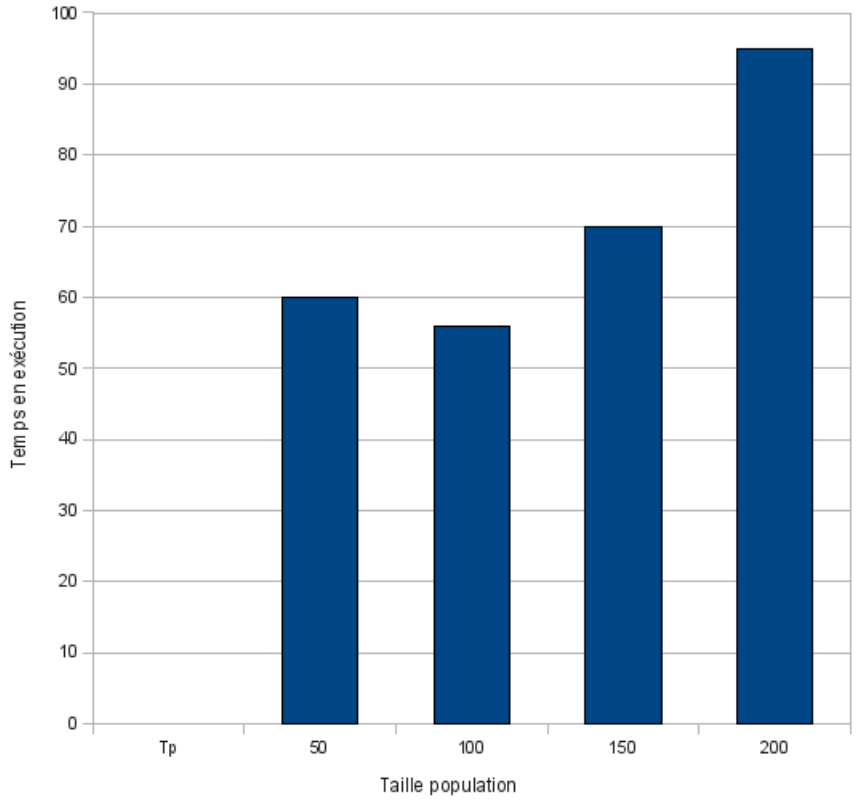


FIG. 4.2 – Diagramme Variations du temps d'exécution selon la taille de la population.

4.11.3.2 Variations de la qualité de chemins par rapport à T_p

Dans cette section nous allons expérimenter la variation de la qualité de chemin par rapport à la variation de la taille de la population, c-à-d qu'elle est l'apport qualitatif qu'engendre l'accroissement du nombre de chromosomes de la population.

Pour ce test nous avons considéré cinq types de populations (50,100,150,200,250 individus). Dans chaque test nous considérons la qualité du chemin, cette dernière est représentée par la longueur du chemin. Plus un chemin est de petite longueur plus il est de qualité. Pour chaque type de population nous avons effectué 05 tests. La qualité finale rapportée sur le graphe pour une population et la moyenne de ces 05 tests.

Le tableau 4.4 représente pour chaque type de population les 05 qualités issues d'expérimentation et la moyenne de ces derniers comme énoncé plus haut les plus petites valeurs représentent les meilleurs résultats.

Le diagramme 4.4 représente pour chaque population la qualité moyenne issue des expérimentations, la barre est d'autant plus petite et le chemin est de bon qualité.

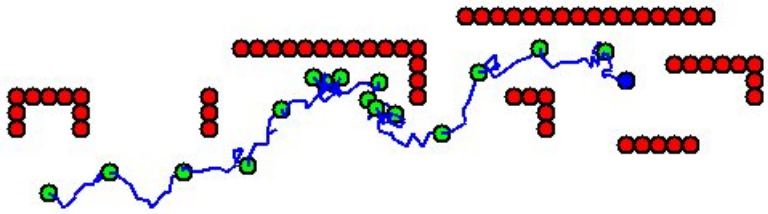


FIG. 4.3 – Une simulation réussie.

T_p	1^{er} $Test$	2^{eme} $Test$	3^{eme} $Test$	4^{eme} $Test$	5^{eme} $Test$	$Moyenne$
50	4237	2906	2519	2604	4438	**3341**
100	2767	2031	3010	2778	2414	**2600**
150	2747	2283	2670	2260	2352	**2474**
200	2352	2438	2078	2322	2310	**2300**
250	2302	2256	2318	2302	2113	**2269**

TAB. 4.4 – Variations de la qualité du chemin par rapport à la taille de la population

Nous remarquons une décroissance de la taille des barres dans le diagramme. Cela signifie une croissance de la qualité de chemins avec l'augmentation de chromosomes de la population. Cela s'explique aisément par le fait que l'augmentation de la population a pour effet d'enrichir le matériel génétique à la disposition de l'algorithme. Autrement dit, nous augmentons la probabilité d'atteindre un optimum local de bonne qualité.

Il faut aussi remarquer que la relation entre l'amélioration de la qualité et l'accroissement de la population n'est pas linéaire. Il existe donc une borne supérieure à ne pas dépasser que notre matériel ne nous a pas permis d'atteindre. Nous situons cette borne vers les 500 chromosomes.

4.11.3.3 Variations de la qualité de chemins par rapport à N_g

Dans cette section nous allons expérimenter la variation de la qualité de chemin par rapport à la variation du nombre de générations, c-à-d qu'elle est l'apport qualitatif qu'engendre l'accroissement du nombre de générations.

Pour ce test nous avons considéré quatre types d'algorithmes (5,15,25,35 générations). Dans chaque test nous considérons la qualité du chemin, cette dernière est représentée tel que dans la section précédente. Pour chaque type d'algorithme nous avons effectuer 05 tests. La qualité finale rapportée sur le graphe pour une population et la moyenne de ces 05 tests.

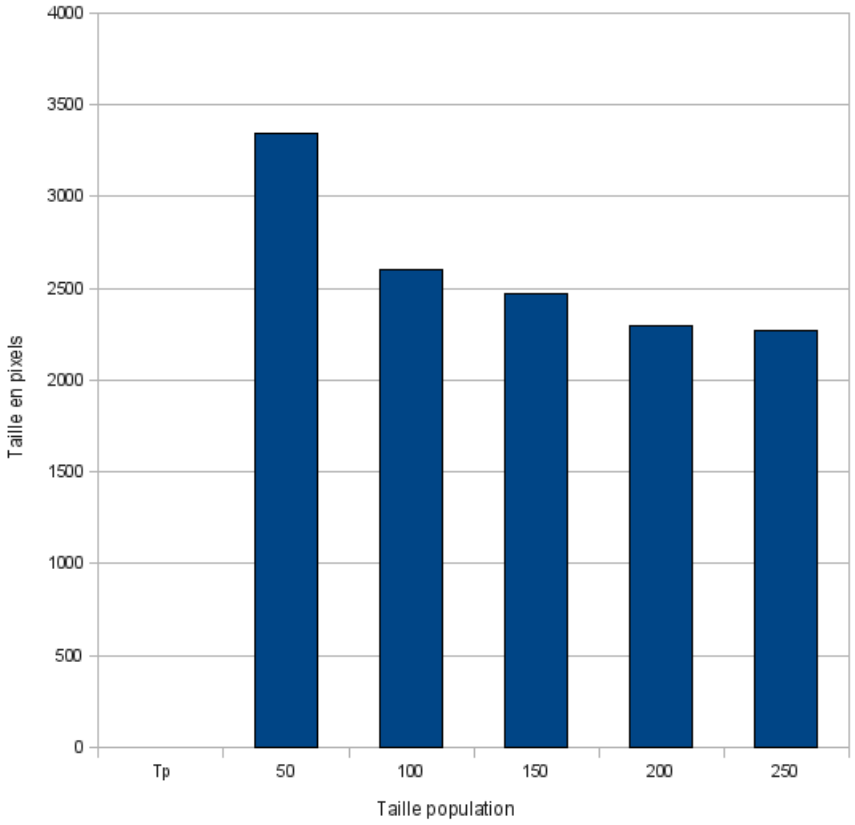


FIG. 4.4 – Diagramme Variations de la qualité selon la taille de population, les barres les plus petites sont les meilleurs

Le tableau 4.5 représente les différentes qualités de chemins obtenues après l'-expérimentation en variant le nombre de génération, cela pour une population constante de 50 individus.

Le tableau 4.6 représente les différentes qualités de chemins obtenues après l'-expérimentation en variant le nombre de génération, cela pour une population constante de 150 individus.

Le diagramme 4.5 représente pour un nombre n de génération la qualité moyenne issue des expérimentations, la barre est d'autant plus petite et le chemin est le bon qualité.

Nous remarquons une décroissance dans les barres oranges (avec 150 individus). Le premier cas signifie une croissance de la qualité de chemins avec l'augmentation de nombre de générations. Cela s'explique aisément par le fait que l'augmentation du nombre de générations a pour effet de prolonger le processus évolutif avec résultat d'améliorer plus finement l'individu final.

