الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière : Électronique

Spécialité : Electronique Systèmes des Systèmes embarquées

Présenté par

Nom et prénom :

ZADOUERKEB BATOUL

Titre du mémoire

DEVELOPPEMENT D'UN PROGRAMME ECRIT SOUS LOGICIEL MATLAB POUR LA NAVIGATION AUTONOME D'UNE STRUCTURE MOBILE

Proposé par : **Monsieur A. BOUNEMRI**

Année Universitaire 2023-2024

Remerciements

Je tiens tout d'abord à remercier mon directeur de mémoire, monsieur

A. BOUNEMERI, pour sa précieuse guidance et son expertise. Sa disponibilité et ses conseils éclairés ont grandement enrichi mon travail de recherche. Je suis reconnaissante d'avoir eu l'opportunité de travailler avec lui.

Je souhaite également exprimer ma gratitude à l'ensemble de l'équipe enseignante de SAAD DAHLEB. Leurs enseignements de qualité, leur passion pour leur domaine et leur dévouement envers leurs étudiants ont été une source d'inspiration constante.

Mes remerciements vont également à ma famille, qui m'a soutenue tout au long de ce parcours exigeant. Leur encouragement, leur soutien moral et leur compréhension ont été des piliers essentiels dans la réalisation de ce mémoire.

Ce mémoire de fin d'études a été une expérience enrichissante et formatrice, et je ne saurais exprimer ma gratitude envers toutes les personnes qui ont joué un rôle dans sa réalisation.

ملخص:

ركزت الدراسة الحالية على التنقل الذاتي لروبوت قادر على اكتشاف وتجنب العقبات باستخدام طريقة الكشف القائمة على التباين بين الأسود والأبيض، المكافئة للطريقة العملية التي تستخدم المستشعرات والعقبات. تم توضيح منهجية تصميم خوارزمية التنقل و بما في ذلك مرحلة التهيئة ومرحلة البحث والوظائف المساعدة. أظهرت نتائج المحاكاة التي أجريت فعالية تنفيذها في برنامج MATLAB. حيث الروبوت قادر على اكتشاف وتجنب العقبات في بيئات متنوعة، بما في ذلك البيئات المعقدة.

كلمات المفاتيح : MATLAB ,الروبوتات، المحاكاة، اكتشاف العوائق، التباين بالأبيض والأسود، تجنب العوائق، الهدف، التوجه

Résumé :

La présente étude a porté sur l'étude de la navigation autonome d'un robot capable de détecter et éviter les obstacles en utilisant la méthode de détection basée sur le contraste noir et blanc, équivalente à l'approche pratique utilisant des capteurs et des obstacles. La méthodologie de conception de l'algorithme de navigation et son implémentation sur MATLAB, incluant la phase d'initialisation la phase de recherche et les fonctions auxiliaires, a été clarifiée .les résultats de simulations réalisées ont montres l'efficacité de la méthode. le robot est capable de détecter et d'éviter les obstacles dans divers environnement, y compris des environnements complexes.

Mots clés : MATLAB , Robotique, Simulation, Détection d'obstacles, Contraste noir et blanc, évitement d'obstacles, cible, orientation.

Abstract :

The present study focused on the autonomous navigation of a robot capable of detecting and avoiding obstacles using the black-and-white contrast detection method, equivalent to the practical approach using sensors and obstacles. The methodology for designing the navigation algorithm and its implementation in MATLAB, including the initialization phase, search phase, and auxiliary functions, was clarified. The results of simulations conducted demonstrated the effectiveness of the method. The robot is capable of detecting and avoiding obstacles in various environments, including complex ones.

Keywords : MATLAB , Robotics , Simulation,Obstacle detection, Black and white contrast ,obstacle avoidance , target , orientation.

Listes des acronymes et abréviations

Table des matières

Liste des figures

Dans le domaine de la robotique mobile, la navigation autonome est un sujet de recherche majeur. Les robots mobiles sont de plus en plus utilisés dans divers domaines, tels que l'exploration spatiale, la logistique en entrepôt, les services de livraison, et les applications domestiques comme les aspirateurs robotisés. La capacité d'un robot à se déplacer de manière autonome dans un environnement inconnu, en évitant les obstacles, est cruciale pour ces applications. L'objectif de ce mémoire de master, en électronique des systèmes embarques, est de développer un programme écrit sous logiciel MATLAB pour la navigation autonome d'une structure mobile. Ce programme utilise une fonction de détection de distance pour détecter l'obstacle, une fonction d'élaboration de la structure sous une forme spécifique, une fonction de conduite basée sur les vitesses des roues de la structure, une fonction principale de recherche, qui représente « le cœur » du travail, et une fonction de navigation pour contourner les obstacles et atteindre un point final connu.

Les algorithmes de navigation autonome sont essentiels pour diverses applications pratiques, améliorant l'efficacité et la sécurité des opérations robotiques.

Travailler sur de tels projets nous permet de développer des compétences en programmation MATLAB, en algorithmique, et en robotique, qui sont très demandées dans le monde industriel et académique. Contribuer à la recherche et au développement dans ce domaine peut mener à des avancées technologiques significatives, facilitant des innovations futures. L'objectif principal de cette étude est donc de développer un algorithme de navigation, c'est-à-dire écrire un programme en MATLAB permettant à une structure mobile de se déplacer de manière autonome dans un environnement avec des obstacles. Ce travail se basera sur les points suivants :

Implémentation de la détection d'obstacles : Utiliser une fonction de détection de distance pour identifier les obstacles.

Conception d'une stratégie de contournement : Élaborer une méthode pour contourner les obstacles détectés et continuer vers le point final.

Validation et tests : Vérifier la performance de l'algorithme dans divers scénarios pour s'assurer de son efficacité et de sa robustesse.

Implémentation en MATLAB : Codage de l'algorithme en MATLAB, incluant toutes les fonctions utilisées.

Simulation et Tests : Utilisation de la simulation pour tester l'algorithme dans divers environnements avec différentes configurations d'obstacles.

Analyse des Résultats : Évaluation des performances de l'algorithme en termes de temps de parcours, efficacité du contournement, et robustesse face à des scénarios variés.

Améliorations : Ajustements et optimisations basées sur les résultats des tests pour améliorer la performance globale du système.

Dans le cadre de ce travail notre mémoire est structuré comme suit :

Une Introduction générale

Le chapitre I : c'est un chapitre relatif à la revue de la Littérature, la navigation autonome en robotique, les techniques de détection d'obstacles et les algorithmes de contournement

Le chapitre II : c'est un chapitre relatif à la méthodologie basée sur la conception de l'algorithme de navigation, son implémentation en MATLAB, la description du code et les explications des fonctions utilisées.

Le chapitre III : c'est un chapitre base sur la simulation des résultats effectues sur divers scénarios.

Conclusion.

CHAPITRE 1 : Navigation autonome, techniques de détection et algorithmes de Contournement

Introduction

La navigation autonome des robots mobiles est un domaine de recherche passionnant qui combine des aspects de robotique, d'intelligence artificielle et de traitement du signal. Dans le cadre de notre mémoire de fin d'études, nous nous intéresserons plus particulièrement aux techniques de détection d'obstacles et aux algorithmes de contournement, qui sont des composantes essentielles pour permettre à un robot mobile de naviguer de manière sûre et efficace dans un environnement inconnu.

I.1 La navigation autonome en robotique mobile

La navigation autonome en robotique mobile est la capacité d'un robot à se déplacer et à accomplir des tâches sans intervention humaine directe, en utilisant des capteurs et des algorithmes pour percevoir l'environnement, planifier un trajet et exécuter des mouvements. Cette compétence est essentielle pour les robots opérant dans des environnements dynamiques et inconnus, tels que les entrepôts, les hôpitaux, ou les espaces domestiques. Par exemple, un robot aspirateur utilise des capteurs infrarouges et des caméras pour détecter les obstacles, tels que les meubles ou les murs, et planifie son parcours pour nettoyer efficacement une pièce. Il doit continuellement ajuster sa trajectoire pour éviter les collisions et s'assurer qu'il couvre toute la surface à nettoyer (figure I-1).

Figure I-1 : Robot aspirateur autonome [1]

Ce type de navigation repose sur des techniques comme la localisation et la cartographie simultanées (SLAM), qui permettent au robot de construire et de mettre à jour une carte de son environnement tout en suivant sa propre position. Un autre exemple est celui des robots utilisés dans les entrepôts, qui se déplacent de manière autonome pour récupérer et transporter des articles. Ils utilisent des capteurs lidar (figure I-2) pour créer une carte 3D de leur environnement et des algorithmes de planification de trajectoire pour naviguer efficacement parmi les étagères et autres robots en mouvement. La navigation autonome est ainsi un domaine crucial en robotique, combinant perception, prise de décision et contrôle, pour permettre aux robots de fonctionner de manière indépendante et fiable dans des environnements variés et complexes.

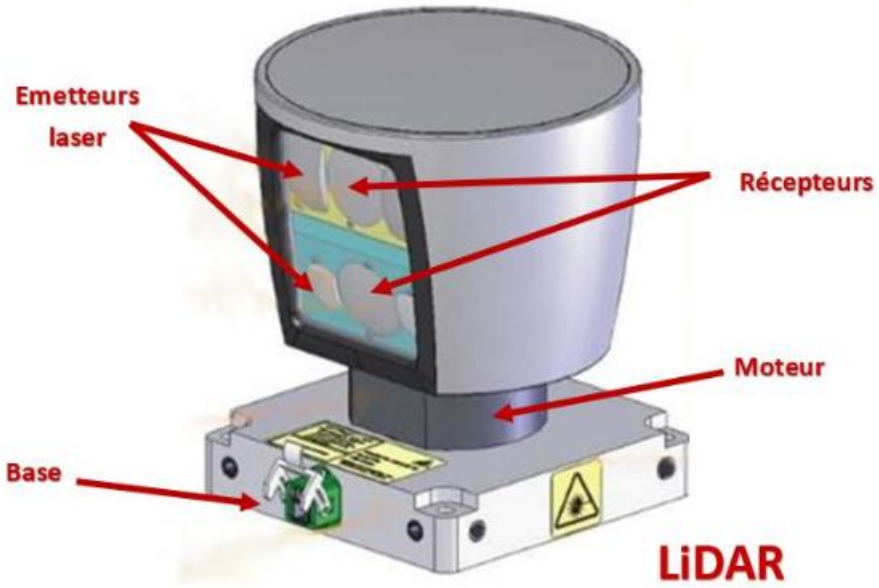


Figure I-2 : Exemple d'un LiDAR utilisé sur les voitures autonomes [2]

I.2 Les techniques de détection d'obstacles en robotique mobile

La détection d'obstacles est une composante cruciale de la navigation autonome en robotique mobile. Plusieurs techniques sont couramment utilisées, chacune avec ses propres avantages et ses inconvénients.

I.2.1 Technique de détection d'obstacles par capteurs ultrasoniques

Les capteurs ultrasoniques émettent des ondes sonores et mesurent le temps de retour de l'écho pour calculer la distance aux obstacles.

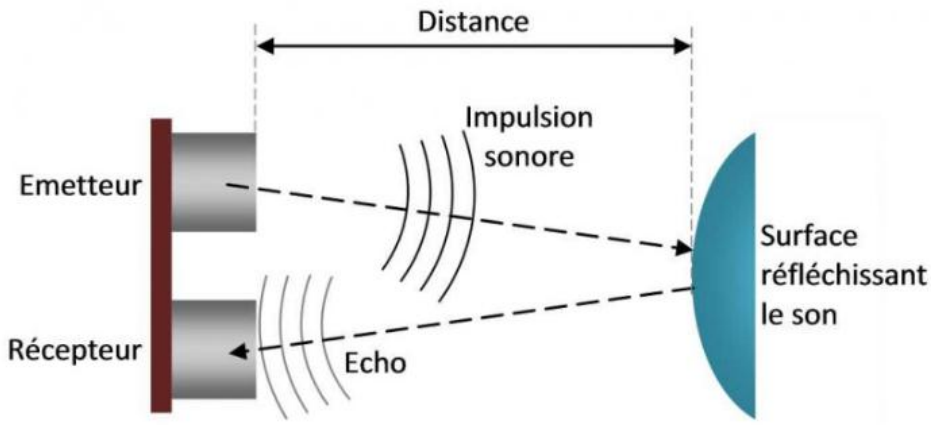


Figure I-3 : Principe de détection d'un objet par un capteur à ultrasons [3]

Un court signal sonore est envoyé (inaudible car dans le domaine des ultrasons – environ 40kHz)

Le son est réfléchi par une surface et repart en direction du capteur : c'est l'écho ;

Une fois revenue à son point de départ, l'onde sonore est détectée par le capteur.

La durée entre l'instant de l'émission et l'instant de la réception peut être mesurée. Le signal ayant parcouru 2 fois la distance entre le capteur et la surface (un aller-retour), on peut la calculer ainsi : distance = (.5) × (vitesse du son) × (durée)

I.2.2 Technique de détection d'obstacles par camera stéréo

Les caméras stéréo utilisent deux caméras (figure I-4) pour obtenir des images légèrement décalées, permettant de calculer la profondeur et la distance aux objets en utilisant la stéréovision. Le principe de fonctionnement est base sur le principe suivant :

- Deux images sont capturées simultanément par deux caméras espacées.
- Les différences entre les deux images (disparité) sont analysées pour calculer la profondeur des objets.