N_g	1^{er} $Test$	2^{eme} $Test$	3^{eme} $Test$	4^{eme} $Test$	5^{eme} $Test$	$Moyenne$
5	3959	3525	3684	3301	3150	**3522**
15	4137	2724	2693	3366	3030	**3190**
25	4237	2906	2519	2604	4438	**3341**
35	3018	3126	2890	2457	3169	**2924**

TAB. 4.5 – Variations de la qualité du chemin par rapport au nombre de génération pour une population de 50 chromosomes

N_g	1^{er} $Test$	2^{eme} $Test$	3^{eme} $Test$	4^{eme} $Test$	5^{eme} $Test$	$Moyenne$
5	3900	2202	2089	3018	2812	**2801**
15	2449	2190	2561	2438	3227	**2573**
25	2747	2283	2670	2260	2352	**2474**
35	2306	2140	2828	2500	2178	**2390**

TAB. 4.6 – Variations de la qualité du chemin par rapport au nombre de génération pour une population de 150 chromosomes

Il faut aussi remarquer qu'il y a une grande différence dans les valeurs de diagramme entre la population qui a 50 et 150 individus. Aussi, les valeurs du diagramme tendent à converger. Nous pouvons conclure qu'il existe une valeur pour laquelle l'algorithme n'améliore le résultat d'une manière que par un calcul démesuré.

4.11.4 Comportement de l'algorithme face au labyrinthe

L'algorithme a échoué à trouver un chemin dans le labyrinthe dans un temps convenable. Il trouve surement un chemin lorsque le temps d'exécution est très grand (figure 4.6).

Lorsque l'algorithme se trouve devant des couloirs étroits ou des environnements necessitant des retours arrières, il ne le fait pas spontanément. Puisque l'algorithme se base sur un processus aléatoire, on peut imaginer que l'algorithme atteigne le but aprés un très grand temps de calcul.

4.12 Conclusion

Lors de ce chapitre nous avons discuté de la mise à l'épreuve de notre algorithme. Nous avons présenté le squelette générale de la classe *genetic*, et la plateforme matériel utilisée. Nous avons aussi exprimé les raisons de nos choix quant au choix du langage de programmation et du système d'exploitation.

Nous avons procédé à 3 tests distincts. Le premier de ces tests portait sur la complexité de l'algorithme (à travers le temps d'exécution) par rapport à la taille de

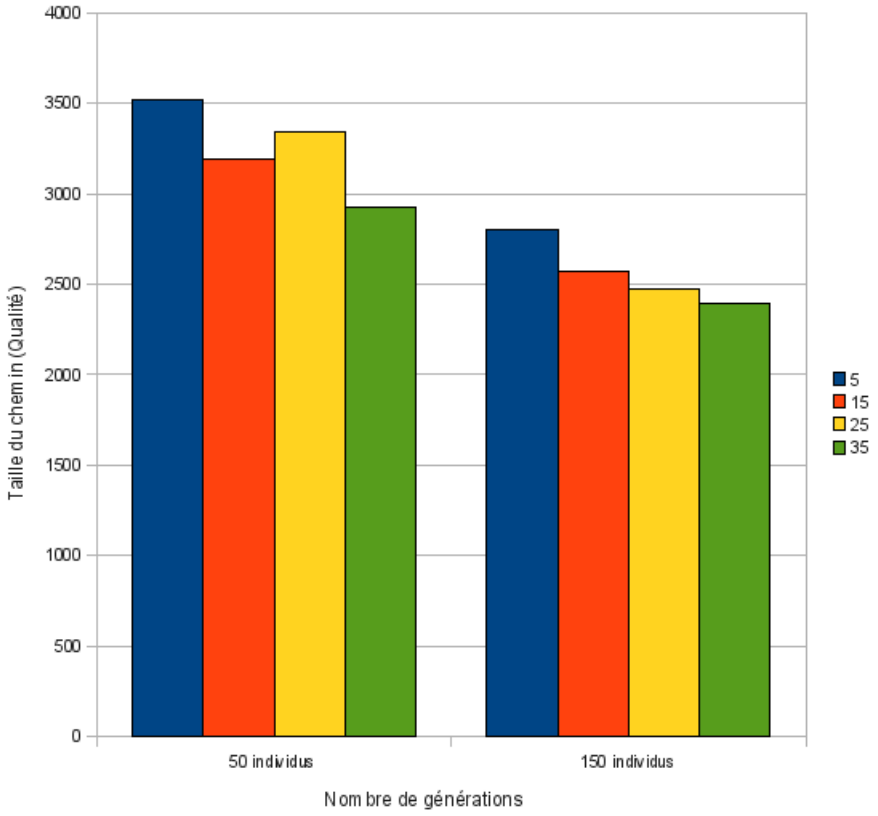


FIG. 4.5 – Diagramme Variations de la qualité selon le nombre de générations pour deux populations de 50 et 150 individus, les barres les plus courtes sont les meilleurs.

la population. Le second test portait sur l'influence de la taille de la population sur la qualité de solution résultant de l'algorithme. Finalement le dernier test portait sur l'influence du nombre de générations sur la qualité du chemin.

Du premier test nous concluons que la complexité de l'algorithme est linéaire. Nous avons remarqué un phénomène intéressant concernant le rapport temps d'exécution d'une ère et nombre d'ère, il existe un équilibre optimale où le temps d'exécution d'une ère multiplié aux nombres d'ères de chemin résulte sur une économie de temps d'exécution. Il serait intéressant de contrôler ce paramètre à l'avenir dans l'optique d'une configuration d'exécution optimale.

Du deuxième test nous concluons que contrairement à l'intuition l'amélioration de la qualité du chemin avec l'augmentation de la taille de population n'est pas linéaire. Bien que croissante elle semble convergente vers une borne supérieure que nos moyens techniques n'ont pu atteindre, nous savons cependant qu'il n'est pas bénéfique de dépasser le seuil de 500 individus.

Le dernier test, nous conduit à la même conclusion que le test précédant c-à-d à

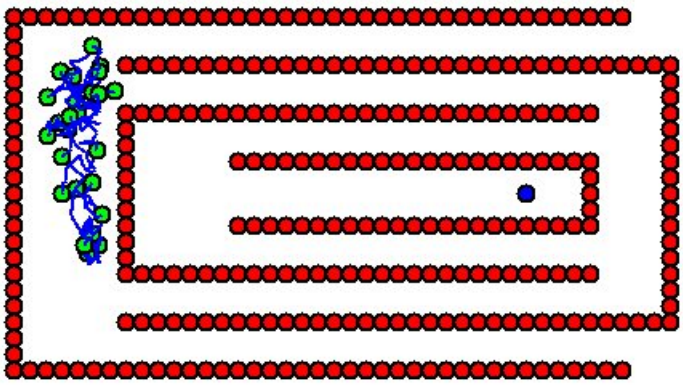


FIG. 4.6 – L'incapacité de l'algorithme face au labyrinthe.

la convergence de la qualité de chemin vers une borne supérieure malgré l'accroissement du nombre de génération. Nous concluons que le résultat ne s'améliore pas vraiment en dépassant 25 générations par ère.

Des tests effectués nous proposons 25 générations par ère pour une population de 120 individus pour bénéficier de l'équilibre optimale issu de premier test tout en restant inférieur aux bornes proposées par les tests suivants (500 individus et 25 générations).

L'algorithme a été incapable de résoudre un labyrinthe dans un temps convenable, il peut le résoudre, grâce à l'aspect aléatoire de l'algorithme en un temps extrêmement long.

Chapitre 5

Proposition d'échappement d'un minima local dans un champs de potentiel normalisé

Nous avons longuement introduit, dans le troisième chapitre, l'utilisation des champs de potentiels électriques dans la planification de chemins. Nous avons aussi remarqué la grande différence entre les formules qu'utilisent les auteurs et les formulations réelles régissant le phénomène électrostatique.

Dans ce chapitre, nous voulons, dans un premier temps, proposer une formulation pour les champs de potentiels qui soit la plus fidèle possible à la réalité. Dans un deuxième temps, nous détaillerons la méthode de normalisation utilisée pour obtenir des chemins réellement exploitables par des robots mobiles.

Finalement, nous proposerons une méthode pour palier à l'inconvénient majeur des champs de potentiels c-à-d les minimas locaux. Cette méthode s'inscrit dans la colonne des méthodes d'échappement utilisant des cibles virtuelles.

5.1 Champs de potentiels normalisés

Notre approche consiste à considérer les formulations réelles des champs de potentiels. Une utilisation directe de ces formules dans les expériences conduit à des résultats qui ne sont pas envisageables. En effet, le robot se trouvant trop loin du but bouge à des vitesses quasiment nulles. Lorsque le robot se trouve proche du but, la force exercée sur celui-ci génère des accélérations irréelles qui conduisent, le plus souvent, le robot a entré en collision avec les obstacles, n'ayant pas la force de le repousser, et à largement dépasser le but, qui n'a plus la force de l'attirer.