La stéréovision permet de percevoir la profondeur en exploitant la disparité entre deux images capturées par des caméras espacées. En trouvant les correspondances de points entre les deux images et en calculant la disparité, on peut déterminer la distance de chaque point de la scène par rapport aux caméras. Cette technique est particulièrement utile en robotique mobile pour la navigation et l'évitement d'obstacles, car elle permet au robot de reconstruire une carte tridimensionnelle de son environnement et de prendre des décisions basées sur la profondeur des objets détectés.

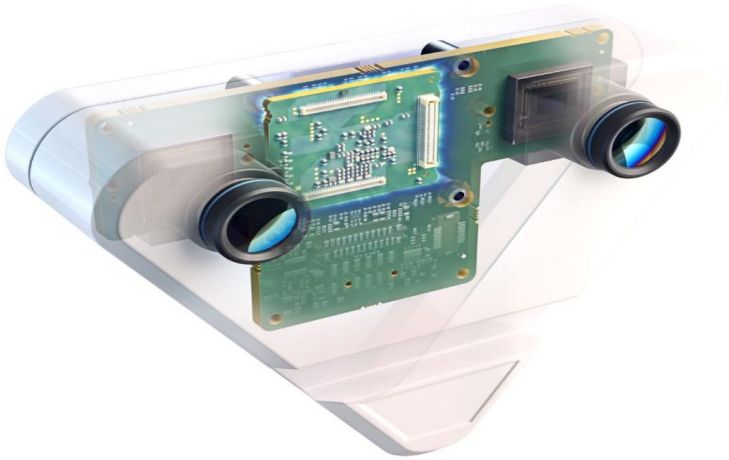


Figure I-4 : Camera stéréo [4]

Le principe de la Stéréovision peut être clarifie en suivant les étapes suivantes :

- Acquisition des Images

Deux caméras (gauche et droite) capturent simultanément des images de la même scène.

Ces caméras sont positionnées avec une distance connue entre elles.

- Correspondance de Points :

Pour chaque point dans l'image de la caméra gauche, le point correspondant dans l'image de la caméra droite doit être trouvé. Cette étape est appelée correspondance de points.

- Calcul de la Disparité :

La disparité est définie comme la différence de position horizontale d'un point dans l'image de gauche et dans l'image de droite.

Si (x_G,y) est la position du point dans l'image gauche et (x_D,y) est la position du point correspondant dans l'image droite, alors la disparité $d = x_G - x_D$

- Calcul de la Profondeur :

Une fois la disparité calculée pour chaque point, la profondeur Z de chaque point peut être déterminée à l'aide de la formule : $Z = (f.B)/d$

f est la distance focale de la caméra (en pixels).

B la distance entre les deux caméras (en mètres ou en unités cohérentes avec f).

d est la disparité calculée pour le point.

La distance focale est une caractéristique essentielle des systèmes optiques, influençant directement le champ de vision et la profondeur de champ. En stéréovision, elle est cruciale pour le calcul de la profondeur à partir des images capturées par des caméras espacées. Comprendre et maîtriser la distance focale permet de développer des systèmes de vision par ordinateur plus précis et efficaces, particulièrement dans des applications de robotique mobile où la navigation autonome et l'évitement des obstacles sont primordiaux.

I.2.3 Technique de détection d'obstacles par capteurs infrarouges

Les capteurs infrarouges (IR) mesurent la distance en émettant une lumière infrarouge et en mesurant l'intensité de la lumière réfléchie. Le principe de fonctionnement est base sur les étapes suivantes :

- Une lumière infrarouge est émise vers un obstacle.
- La lumière réfléchie est captée par un récepteur.
- L'intensité de la lumière réfléchie est utilisée pour estimer la distance à l'obstacle.

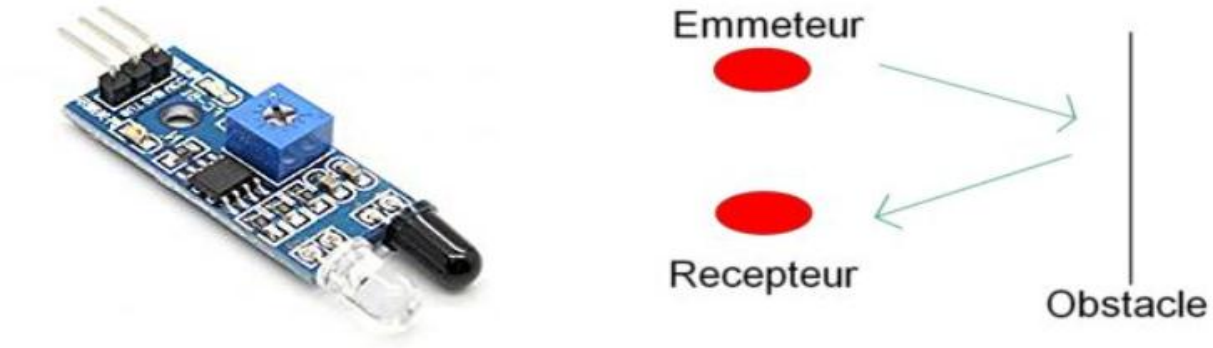


Figure I-5 : Capteur infrarouge – Principe de la détection [5]

I. 3 Les algorithmes de contournement d'obstacles

La robotique mobile se concentre sur le développement de robots capables de se déplacer de manière autonome dans leur environnement. Un aspect crucial de cette autonomie est la capacité à détecter et contourner les obstacles, permettant aux robots de naviguer efficacement et en toute sécurité dans des environnements souvent imprévisibles et dynamiques. Les algorithmes de contournement d'obstacles jouent un rôle central dans cette capacité, en combinant des techniques avancées de perception, de planification de trajectoire et de contrôle de mouvement. Les algorithmes de contournement d'obstacles [6-9] sont essentiels pour plusieurs raisons :

- Sécurité : Ils permettent aux robots d'éviter les collisions, protégeant ainsi non seulement le robot mais aussi les personnes et les objets dans son environnement.
- Efficacité : En évitant les obstacles de manière optimale, les robots peuvent accomplir leurs tâches plus rapidement et avec une consommation d'énergie réduite.
- Adaptabilité : La capacité à naviguer dans des environnements inconnus et changeants est fondamentale pour les applications robotiques, qu'il s'agisse de robots domestiques, de drones ou de véhicules autonomes.

Les algorithmes de contournement d'obstacles peuvent être classés en plusieurs catégories, chacune ayant ses propres avantages et limitations. On peut citer les algorithmes bases sur les

capteurs, les algorithmes basés sur des modèles et les algorithmes bases sur la planification des trajectoires.

Les principaux défis dans le contournement d'obstacles incluent :

- Gestion des Environnements Dynamiques : Adaptation en temps réel aux changements dans l'environnement.
- Précision de la Détection : Minimisation des faux positifs et négatifs pour une détection fiable des obstacles.
- Efficacité des Algorithmes : Implémentation des algorithmes dans des systèmes embarqués avec des ressources limitées.

I.3.1 Algorithme de Bug 1

L'algorithme de Bug 1 (Voir Annexe) fonctionne de la manière suivante :

- Le robot se dirige en ligne droite vers la cible jusqu'à rencontrer un obstacle.
- Ensuite, il suit le contour de l'obstacle jusqu'à retrouver le chemin le plus proche de la cible.
- Il contourne complètement l'obstacle avant de reprendre sa trajectoire vers la cible.

I.3.1 Algorithme de Bug 2

L'algorithme de Bug 2 est similaire au Bug 1 (Voir Annexe), mais avec une différence clé :

- Le robot se dirige vers la cible jusqu'à rencontrer un obstacle.
- Ensuite, il suit le contour de l'obstacle jusqu'à trouver un point où la ligne droite vers la cible est à nouveau dégagée.

I.3.1 Algorithme de Tangent Bug

L'algorithme de Tangent Bug (voir Annexe) utilise des capteurs pour mesurer la distance aux obstacles et contourne l'obstacle le plus proche en suivant sa tangente jusqu'à ce qu'un chemin dégagé vers la cible soit trouvé.

L'illustration ci-dessous (figure I-6) est une application simplifiée des trois algorithmes. Cet exemple nous permet de voir que la cible de coordonnées (10,10) est atteinte par application de l'algorithme Bug 1. Par contre l'algorithme Bug 2 reste bloqué au niveau du point de coordonnées (3,3) et la cible n'est pas atteinte. L'algorithme Tangent Bug permet quant à lui aussi d'atteindre la cible. L'efficacité des algorithmes n'est « en générale » jamais garantie pour tous les scenarios.

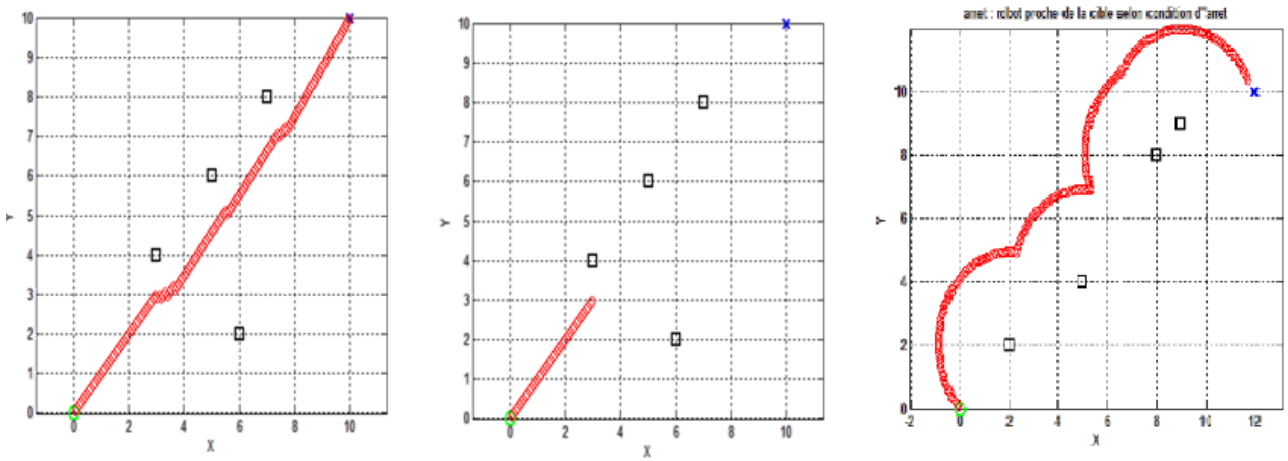


Figure I-6 : Application des algorithmes Bug 1, Bug 2 et Tangent Bug

Conclusion

La détection d'obstacles est une compétence fondamentale pour la navigation autonome en robotique mobile, permettant aux robots de percevoir et de réagir à leur environnement en temps réel. Les techniques de détection d'obstacles les plus couramment utilisées incluent les capteurs ultrasoniques, les capteurs infrarouges et les caméras stéréo, chacune offrant des avantages spécifiques en termes de précision, coût et complexité.

Les capteurs ultrasoniques et infrarouges sont populaires pour leur simplicité et leur coût réduit, bien qu'ils soient généralement limités en portée et précision. Les caméras stéréo permettent de reconstruire des scènes en trois dimensions en utilisant la stéréovision, où la disparité entre deux images capturées par des caméras espacées est analysée pour calculer la profondeur des objets.

La distance focale joue un rôle crucial dans la stéréovision, influençant directement la précision du calcul de la profondeur. Comprendre et utiliser correctement cette caractéristique permet de développer des systèmes de vision par ordinateur plus efficaces.

Ainsi, la compréhension et l'application des différentes techniques de détection d'obstacles sont cruciales pour le développement de systèmes robotiques capables de naviguer de manière autonome, en évitant efficacement les obstacles pour atteindre leurs objectifs.

CHAPITRE 2 : METHODOLOGIE BASEE SUR LA CONCEPTION DE L'ALGORITHME DE NAVIGATION ET SON IMPLEMENTATION SUR MATLAB

Introduction

Le développement d'un système de navigation autonome pour un robot mobile nécessite une approche méthodologique rigoureuse et une mise en œuvre efficace. Dans ce chapitre, nous nous concentrons sur la conception et l'implémentation d'un algorithme de navigation pour un robot mobile, en utilisant le langage de programmation MATLAB. Notre objectif est de créer un système capable de localiser et de se déplacer dans un environnement inconnu, tout en évitant les obstacles et atteignant sa destination souhaitée.

II.1 Conception de l'algorithme de navigation

Dans le domaine de la robotique mobile, la capacité d'un robot à naviguer de manière autonome dans un environnement inconnu est une compétence essentielle. Le sujet de ce mémoire, de master, se concentre sur le développement d'un programme en MATLAB pour permettre à une structure mobile de se déplacer d'un point initial défini par l'utilisateur à un point final spécifié aussi par l'utilisateur, en évitant les obstacles rencontrés en chemin.

La navigation autonome repose sur plusieurs composants clés, notamment la détection d'obstacles et leur évitement. Dans ce projet, nous formulerons un code MATLAB qui intègre ces éléments dans des fonctions. Le programme commence par positionner la structure mobile au point initial fourni par l'utilisateur. Il définit également un point final que la structure mobile doit atteindre.

Une fois ces points définis, le programme principal appelle une fonction de recherche, qui joue un rôle central dans la navigation autonome. Cette fonction de recherche, soutenue par d'autres fonctions spécifiques, guide la structure mobile à travers l'environnement en utilisant des techniques de détection pour identifier et contourner les obstacles.

L'environnement est considéré comme inconnu, ce qui signifie que la structure mobile ne dispose pas d'une carte préétablie des positions des obstacles. Elle doit s'appuyer sur des capteurs pour détecter les obstacles en temps réel et adapter sa trajectoire en conséquence. Ce processus

dynamique d'exploration et d'évitement d'obstacles est crucial pour atteindre le point final de manière efficace et sûre. D'une manière générale l'organigramme définissant l'algorithme est établi comme suit (figure II.1) :

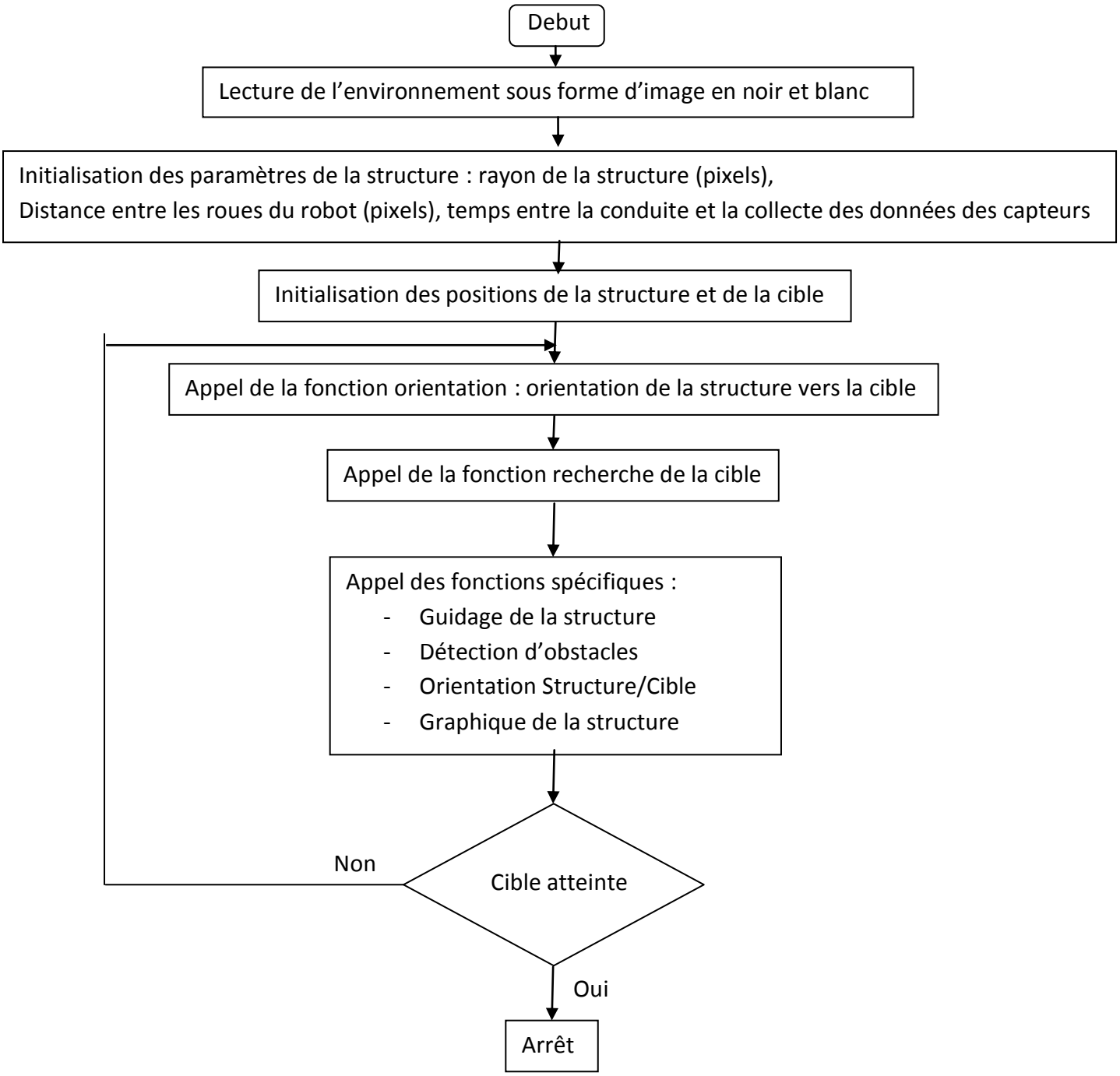


Figure II-1 : Organigramme définissant l'algorithme de navigation.

II.2 Phase initialisation

Cette partie de l'algorithme (programme principal) permet de définir l'initialisation des paramètres de la structure mobile qui sont définis comme suit :

```
rad  = 4;                % Dimension de la structure mobile
wdia = 7;               % Dimensions entre les roues de la structure mobile
dt   = .5;                % Temps entre la conduite et la collecte des donnees
course = rgb2gray(imread('test4.png'));  % Lecture de l'environnement, du robot,
                                         % nomme test4
imagesc(course), axis image on; colormap gray;
```

Les deux lignes dernières du pseudo-code permettent de lire l'image définissant l'environnement ou doit évoluer la structure mobile. L'environnement est représenté par une image en noir et blanc. Cette notion est utilisée car elle est à la base de la fonction de détection des obstacles. En effet cette dernière se base sur le contraste entre le fond de l'image, en blanc, et l'obstacle, en noir pour pouvoir détecter l'obstacle et calculer a quelle distance il se situe de la structure.
Une partie interactive, dans le programme principal, permet d'établir l'initialisation de la position de la structure mobile et de la cible (point d'arrivée de la structure mobile). Cette partie est assurée par les instructions suivantes :

```
%
title('Initialisation de la position initial de la structure mobile');
%Initialisation
%collecte des coordonnees initiales de la structure mobile (vecteur posn)
[posn(1) posn(2)] = ginput(1);
drawbot1(posn,rad, course, 0, posc,poso);
%
title('Click to specify robots inital heading');
%collecte  des coordonnees de la cible
[y x] = ginput(1);
% Orientation de la structure mobile dans la direction de la cible
cart2pol(x,y);
```

L'instruction gin put(1) de Matlab, permet d'assurer l'action interactive avec l'utilisateur qui peut positionner à la demande (apparition du titre) et la structure et la cible. Un exemple d'initialisation dans un environnement est illustre en figure II.2. La figure II.3, illustre l'orientation de la structure mobile vers la direction de la cible, assurée par l'instruction cart2pol(x,y).

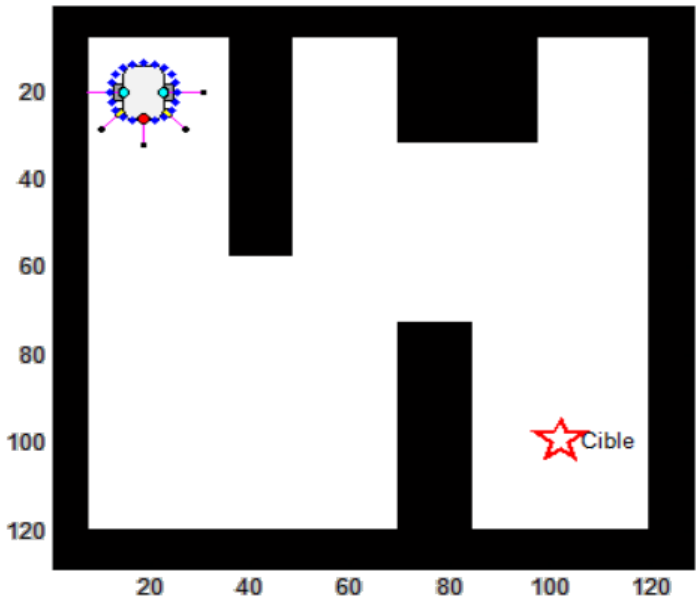


Figure II-2 : Initialisation de la structure mobile et de la cible

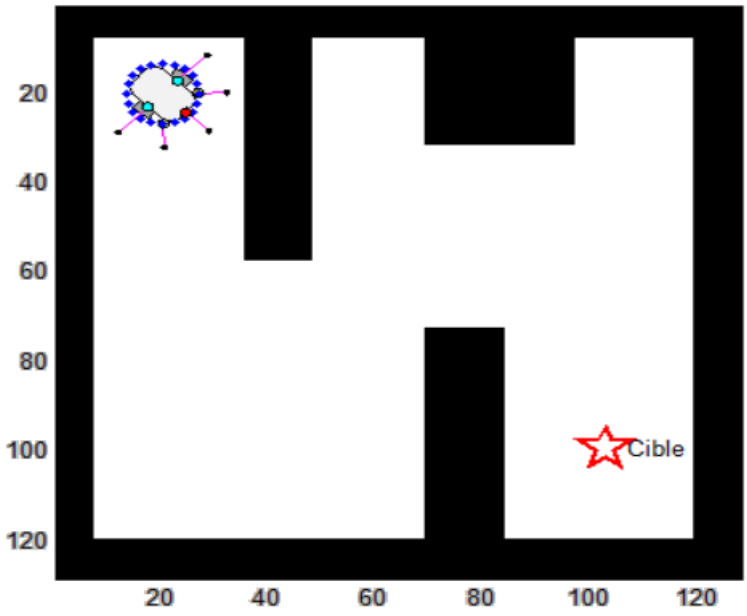


Figure II-3 : Orientation de la structure mobile selon la direction de la cible

II.2.1 Fonction de représentation de la structure mobile

La fonction qu'on a programmée, dans Matlab, pour l'illustration de la structure mobile sur une image (l'environnement) peut être résumée selon l'organigramme suivant (figure II.4) :

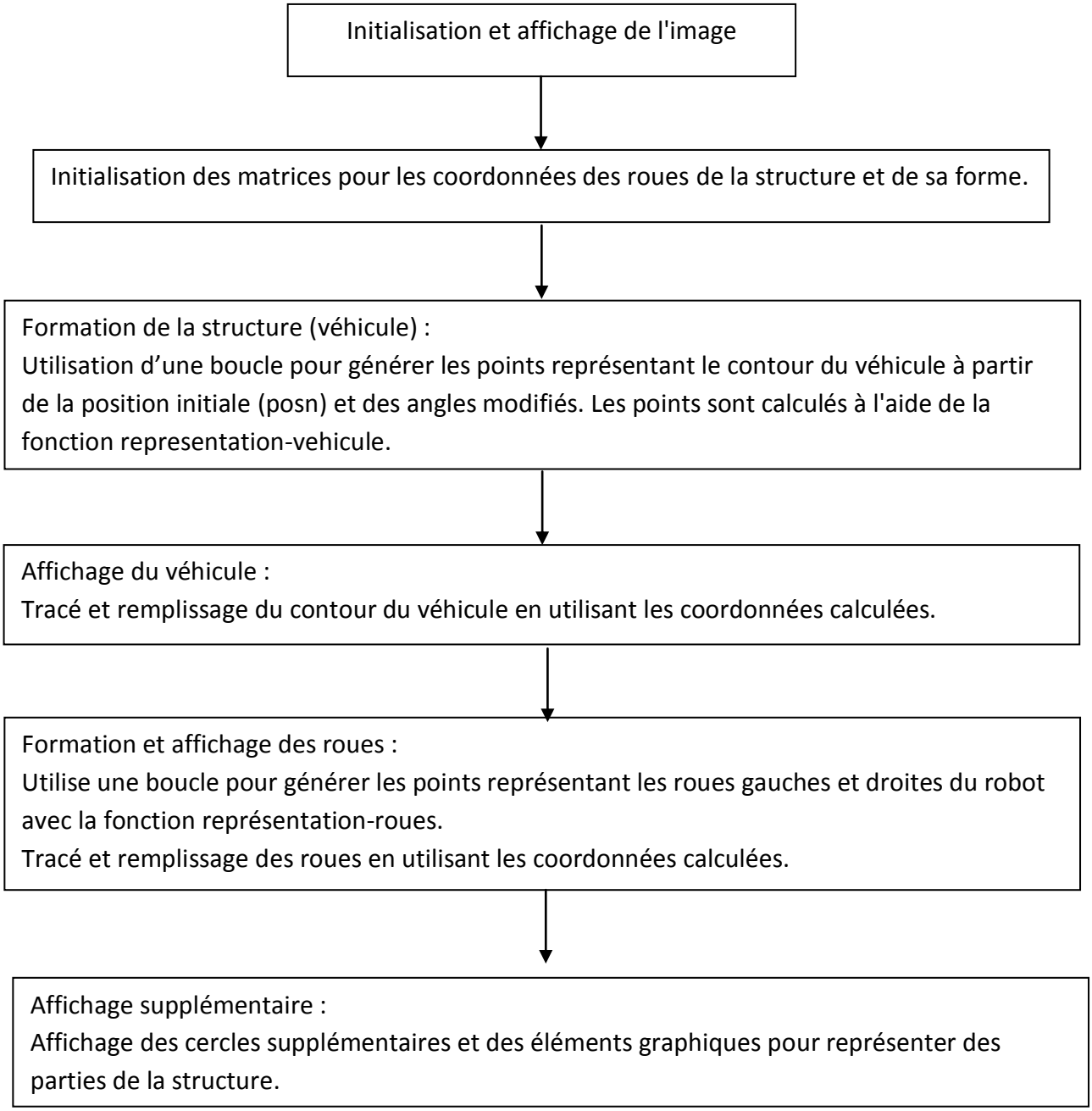


Figure II-4 : Organigramme de la fonction illustration de la structure mobile sur une image

En résumé, la fonction représentation de la structure mobile visualise un véhicule et ses composants (véhicule et roues) sur une image donnée en utilisant des coordonnées et des transformations géométriques basées sur des angles et des rayons. Elle intègre également la possibilité d'afficher une cible si spécifié.