5.1.1 Détail du problème des équations réelles

Dans la sous-section 3.2.1 du troisième chapitre, il est question de généralités concernant les champs électrostatiques. L'équation 3.7 décrit la force qu'exercent l'une sur l'autre deux charges q_1 et q_2. Cette équation est facteur de la distance entre les deux charges puissance -3. Cela veut dire que la formule est équivalente au monôme de l'équation suivante :

$$\vec{F_{12}} = \frac{q_1 q_2}{4\pi\epsilon_0 ||\vec{r_{12}}||^3}\vec{r_{12}} \Rightarrow \frac{K}{r^3} \tag{5.1}$$

Pour les besoins de la simulation, et pour déduire un mouvement à partir de la force électrostatique, nous utilisons le deuxième principe de Newton pour obtenir l'accélération générée sur le robot dont la masse est considérée être de $1Kg$:

$$\Sigma_i F_i = \gamma m_{robot} \Rightarrow \gamma = \frac{\Sigma_i F_i}{m_{robot}} \Rightarrow \gamma = \frac{K'}{r'^3} \tag{5.2}$$

La vitesse étant un dérivatif de l'accélération par rapport au temps, et en utilisant la méthode d'Euler pour implémenter les équations différentielles, nous pouvons estimer la variation de la vitesse :

$$v_{t+1} = v_t + \gamma \Rightarrow v_{t+1} - v_t = \gamma \Rightarrow \Delta v = \frac{K'}{r'^3} \tag{5.3}$$

De la même manière, la position étant un dérivatif de la vitesse par rapport au temps, et toujours en utilisant la méthode d'Euler pour résoudre (approximativement) les équations différentielles, nous pouvons estimer la variation de la position :

$$p_{t+1} = p_t + v_t \Rightarrow p_{t+1} - p_t = v_t \Rightarrow \Delta p = \frac{K'}{r'^3} \tag{5.4}$$

Ces séries d'implications nous démontre que la variation de la position est lui aussi facteur de la distance à la puissance -3. Le potentiel électrique du but étant très supérieur à celui des obstacles, nous pouvons considérer la distance comme étant celle séparant le robot du but en négligeant la distance des obstacles dont les charges sont petites et la présence occasionnelle.

Le graphe 5.1 suivant nous montre la variation de la variation de la position par rapport à la distance entre le robot et le but. L'unité de distance utilisée étant le pixel et K' ayant une valeur de 10000.

Il est clair sur le graphe que la variation de la position est quasiment nulle pour une distance robot-but supérieur à 10 pixels. C-a-d que le robot bougera de manière insignifiante s'il se trouve à une distance supérieur à 10 pixels (c-a-d s'il n'est pas tout prés du but). D'un autre coté, la variation de la position prend une allure exponentielle lorsqu'elle se rapproche à moins de 10 pixels du but. Autrement dit, le robot subit des accélérations astronomique lorsqu'il se rapproche trop.

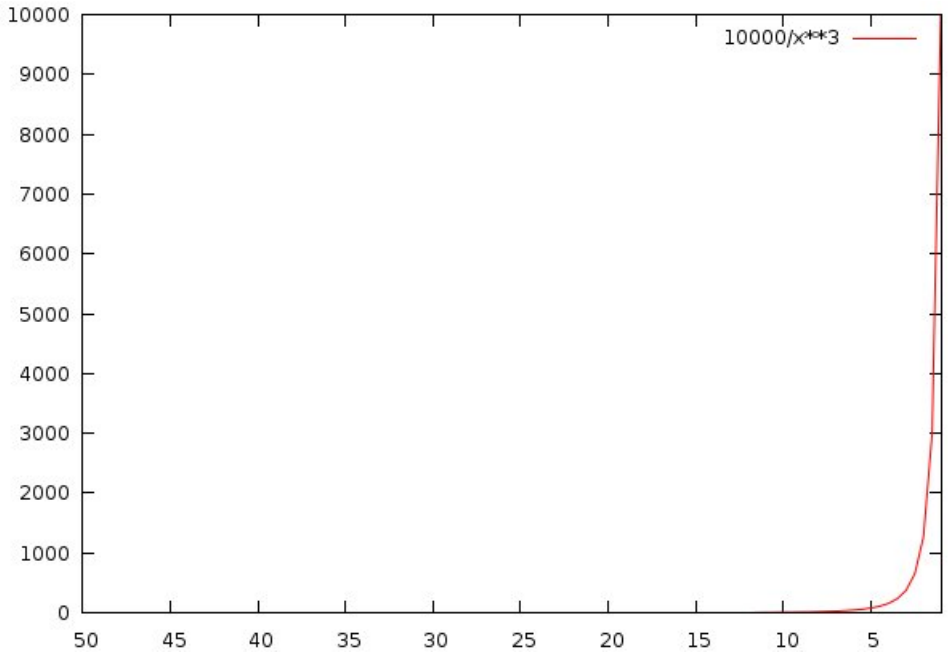


FIG. 5.1 – Variation de la variation de la position par rapport à la distance.

De telles propriétés sont analogues (en terme d'exponentialité) à celles des trous noirs en astrophysique. L'on assiste à une singularité. après le cap des 10 pixels, le robot subit une accélération si forte que le but est incapable de le stopper. Il traverse littéralement le but et se met en orbite extrêmement excentrique autour de lui. De telles états seraient catastrophiques pour la navigation du robot.

5.1.2 Formulation des champs de potentiels normalisés

Pour palier à ces inconvénients, la solution que nous proposons est d'annuler les effets de l'accélération en gardant une vitesse constante de valeur 1. Le calcul de la nouvelle position passe par trois étapes. La première étape consiste à utiliser les formules classiques de l'électrostatique pour calculer la force exercée sur le robot par tous les obstacles et le but par la formule qui suit qui est une généralisation par la sommation de l'équation 3.7 :

$$\vec{F_T} = q_1 \vec{E_T} = q_1 \sum_{i=1}^{n} \frac{q_i}{4\pi\varepsilon_0} \frac{P_i\vec{M}}{|P_i\vec{M}|^3} \tag{5.5}$$

Dans cette équation nous considérons que q_1 est la charge du robot et que M est sa position, q_i est la charge des entités influentes sur le robot et P_i leurs positions respectives. La seconde étape consiste à normaliser la force $\vec{F_T}$ obtenue par l'équation précédente. En normalisant la force, on annule la singularité observée dans l'utilisation des formules réelles dues à la distance du robot par rapport aux obstacles et

au but.

$$\vec{F_n} = \frac{\vec{F_T}}{|\vec{F_T}|} \tag{5.6}$$

Le vecteur $\vec{F_n}$ obtenu ne représente plus dans ce cas une force mais plutôt la direction du mouvement à entreprendre. Puisque $\vec{F_n}$ est de module 1, nous le sommons directement au vecteur de position $\vec{OM}$ (O est considéré comme origine) :

$$\vec{OM}_n = \vec{OM}_{n-1} + \vec{F_n} \tag{5.7}$$

Cette équation sous-entend que le mouvement se fait indépendamment de la distance, selon une accélération nulle et une vitesse constante de valeur 1. Ces deux propriétés sont une solution aux problèmes observés lors de l'application des formules brutes. Le robot adopte un mouvement fluide, uniforme et réellement applicable.

5.2 Les corps virtuels

Lorsque le robot se retrouve dans un minimum local, cela sous entend que la sommation de toutes les forces exercées sur le robot est nulle. La méthode des entités virtuels vise à considérer un but (ou un obstacle) qui ne représente rien dans le réel perçu mais qui malgré tout possède une charge. Placée à un endroit stratégique, cette entité virtuelle perturbe la force exercée sur le robot. La sommation n'étant plus nulle, le robot se met en mouvement et sort du minima local.

5.2.1 Les attracteurs virtuels

Le premier type d'entité virtuelle possible est l'attracteur virtuel. C'est une charge ponctuelle de signe inverse à celle du robot. Donc, elle attire le robot comme le ferait le but. Cette charge ne représente rien dans le réel perçu, elle est générée par le robot dans le but de déstabiliser le vecteur nul issu de la sommation des force dans un minimum local.

5.2.1.1 Problème de positionnement de l'attracteur

Un des problèmes rencontrès en utilisant un attracteur est celui de son positionnement. Pour qu'il soit efficace, et qu'il puisse extraire le robot de sa condition de blocage dans le minima, l'attracteur doit être positionné au-delà de l'horizon du robot, c-à-d à une distance supérieure à celle de sa portée.

Dans notre travail, nous utilisons un positionnement vectoriel virtuel qui dépend directement du vecteur entre le robot et le but $\vec{rb}$. Soit $\vec{P}$ un vecteur tel que :

$$\vec{P} = \frac{40}{|\vec{rb}|}\vec{rb} \tag{5.8}$$

Si P_x et P_y désignent respectivement la première et la seconde composante du vecteur $\vec{P}$. Si $\vec{OM}$ désigne la position du robot par rapport à l'origine et OM_x, OM_y ses composantes. Alors nous avons le vecteur $\vec{Q}$ qui désigne la position de l'attracteur par rapport à l'origine telle que :

$$\vec{Q} = \begin{pmatrix} OM_x - P_x - P_y \\ OM_y + P_x + P_y \end{pmatrix} \tag{5.9}$$

L'inconvénient majeur de ce positionnement est qu'il se produit dans un espace inconnu, non couvert par les capteurs du robot. Il existe une probabilité non nulle que l'attracteur se superpose sur un obstacle réel.

Puisque l'attracteur est muni d'une charge supérieure à celle de n'importe quel obstacle, la sommation directe de la force d'attraction de l'entité virtuelle avec la force de répulsion de l'obstacle réel conduit à une force attractive vers l'entité virtuelle donc vers l'obstacle réel. La collision est alors inévitable. La solution proposée dans ce mémoire est de considérer un attracteur munî d'une charge variable, facteur de la distance.

5.2.1.2 La fonction de charge de l'attracteur

Pour résoudre le problème de collision dû à l'attracteur, nous proposons de munir cet attracteur d'une charge variable, fonction de la distance. Puisque la charge est virtuelle et entièrement inventée par le robot, ce dernier est capable de déterminer sa distance (virtuelle) et sa direction (odométrie ou par rapport aux obstacles). Si la charge de l'attracteur est notée q_{att} et est généralement à -150, la distance d, nous proposons une fonction de charge de forme sigmoide qui tend vers le zéro lorsque la distance diminue :

$$q_{att} = \frac{150}{1 + e^{-20+d}} - 150 \tag{5.10}$$

Cette fonction tend vers 0 lorsque la distance est inférieure à 15. Elle augmente soudainement dans l'intervalle $[15, 25]$ pour se stabiliser finalement à -150 pour toute valeur supérieure à 25. Dans la figure 5.2, le tracé rouge représente la charge de l'attracteur. La charge de l'obstacle est constante et égale à 10. La somme des deux charges est présentée par le tracé bleu. Il est clair que lorsque le robot s'approche de l'obstacle c'est la charge de l'obstacle qui l'emporte (celle de l'attracteur étant nulle). Pour de grandes distances la charge de l'attracteur devient largement influente par rapport à l'obstacle.

5.2.2 Les répulseurs virtuels

Le second type d'entité virtuelle possible est le répulseur virtuel. C'est une charge ponctuelle de même signe que celle du robot. Donc, elle répulse le robot comme le ferait un obstacle. Cette charge ne représente rien dans le réel perçu, elle est générée par le robot dans le but de déstabiliser le vecteur nul issu de la sommation des forces

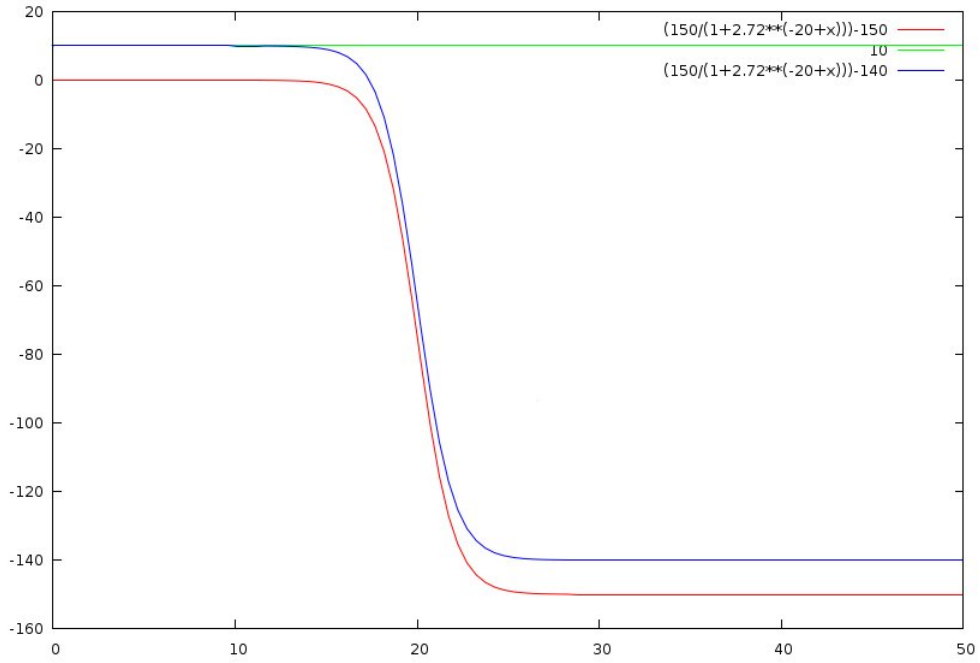


FIG. 5.2 – Variation de la charge de l'attracteur (rouge), de la charge de l'obstacle (vert) et de la somme des deux (bleu) par rapport à la distance entre le robot et l'attracteur (obstacle)

dans un minimum local.

5.2.2.1 Positionnement du répulseur

Le positionnement du répulseur ne demande pas autant de précautions que celui de l'attracteur. La raison est que quelque soit sa position (au-delà de l'horizon du robot), le répulseur ne conduira jamais à une collision avec un obstacle. Pour son positionnement, nous avons opté pour que le robot soit entre le répulseur et le but, de manière à ce que le répulseur propulse le robot en direction du but (si c'est possible). Soit $\vec{P}$ tel que considéré dans la formule 5.8. Si P_x et P_y désignent respectivement la première et la seconde composante du vecteur $\vec{P}$. Si $\vec{OM}$ désigne la position du robot par rapport à l'origine et OM_x, OM_y ses composantes. Alors nous avons le vecteur $\vec{Q}$ qui désigne la position du répulseur par rapport à l'origine telle que :

$$\vec{Q} = \begin{pmatrix} OM_x + P_x \\ OM_y - P_y \end{pmatrix} \tag{5.11}$$

Le vecteur $\vec{Q}$ positionne le répulseur quasiment sur la même ligne que le robot et le but. Le robot se trouve entre le répulseur et le but. Le fait que le répulseur se trouve derrière le robot (par rapport au but) permet un effet réacteur vers l'avant très utile.

5.3 L'algorithme de résolution de trajectoire

L'algorithme permettant de résoudre une trajectoire pour le robot est résumé dans le tableau 5.1. Initialement, l'algorithme alloue des positions pour le robot et le but. Il alloue aussi des positions pour les obstacles en rapport avec l'environnement considéré.

La seconde ligne de l'algorithme est la boucle "Tant que" principale qui génère la simulation. Elle contient en condition le critère d'arrêt général du processus. Dans ce cas, l'algorithme tournera tant que la distance entre le robot et le but n'est pas inférieure à un seuil. Nous avons préférer une inégalité à une égalité vu qu'il n'est pas toujours certain qu'une égalité stricte des coordonnées du robot et du but puisse être atteinte, la présence d'un obstacle à proximité ou la trop grande précision des variables du langage utilisé peuvent conduire à une boucle infinie.

Algorithme : Résolution de trajectoire

Générer les positions initiales des obstacles, du robot et du but.
Tant que distance(robot,but)>seuil faire
Début
  Lister les obstacles se trouvant à portée de capteurs
  Bruiter la position exactes de ces obstacles (environnement incertain)
  Calculer la force exercée sur le robot (équations 5.5 et 5.6)
  Si (cette force est nulle (minima local)) alors
  Début
  $dp \leftarrow$ dernière position du robot avant le minima local
  Positionner un corps virtuel (attracteur ou repulseur)
  Tant que distance(robot,dp)<seuil_minima faire // *Tant que dans les parages du ML*
  Début
  Calculer la charge du corps virtuel (s'il y a lieu de le faire)
  Calculer la force d'influence du corps
  Calculer la force exercée sur le robot
  Mettre à jour la position du robot (équation 5.7)
  Fin
  Fin
  Sinon
  Début
  Mettre à jour la position du robot (équation 5.7)
  Fin
Fin

TAB. 5.1 – Algorithme permettant la résolution de la trajectoire d'un robot dans un champs de potentiels normalisé en utilisant la méthode des corps virtuels pour échapper d'un minima local

A l'entrée de la boucle, nous listons les obstacles qui se trouvent à portée des

capteurs du robot. L'environnement étant une liste des obstacles qui le constitue. Nous insérons un bruit gaussien sur les valeurs des coordonnés de ces obstacles, ce bruit est sensé représenter l'imprécision des capteurs (que nous considérons être des sonars).

Ensuite, nous calculons grâce aux coordonnées bruitées des obstacles et du but (donnée nécessaire) la force exercée sur le robot. Nous évaluons cette force pour déduire si le robot est dans un minima local. En effet, dans un minima local le robot se retrouve bloqué car se trouvant dans un état d'équilibre dynamique dû à l'annulation de la force exercée sur lui.

Si la force exercée est nulle (en réalité ϵ), le robot est dans un minima local. Nous positionnons virtuellement un corps (attracteur ou répulseur) puis l'algorithme entre dans une seconde boucle, c'est la boucle qui permet l'extraction du robot du minima local. La condition d'arrêt de cette boucle est un éloignement suffisant depuis le minima local que nous exprimons sous forme d'une distance entre la position actuelle du robot et celle juste au moment ou il se bloque.

Pour extraire le robot, un calcul de l'action du corps virtuel et des obstacle sur le robot est effectué. Il faut remarque que la charge du corps peut être variable dans le cas d'un attracteur à charge variable, ce qui conduit à un recalcule constant de cette charge. La position du robot est mise à jour jusqu'à ce qu'il s'éloigne assez du minima, dans ce cas un traitement normal peut reprendre.