II.3 Phase de recherche

La fonction de recherche est conçue pour simuler une structure mobile se déplaçant dans un environnement avec des obstacles. Cette fonction se base sur l'utilisation de capteurs simulés pour mesurer les distances aux obstacles et ajuster les vitesses des roues pour éviter les collisions et se diriger vers une cible. La boucle principale de la fonction gère le déplacement en ajustant constamment la direction de la structure en fonction des lectures des capteurs. En cas de détection d'obstacles proches, la structure change de direction pour éviter les collisions. La fonction inclut également des mécanismes pour arrêter la structure lorsqu'elle est proche de la cible.

La stratégie principale repose sur la lecture des distances aux obstacles à différents angles, l'ajustement des vitesses des roues pour éviter les obstacles, et le recalcule constant de la position de la structure. Des fonctions auxiliaires comme 'distance à l'obstacle', 'conduite' et 'détection de collision' sont utilisées pour les calculs de distances, les mises à jour de position, et la détection des collisions, respectivement.

La description visuelle de cette fonction peut être illustrée selon l'organigramme présenté en figure II.5. L'organigramme visualise le flux de contrôle et les décisions principales prises par la fonction de recherche, permettant une compréhension claire du processus de déplacement et d'évitement d'obstacles du robot.

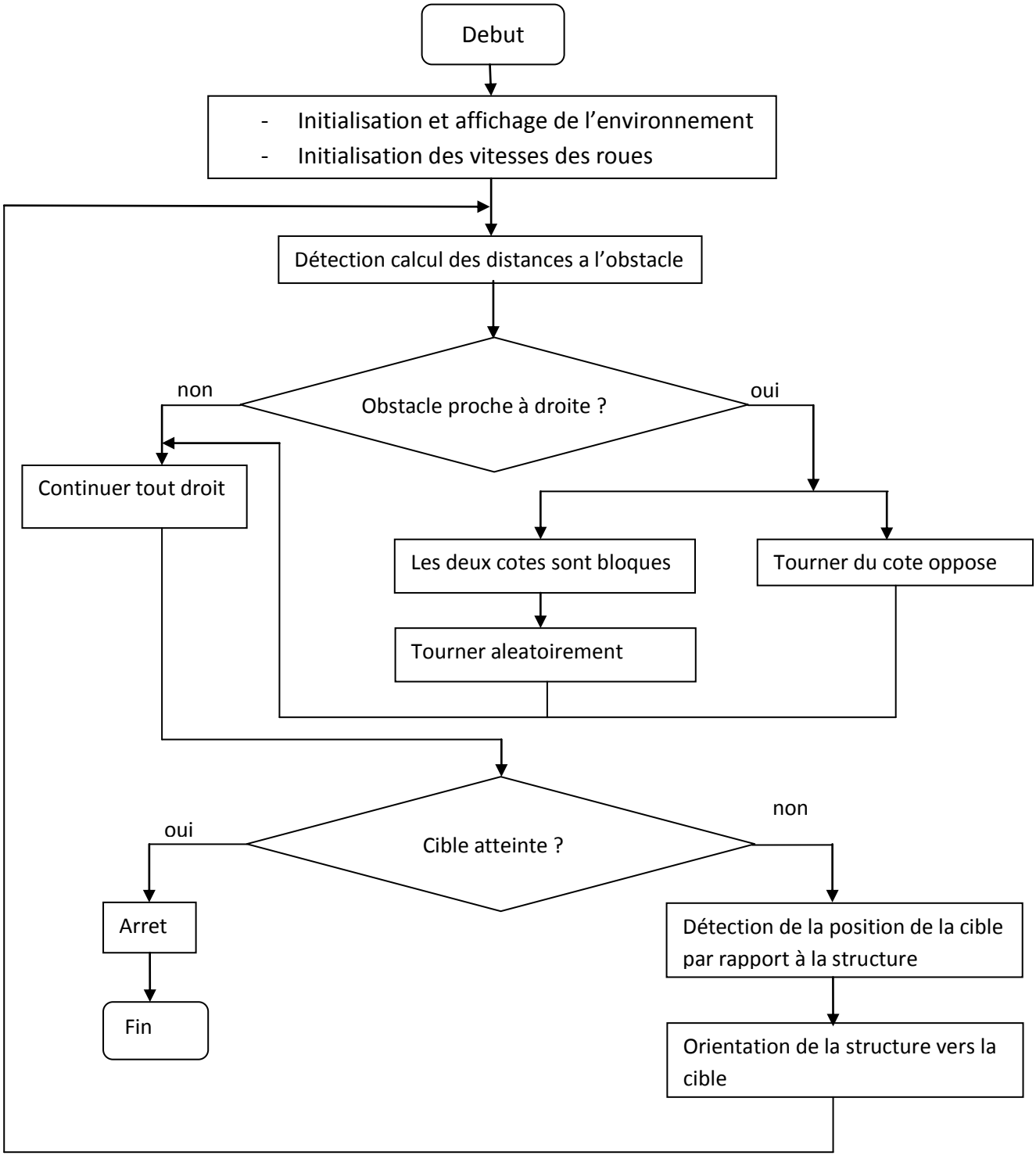


Figure II-5 : Organigramme de la fonction de recherche

II.3.1 Fonction auxiliaire ‘distance à l’obstacle’

La fonction ‘distance à l’obstacle’ en MATLAB simule un capteur de distance pour une structure mobile. L’utilisation de cette fonction peut être résumée selon les étapes suivantes :

a/ Initialisation :

La fonction prend en entrée la position actuelle du robot (posn), le rayon du robot (rad), et la matrice du parcours (course) où les obstacles sont représentés par des pixels de valeur 0.

b/ Boucle de Détection :

Pour une plage de distances allant de rad à rad + 100 pixels, la fonction calcule les coordonnées (x, y) du point à tester basé sur l'angle actuel du robot (posn(3)).

c/ Validation des Coordonnées :

Les coordonnées (x, y) sont ajustées pour s'assurer qu'elles restent dans les limites de la matrice course.

d/ Détection d'Obstacles :

Si le point testé est un obstacle (pixel de valeur 0), la fonction retourne la distance depuis le bord du robot jusqu'à cet obstacle.

e/ Distance par Défaut :

Si aucun obstacle n'est détecté dans un rayon de 100 pixels, la fonction retourne une distance par défaut de 100.

La figure II.6, illustre un calcul de distance entre la structure mobile et un obstacle. Le calcul se fait par rapport au capteur central et aux deux capteurs latéraux, positionnes par rapport au capteur central. Les notions de gauche et de droite sont relatives à des directions par rapport à un individu à l’intérieur de la structure dont l’orientation est vers l’avant du véhicule (exemple d’un conducteur de voiture). On a donc :

Distance capteur central – obstacle = 55 pixels.

Distance capteur latéral droit – obstacle = 44 pixels

Distance capteur latéral gauche – obstacle = 41 pixels

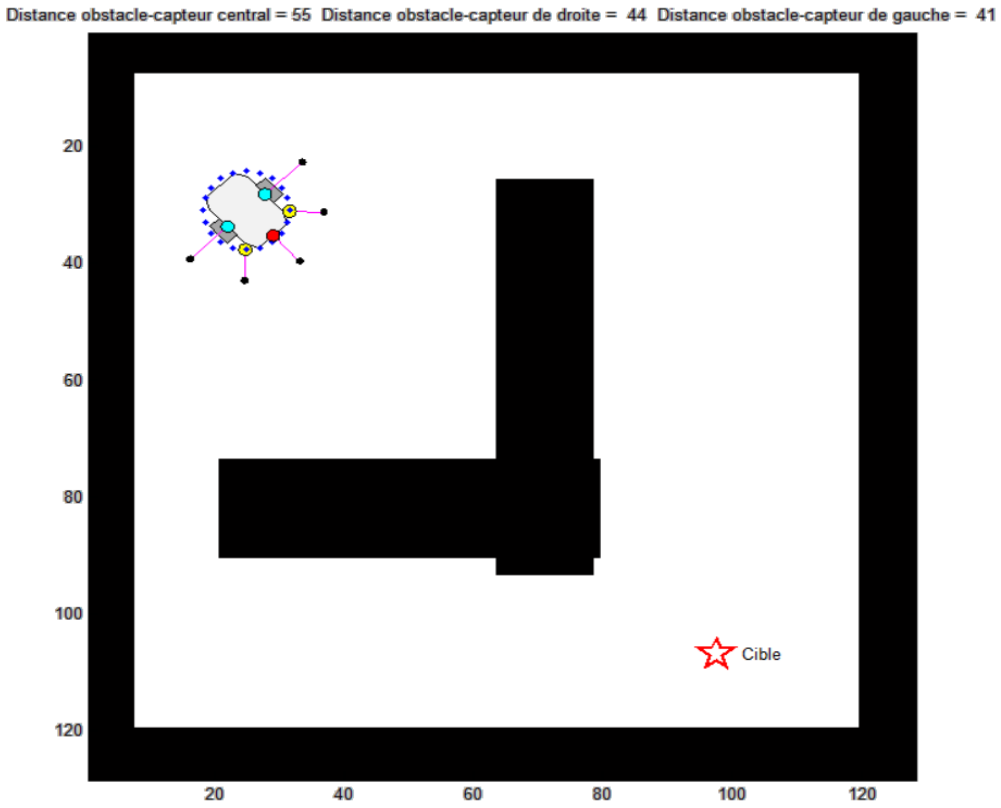


Figure II-6 : Illustration d'un calcul de distance entre la structure mobile et un obstacle.

II.3.2 Fonction auxiliaire 'conduite'

La fonction 'conduite' détermine la nouvelle position de la structure à deux roues après un temps dt, en fonction des vitesses des roues gauche et droite. Si les roues tournent à la même vitesse, la structure se déplace en ligne droite. Sinon, elle suit une trajectoire « courbee ». La fonction prend en compte la distance entre les roues (wdia) pour ajuster l'angle et les coordonnées de la structure. Elle peut être résumée comme suit :

a/ Calcul de la différence et de la somme des vitesses des roues.

b/ Mise à jour de l'angle d'orientation.

c/ Calcul des nouvelles coordonnées :

- Ligne droite si les vitesses des roues sont égales.
- Trajectoire circulaire si les vitesses des roues sont différentes.

La fonction retourne la nouvelle position du robot en termes de coordonnées [y,x] et d'angle d'orientation θ (posn(3)).

Un code de la fonction conduite peut être écrit comme suit :

```
function [np] = conduite(posn, wdia, vg, vd, t)
 % evaluation de la difference et de la somme des vitesses
vdiff = vd-vg;
vsum = vd+vg;
 % calcul du nouveau angle d'orientation
np(3) = posn(3) + vdiff*t/wdia;

if(vdiff == 0)
    % calcul des coordonnees [y x] dans le cas où les roues tournent
    % à la meme vitesse.
    np(1) = vg*t*sin(posn(3))+posn(1);
    np(2) = vd*t*cos(posn(3))+posn(2);
else
    % calcul des coordonnees [y x] dans le cas où les roues tournent
    % à des vitesses differentes.
np(1) = posn(1)- wdia*vsum/(2*vdiff)*(cos(vdiff*t/wdia+posn(3))- cos(posn(3)));
np(2) = posn(2) + wdia*vsum/(2*vdiff)*(sin(vdiff*t/wdia+posn(3))- sin(posn(3)));
end
```

Conclusion

Ce chapitre a présenté une méthodologie détaillée pour la conception et la mise en œuvre d'un algorithme de navigation à l'aide de MATLAB. La conception de l'algorithme se concentre sur une approche structurée, englobant des phases clés telles que l'initialisation, la recherche et l'évitement d'obstacles.

La phase d'initialisation pose les bases en définissant la représentation de la structure mobile, ouvrant la voie aux tâches de navigation ultérieures. La phase de recherche exploite des fonctions auxiliaires telles que « distance à l'obstacle » et « conduite » pour guider efficacement la structure mobile.

Ce chapitre a fourni un aperçu complet du développement de l'algorithme, mettant en évidence ses composants et fonctionnalités clés. L'implémentation de cet algorithme dans MATLAB offre un cadre robuste pour naviguer dans des structures mobiles dans des environnements complexes.

Des recherches plus approfondies pourraient explorer l'intégration de données de capteurs avancées, la détection d'obstacles en temps réel et des stratégies de planification de trajectoire adaptative pour améliorer les performances et l'adaptabilité de l'algorithme

Introduction

Après avoir conçu et implémenté l'algorithme de navigation pour un robot mobile dans le chapitre précédent, il est maintenant temps de présenter les résultats obtenus et de les discuter. Ce chapitre sera consacré à l'évaluation des performances de notre système de navigation dans différents environnements, allant des plus simples aux plus complexes. Nous comparerons également notre approche à d'autres méthodes existantes afin de mettre en évidence les avantages et les limites de notre solution.

III.1 Présentation des résultats

Dans le cadre de l'application du programme conçu, nous avons utilisé un ensemble de scenarios que nous exposons, dans ce chapitre, comme résultats.

III.1.1 Environnement sans obstacles sur la direction structure mobile – cible

Dans ce cas-là nous avons utilisé l'environnement illustre en figure III-1. Cet environnement illustre une direction structure mobile – cible, sans obstacles. Les obstacles qui apparaissent dans cet environnement ne nécessitent aucun contournement de la part de la structure mobile. Le chemin entre la structure mobile et la cible parait donc, pour la structure mobile, sans obstacles. rrtLa structure se déplacera en ligne droite sur la direction initiée au départ lors de son orientation

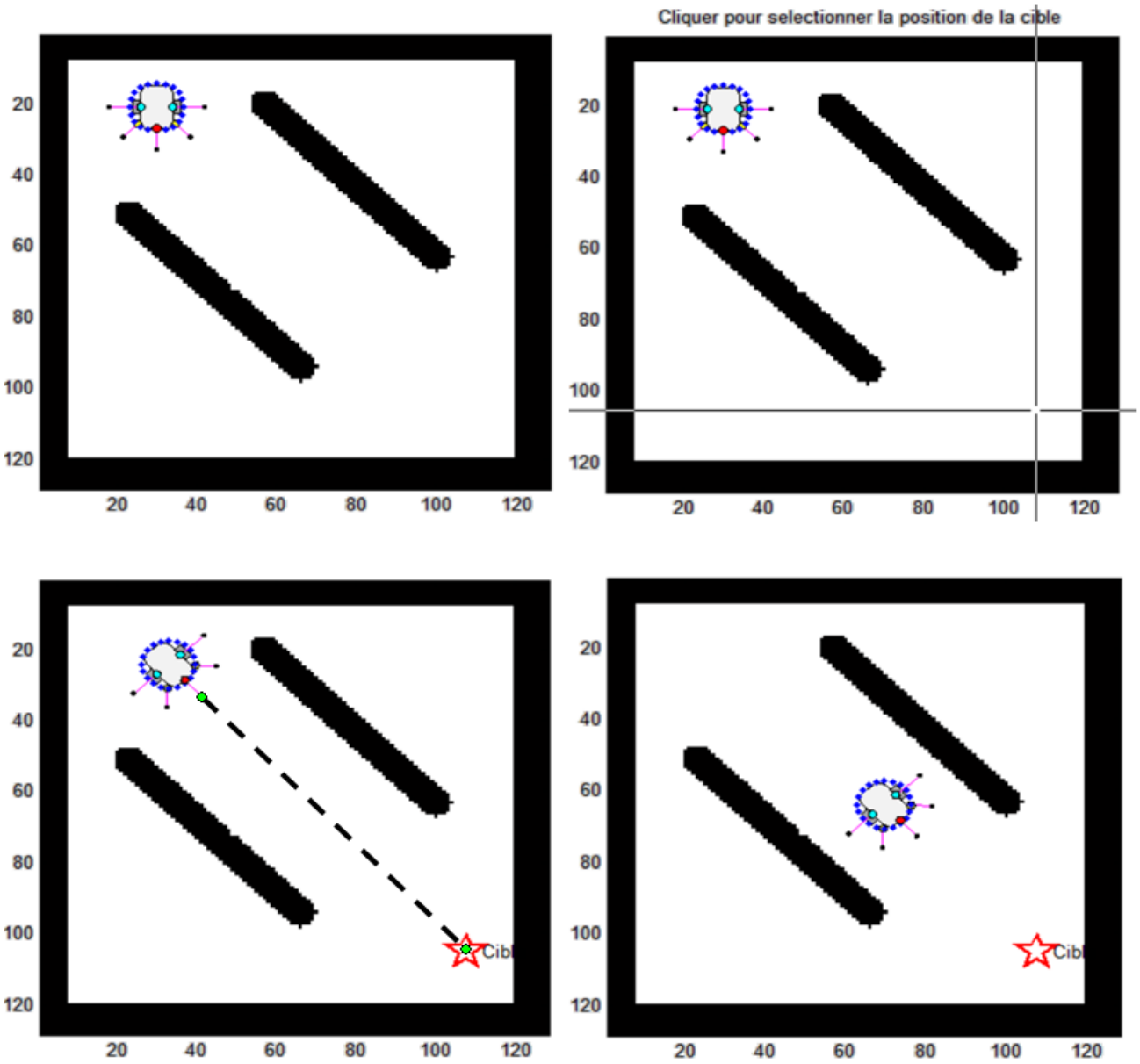
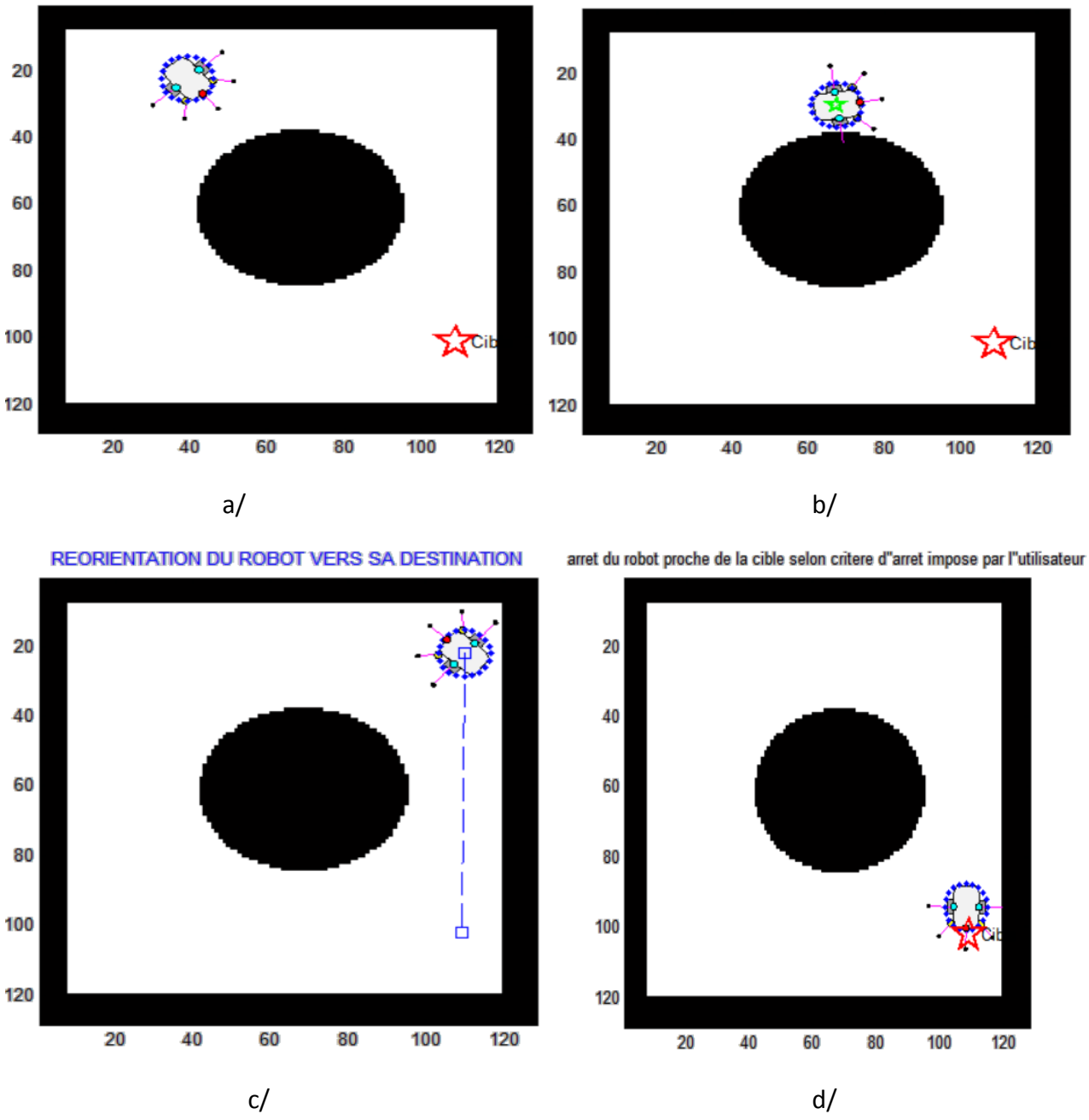


Figure III-1 : Parcours sans obstacles a contourner entre la structure mobile et la cible

Le temps alloué à cette opération sur un ordinateur personnel dont les caractéristiques sont :
Intel(R) Core TM i-7-4610M CPU @ 3.00 GHz - RAM 8.00 Go
est d'environ 0,75 minutes.

III.1.2 Environnement avec obstacle circulaire sur la direction structure mobile – cible

L'environnement pris comme exemple dans ce cas est illustré comme suit en figure III-2



a/ b/

c/ d/

Figure III-2 : Environnement avec obstacle circulaire sur la direction structure mobile – cible

La structure mobile en se déplaçant dans la direction d'initialisation (partie a/ de la figure III-2), rencontre l'obstacle et commence l'opération de contournement selon les directives de l'algorithme (partie b/ de la figure III-2). La structure se déplaçant alors en ligne droite arrive directement au niveau du coin du cadre (limites de l'environnement d'évolution partie c/ de la figure II-2).

Nous avons introduit notre propre stratégie qui permet à la structure mobile de se réorienter vers la cible pour ne pas « errer » aléatoirement dans l'espace environnement. Cette stratégie consiste à comparer les coordonnées, en abscisse ou en ordonnée, de la structure mobile avec celle de la cible. Cette comparaison est basée sur un calcul d'erreur des coordonnées (en x ou en y) exprimée par :

```
erx = abs(posn(1) - posc(1));
ery = abs(posn(2) - posc(2));
```

Ou erx représente l'erreur entre les abscisses et ery représente l'erreur entre les ordonnées.

Dans le cas de cet exemple la structure compare son abscisse par rapport à celle de la cible.
Si erx < 2 (valeur fixée par le programmateur selon les dimensions de l'environnement), elle s'arrête et se réoriente vers la cible dont la direction est maintenant connue.

Cette orientation lui permet de se déplacer dans cette direction vers la cible. Finalement comme le parcours, dans cette direction est libre, sans obstacles, la structure arrive à atteindre la cible (partie d/ de la figure III-2).
Le temps alloué à cette opération sur un ordinateur personnel dont les caractéristiques sont :
Intel(R) Core TM i-7-4610M CPU @ 3.00 GHz - RAM 8.00 Go
est d'environ 1,45 minutes.