Le traitement normal désigne le calcul de l'action du but réel et des obstacle sur le robot en accord avec les équations vues dans les sections précédentes.

5.4 Test de l'approche par champs de potentiels normalisé

Dans cette section nous allons effectué des tests sur, premièrement, l'efficacité des champs de potentiels normalisés à générer des trajectoires fiables pour le robot mobile autonome, puis sur l'efficacité des corps virtuels à faire sortir le robot du minima local lorsqu'il s'y trouve.

Nous avons développé pour cela un programme écrit en Python. Nous avons utilisé le même matériel et le même système d'exploitation que celui décrit dans l'expérimentation de l'approche évolutionniste.

Nous allons tester trois corps virtuels. Un répulseur muni d'une charge statique, un attracteur muni lui aussi d'une charge statique et finalement un attracteur muni d'une charge variable obéissant à la formulation 5.2.

Nous avons fait des tests dans trois environnements différents : aléatoire, en

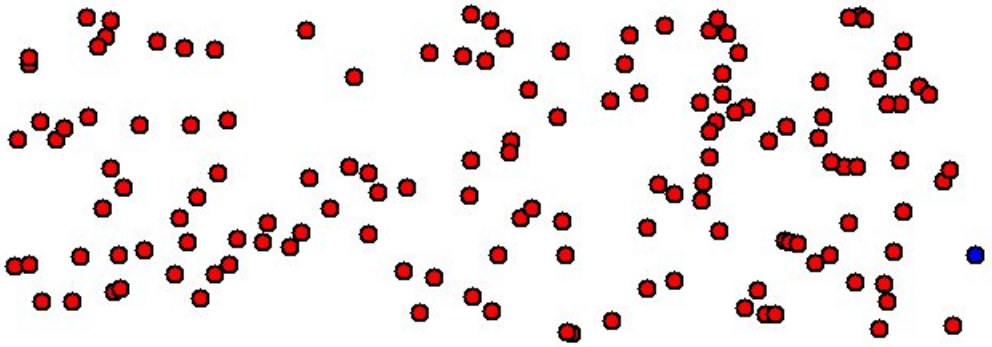


FIG. 5.3 – Un environnement aléatoire de 140 obstacles.

étau et labyrinthe. Nous avons tester le temps d'exécution T_{exe} de l'algorithme sur notre machine dans des environnements aux nombres d'obstacles différents. Nous avons aussi calculé la proportion des minimas locaux qui apparaissent, nous l'avons noté T_{ML}. Finalement, nous avons calculé le taux de succès de chaque méthode à produire un chemin complet (atteignant le but) et sans collision, nous avons noté ce taux T_{succes}.

5.4.1 Les environnements de test

Pour réellement avoir une idée de l'efficacité de l'algorithme à produire des chemins fiables pour un robot mobile autonome, il est nécessaire de considérer divers environnements. Dans notre travail nous avons considéré trois environnement très différents.

Le premier environnement, dont un exemple est visible sur la figure 5.3, est un environnement aléatoire. Les obstacles sont disséminés de manière hasardeuses dans l'espace de l'expérience. Nous allons considéré 8 versions d'environnements aléatoires, les environnement munis de : 20, 40, 60, 80, 100, 120, 140 et 160 obstacles.

Le deuxième environnement considéré est l'environnement en étau visible sur la figure 5.4. C'est l'environnement typique produisant un minima local utilisé dans la littérature. Cet environnement est un bon test pour connaitre l'aptitude de l'algorithme à extraire le robot d'un minima local "profond".

Le troisième environnement est le labyrinthe visible sur la figure 5.5. Cet environnement est le plus complexe pour un algorithme de navigation de robot. Nous allons tester notre algorithme dans ce type d'environnement pour constater son efficacité à faire face à la complexité géométrique de son tracé et à retrouver un but situer au coeur cette complexité.

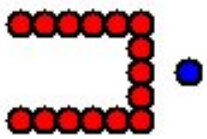


FIG. 5.4 – L'étau.

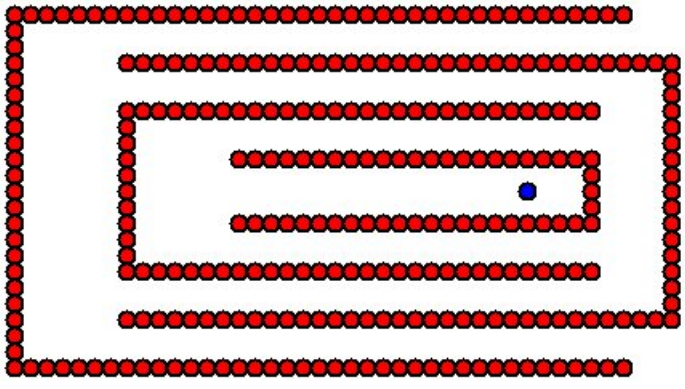


FIG. 5.5 – Le labyrinthe.

5.4.2 Le répulseur dans un environnement aléatoire

Nous commençons nos tests avec un répulseur muni d'une charge de -240 et positionné selon le vecteur $\vec{Q}$ de la formule 5.11 dans un environnement aléatoire. Tous les résulats obtenus sont une moyenne de 10 expériences.

5.4.2.1 Variations de T_{ML} par rapport à N_{obs}

Nous commençons par tester le taux de minimas locaux que rencontre le robot par rapport aux nombre d'obstacles dans l'environnement. Le tableau 5.2 contient dans sa première ligne le nombre d'obstacles de l'environnement. Dans la seconde ligne, nous trouvons le nombre de minima locaux rencontrès dans les expériences réussies. Par "réussies", nous désignons les expériences qui génère un chemin complet et sans collisions. La troisième ligne présente le taux de minimas locaux présents par rapport aux expériences réussies.

N_{obs}	20	40	60	80	100	120	140	160
N_{ML}	4	13	23	27	22	26	28	20
T_{ML}	0.4	1.4	2.5	3	2.75	3.71	4.66	5

Tab. 5.2 – Variations du taux de minimas locaux par rapport au nombre d'obstacles

Nous remarquons une augmentation du taux de minimas locaux. Ceci s'explique aisément par le fait de l'augmentation du nombre d'obstacles qui conduit à une élévation de la densité à proximité des capteurs et par conséquent à l'augmentation de la probabilité de tomber dans un minimas local.

5.4.2.2 Variations de T_{exe} par rapport à N_{obs}

Nous passons au test des performances logicielles du programme utilisé et cela en testant son temps d'exécution. Les résultats obtenus sont une moyenne de 10 expériences pour chaque environnement. Nous ne considérons que les temps des expériences réussies.

N_{obs}	20	40	60	80	100	120	140	160
T_{exe}	$1s$	$6s$	$13s$	$24s$	$39s$	$57s$	$90s$	$90s$

Tab. 5.3 – Variations du temps d'exécution par rapport au nombre d'obstacles

L'augmentation du temps de calcul s'explique par deux facteurs ayant une même cause. La cause étant l'augmentation de la densité des obstacles à portée des capteurs. Cette augmentation conduit à une complexité accrue dans le calcul de la force (la sous-liste d'obstacles est plus grande) comme premier facteur, et à l'augmentation de minimas locaux qui représente une perte de temps considérable lors de l'extraction, comme deuxième facteur.

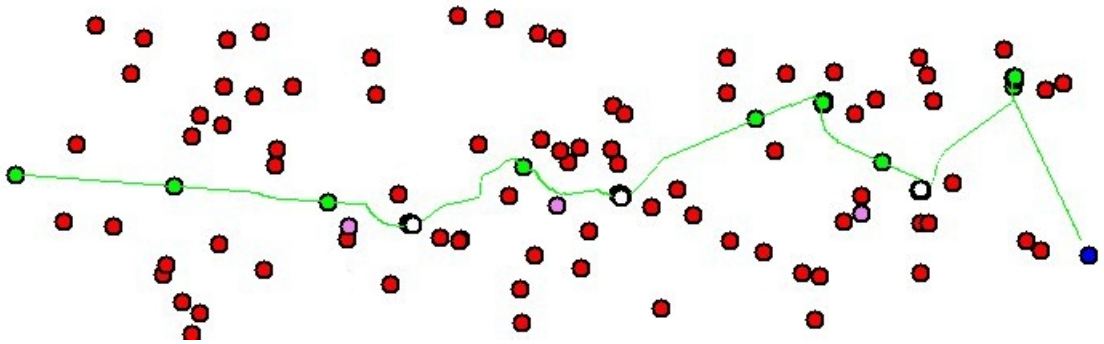


FIG. 5.6 – Exemple d'une expérience réussie avec un répulseur et un environnement à 80 obstacles.

5.4.2.3 Variations de T_{succes} par rapport à N_{obs}

Finalement, nous testons le taux de succès de l'algorithme quant à la résolution d'un chemin complet (atteignant le but) et sans collision. Ces chiffres sont obtenus après 10 expériences pour chaque environnement.