III.1.3 Environnement avec un nombre d'obstacles circulaires sur la direction structure mobile – cible

L'illustration en figure III-3 représente ce scenario :

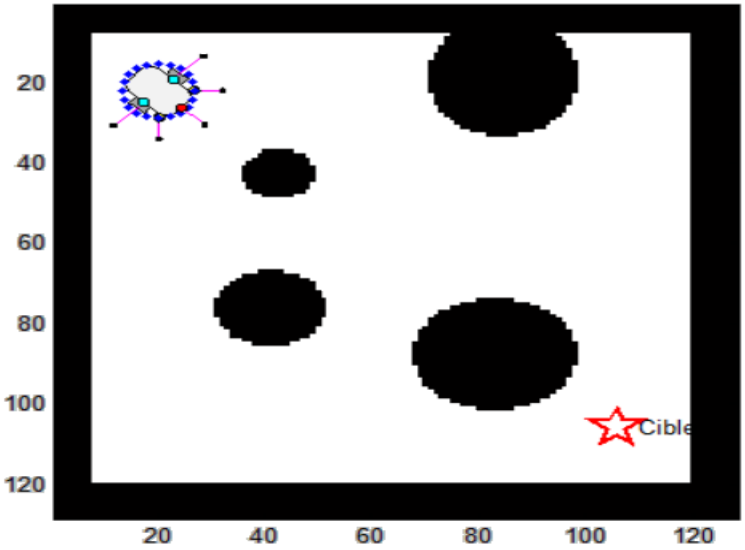
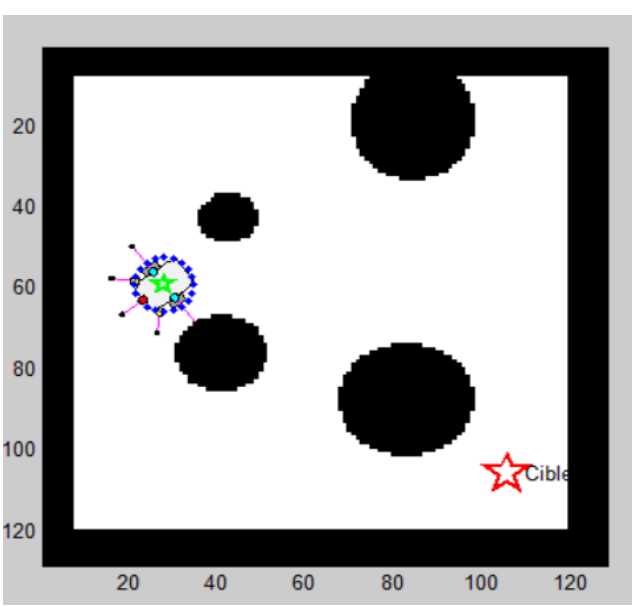
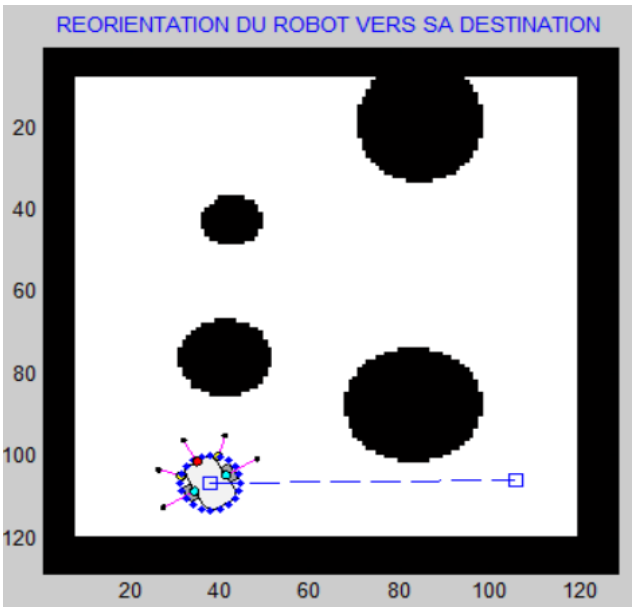


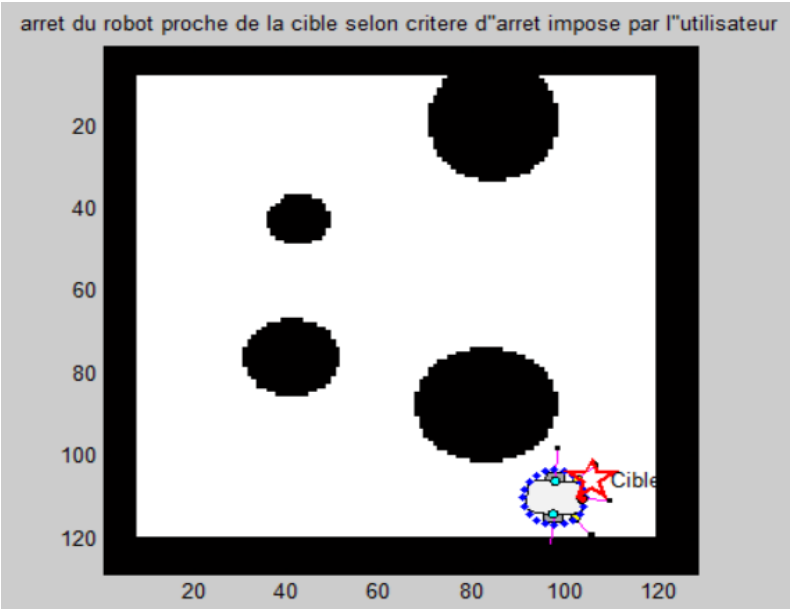
Figure III-3 : Environnement avec un nombre d'obstacles circulaires sur la direction structure mobile – cible



a/



b/



c/

Figure III-4 : Stratégie de déplacement avec contournement et arrivée a la cible

La structure mobile après son orientation vers la cible (initialisation) se dirige vers cette dernière dans la direction linéaire structure mobile-cible. L'environnement comprenant des obstacles dans cette direction, la structure se « faufile » pour réaliser des contournements d'obstacles (partie a/ de la figure III-4). La stratégie de réorientation de la structure mobile vers la cible se base, dans ce cas d'exemple, sur le calcul de l'erreur ery (partie b/ de la figure III-4). Finalement la structure suit la nouvelle direction vers la cible, chemin en ligne droite (partie c/ de la figure III-4) est l'atteint au bout d'un temps approximé a 20,23 minutes.

III.1.4 Environnement complexe

Le scenario considéré dans cet exemple est tel qu'illustré en figure III-5. Le qualificatif complexe est utilisé ici car l'obstacle central a une dimension qui couvre une zone assez large de l'environnement. Dans ce cas-là les manœuvres pour la structure mobile vont prendre du temps pour pouvoir atteindre la cible.

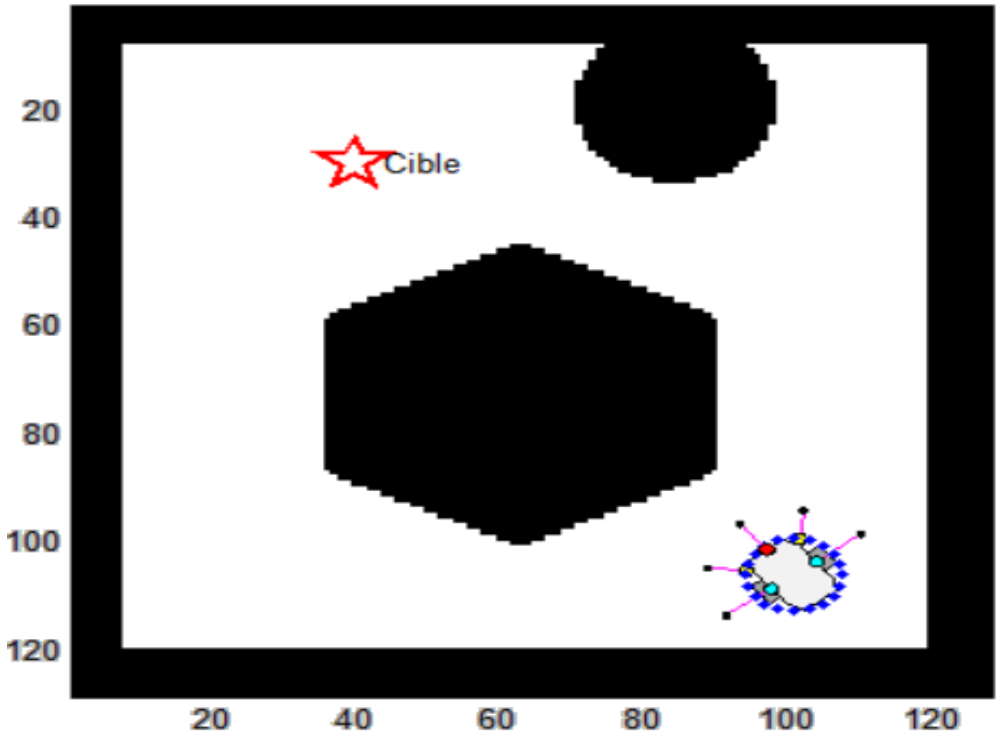
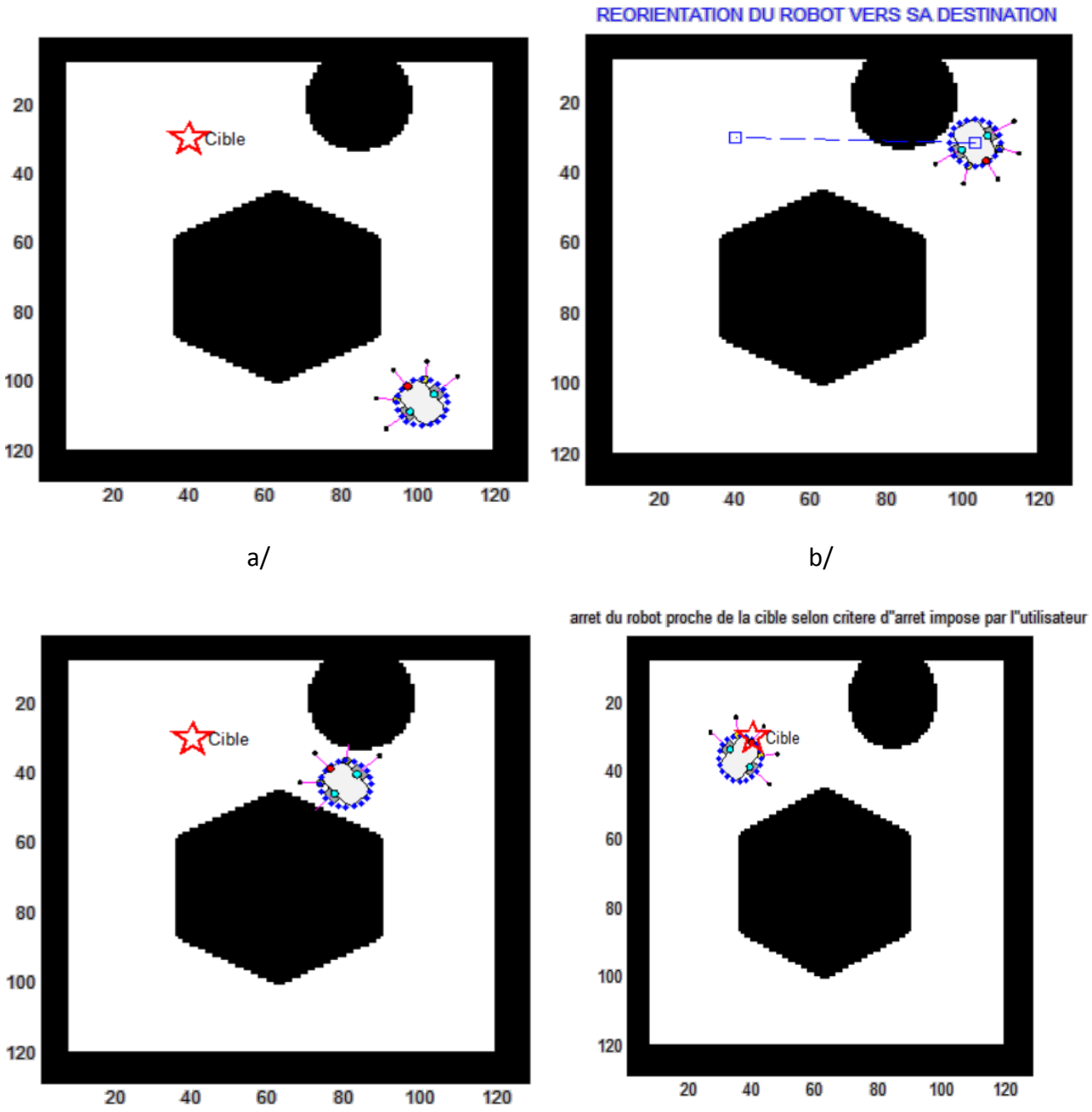


Figure III-5 : Dimension de l'obstacle couvrant une zone large de l'environnement

L'illustration en figure III-6 permet de voir quelques étapes des manœuvres opérées par la structure mobile, pour atteindre la cible.



a/ b/

c/ d/

Figure III-6 Quelques étapes des manœuvres opérées par la structure mobile, pour atteindre la cible.

L'illustration en figure III-6, partie c/, nous permet de constater que la structure mobile se trouve dans une région sous forme de couloir. Dans ce cas-là les manœuvres d'évitement de collision sont tel que le mouvement de la structure mobile est un mouvement de déplacement tantôt à droite tantôt à gauche. La cible est atteinte après un (qui n'apparait pas sur l'illustration) du haut vers le bas puis du bas vers le haut (donc vers la cible). Cet ensemble de manœuvres est effectué en un temps approximatif de 7,4 minutes.