N_{obs}	20	40	60	80	100	120	140	160
T_{succes}	100%	100%	90%	90%	80%	70%	60%	50%

TAB. 5.4 – Variations du taux de succès par rapport au nombre d'obstacles

Le taux de succès diminue progressivement avec l'augmentation du nombre d'obstacles. Cela s'explique par le fait que la trop grande concentration des obstacles peut conduire à des configuration où le répulseur est incapable d'extraire le robot de sa condition.

5.4.3 Le répulseur dans un environnement en étau

Le répulseur a été incapable de sortir le robot de l'étau. L'emplacement du répulseur en est la cause. Puisque celui-ci était quasiment sur la même ligne que le robot et le but, son action était de pousser le robot vers le fond de l'étau, ce qui est complètement inefficace.

Pour pouvoir sortir le robot tel que visible sur la figure 5.7, nous avons dû changer le vecteur $\vec{Q}$. Plutôt que d'être derrière le robot, nous lui avons fait changer de position de manière à ce qu'il soit entre le robot et le but. Pour cela, nous l'avons positionner aux coordonnées où habituellement sont placés les attracteurs. Nous avons donc utilisé la formule 5.9.

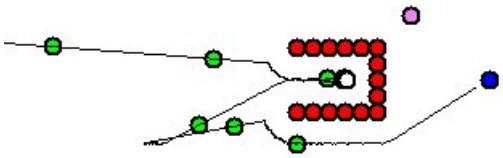


FIG. 5.7 – Le répulseur (en violet) extrait le robot (en vert) du minima local (en blanc). En rouge, les obstacles. En bleu, le but. En noir, le trajet du robot.

5.4.4 Le répulseur dans un labyrinthe

L'algorithme a été incapable de trouver un chemin entre l'emplacement initial et le but. Quelque soit la position du répulseur, cette position ne peut être optimale tout le long de la simulation. Quelque soit la durée du test, le robot se retrouvera tôt ou tard dans un minima local profond duquel il ne pourra être extré.

5.4.5 L'attracteur dans un environnement aléatoire

Nous continuons nos tests avec deux types d'attracteurs. Le premier est muni d'une charge statique de +190, le second possède une charge variable obéissant à la fonction de la formule 5.2. Ils sont tous les deux positionnés selon le vecteur $\vec{Q}$ de la formule 5.9 dans un environnement aléatoire. Tous les résulats obtenus sont une moyenne de 10 expériences.

5.4.5.1 Variations de T_{ML} par rapport à N_{obs}

Nous commençons par tester le taux de minimas locaux que rencontre le robot par rapport au nombre d'obstacles dans l'environnement. Le tableau 5.5 contient dans sa première ligne le nombre d'obstacles de l'environnement. Dans la seconde ligne, nous trouvons le taux de minima locaux rencontrès dans les expériences réussies de l'attracteur à charge statique. Dans la dernière ligne, nous trouvons le taux de minima locaux rencontrès dans les expériences réussies de l'attracteur à charge variable.

N_{obs}	20	40	60	80	100	120	140	160
$T_{ML}(charge\ statique)$	0.1	0.8	2	3.3	3	4	–	–
$T_{ML}(charge\ variable)$	0.4	1.1	1.88	3.5	5	6	7	–

TAB. 5.5 – Variations du taux de minimas locaux par rapport au nombre d'obstacles dans le cas d'un attracteur à charge statique et d'un attracteur à charge variable

Attracteur à charge statique Le taux de minimas locaux augmente proportionnellement au nombre d'obstacles, cela s'explique comme précédemment par la

densité croissante des obstacles qui augmente la probabilité de tomber dans un minima. L'algorithme n'a pu résoudre de chemins pour plus de 120 obstacles, de ce fait, le taux est infini (but inatteignable).

Attracteur à charge variable Le taux de minimas locaux augmente proportionnellement au nombre d'obstacles, l'explication a été donnée précédemment. L'algorithme n'a pu résoudre de chemins pour plus de 140 obstacles, de ce fait, le taux est infini (but inatteignable).

5.4.5.2 Variations de T_{exe} par rapport à N_{obs}

Nous passons au test des performances logicielles du programme utilisé et cela en testant son temps d'exécution. Les résultats obtenus sont une moyenne de 10 expériences pour chaque environnement. Nous ne considérons que les temps des expériences réussies.Le tableau 5.6 contient dans sa première ligne le nombre d'obstacles de l'environnement. Dans la seconde ligne, nous trouvons le temps d'exécution dans les expériences réussies de l'attracteur à charge statique. Dans la dernière ligne, nous trouvons le temps d'exécution dans les expériences réussies de l'attracteur à charge variable.

N_{obs}	20	40	60	80	100	120	140	160
$T_{exe}(chargestatique)$	$1s$	$6s$	$19s$	$42s$	$58s$	$74s$	–	–
$T_{exe}(chargevariable)$	$1s$	$6s$	$11s$	$23s$	$32s$	$96s$	$110s$	–

Tab. 5.6 – Variations du temps d'exécution par rapport au nombre d'obstacles dans le cas d'un attracteur à charge statique et d'un attracteur à charge variable

Globalement, le temps d'exécution augmente avec l'augmentation du nombre d'obstacle. L'explication est la même que celle donnée pour le répulseur. Dans le cas de l'attracteur à charge statique, le temps d'exécution pour plus de 120 obstacles est infini, de même pour l'attracteur à charge variable au-delà de 140 obstacles.

5.4.5.3 Variations de T_{succes} par rapport à N_{obs}

Finalement, nous testons le taux de succès de l'algorithme quant à la résolution d'un chemin complet (atteignant le but) et sans collision. Ces chiffres sont obtenus après 10 expériences pour chaque environnement. Le tableau 5.7 contient dans sa première ligne le nombre d'obstacles de l'environnement. Dans la seconde ligne, nous trouvons le taux de succès dans les expériences réussies de l'attracteur à charge statique. Dans la dernière ligne, nous trouvons le taux de succès dans les expériences réussies de l'attracteur à charge variable.

Attracteur à charge statique Le taux de succès de cet attracteur diminue de manière rapide jusqu'à ce qu'il n'y a plus de test réussi au-delà de 120 obstacles.

N_{obs}	20	40	60	80	100	120	140	160
$T_{succes}(chargestatique)$	100%	90%	80%	40%	30%	10%	0%	0%
$T_{succes}(chargevariable)$	100%	100%	90%	80%	60%	30%	20%	0%

TAB. 5.7 – Variations du taux de succès par rapport au nombre d'obstacles dans le cas d'un attracteur à charge statique et d'un attracteur à charge variable

Attracteur à charge variable Le taux de succès de cet attracteur diminue de manière modérée jusqu'à ce qu'il n'y a plus de test réussi au-delà de 140 obstacles.

5.4.6 L'attracteur dans un environnement en étau

L'algorithme ont été incapables de sortir le robot de l'étau. L'emplacement des attracteurs en est la cause. Puisque ceux ci était entre le robot et le but, leurs action était d'attirer le robot vers le fond de l'étau, ce qui est complètement inefficace.

FIG. 5.8 – L'extraction du robot (en vert) par un attracteur (en violet). A gauche, un attracteur à charge simple, A droite, un attracteur à charge variable.

Pour pouvoir sortir le robot tel que visible sur la figure 5.8, nous avons dû changer le vecteur $\vec{Q}$. Plutôt que d'être devant le robot, nous leurs avons fait changer de position de manière à ce qu'ils soit derrière le robot et un peu vers le haut (voir la figure 5.8). Pour cela, nous avons légèrement modifier le vecteur $\vec{Q}$ habituellement utilisé pour les répulseur. Soit $\vec{P}$ tel que considérer dans la formule 5.8, le vecteur $\vec{Q}$ devient :

$$\vec{Q} = \begin{pmatrix} OM_x + P_x + \alpha \\ OM_y - P_y + \beta \end{pmatrix} \tag{5.12}$$

α et β étant deux constantes telles que $\alpha < 0$ et $\beta > 0$. Dans nos expérimentations, nous avons pris $\alpha = -10$ et $\beta = 40$.