Une seconde tentative de simulation sur le même scenario, nous a permis de constater que le parcours s'est fait différemment. La cible est atteinte dès la sortie de la structure mobile du couloir. Le temps d'exécution n'est que de 1,76667 minutes (figure III-7)

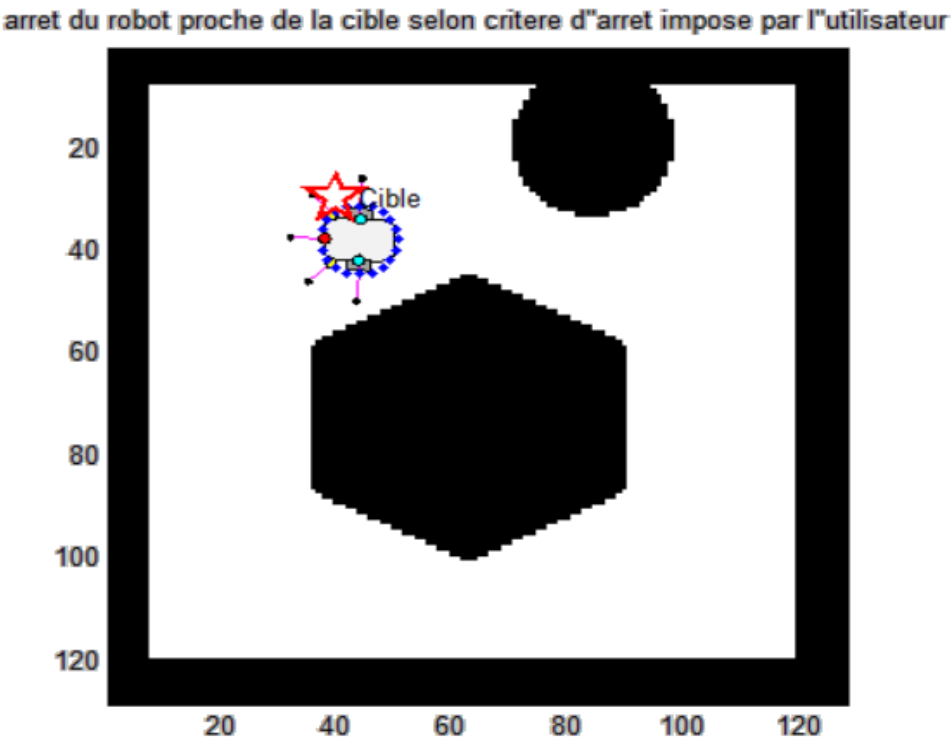
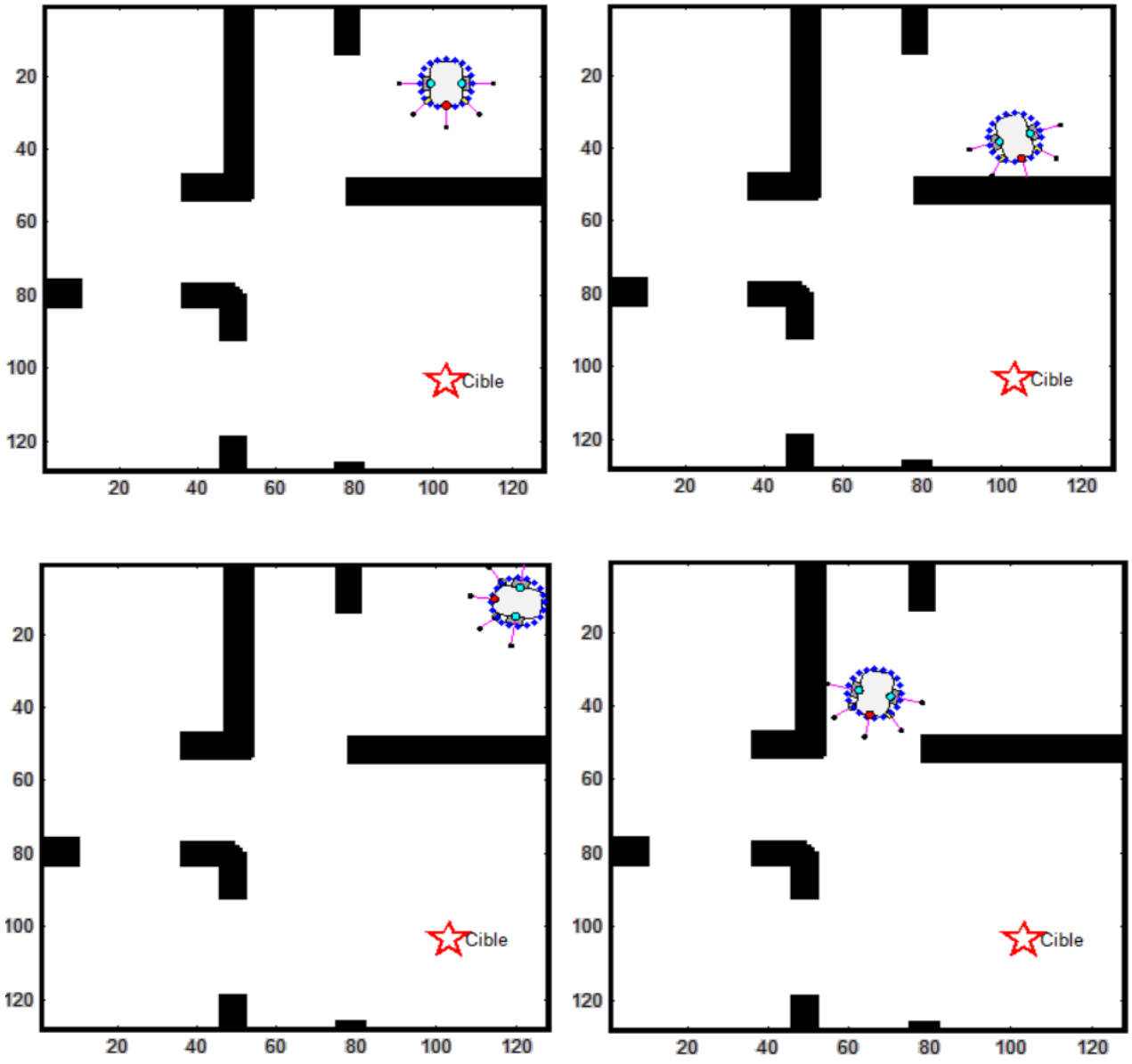


Figure III-7 : Atteinte de la cible grâce à un parcours raccourci

Le mouvement de déplacement de la structure mobile dans le couloir est géré par un ensemble de « fluctuations » gauche-droite, pour éviter les obstacles. Le dernier mouvement permettant à la structure d'être a la sortie du couloir l'entrainera en ligne droite vers la cible. Cet état fait que le parcours est raccourci par rapport au raccourci précèdent (figure III-6). Il y a d'autres paramètres qui entraine que pour le même scenario le parcours peut être diffèrent. Parmis ces paramètres le temps d'acquisition de l'information (existence d'obstacle ou non), par les capteurs.

III.1.5 Environnement sous forme de locaux pour bureaux

La figure III-8 illustre un cas d'environnement sous forme de locaux pour bureaux. Les étapes de déplacements de la structure mobile y figurent aussi. On peut constater que les instructions définissant la stratégie permettant à la structure mobile de ne pas errer aléatoirement ne sont pas exécutées. La structure atteint le bas de l'environnement et fait une rotation qui lui permet de se déplacer en ligne droite (aucun obstacle). Ce déplacement l'oriente directement vers la cible. Comme il n'y a, dans cette zone, aucun obstacle dans la direction structure mobile-cible, la structure s'arrêtera lorsqu'elle est à proximité de la cible (critère d'arrêt). Le temps d'exécution est de 3,2666667 minutes.



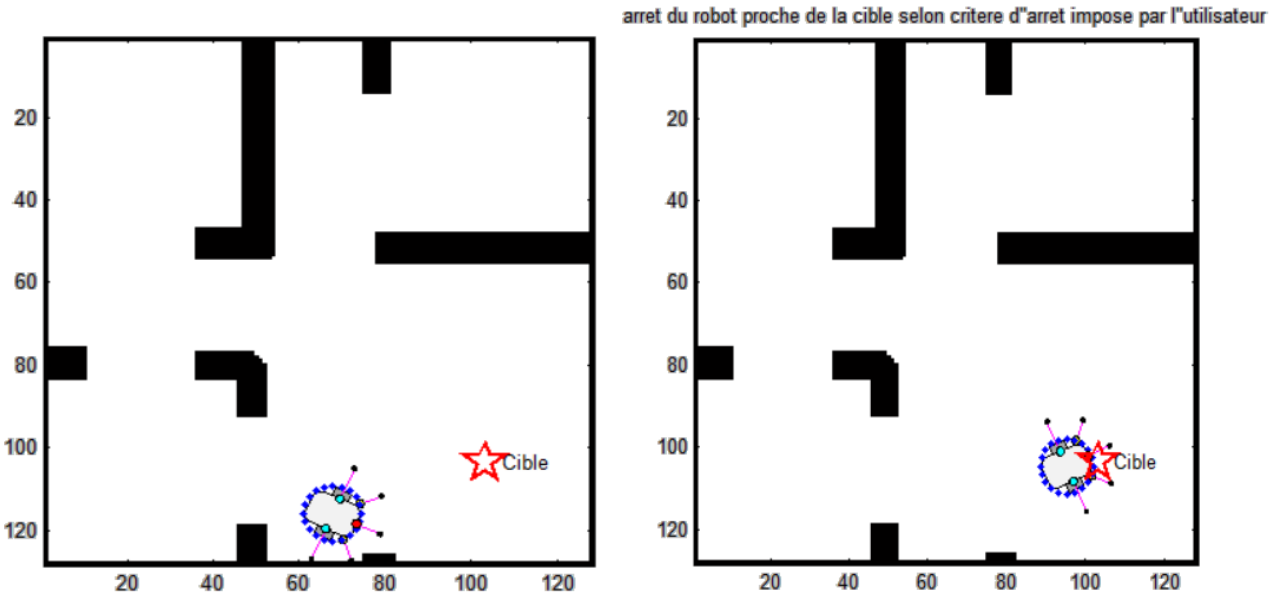


Figure III-8 : Déplacements de la structure mobile dans un environnement sous forme de locaux pour bureaux.

III.2 Comparaison avec d'autres approches

Pour essayer de comprendre si les résultats obtenus dans ce travail sont valides, nous avons décider de faire une comparaison avec d'autres méthodes. Les méthodes ou approches choisies représentent celles qu'on a pu programmer selon le laps de temps alloué à la préparation de ce mémoire. On a pu donc travailler sur les algorithmes suivants :

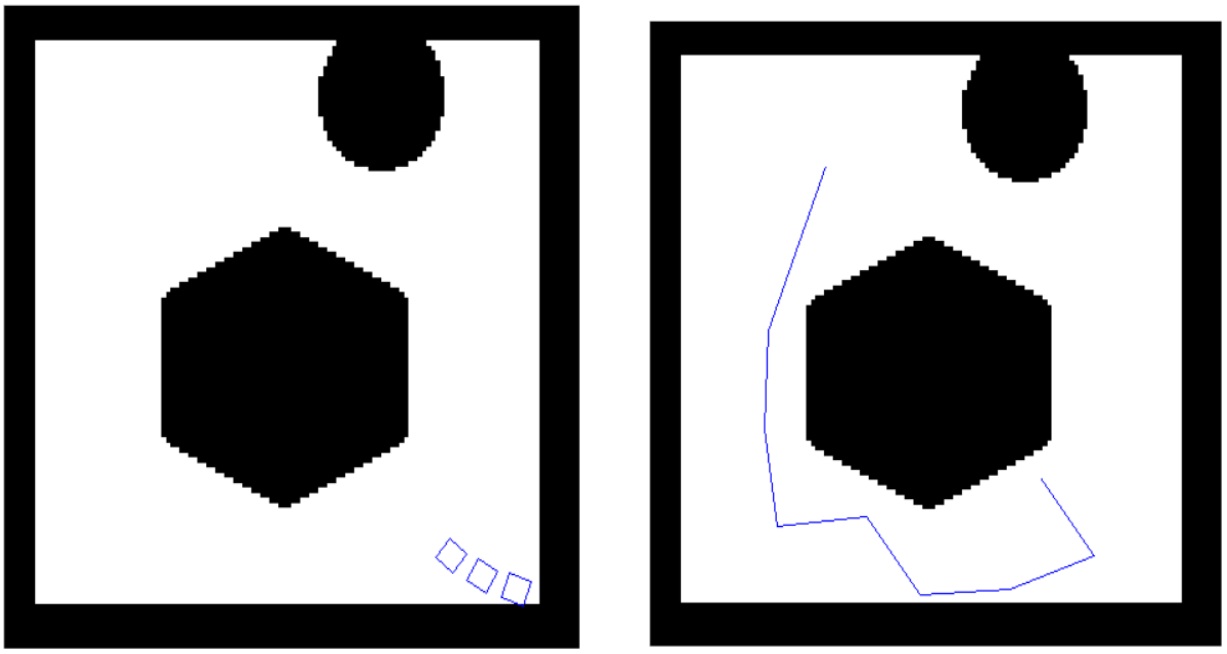
[9] : Siciliano, B., & Khatib, O. (Eds.). (2016). "Springer Handbook of Robotics." Springer.

- Champ des potentiels.
- Exploration rapide d'arbrescence aléatoire (RRT).
- Exploration rapide d'arbrescence aléatoire dans les deux sens (BRRT).