5.4.7 L'attracteur dans un labyrinthe

L'algorithme a été incapable de trouver un chemin entre l'emplacement initial et le but. Quelque soit le type de l'attracteur (charge statique ou variable) et quelque soit sa position, cette position et cette charge ne peuvent être optimales tout le long

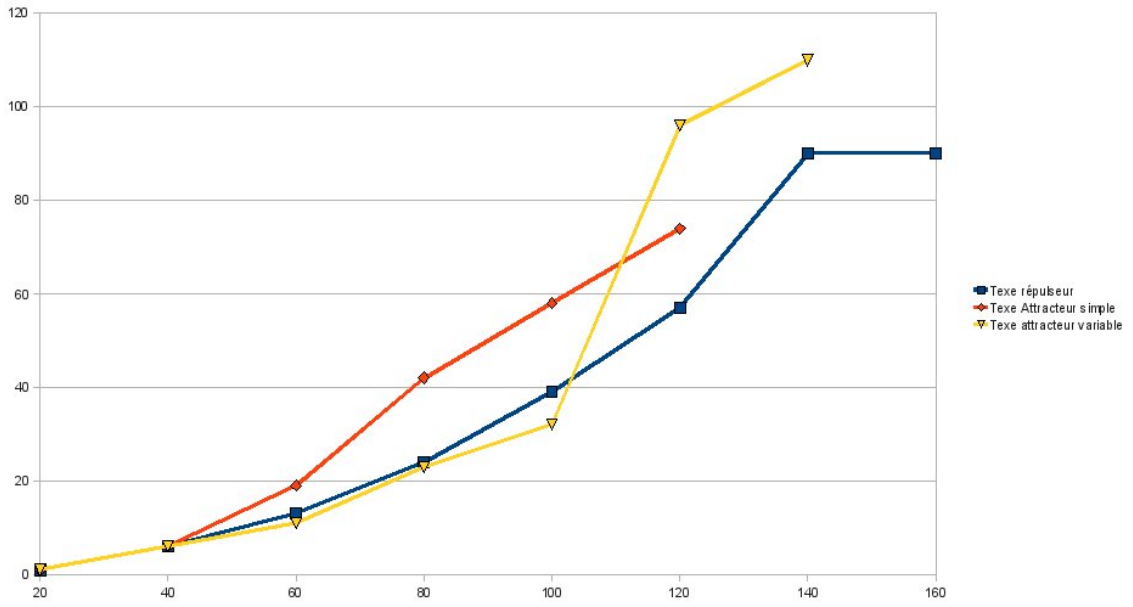


FIG. 5.9 – Graphe représentant la variation du temps d'exécutions pour les différentes technique d'échappement : le répulseur en bleu, l'attracteur simple en rouge et l'attracteur variable en jaune. En abscisse, le nombre d'obstacles. En ordonnée, le temps d'exécution.

de la simulation. Quelque soit la durée du test, le robot se retrouvera tôt ou tard dans un minima local profond duquel il ne pourra être extrait.

5.5 Attracteurs ou répulseurs ?

La question qui se pose est de savoir lequel des ces corps virtuels est le meilleur pour extraire le robot d'un minima local. En ce qui concerne le labyrinthe, les trois méthodes ont échoué, donc, ce test ne pas être pris en compte pour décider de la supériorité d'une des méthodes. De même pour le test de l'étau, les trois méthodes n'ont été concluantes qu'après une modification du vecteur d'emplacement, le calcul a été instantané et la réussite de 100%. Donc ce test ne peut être utile.

Le test de l'environnement aléatoire a, quant à lui, produit des résultats différents pour chacun des trois corps virtuels utilisés. Nous allons discuté des taux de réussite et du temps d'exécution de chacune de ces méthodes pour pouvoir tirer une conclusion.

Le graphe 5.9 représente la variation du temps d'exécution pour les trois techniques utilisées. Il est clair que le tracé du répulseur est le meilleur des trois car l'algorithme arrive à trouver un chemin pour tout les environnement dans des délais moindres. En seconde position, l'attracteur variables n'échoue qu'après 140 obsta-

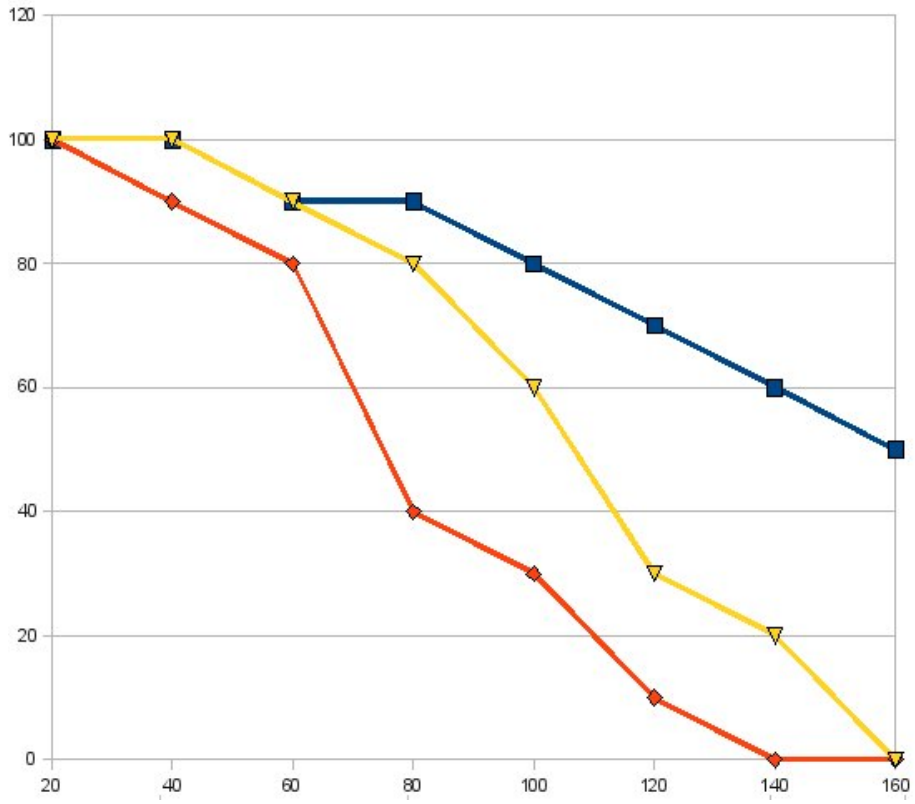


FIG. 5.10 – Graphe représentant la variation du taux de succès pour les différentes technique d'échappement : le répulseur en bleu, l'attracteur simple en rouge et l'attracteur variable en jaune. En abscisse, le nombre d'obstacles. En ordonnée, le taux de succès.

cles. Il présente de bons résultats au début bien qu'il se retrouve en difficulté dans les environs de 120-140 obstacles. En dernier, l'attracteur simple signe les plus mauvais temps. Il se retrouve en dernier dans toutes les expériences et s'avère incapable d'aller au-delà de 120 obstacles.

Le graphe 5.10 représente la variation du taux de réussite pour les trois techniques utilisées. Il s'en dégage que le répulseur s'en sort haut la main en signant les meilleurs scores. Par titre de comparaison, il présente un succès de 60% pour des environnements de 140 obstacles alors que l'attracteur variable arrive à peine à 20% et que l'attracteur simple échoue tout le temps. Il est le seul à s'en sortir avec 50% de réussite pour des environnement de 160 obstacles. Sa régression est linéaire et trouve ses limites, apparemment, vers les 240 obstacles. La régression de l'attracteur simple est la plus brutale, celle de l'attracteur variable est plus douce.

Nous pouvons en conclure que le répulseur est le corps virtuel qui donne de meilleurs résultats dans les environnements aléatoire, suivi de l'attracteur variable puis de l'attracteur simple.

5.6 Conclusion

Dans ce chapitre nous avons proposé une méthode pour contourner les inconvénients des formulations réelles du champs électrostatique pour la planification de chemin. Pour cela, nous proposons une étape de normalisation du vecteur de la force électrostatique et de l'utiliser comme un vecteur de direction. Ceci dans le but d'annuler l'accélération et d'obtenir des vitesses stables.

Nous avons aussi proposé et comparé trois techniques issues des techniques des corps virtuels pour l'échappement des minimas locaux. Nous avons proposé un attracteur à charge constante, un attracteur à charge variable et un répulseur. Nous les avons testé dans des environnements aléatoires, en étau et dans un labyrinthe.

Les trois méthodes se sont révélées inefficaces pour le labyrinthe. Elles réussissent dans un environnement en étau à condition de changer le vecteur de positionnement. Finalement, dans l'environnement aléatoire, nous avons conclu que le répulseur est le corps virtuel le plus adapté à ce type de dispersions d'obstacles.

Conclusion Générale

Le long de ce travail, nous avons présenté un état de l'art global du domaine de la robotique mobile autonome. Nous avons pour cela exposé un état de l'art concernant les robots eux-mêmes à travers leurs types, leurs capteurs et actionneurs, mode de locomotion et environnement dans lequel ils évoluent. Nous avons constater que les organes de perception de ces robots transmettent généralement des informations bruitées et à faible distance conduisant à des cartographies infidèles et incomplètes par rapport à la réalité.

Nous avons aussi établi un état de l'art de la localisation et de la navigation des robots dans ces environnements. Nous avons présenté des méthodes aussi variées que les algorithmes génétiques, les réseaux de neurones, les champs de potentiel, la logique floue et les roadmaps pour la navigations. Les filtres de Kalman et particulaires ainsi que les cartes de Kohonen pour la localisation.

Nous avons proposé deux méthodes de navigation. La première méthode se base sur les algorithmes génétiques. Nous avons émis l'hypothèse que la position du but est tout le temps connue et que le robot est muni de capteurs à faible portée. Les résultats ont démontré la capacité de notre algorithme à trouver un chemin sans collisions vers le but dans des environnements moyennement complexes. Cependant, notre algorithme a été incapable de résoudre un labyrinthe ou des environnements nécessitant des retours arrières. Une critique a émettre concernant l'algorithme génétique est le temps de calcul. En effet, sur notre machine, le robot prend un temps important pour décider d'un mouvement (un mini-chemin), cela entrave grandement la recherche de chemins dans un environnement dynamique qui nécessite des décisions en temps réel. Mais puisque le problème est purement matériel et en respect avec la loi de Moore, nous sommes optimiste quant à l'apparition dans le futur de machines capables de traiter le mini-chemin instantanément.

La seconde proposition se base sur les champs de potentiel. Tout comme les algorithmes génétiques, nous émettons l'hypothèse que la position du but et connue et que les capteurs du robot ont une faible portée (cartographie locale). La nouveauté de notre approche réside dans l'emploie d'une forme normalisée des équations réelles qui régissent le phénomène électrostatique. Notre travail s'est aussi basé sur la résolution du problème de minima local qui entrave cette approche. Nous avons utiliser et comparer trois types d'entités virtuelles dont nous nous sommes servi comme extracteur du robot depuis le minima local. Ces trois types d'entités sont : l'attracteur simple,

l'attracteur à charge variable et le répulseur. Les tests ont démontré la supériorité du répulseur à sortir le robot de sa condition de blocage. Nous avons aussi prouvé que l'attracteur à charge variable était plus efficace qu'un attracteur à charge statique en garantissant des chemins sans collisions. L'algorithme s'est montré efficace pour résoudre des environnements moyennement complexes, il peut aussi résoudre les environnements en étau moyennant un changement dans le vecteur de positionnement. Mais comme la première proposition, les champs de potentiels ont été incapable de résoudre un chemin dans un labyrinthe ou des environnements nécessitant des retours arrières.

Nous envisageons comme perspective d'élaborer des algorithmes basés sur des cartographies locales mais capables de résoudre des labyrinthes. Selon notre intuition, nous déduisons que si les méthodes proposées dans ce travail ont été incapable de résoudre un labyrinthe cela est principalement dû à l'absence de mémoire spatiale pour ces algorithmes. Nous envisageons fortement pour nos travaux futurs de nous pencher sur la voie des réseaux de neurones avec lesquels il est possible de construire cette mémoire spatiale si indispensable à la résolution d'environnements avec rebroussement de chemins.

Bibliographie

[Abo03] Ashraf Aboshosha. Robust mapping and path planning for indoor robots based on sensor integration of sonar and a 2d laser range finder. *IEEE 7th international Conference on Intelligent Engineering Systems*, 2003.

[Bra06] Thomas Braunl. *Embedded Robotics Mobile Robot Design and Applications with Embedded Systems*. Springer, 2006.

[Cyb89] G. Cybenko. Approximation by superpositions of sigmoidal fonction. *Mathematics of Control, Signals and Systems page 303-314*, 1989.

[Fri07] Terrence P. Fries. Evolutionary navigation of autonomous robots under varying terrain conditions. *Mobile Robots : The Evolutionary Approach*, 2007.

[GSC00] Yunfeng Wang Gregory S. Chirikjian. A new potential field method for robot path planning. *Proceedings of the 2000 IEEE international Conference on Robotics 8, Automation San Francisco page 977-982*, 2000.

[HJ04] Harald Haas Herbert Jaeger. Harnessing nonlinearity : predicting chaotic systems and saving energy in wireless telecommunication. *Science 2 April 2004 : Vol. 304. no. 5667, pages 78-80*, 2004.

[HJ08] Lee Gim Hee and Marcelo H. Ang Jr. An integrated algorithm for autonomous navigation of a mobile robot in an unknown environment. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 2008.

[Hop82] John J. Hopfield. Neural networks and physical systems with emergent collective computational properties. *Proc. Nat. Acad. Sci. (USA) 79, 2554-2558*, 1982.

[JAJ96] Troy A. Chase Mark W. White John C. Sutton Jason A. Janet, Ricardo Gutierrez. Autonomous mobile robot global self-localization using kohonen and region-feature neural networks. *Journal of Robotic Systems*, 1996.

[Jan04] Danica Janglova. Neural networks in mobile robot motion. *International Journal of Advanced Robotic Systems, Volume 1, Number 1, March 2004, pp.15-22*, 2004.

[Jor98] Berg Jorg. Mobile robot sonar sensing with pseudo-random codes. *IEEE International Conference on Robotics and Automation 1998*, 1998.

[KHS03] Theodore W. Manikas Roger L. Wainwright Heng-Ming Tai Kamran H. Sedighi, Kaveh Ashenayi. Autonomous local path-planning for a mobile robot using a genetic algorithm. *University of Tulsa*, 2003.

[Lar03] Frederique Large. *Navigation Autonome d'un Robot dans un Environnement Dynamique et Incertain*. PhD thesis, Universtité de Savoie, Juillet 2003.

[MGP01] Min Cheol Lee Min Gyu Park, Jae hyun Jeon. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. *ISIE Pusan Korea page 1530-1535*, 2001.

[MGP03] Min Cheol Lee Min Gyu Park. Artificial potential field based path planning for mobile robots using a virtual obstacle concept. *Proceedings of the 2003 IEEWASME InternationalConference on Advanced Intelligent Mechatronics (AIM 2003)*, 2003.

[Mit98] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[MM08] C.R. McInnes M.H. Mabrouk. Solving the potential field local minimum problem using internal agent states. *Robotics and Autonomous Systems 56 (2008)*, 2008.

[Mos00] Matt Moses. A minimalist approach to design of walking robots. *Unmanned systems 2000 Symposium and Exhibition*, 2000.

[NN05] Luiza de Macedo Mourelle Nadia Nedjah. *Evolvable machines : theory and practice*. Springer, 2005.

[Oua06] L. Hamerlain M. Boudjema F. Ouadah, N. Ourak. Implementation of an oriented positioning on a car-like mobile robot by fuzzy control. *IEEE Industrial Electronics, IECON 2006*, 2006.

[PRN02] Djitt Laowattana Panrasee Ritthipravat, Thavida Maneewarn and Kenji Nakayama. Obstacle avoidance using modified hopfield neural network for multiple robots. *International Conference of Artificial Intelligence 2002*, 2002.

[RH04] Sue Ellen Haupt Randy Haupt. *Practical Genetic Algorithms*. Wiley, 2004.

[SS96] Kazuo Sugihara and John Smith. A genetic algorithm for 3-d path planning of a mobile robots. *CiteSeerX*, 1996.

[VMP06] Dan Simon Vamsi Mohan Peri. Fuzzy logic control for an autonomous robot. *Fuzzy Information Processing Society, 2005. NAFIPS 2005. Annual Meeting of the North American pages 337-342*, 2006.

[Whi00] Louis L. Whitcomb. Underwater robotics : Out of the research laboratory and into the field. *Industrial Robotics IEEE 2000 International Conference on Robotics and Automation*, 2000.

[WSM43] Walter Pitts Warren S. McCulloch. A logical calculus of the ideas immanent nervous activity. *Bulletin of Mathematical Biophysics 5 :115-133*, 1943.

[YK91] J. Borenstein Y. Koren. Potential field methods and their inherent limitations for mobile robot navigation. *IEEE International Conference on Robotics and Automation 1991*, 1991.

[Zen04] Youcef Zennir. *Apprentissage par renforcement et systèmes distribués : Application à l'apprentissage de'un robot hexapode*. PhD thesis, INSA de Lyon, Juillet 2004.

[Z.H05] Z.Hendel. Collision free path planning and control of wheeled mobile robot using kohonen self-organising map. *Bulletin of the Polish Academy of Sciences Technical Sciences Vol. 53, No. 1*, 2005.

[ZXy03] Zhu Jing Zou Xi-yong. Virtual local target method for avoiding local minimum in potential field based robot navigation. *Journal of Zhejiang University SCIENCE June 2003*, 2003.

ملخص:

في هده الدراسة, نقوم بتقديم الوضع الفني لمجال الروبوتات المتحركة المستقلة. ثم, نقترح طريقتين للملاحة. الاولى ترتكز على خوارزم جينية معدلة لروبوتات مجهزة بملتقطات دو بعد محلي. الثانية ترتكز على اقتراح لحقول كمونات نضامية و استخراج من دروة صغرى محلية اعتمادا على طريقة الاجسام الافتراضية

الكلمات المفتاح: الروبوتات المتحركة المستقلة, الملاحة, خوارزم جينية, حقول كمونات, الاجسام الافتراضية

Résumé

Dans cette étude, nous présentons un état de l'art du domaine de la robotique mobile autonome. Ensuite, nous proposons deux approches de navigation. La première est basée sur les algorithmes génétiques et est adaptée aux robots munis de capteurs à portée locale. La seconde est basée sur une proposition de champs de potentiels normalisé avec une solution d'échappement depuis les minimas locaux grâce à une technique inspirée des entités virtuelles.

Mots Clés Robot Mobile Autonome, navigation, algorithme génétique, champs de potentiel, cibles virtuelles.

Abstract

In this study, we present a state of the art in the field of autonomous mobile robots. Then, we propose two navigation approaches. The first one is based on genetic algorithms and is suitable for robots equipped with local range sensors. The second one is based on a proposal for a normalized potential field with a solution for escape from local minima through a technique based on virtual entities.

Keywords Autonomous Mobile Robots, navigation, genetic algorithms, potential fields, virtual entities.