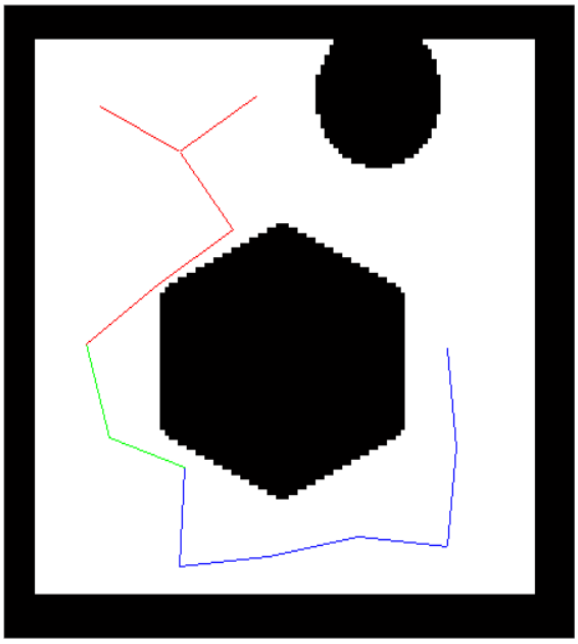
Ces approches définissent en réalité la recherche d'un chemin de parcours, possible, entre le point de départ de la structure mobile et la cible. Pour cela nous avons utilisé deux des scenarios utilises précédemment et nous avons obtenu les résultats illustres en figure III-9 et en figure III-10

La technique du champ de potentiel est la seule parmi les techniques choisies qui ne permet pas d'avoir un chemin de parcours pour le scenario « environnement complexe ». La structure étant relativement proche de l'obstacle, au départ, il y a existence d'un point d'arrêt a partir duquel la

trajectoire n'évolue plus : le chemin entre le point de départ de la structure mobile et la cible ne peut être évalué.



Artificial potential fields **Rapidly-exploring Random Trees**



Bidirectional Rapidly-exploring Random Trees

Figure III-9 : Recherche de chemin par différentes techniques, scenario environnement complexe

Le scenario relatif aux locaux pour bureaux a été utilisé lui aussi pour l'élaboration de la trajectoire par les trois techniques choisies. Les résultats apparaissent sur la figure III-10

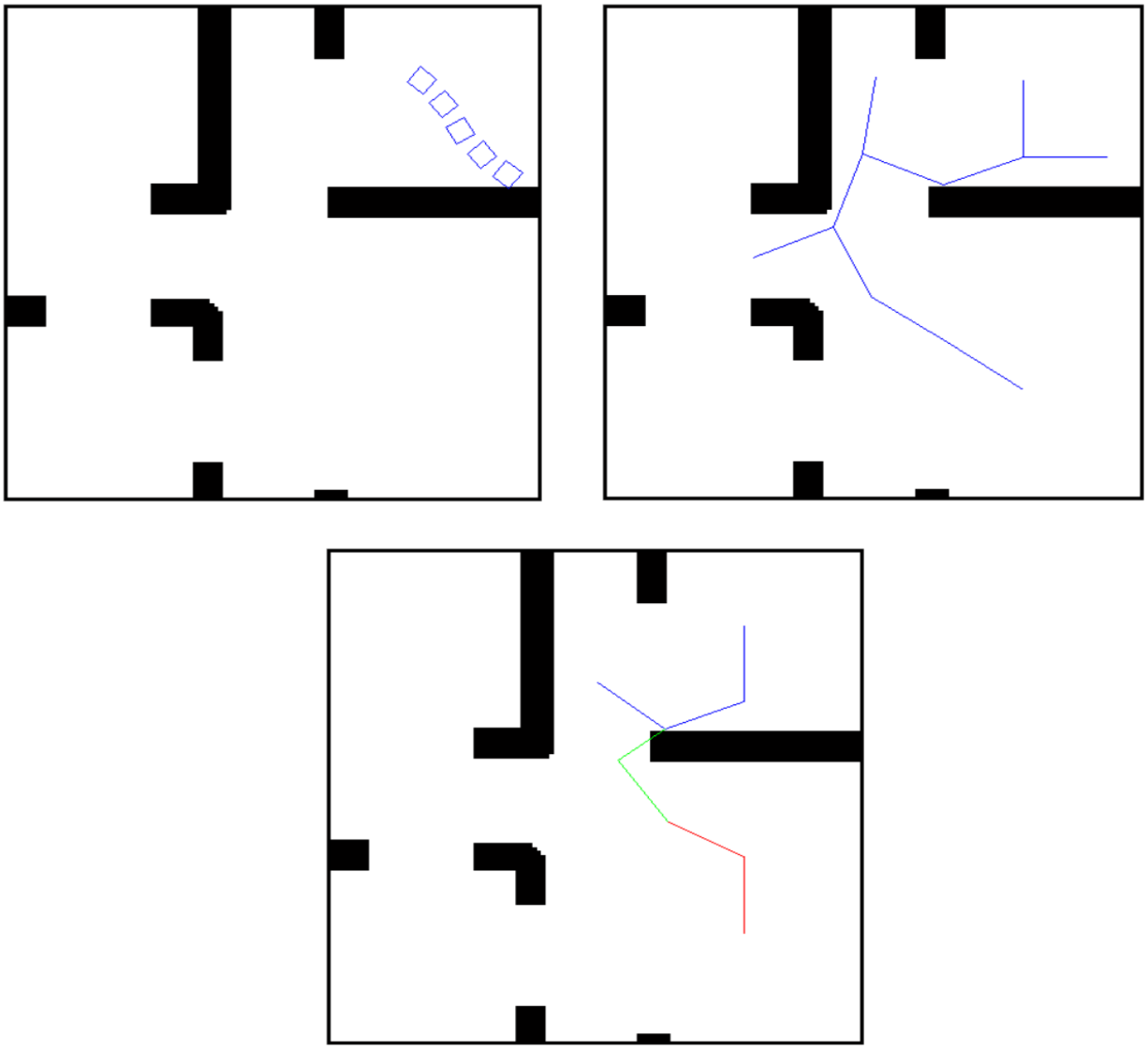


Figure III-10 : Recherche de chemin par différentes techniques, scénario locaux pour bureaux

On dispose des mêmes résultats que précédemment ou la technique du champ de potentiel est la seule qui est incapable d'évalué la trajectoire recherche. On remarque aussi que la technique BRRT ainsi que RRT définissent des trajectoires qui « frôlent » l'obstacle : il y a collision avec l'obstacle.

Conclusion

En résumé, les résultats de cette étude montrent que la méthode de détection basée sur le contraste noir et blanc est efficace et robuste pour détecter et éviter les obstacles, même dans des environnements complexes. Les simulations réalisées dans cet étude ont démontré que le robot est capable de détecter et éviter des obstacles circulaires, des obstacles multiples et des obstacles dans des environnements complexes, tels que des locaux pour bureaux. Cette méthode est également comparable à d'autres approches Ces résultats confirment l'efficacité de la méthode de détection basée sur le contraste noir et blan Un additive d’appproches pour eliminer la perte de temps dans le cas de recherché aleatoire doit obligatoirement incorpore permettant une nette amelioration de cette technique .

La présente étude a porté sur l'étude de la navigation autonome d'un robot capable de détecter et éviter les obstacles en utilisant la méthode de détection basée sur le contraste noir et blanc, équivalente à l'approche pratique utilisant des capteurs et des obstacles. Les chapitres I et II ont présenté les techniques de détection et les algorithmes de contournement utilisés dans la navigation autonome, ainsi que la méthodologie de conception de l'algorithme de navigation et son implémentation sur MATLAB. Les résultats de simulations réalisées dans le chapitre III ont montré que le robot est capable de détecter et éviter les obstacles dans divers environnements, y compris des environnements complexes. Cette étude confirme l'efficacité de la méthode de détection basée sur le contraste noir et blanc pour les applications pratiques telles que la navigation de robots dans des locaux pour bureaux.

```
%%%%%%% Algorithme Bug 1
function bug1(start, goal, obstacles)
    position = start;
    while norm(position - goal) > 0.1
        direction = (goal - position) / norm(goal - position);
        next_position = position + direction * 0.1;
        if ~isCollision(next_position, obstacles)
            position = next_position;
        else
            position = followObstacle(position, direction, obstacles);
        end
        plot(position(1), position(2), 'ro');
        hold on;
    end
    hold off;
end
%%%%%%%
function collision = isCollision(position, obstacles)
    collision = false;
    for i = 1:length(obstacles)
        if norm(position - obstacles(i, :)) < 1
            collision = true;
            return;
        end
    end
end
%%%%%%%
function new_position = followObstacle(position, direction, obstacles)
    while isCollision(position + direction * 0.1, obstacles)
        direction = [direction(2), -direction(1)]; % Change direction
        position = position + direction * 0.1;
    end
    new_position = position;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function bug2(start, goal, obstacles)
    position = start;
    while norm(position - goal) > 0.1
        direction = (goal - position) / norm(goal - position);
```

```
        next_position = position + direction * 0.1;

        if ~isCollision(next_position, obstacles)
            position = next_position;
        else
            position = followObstacleToGoal(position, direction, goal,
obstacles);
        end

        plot(position(1), position(2), 'ro');
        hold on;
    end
    hold off;
end

function collision = isCollision(position, obstacles)
    collision = false;
    for i = 1:length(obstacles)
        if norm(position - obstacles(i, :)) < 1
            collision = true;
            return;
        end
    end
end

function new_position = followObstacleToGoal(position, direction, goal,
obstacles)
    while true
        direction = [direction(2), -direction(1)]; % Change direction
        next_position = position + direction * 0.1;
        if ~isCollision(next_position, obstacles)
            if dot(next_position - goal, goal - position) > 0
                new_position = next_position;
                return;
            end
        end
        position = next_position;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Exemple d'application de l'algorithme de Tangent Bug
start = [0, 0];
goal = [10, 10];
obstacles = [3, 4; 5, 6; 7, 8; 6, 2];
tangentBug(start, goal, obstacles);
function tangentBug(start, goal, obstacles)
    % Paramètres
    sensing_range = 5.0;  % Portée du capteur
    step_size = 0.1;      % Taille du pas de déplacement

    % Position initiale
    position = start;
    plotEnvironment(start, goal, obstacles);

    while norm(position - goal) > 0.1
        % Mesurer la distance aux obstacles
        [min_dist, nearest_obstacle] = measureDistance(position, obstacles,
sensing_range);
```

```
        if min_dist < sensing_range
            % Contourner l'obstacle
            direction = (nearest_obstacle - position) / norm(nearest_obstacle -
position);
            tangent_direction = [-direction(2), direction(1)];  %
Perpendiculaire à l'obstacle
            position = position + tangent_direction * step_size;
        else
            % Se déplacer vers la cible
            direction = (goal - position) / norm(goal - position);
            position = position + direction * step_size;
        end

        % Afficher la position actuelle
        plot(position(1), position(2), 'ro');
        hold on;
        pause(0.1);
    end
    hold off;
end

function [min_dist, nearest_obstacle] = measureDistance(position, obstacles,
sensing_range)
    min_dist = sensing_range;
    nearest_obstacle = position;
    for i = 1:size(obstacles, 1)
        dist = norm(position - obstacles(i, :));
        if dist < min_dist
            min_dist = dist;
            nearest_obstacle = obstacles(i, :);
        end
    end
end

function plotEnvironment(start, goal, obstacles)
    figure;
    plot(start(1), start(2), 'go', 'MarkerSize', 10, 'LineWidth', 2);
    hold on;
    plot(goal(1), goal(2), 'bx', 'MarkerSize', 10, 'LineWidth', 2);
    for i = 1:size(obstacles, 1)
        plot(obstacles(i, 1), obstacles(i, 2), 'ks', 'MarkerSize', 10,
'LineWidth', 2);
    end
    axis equal;
    grid on;
    xlabel('X');
    ylabel('Y');
    title('Tangent Bug Algorithm');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

[1] : https://www.usinenouvelle.com/editorial/un-robot-aspirateur-capable-de-cartographier-votre-maison.N740479 – consulte le 22-05-2024

[2] : Marine du Chapelet, Zifeng fang, Quantin Gautier, Anthony Ho WenTsai, Julie Rouzee, Bachar Salame
Lidar, Etat de l'art, application en perception de l'environnement pour le véhicule autonome. Projet de physique – STP12- 2016-2017

[3] : https://arduino.blaisepascal.fr/capteur-de-distance-a-ultrasons/
Consulte le 22-05-2024

[4] : https://www.edge-ai-vision.com/2022/05/stereo-camera-for-embedded-vision/
Consulte le 22-05-2024

[5] : https://mataucarre.fr/index.php/2017/05/24/capteur-de-proximite-infra-rouge-fc-51-arduino/ - Consulte le 22-05-2024

[6] : Khatib, O. (1986). "Real-time obstacle avoidance for manipulators and mobile robots." The International Journal of Robotics Research, 5(1), 90-98.

[7] : Borenstein, J., & Koren, Y. (1991). "The Vector Field Histogram—Fast Obstacle Avoidance for Mobile Robots." IEEE Transactions on Robotics and Automation, 7(3), 278-288.

[8] : Choset, H. M., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., & Thrun, S. (2005). "Principles of Robot Motion: Theory, Algorithms, and Implementations." MIT Press.

[9] : Siciliano, B., & Khatib, O. (Eds.). (2016). "Springer Handbook of Robotics." Springer.